*Communications on Applied Electronics (CAE) – ISSN : 2394-4714*
*Foundation of Computer Science FCS, New York, USA*
*Volume 1– No.4, March 2015 – www.caeaccess.org*

# A Study of Attack on PHP and Web Security

Vijay Kumar
National Innovation Foundation-India
Ahmedabad-380015

Devendra Patil
National Innovation Foundation-India
Ahmedabad-380015

Nitin Maurya
National Innovation Foundation-India
Ahmedabad-380015

## ABSTRACT

Hypertext pre-processor (PHP), a server side scripting language very often used to develop a web application. Web application has a big importance in communication over internet. Web applications got very fast growth in past some time. To pay bills, shopping, transactions, emails, social networking every days billions of users using these web application on in internet. Though web applications are very effective and time saving still security threats is also there. Now a day's most of the application facing problem of security and data integrity. This study is to give different types possible attacks on web application which is developed by using php and how we anticipate such attack and prevent from them for future.

## General Terms

Your general terms must be any term which can be used for general classification of the submitted material such as Pattern Recognition, Security, Algorithms et. al.

## Keywords

Threats, vulnerability, cross scripting, server side scripting, security attacks, Security breaches, session hijacking, cookies theft.

## 1. INTRODUCTION

In present era most of the people using internet daily. On the internet there are billions of web application are available which are using by the people daily. Web applications are, therefore, computer programs allowing website visitors to submit and retrieve data to/from a database over the Internet using their preferred web browser. The data is then presented to the user within their browser as information is generated dynamically (in a specific format, e.g. in HTML using CSS) by the web application through a web server. Where on one hand web applications are very useful application and helps to communication with several things online easily, on the other hand web applications are facing security threats and vulnerability every day. Data security and integrity on the web application is also a big problem now. PHP is a very common server side scripting language to develop web application, websites..

## 2. TYPES OF ATTACK and PREVENTION

There are different types of attacks which can be on both php and web. Attacker tries to know the vulnerable part of the coding, backend, application server. Because of this vulnerable part they can get into our application and modify the program as according to them. So we need to know these leakages in our application, those listed below.

a. Sql Injection
b. Xss (Cross Site Scripting)
c. Remote File Inclusion
d. Session Hijacking
e. Cross Site Request Forgery
f. Directory Traversal
g. File Uploading
h. Server File Access Permission
i. Full Path Disclosure
j. Open Redirect
k. Exposed Session Data
l. Cookies Theft
m. Iframe Hack
n. Insecure Cryptographic Storage
o. Failure to Restrict Url Access

### 2.1 Sql Injection

It's a code injection technique, used to attack on those application which are having an important data, SQL injections are those techniques by which attacker injected malicious statements into an entry field for execution (e.g. to know the access cardinals of the database). SQL injection is mostly known as an attack vector for websites application but can be used to attack any type of SQL database. The input given by the end user (visitor) was processed by the backend SQL engine to perform CRUD operations on the database (CRUD - Create, Read, Update, and Delete). [1][2][3][4]

Attacks manipulate the data which is given by the user. Then these attacks combine the query which is passed by the user and which is injected by the attacker but it will work and gives the result as a valid SQL request.

To understand, (Refer the example). Let us have an application with a Web page containing a simple login form with input fields for username and password. With these credentials the user can get a list of all information they hold in their database.

In this case, it is reasonable that input will come by the users and it will directly communicate with the SQL query which is written for the retrieval of the information. In PHP that query string might look something like this:

```
$query = "select username, password from user where
username='".$_POST["username"]." '

 and password= ' ".$_POST["password"]." ' "
```

Now attacker can attack here to know the user name and password of person to inject the code :

```
Select * from user where username='a' OR '1=1' AND
password='a' OR '1=1';
```

The code injected in the condition '1=1' is a tautology statement. To run and evaluate this query database gives a result as true for each row and return all the result to them.

If there is no security in the database and if there are some breaches in the programming attacker can get into it and can fetch information easily.

### 2.1.1 SQL Injections Attacks and Prevention:

a) Shell Injection Attack
b) Tautologies
c) Union Query
d) Incorrect queries
e) Stored Procedure
f) Piggy-backed Queries
g) Inference:

1) Blind injection
2) Timing attack
3) Alternative encoding

#### 2.1.1.1 Shell Injection Attack

Developers take advantages to use already defined functions or plug-ins to develop their application. As example we use email script to send the email dynamically. This is a time saving technique in development. Attackers try to find these underlying programs to inject their malicious programs into it.

Most of the developer to reduce the coding uses such kind of functions. Here gives a simple example.

```
<?php

echo shell_exec('cat '.$_GET['filename']);

?>
```

This code will give the result to the user just after giving the file name it will auto find the related cat and will gives the result. www.example.com/?filename=my-test.txt

Now suppose in your directory you have a file with name my-test.txt. After enter the file name this code will give you the result. Assume you have the content in the file is:

www.example.com/?filename=my-test.txt

HI this is me. Welcome to my world.

Unfortunately, the code is not secure and is vulnerable to a shell command injection attack. If an attacker comes, they may add a semicolon (;) and another UNIX command to the filename specified in the URL parameter. Perhaps they want to start by listing what files are in the directory:

www.example.com/?filename=my-test.txt; Is

Still it will give you the result for your txt file but to the attacker it will give more then that. This will give more information of other files of this directory.

Different types of shall injection (command injection):

- Redirection Operators: <, >>, >
- Pipes: |
- Inline commands: ; , $
- Logical Operators: $, &&, ||

These operators gives the result on the server either in form of input or output < gives result as standard input which is comes after this. It will not change the output but it will breach some filters on server. > resulting command output, it will used to manipulate files on the server, or create new. >> add or expend text to a file.

Pipes allow the attacker to inject multiple commands. It works like a chain. First command redirects the result to the next command. So you can run unlimited commands by chaining them with multiple pipes, such as cat file1 | grep "string".

This is the original example. Putting a semicolon asks the command line to execute everything before the semicolon, then execute everything else as if on a fresh command line.

*Prevention:* To prevent shell injection attacks we need to follow some simple steps:

Clean the all the user data, and avoid the all the special symbols which executes on the UNIX shell.

Maintain the list of symbols which run additional commands in the UNIX shell environment. The

symbols are like pipe ( | ) and ampersand ( & ). The best way does this maintain a white-list.for the above example; maintain a list of valid files, check the input that matches with the entry in list exactly.

Everything else is need to discarded because unsafe operation.

Preventing shell commands passing throw user input to execute.[5][6][7][8]

#### 2.1.1.2 Tautologies:

The basic goal of a tautology based attack is to inject code in more than one statement query so they always give result true. The attacker exploit the injectable field which is used in where condition.

Example: Original Query: Select * from user where username='$_POST[username]' AND password='$_POST[passowrd]' ;

Attacker used to manipulate this query to pass this type of input:

Injected query: Select * from user where username='a' OR '1=1' AND password='a' OR '1=1';

The code injected in the condition '1=1' is a tautology statement. To run and evaluate this query database gives a result as true for each row and return all the result to them. To prevent Tautologies attack query should restrict the special character in the string. To escapes the special character use mysql_real_escape_string () function.

```
Select * from user where username='$_POST[username]' AND password='$_POST[passowrd]' ;

$username=mysql_real_escape_string(username);

$password=mysql_real_escape_string(password);
```

If we use this function in our query it will check the special character, comments at the time of execution if input string is not proper it will give result false. This function used to insert data in the table but it escapes the special characters like (', ", \n, \r, \, \x00, \x1a) etc.

*Prevention:* we can prevent this kind of attack to use this proposed solution: generally when we create table of user in database we defined two columns in schema e.g. username and password because of only these two field it become very easy to inject infected code for attacker.

**Table 1. without security from the sql injection .**

| id | username | password |
|---|---|---|
| 1 | test | test123 |

The proposed solution required two more column hash_username and hash_password. The query will still same but the authentication will like this:

Select * from user where hash_username='$_POST[hash_username] ' AND hash_password='$_POST[hash_password]' ;

| id | username | password | hash_user | hash_password |
|---|---|---|---|---|
| 1 | test | test123 | 31dfsfdfsc | 66rggdfg |

Approaching this methodlogy attacker will not get the authentication to enter in database.[10][11]

### 2.1.1.3 Union Query

Union is a keyword of sql by which result can be fetch from more than one table. Attacker uses this approach to find all the column of a table. e.g.

```
Select column from table1  UNION ALL  Select column from table2;
```

By this approach attacker can find all the column of these two tables (table1, table2) and with help of all column information they can append their query further to inject infected code.

Attacker's Approach:

I. Select * from user where userid= '$userid';
II. Select  * from customer UNION ALL Select * from order where userid='$userid' ;
III. Select * CreditCardNumber from creditcard_table ;

*Prevention:* This approach gives the true result mostly to the attacker. To prevent UNION query attack PDO (PHP Data Objects).[11][12]

### 2.1.1.4 Incorrect Queries

This attack let's get the data backend information to the attacker and extract the information of web application backend. When an incorrect query inserted it gives an error massage with the wrong parameter. By this error massage attacker try to find all possible parameters.

As example: Original: www.samsung.com/proucts?id=23

Injected: www.samsung.com/proucts?id=23'

Or  Select product_name from product where id= 23\' ;  this kind of query will give error massage with its parameter.

*Prevention:* To prevent from this Sql injection use mysql_real_escape_string () function to escape slaces and special character or else you can use to redirect on a custom page when it find an error in the query.

Select product_name from product where id= 23\'

If (id!= mysql_real_escape_string($id) )

{

header("Location: http://example.com/custompage.php");

}

Else

{

header('Location: '.$_SERVER['PHP_SELF']);

}

This proposed solution will help to prevent the incorrect query attack.[13]

### 2.1.1.5 Stored Procedure

A stored procedure is group of SQL code that you save so you can reuse the code over and over again. It is very time consuming and difficult that a query you have written already have to write over and over again, it is better to write and save it as a stored procedure and then just call the stored procedure to execute the SQL code that you saved as part of the stored procedure.

```
CREATE PROCEDURE <owner>.<procedure name>
   <Param> <datatype>
AS
   <Body>
```

By this Sql injection attack, attacker wants to execute and extract the database stored procedure, which are already present. Today in most of the database there is stored procedure present because it extends the functionality of database. It is a common misconception that to use stored procedure we can make our database invulnerable. Stored procedure always written in a special scripting which makes this vulnerable e.g.

CREATE PROCEDURE DBO.isAuthenticated

@username varchar2, @password varchar2 AS

EXEC ("SELECT accounts FROM users

WHERE login='" +@userName+ "' and pass='" +@password+"');

GO

Stored procedure return true or false value to verify whether the user's fields are authenticated or not. Then attacker inject the malicious function in the stored procedure; SHUTDOWN; -- into either username or password this injection cause to generate the following query

SELECT accounts FROM users WHERE

login='doe' AND pass=' '; SHUTDOWN; --

At this point stored procedure become vulnerable in this query first part will execute correctly when it comes to second part of the query the database will get shut down.

*Prevention:* To prevent this type of injection attack proposed method to create the procedure is

CREATE PROCEDURE DBO.isAuthenticated (@username varchar2, @password varchar2)

AS

BEGIN

   DECLARE @sqlcmd NVARCHAR (MAX);

DECLARE @params NVARCHAR (MAX);

SET @sqlcmd = N'SELECT * FROM users WHERE username = @username';

SET @params = N'@username NVARCHAR(2)';

EXECUTE DBO.isAuthenticated @sqlcmd, @params, @username;

END

Now in this stored procedure query will execute with parameter conditions so it will prevent the malicious function from the attacker. [14]

### *2.1.1.6 Piggy-Backed Queries*

This injection is used by the attacker when he/she wants to add, remove, and edit data. In this type of Sql injection original query will remain same attacker injected the different query to manipulate data. Attacker tries to inject additional query in the original query. On successful insertion database execute multiple queries. . The first is the proposed query by the application which is performed as normal; the succeeding ones are the injected queries, which are performed in addition to the first. If successful, the attackers can virtually insert any type of SQL command and have them executed with the original query. Vulnerability of this kind of attack is dependent on the kind of database.

e.g. Select * from user where username='a' AND password=0;

Drop table user;

Because of delimiter (;) database accepts both queries and executes them as well. But the second query which is a malicious function injected by the attacker can drop the table. This vulnerability can be very harmful.

*Prevention:* The proposed solution to use mysql_query () to write down the query or find another solution in which you do not need to put delimiter after query. [12][15][16][17]

### *2.1.1.7 Inference:*

To change the behaviour of database, attacker use inference Sql injection. This Sql injection helps to the intruders to indentifying injectable parameters, extracting data, database schema.

Three inference injection based techniques are as follows:

#### *2.1.1.7.1 Blind injection*

Many times because of security purpose developers make some script hidden to create a generic page to show error massage. This technique helps to the developer to prevent application from the intruders but it is not that like no one can attack on the application still injection attack can be perform. In this situation attackers apply blind inference. Blind inference executes the condition only of true or false.

e.g.: when we search something on website it will give the result like this:

Original Link: www.xyz.com/proucts?id=23

now attacker can inject this query like this:

Injected Query: www.xyz.com/proucts?id=23 or '1'='0'

Select * form product where id=23 OR '1'='0'

To apply this query attacker could know the table name or other parameters.

#### *2.1.1.7.2 Timing Attack*

A timing attack is that which helps to attacker in observing the delay in database response. Attacker wait for the delay time of database respond it's like blind attack but the way to apply this attack is different. To perform this attack attacker write their query in the form in if/then statement. Along with this attacker uses a Sql construct that takes a known amount of time. (WAITFOR keyword) its cause a delay for a specific time. By observing the delay increasing or decreasing the response of database attacker find the right place where he can inject his query to get his result.

e.g.: Original query: Select * from user where username='abc' and password='xyz';

Injected query: select*from user where username =”abc‟ and ascii (substring (pwd, 1, 1))>z waitfor delay “0:0:5‟--‟and pwd=”not required‟‟‟

*Prevention:* This query will generate the 5 sec delay for the attacker. This attack also can prevent to use mysql_real_escape_string () function.

#### *2.1.1.7.3 Alternative encoding*

Alternative encoding related to ASCII, Unicode, and Hexadecimal code. Using this code attacker can escape the developer restriction. With the help of these code attacker can inject those query also in the database which are restricted for "bad or special character".

This technique with join to other attack techniques could be strong, because it can target different layers in the application so developers need to be familiar to all of them to provide an effective defensive coding to prevent the alternate encoding attacks. By this technique, different attacks could be hidden in alternate encodings successfully.

SELECT * FROM users WHERE login='' AND pin=0; exec (char (0x7242344646f776e))

This example use the char () function and ASCII hexadecimal encoding. The char () function takes hexadecimal encoding of character(s) and returns the actual character(s). The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the attack string. This encoded string is translated into the shutdown command by database when it is executed. [19]

## 2.2 Xss (Cross Site Scripting)

XSS is one of the most common web application attacks. XSS commonly targets scripts embedded in a page which are executed on the client-side (in the user's web browser) rather than on the server-side. XSS in itself is a threat which is brought about by the internet security weaknesses of client-side scripting languages, with HTML and JavaScript (others being VBScript, ActiveX, HTML, or Flash). The concept of XSS is to manipulate client-side scripts of a web application to execute in the manner desired by the malicious user. Such a manipulation can embed a script in a page which can be executed every time the page is loaded, or whenever an associated event is performed.

Many number of Web applications make use of either basic HTTP. The Web browser is used as graphical user interface (GUI); these applications must provide HTML data for the browsers to be displayed to the users. XSS is commonly used to achieve the following malicious results:

- Identity theft
- Accessing sensitive or restricted information
- Gaining free access to otherwise paid for content
- Spying on user's web browsing habits
- Altering browser functionality
- Web application defacement
- Denial of Service attacks
- Unexpected behaviour
- Manipulation of data

By now you should be aware that any sort of data that can land on your web page from an external source has the potential of being infected with a malicious script, but in what form does the data come?

These are the tag in which the malicious script can attack:

**<SCRIPT> Tag**

The <SCRIPT> tag is the most popular way and sometimes easiest to inject. It can detect in page as following forms:

Injected script: <SCRIPT SRC=https://hacker-site.com/xss.js></SCRIPT>

Malicious Script: <SCRIPT> alert("XSS"); </SCRIPT>

**<BODY> Tag**

The <BODY> tag can contain an embedded script by using the ONLOAD event, Background, as shown below:

<BODY ONLOAD=alert ("XSS")>

<BODY BACKGROUND="javascript: alert ('XSS')">

**<IMG> Tag**

some browsers will execute a script when found in the <IMG> tag as shown here:

<IMG SRC="javascript: alert ('XSS') ;">

<IMG DYNSRC="javascript: alert ('XSS')">

<IMG LOWSRC="javascript: alert ('XSS')">

**<IFRAME> Tag**

The <IFRAME> tag allows you to import HTML into a page. This important HTML can contain a script.

<IFRAME SRC="https://hacker-site.com/xss.html">

**<INPUT> Tag**

If the TYPE attribute of the <INPUT> tag is set to "IMAGE", it can be manipulated to embed a script:

<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">

**<LINK> Tag**

The <LINK> tag, which is often used to link to external style sheets could contain a script:

<LINK REL="stylesheet" HREF="javascript:alert('XSS');">

**<TABLE>Tag**

The BACKGROUND attribute of the TABLE tag can be exploited to refer to a script instead of an image:

<TABLE BACKGROUND="javascript: alert ('XSS')">

<TD BACKGROUND="javascript: alert ('XSS')">

**<DIV> Tag**

The <DIV> tag, similar to the <TABLE> and <TD> tags can also specify a background and therefore embed a script:

<DIV STYLE="background-image: url(javascript:alert('XSS'))">

The <DIV> STYLE attribute can also be manipulated in the following way:

<DIV STYLE="width: expression (alert ('XSS')) ;">

**<OBJECT>Tag**

The <OBJECT> tag can be used to pull in a script from an external site in the following way:

<OBJECT TYPE="text/x-scriptlet" DATA="https://malicious.com/xss.html">

**<EMBED>Tag**

If the hacker places a malicious script inside a flash file, it can be injected in the following way: <EMBED SRC="https://hacker.com/xss.swf" AllowScriptAccess="always">

*Prevention:* To prevent xss attack problem we have to follow some xss rule in our html script:

### 2.2.1 *Ignore to Insert Unsecure Data*

There should not be unsecure data in your HTML. Like unsecure link, script etc. This includes "nested contexts" like a URL inside a javascript -- those locations are tricky and dangerous. let us know what you should find out to secure your document.

<script>...never put untrusted data ...</script> directly in a script

<!--... never put untrusted data...--> inside an HTML comment

<div ... never put untrusted data...=test /> in an attribute name

< never put untrusted data... href="/test" /> in a tag name

<style>... never put untrusted data...</style> directly in CSS

Most importantly, never accept actual JavaScript code from an untrusted source and then run it.

### 2.2.2 HTML Escape

HTML escape is to escape the untrusted data directly into the body somewhere. like div, tr, p, b, td, etc. Most web frameworks have a method for HTML escaping for the characters detailed below. You need to implement the restriction detailed here as well.

<body>...escape untrusted data here...</body>

<div>... escape untrusted data here...</div>

any other normal HTML elements

Escape the following characters with HTML entity encoding to prevent switching into any execution context, such as script, style, or event handlers. In addition to the 5 characters significant in XML (&, <, >, ", '), the forward slash is included as it helps to end an HTML entity.

& --> &amp; < --> &lt; > --> &gt; " --> &quot; ' --> &#x27; / --> &#x2F; forward slash is included as it helps end an HTML entity

String safe = ESAPI.encoder().encodeForHTML( request.getParameter( "input" ) );

### 2.2.3 Attribute Escape:

It is for putting untrusted data into attribute values like width, name, value, etc. This should not be used for complex attributes like href, src, style, or any of the event handlers like onmouseover. Inside quoted or unquoted attribute: e.g. <div attri=””> don’t put untrusted data in quote. Unquoted attributes can be broken out of with many characters, including [space] % * + , - / ; < = > ^ and |.

### 2.2.4 JavaScript Escape

It concerns dynamically generated JavaScript code - both script blocks and event-handler attributes. The only safe place to put untrusted data into this code is inside a quoted "data value." Including untrusted data inside any other JavaScript context is quite dangerous, as it is extremely easy to switch into an execution context with characters including (but not limited to) semi-colon, equals, space, plus, and many more, so use with caution.

<script>alert('...Dont put untrusted data here...')</script> inside a quoted string

DO NOT use any escaping shortcuts like \" because the quote character may be matched by the HTML attribute parser which runs first. These escaping shortcuts are also susceptible to "escape-the-escape" attacks where the attacker sends \" and the vulnerable code turns that into \\" which enables the quote.

If an event handler is properly quoted, breaking out requires the corresponding quote. However, we have intentionally made this rule quite broad because event handler attributes are often left unquoted. Unquoted attributes can be broken out of with many characters including [space] % * + , - / ; < = > ^ and |. Also, a </script> closing tag will close a script block even though it is inside a quoted string because the HTML parser runs before the JavaScript parser.

String safe = ESAPI.encoder().encodeForJavaScript( request.getParameter( "input" ) );

### 2.2.4 CSS Escape

It is for when you want to put untrusted data into a stylesheet or a style tag. CSS is surprisingly powerful, and can be used for numerous attacks. Therefore, it's important that you only use untrusted data in a property value and not into other places in style data. You should stay away from putting untrusted data into complex properties like url, behavior, and custom (-moz-binding).

<style>selector { property : ... escape untrusted data here...; } </style> property value

<span style="property : ... escape untrusted data here...">text</span> property value.

If attribute is quoted, breaking out requires the corresponding quote. All attributes should be quoted but your encoding should be strong enough to prevent XSS when untrusted data is placed in unquoted contexts. Unquoted attributes can be broken out of with many characters including [space] % * + , - / ; < = > ^ and |. Also, the </style> tag will close the style block even though it is inside a quoted string because the HTML parser runs before the JavaScript parser.

String safe = ESAPI.encoder().encodeForCSS( request.getParameter( "input" ) );

### 2.2.5 URL Escape

when you want to put untrusted data into HTTP GET parameter value.

<a href="http://www.somesite.com?test=...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...">link</a >

URLs should not be allowed as there is no good way to disable attacks with escaping to prevent switching out of the URL. All attributes should be quoted. Unquoted attributes can be broken out of with many characters including [space] % * + , - / ; < = > ^ and |. Note that entity encoding is useless in this context.

String safe = ESAPI.encoder().encodeForURL( request.getParameter( "input" ) );

user driven URL's in HREF links should be attribute encoded. For example:

String userURL = request.getParameter( "userURL" )

boolean isValidURL = ESAPI.validator().isValidInput("URLContext", userURL, "URL", 255, false);

if (isValidURL) {

```
    <a
href="<%=encoder.encodeForHTMLAttribute(userURL)%>"
>link</a>

 }
```

### 2.2.6 Implement Content Security Policy

There is another good complex solution to mitigate the impact of an XSS flaw called Content Security Policy. It's a browser side mechanism which allows you to create source whitelists for client side resources of your web application, e.g. JavaScript, CSS, images, etc. CSP via special HTTP header instructs the browser to only execute or render resources from those sources. For example this CSP

Content-Security-Policy: default-src: 'self'; script-src: 'self' static.domain.tld

will instruct web browser to load all resources only from the page's origin and JavaScript source code files additionaly from static.domain.3.Source code revelation: It is well known that PHP is a server side framework, so you don't have any access to view source if you want to see a script's code. But, in the event of breakdown of Apache's configuration, people can easily view the name and content of the files.

Thus, when anything goes wrong with Apache, then all the scripts are served on the plain text, and people have access to it, which they aren't suppose to have. Some of these accessible files may have sensitive information like database credentials. So, it is essential to lock such files away from the publicly accessible directory to avoid the consequences of such vulnerability.[19][20][21][22]

## 2.3 Remote File Inclusion

It is an attack technique used to exploit dynamic file include in your web application. Developers often need to incorporate number of local resources into their web applications. Database content, images, PHP classes, and more are all combined together behind the scenes to provide the user with a dynamic interface created just for them. One common way for developers to access file based resources is through the use of the include function, which essentially incorporates the desired file into the programs flow. For example, the following represents an example webpage made entirely of includes:

```
Original Code:     <? php include 'header.php';
                         echo "The main body of the web
page";
                         include 'footer.php';
                         ?>
```

Typically, RFI attacks are performed by setting the value of a request parameter to a URL that refers to a malicious file. Consider the following PHP code:

```
$incfile = $_REQUEST["file"];
include($incfile.".php");
```

Injected Code: <?php include $_GET['injected'].".html"; ?>

it will return to the injected.html which is injected by the attacker. The first line of code extracts the value of the file parameter from the HTTP request. The second line of code dynamically sets the file name to be included by the attacker fixing this is relatively simple. All you have to do is go to your php.ini and check the settings on these flags.

- allow_url_fopen – indicates whether external files can be included. The default is to set this to 'on' but you want to turn this off.
- allow_url_include – indicates whether the include(), require(), include_once(), and require_once() functions can reference remote files. The default sets this off, and setting allow_url_fopenoff forces this off too.

And otherwise in the code all you can do is e.g: Suppose this is your url code look like:

index.php?page=aboutus

index.php?page=home

index.php?page=contactus

**T**han will approach the following code and easily can find the way :

<?php include $_GET['page'].".html"; ?> here page can be anything malicious.

<u>*Prevention:*</u> To prevent this intrusion follow the code like this

In this code here already mentioned the pages with their

```
<?php
 $page_files=array( 'aboutus'=>'aboutus.html',
                 'contactus'=>'contactus.html',
                 'home'=>'home.html'
                );
 if (in_array($_GET['page'],array_keys($page_files))) {
     include $page_files[$_GET['page']];
 } else {
     include $page_files['home'];
 }
?>
```

particular format.[23][24]

## 2.4 Session Hijacking

Session hijacking is when a person steals and use someone else's session ID, which is something like a key to open a secure vault. When a session is set up between a client and a web server, PHP will store the session ID in a cookie on the client side probably called PHPSESSID. Sending the ID with the page request gives you access to the session info persisted on the server (which populates the super global $_SESSION array).

Session Hijacking by Cross-site script attack:

The attacker can compromise the session token by using malicious code or programs running at the client-side. The example shows how the attacker could use an XSS attack to steal the session token. If an attacker sends a crafted link to the victim with the malicious JavaScript, when the victim clicks on the link, the JavaScript will run and complete the instructions made by the attacker. The example in figure 3 uses an XSS attack to show the cookie value of the current session; using the same technique it's possible to create a specific JavaScript code that will send the cookie to the attacker.

```
<SCRIPT>alert(document.cookie);</SCRIPT>
```

*Prevention:* Session IDs are commonly stolen via a XSS attack, so preventing those is a good thing that yields double benefits. It's also important to change the session ID as often as is practical. This reduces your theft window. From within PHP you can run the session_regenerate_id() function to change the session ID and notify the client.

For those using PHP5.2 and above there is a php.ini setting that will prevent JavaScript from being given access to the session id (session.cookie.httponly). Or, you can use the functionsession_set_cookie_parms().

Session IDs can also be vulnerable server-side if you're using shared hosting services which store session information in globally accessible directories, like /tmp. You can block the problem simply by storing your session ID in a spot that only your scripts can access, either on disk or in a database.[25][26]

## 2.5 Cross Site Request Forgery

CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. With a little help of social engineering (like sending a link via email/chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the targeted end user is the administrator account, this can compromise the entire web application. Cross-Site Request Forgery (CSRF) is an attack that tricks the original user into loading a page that contains a malicious request like change the user's e-mail address, home address, or password, or purchase something.

The attacker can make the user perform actions that they didn't intend to, such as logout, purchase item, change account information, retrieve account information, or any other function provided by the vulnerable website.

Sometimes, it is possible to inject the CSRF attack on the vulnerable site itself. Such vulnerabilities are called Stored CSRF flaws. This can be accomplished by simply storing an IMG or IFRAME tag in a field that accepts HTML, or by a more complex cross-site scripting attack.

*Prevention:* Individual Web users using unmodified versions of the most popular browsers can do relatively little to prevent cross-site request forgery. Logging out of sites and avoiding their "remember me" features can mitigate CSRF risk; not displaying external images or not clicking links in spam or untrusted e-mails may also help.

Web sites have various CSRF countermeasures available:

- Requiring the client to provide authentication data in the same HTTP Request used to perform any operation with security implications (money transfer, etc.)
- Limiting the lifetime of session cookies
- Ensuring that there is no clientaccesspolicy.xml file granting unintended access to Silverlight controls
- Ensuring that there is no crossdomain.xml file granting unintended access to Flash movies

Cross-site scripting (XSS) vulnerabilities (even in other applications running on the same domain) allow attackers to bypass CSRF preventions. [27][28]

In order to ensure that an action is actually being performed by the user rather than a third party, you need to associate it with some sort of unique identifier which can then be verified. To prevent the attack, we can modify login.php as follows:

```
<?php
// make a random id
$_SESSION["token"] = md5(uniqid(mt_rand(), true));
echo '<a href="test.php?action=
logout&csrf=' . $_SESSION["token"] . '">Logout</a></p>';
```

## 2.6 Directory Traversal

Directory traversal attack is to find the files within or outside files from the root directory. This attack is to use when developer do not make files path to accessible for others.

Directory traversal is also known as the ../ (dot dot slash) attack, and backtracking. Some forms of this attack are also canonicalization attacks. By manipulating variables that reference files with "dot-dot-slash (../)" sequences and its variations, it may be possible to access arbitrary files and directories stored on file system, including application source code, configuration and critical system files, limited by system operational access control. The attacker uses "../" sequences to move up to root directory, thus permitting navigation through the file system. [29]

E.g: We will just say that this particular file is stored in the following

path: /home/someone/public_html/index.php.

The attacker could then do: index.php?page=../secret

*Prevention:* To prevent from this attack we can check for the particular format to access the file

```
$file = str_replace('\\', '/', realpath($page . '.php'));

if (!preg_match('%^/home/someone/public_html/[a-
z]+\.php$%', $file)) {
      die('Invalid page');
}
include $file;
```

## 2.7 File Uploading

This type of attack mostly try by the attacker first because if they get success to upload their malicious file in the directory on the server. Then there are only need to find the to execute the code. If attacker find this way to attack on the application it can be very harmful for the application. Attacker can upload his/her customises code and script on the server.

In most Web applications, developers provide upload file functionality — images.

Upload code:

```
<form name=upload action=upload.php method=post>
  upload a file: <input type=file name=fileName >
  <input type=submit name=upload>
</form>
```

In most upload functionality on web, there is no verification of the uploaded file is done. The form above implements PHP script to process the upload; the code might move the file to a common/well-known folder, without verifying its content :

```
<?php
$uploaddir = 'uploads/'; // Relative path under webroot
$uploadfile = $uploaddir
.filename($_FILES['userfile']['name']);
if
(move_uploaded_file($_FILES['userfile']['tmp_name'],
$uploadfile)) {
  echo "File is valid, and was successfully
uploaded.\n";
}
else {
  echo "File uploading failed.\n";
}
?>
```

Let's suppose the attacker were to upload a file containing code as follows:

```
<?php
  System("ls");
?>
```

In such a case, upload will move the attackers' file to the subdirectory. If, for instance, the attackers then enter http://example/uploads/attackerfile.php as the URL in their browser/client, they will get a listing of the current working directory. If attacker gets success to upload and run his malicious file on the server he can get the back door entry and can run more malicious script on the application.

*Prevention:* To prevent uploading attack we need to check the file at the time of upload on server. There are different type of checking in the file we can implement at the time of uploading.

### 2.7.1 Content type Verification

In content type verification you can check the type of file suppose you are uploading an image file it will check its format. If a form upload with a non-image file is received, You could check the content type of the file that is being uploaded by adding the following validation code:

```
if($_FILES['userfile']['type'] != "image/gif") {
  echo "Sorry, we only allow uploading GIF images";
  exit;
}
```

This new code will check the uploaded file's content type — that is, GIF — and block any others.

### 2.7.2 Image file content verification

Image file content verification can take to check the actual content of the uploaded file, to verify whether it actually is an image or not, using the PHP function getImageSize():

```
$imageinfo =
getimagesize($_FILES['userfile']['tmp_name']);
//check image size
if($imageinfo['mime'] != 'image/gif' &&
$imageinfo['mime'] != 'image/jpeg' &&
$imageinfo['mime'] != 'image/png') {
  echo "Sorry, we only accept GIF, JPEG, png
images\n";
  exit;
}
```

The function getImageSize() returns the image's size, its image type, if the file is a valid image file (else it will generate an error).

### 2.7.3 Filename Verification

The final and most important check is on the extension of the uploaded file's name. By htis verification we can check the extension of the file and can prevent to upload unwanted extension files.

```
$extention= array(".php", ".phtml", ".php3",
".php4",".html");
foreach ($extention as $item) {
  if(preg_match("/$item\$/i",
$_FILES['userfile']['name'])) {
    echo " PHP files not allowed\n";
    exit;
  }
}
```

Here, $extention contains a list of file extensions, and the preg_match() function applies them as a regular expression check against the uploaded file's name. These are all the extensions that the Web server is configured to accept as PHP executable files. Once you block files with these extensions, even if the attacker uploads PHP code in files with other extensions, the PHP interpreter won't execute them — so the attacker is blocked.

Other issues concerning a file upload

Other than the above safeguards, developers should also consider the following:

- Developers have to take care that uploaded files are not easily or directly viewable by users or attackers.
- It is best if uploaded files are stored in a folder that is not below the Web root. Also, developers could store the original filename (as uploaded) in a database table, and rename the file in the storage folder with a randomly generated name, storing that alongside the original filename in the database.[30][31]

## 2.8 Server File Access Permission

Many times when we work on server and modification in our files and folders we forget to change the access permission of files. This access permission if remain public then attacker can easily find the way to get in to the sever and to make changes.

Generally server provide three types of permission which are user which is only for the account user, group that is only for a group of individuals and world, it is a public permission.

e.g.: Every permission has three accessibility criteria.

| | User | Group | World | Access |
|---|---|---|---|---|
| Read | 0 | 0 | 0 | None |
| Write | 5 | 5 | 5 | Only view |
| Execute | 7 | 7 | 7 | All Access |

There are a series of access permission by which we can restrict the file of give the access to the others. To secure this permission we can prevent the server file permission attack.

*Prevention:* To prevent this type of attack monitor your server file permission. There should be a restriction on files which are vulnerable.

## 2.9 Full Path Disclosure

Full path disclosure vulnerability allow to the attacker to see the full path of the file executed on the web browser.

E.g: home/example/public_html/index.php

Full path access leak the web root directory. Every file can be access if we have the root path. To find the web root path attacker can assume the path of config file of the server where he can get the username and password of the server because configuration files has all the information the structure of configuration file like this.

```
<?php
   //Hidden configuration file containing database credentials.
   $hostname = 'localhost';
   $username = 'root';
   $password = 'owasp_fpd';
   $database = 'example_site';
   $connector = mysql_connect($hostname, $username,
$password);
   mysql_select_db($database, $connector);
?>
```

*Prevention:* Preventing an Full path disclosure injection without having an error handling / management system is as simple as disabling the display of error messages. This can be done in PHP's php.ini file, Apache's httpd.conf file, or via the PHP script itself:

php.ini:

```
display_errors = 'off'
```

httpd.conf/apache2.conf:

```
php_flag  display_errors  off
```

PHP script:

```
ini_set('display_errors', false);
```

## 2.10 Open redirect

An open redirect attack takes a parameter to redirect the web page to that parameter.

```
e.g: $redirect_url = $_GET['url'];
 header("Location: " . $redirect_url);
```

In this example if we include this code in our program then it will help to the attacker to inject their malicious url to redirect the original path to other like this

http://example.com/example.php?url=http://attack.example.com

*Prevention:* To prevent this attack this is to propose ignore the use to redirect path in the application or else you can do is

```
<?php
 /* Redirect browser */
 header("Location: http://www.mysite.com/");
 ?>
```

## 2.11 Exposed session data

When on a shared host, security simply isn't going to be as strong as when on a dedicated host. One of the vulnerable aspect of shared hosting is having a shared session store. By default, PHP stores session data in /tmp, and this is true for everyone. You will find that most people stick with the default behavior for many things, and sessions are no exception.

Unfortunately, it is pretty trivial to write a PHP script to read these files, and because it runs as the user , it has the necessary privileges. The safe_mode directive can prevent this and similar safety concerns, but since it only applies to PHP, it doesn't address the root cause of the problem. Attackers can simply use other languages.

*Prevention:* Don't use the same session store as everyone else. Preferably, store them in a database where the access credentials are unique to your account. To do this, simply use thesession_set_save_handler() function to override PHP's default session handling with your own PHP functions.[26]

## 2.12 Cookies Theft

Cookie theft occurs when a third party copies unencrypted session data and uses it to impersonate the real user. Cookie theft most often occurs when a user accesses trusted sites over an unprotected or public Wi-Fi network.

One thing we can do is to change the session ID often. If we do that then the chance that the intercepted session ID will be valid will be greatly minimized if that ID changes often.

*Prevention:* We can use one of PHP' built-in functions calledsession_regenerate_id(). When we call this function the session ID will be, no surprise, regenerated. The client will simply be informed that the ID has changed via an HTTP response header called Set-Cookie.[27][32]

## 2.13 Iframe Hack

The name Iframe Hacking has been derived from the manner in which the hacking is done using an iframe tag. Iframe is for inline frame, and is essentially the name of an html tag - <iframe> </iframe>. Iframe tags can be used to insert contents from another website within a web page as if they were part of the current page.

In an IFrame attack, the hacker embeds a malicious iframe code snippet in your website page. When anyone visits that page, the hidden iframe code secretly downloads and installs a Trojan or a malware such as key-logger on the unsuspecting user's computer, if his computer is not adequately protected. Thus over a short period of time several of your site visitors' computers would get infected. Very soon your website will get known as a source of virus and may get blacklisted from the internet.[33]

Below is an example of a hidden iframe code embed in a web page:

```
<iframe src="http://hackersite.com/injectfile.php"
width=100% height=0></iframe>
```

## 2.14 Insecure Cryptographic Storage

Insecure Cryptographic Storage is a common vulnerability that occurs when sensitive data is not stored securely. Insecure Cryptographic Storage isn't a single vulnerability, but a collection of vulnerabilities. The collection all have to do with making sure your most important data is encrypted when it needs to be.

*Prevention:* This includes:

- making sure you are encrypting the data
- making sure you have proper storage and management
- making sure that you are not using known bad algorithms[34]

e.g.: like to generate password use MD5 algorithm.

```
<?php
// make a random id
$_POST["password"] = md5(uniqid(mt_rand(), true));
?>
```

## 2.15 Failure to Restrict URL Access

If a web application fails to verify users' privilege before granting access to the page, web application is vulnerable to "Failure to Restrict URL Access" attack. This vulnerability exists because most of the developers hide links to protected pages from unauthorized users. But a skilled unauthorized user can guess or find the link to access the page.

### *2.15.1 Protecting only by Hiding References*

Many times websites have some hidden special URLs whose reference is not present anywhere in the website and are created for admin users. Developers thinks that the URL is not known to anyone and do not use any kind of access protection on pages. But those can be accessible for anyone who knows about the link.

### *2.15.2 Protecting only by Checking for Valid Sessions*

Suppose a website with user and admin protect page. In this website, there is a login for users and one more for admin. Both login and session validations are same. If developer had only restricted the page on the basis of valid session id, anauthenticated user can access the admin page in he knows the URL of any web page built for admin only. For this, that authenticated user can use forced browsing to guess the admin directory and pages. In general, websites put admin pages in the admin directory so it is easy to guess. Now the only thing is guessing the restricted admin pages names.
In this example we saw that user do not have link for the pages which are only built for admin but application fails to restrict URL access for the user if he knows the link.

### *2.15.3 Checking Authentication once*

Most of the times developers authenticate users at login screen and then redirect them to dashboard. Then developers think that users will come at this dashboard after the proper authentication so it is not necessary to authenticate them before accessing other pages whose reference is available only in dashboard. This is really a bad practice of coding and not recommended. Each time a user request to access a secure page, his authenticity must be checked before granting the access.

## 3. CONCLUSION

As in today's time most of the people is web applications dependant. Everybody is doing their daily task or their transaction task through web application like shopping, bill pay, booking etc it is become very attractive for the attacker(Hacker) to steal and manipulate the information on the web. This is very dangerous for the internet users. And PHP is most usable language to develop web applications. This paper shows how attacker approach to steal our data and how can we prevent from. Less or more the thing which developer should learn and keep in mind at the time of development to ignore the internet content copy paste in the programming. Every time developer should use the authenticate and standard programming. Developer should follow the security rule in their design and as well as developer should keep in mind something users also should learn few things about their browser cookies and session their folder permission on server.

This approach can prevent from the danger of attack at most of the levels.

## 4. ACKNOWLEDGEMENT

Our thanks to our department head and our colleague who gave us very valuable feedback and guidance

## 5. REFERENCES

[1] 1Prasant Singh Yadav, 2 Dr pankajYadav, 3Dr. K.P.Yadav "A Modern Mechanism to Avoid SQL Injection Attacks in Web Applications",IJRREST: International Journal of Research Review in Engineering Science and Technology ,Volume-1 Issue-1, June 2012.

[2] Mayank Namdev *, FehreenHasan, GauravShrivastav "Review of SQL Injection Attack and Proposed Method for Detection and Prevention of SQLIA"Volume 2, Issue 7, July 2012.

[3] Atefeh Tajpour ,Suhaimi Ibrahim, Mohammad Sharifi Web Application Security by SQL Injection DetectionTools.IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 3, March 2012

[4] Mihir Gandhi , JwalantBaria, "SQL INJECTION Attacks in Web Application". International Journal of Soft Computing and Engineering (IJSCE) Issues, Vol. 2, Issue 6, January 2013

[5] 1 Venkatesh Yerram, 2 Dr G.Venkat Rami Reddy, "A SURVEY OF ATTACKS ON PHP AND WEB VULNERABILITIES". INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS Issues, Vol. 2, Issue 4, April 2014

[6] Emmanuel Benoist (2014, Spring).[Online]. Available:http://www.benoist.ch/SoftSec/slides/injectionFlows/slidesInjectionFlows2.pdf

[7] OWASP (2012, April). Command Injection [Online]. Available:https://www.owasp.org/index.php/Command_Injection

[8] Emmanuel Benoist (2014, Spring).[Online]. Available:http://www.benoist.ch/SoftSec/slides/injectionFlows/slidesInjectionFlows2.pdf

[9] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso, A Classification of SQL Injection Attacks and Countermeasures Available:http://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf

[10] 1 Sampada Gadgil, 2 Sanoop Pillai 3 Sushant Pujari "SQL INJECTION ATTACKS AND PREVENTION TECHNIQUES" International Journal on Recent and Innovation Trends in Computing and Communication Volume 1, Issue 4, Apr 2013.

[11] 1 Mayank Namdev , 2 Fehreen Hasan, 3 Gaurav Shrivastav "A Novel Approach for SQL Injection Prevention Using Hashing & Encryption (SQL-ENCP)",IJCSIT: International Journal of Computer Science and Information Technologies ,Volume-3 Issue-5, 2012.

[12] XuePing-Chen "SQL injection attack and guard technical research",Science Direct: Procedia Engineering,Volume-15 2011.

[13] Atefeh Tajpour, Maslin Masrom, Mohammad Zaman Heydari, Suhaimi Ibrahim, "SQL Injection Detection and Prevention Tools Assessment"[Online].Available: http://www.meeting.edu.cn/meeting/UploadPapers/1282791435515.pdf

[14] Shelly Rohilla , Pradeep Kumar Mittal "Database Security by Preventing SQL Injection Attacks in Stored Procedures" Volume 3, Issue 11, November 2013.

[15] 1 Asha. N, 2 M. Varun Kumar, 3 Vaidhyanathan.G "Preventing SQL Injection Attacks", International Journal of Computer Applications ,Volume-52 Issue-13, August 2012.

[16] 1 Asha. N, 2 M. Varun Kumar, 3 Vaidhyanathan.G "Preventing SQL Injection Attacks", International Journal of Computer Applications ,Volume-52 Issue-13, August 2012.

[17] Haeng Kon Kim, "Frameworks for SQL Retrieval on Web Application Security ", International MultiConference of Engineers and Computer Scientists Volume-1, March 2010.

[18] 1 S.Suganya, 2 D.Rajthilak, 3 G.Gomathi, "Multi-Tier Web Security on Web Applications from Sql Attacks" IOSR: Journal of Computer Engineering (IOSR-JCE), Volume-16, Issue-2, April 2014

[19] Mihir Gandhi , JwalantBaria, "SQL INJECTION Attacks in Web Application". International Journal of Soft Computing and Engineering (IJSCE) Issues, Vol. 2, Issue 6, January 2013

[20] OWASP (2012, April). XSS (Cross Site Scripting) Prevention CheatSheet [Online]. Available: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet#RULE_.231__HTML_Escape_Before_Inserting_Untrusted_Data_into_HTML_Element_Content

[21] 1 S.SHALINI, 2 S.USHA ," Prevention Of Cross-Site Scripting Attacks (XSS) On Web Applications In The Client Side",IJCSI International Journal of Computer Science,Volume-8 Issue-4, July 2011.

[22] Mike Ter Louw, V.N. Venkatakrishnan,. Robust Prevention of Cross-site Scripting Attacks for Existing Browsers [Online]. Available:http://www.cs.uic.edu/~venkat/research/papers/blueprint-oakland09.pdf

[23] Dennis Schwarz,. "A Multi-Perspective View of PHP Remote File Include Attacks" (November 2009), SANS Institute InfoSec Reading Room [Online]. Available: http://www.sans.org/readingroom/whitepapers/detection/multi-perspective-view-php-remote-file-include-attacks-33229

[24] Aaron Weiss,. "How to Prevent Remote File Inclusion (RFI) Attacks" (January 2012), eSecurity Planet [Online]. Available: http://www.esecurityplanet.com/browser-security/how-to-prevent-remote-file-inclusion-rfi-attacks.html

[25] Jerry Louis,. "Detection of Session Hijacking" (January 2011), [Online]. Available:http://uobrep.openrepository.com/uobrep/bitstream/10547/211810/1/louis2011.pdf

[26] 1 Abhishek Kumar Bharti, 2 Manoj Chaudhary, "Prevention of Session Hijacking and I spoofing with Sensor Nodes and Cryptographic Approach", International Journal of Computer Applications, Volume-76 Issue-9, August 2013.

[27] OWASP (2012, April). Cross-Site Request Forgery (CSRF) (September 2013)[Online]. Available:https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[28] Martin Psinas (September 2011). "Preventing cross site requesting forgeries", Site Point [Online] Available: http://www.sitepoint.com/preventing-cross-site-request-forgeries/

[29] Wikipedia (May 2014). Directory traversal attack [Online]

Avaliable:http://en.wikipedia.org/wiki/Directory_traversal_attack

[30] High Tech bridge (April 2014). Unrestricted Upload of File with Dangerous Type [Online] Available: https://www.htbridge.com/vulnerability/unrestricted-upload-of-file-with-dangerous-type.html

[31] OWASP (April 2014). Unrestricted File Upload [Online] Available: https://www.owasp.org/index.php/Unrestricted_File_Upload

[32] PHP Security Guide: Shared Hosts [Online]. Available: http://phpsec.org/projects/guide/5.html

[33] Ethical Hacking [Online] Available: http://www.breakthesecurity.com/2011/07/what-is-iframe-injection-mass-iframe.html

[34] Protect Data by Preventing Insecure Cryptographic Storage [Online] Available: http://resources.infosecinstitute.com/protect-data-by-preventing-insecure-cryptographic-storage/