INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

A SURVEY OF ATTACKS ON PHP AND WEB VULNERABILITIES

Venkatesh Yerram[1], Dr G.Venkat Rami Reddy[2]

[1]Computer Networks and Information Security, *venkatesh.yerrams@gmail.com*

[2]Computer Science Engineering, 2nd gvr_reddi@yahoo.co.in
JNTU Hyderabad, India

Abstract

Web application is the one of the important feature of internet which plays a vital role in communication. Web application got a rapid growth over a decade. Nowadays we are using web application in social networking, online transaction, electronic mail etc. billion's of users using internet daily. Of course it has many benefits the impediment is security. Most of web applications are facing security treats and issues. PHP is a server side scripting language used to build the web applications. This survey gives a different type of possible attacks on web application which built using php and how to anticipate such attacks. Here we are providing the different types of web application vulnerabilities and countermeasures against it.

Keywords*:* threats; vulnerabilities; attacks; security; eval injection; shell injection; SQL-Injection; cross-site scripting

I. Introduction

Internet has reached nook and corner of world. There are different types of services available in internet. There are millions of new users are adding to the internet daily, generally this users interact with internet through the web applications. Knowledge about the internet has been improved which leads to increase cyber crimes. "There are more than 80% of web applications are vulnerable to cross site script attack" [1].

The Open Web Application Security Project (OWASP) is an open source web application security project which finds the causes of insecure software. OWASP was come in 2001.it is a non profitable charity organization. It provides powerful security awareness for web application [2].

Web applications allow the legitimate user to store and retrieve the data to/from the database trough an internet using the web browser. The databases are central to the web applications, that stores the data needed for websites to deliver it to specific users.

To handle the user's huge data web applications store the data in the databases. OWASP has given the first position for SQL injection on TOP-10 web application vulnerabilities in 2013. Injection attacks are SQL injection attack where attacker injects the malicious code into database statement into a web form that executed by the database by an inefficient validation. In some situation maliciously crafted input that contained the SQL statement or the parts of these, produces the output that's semantically different from the one meant by the designer and which threatens the security policies of the underlying databases.

In 2001 the famous well known sites Twitter, Facebook, Orkut, MySpace are affected by this code injection attacks. Present these attacks are known as most common publicly reported security vulnerability. With Some researchers reporting that 68% of websites are probably prone to these attacks.

Cross-site scripting attack (XSS attack) is an attacker gaining access to the victim's browser in order to execute the malicious code (this is usually HTML or JavaScript) within the context of trust of web application site. As a result, if this modified code is successfully executed, the attacker may get the chance to access the user information, the browser URL contains the different type of information belongs to the user for example cookies, usernames, session information, session keys etc, the attacker can stole the information regarding user. OWASP has given the first position for Cross site scripting attack on TOP-10 web application vulnerabilities in 2007[2]. There are two types of cross site scripting attacks are their

- Persistent XSS attack
- Non-persistent XSS attack

Persistent XSS attack is traditional type of attack. Malicious script is injected into web applications and that is executed by a week input validation.

Non-persistent XSS attack is unlike the common type of XSS attack usually this type of attack is done by mixing the phishing, social engineering. The goal of this attack is to steal the user sensitive information such as credit card detail, passwords. Due to this difference this code not persistently stored into the web application.

II. Injection Attacks

OWASP definition for injection attack is injection attack occur when untrusted data in the form command or query sent to an interpreter. Without proper validation the attacker hostile data can trick interpreter into executing unintended commands or accessing data. Injection attack is placed first position in OWASP top-10 web vulnerabilities 2010 and 2013 [2].

A. Shell Injection Attack

Shell infection also known as command injection. While it is not more frequently talked it is one of most critical. Shell injection where attacker executes the system commands and gains unauthorized access to the user's data [3][4].

To understand briefly how the shell injection works. Assume a simple case. A script is to display the file contents to the end user, but developer don't have interest to spend time on write a procedure read this files. Instead of this user allowed selecting a file and UNIX cat command to execute and display results. The code something looks like this.

```
<?php
echo shell_exec(`cat`.$_ GET[`file_name`]);
?>
```

The above code works well with different get parameters and displays different outputs. If we add new files to this directory it will automatically recognizes reads the file and outputs it.

A user opens the page and type the URL like this:

www.site2u.com/viewfile.php?filename=my_context.txt

This PHP page will show the results as the user expected. For the legitimate users page works as expected. If an attacker looks at this URL he will surely entice with the vulnerable. Attacker may append the semicolon (;) and another UNIX command to the file which specified in the URL

www.site2u.com/viewfile.php?filename=my_context.txt;ls

This page will run expected result that the file content and the execution will not stop at their, it will continue by listing the all the files in the directory. The above code is not secure and which have vulnerable to shell or command injection attack.

II. How to prevent shell injection attacks:

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

- Sanitize the data
    - Sanitize the all the user data, and avoid the all the special symbols which executes on the UNIX shell.
- White-listing the data
    - 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 [3]

B. Eval Injection Attack

Eval () is s PHP function which allows to interpret the given string as a PHP code, often the eval () function is used in web applications, the interpretation of the PHP code is widely acceptable, eval () executes regularly the PHP code containing the previously defined variables. The problem with this is if eval () executes a variable that you can modify the code contained by PHP [5].

To understand the how the eval injection attack works, consider a simple example of eval ();

```
<? php
$color = “blue”;
$animal = “tiger”;

$str = ‘A $animal jump over a $color fence!’ ;
echo $str. “<br />”;

eval (“$str=\”$str\”;”);
echo $str ;
?>
```

The output of the above code will be

A $animal jump over a $color fence!
A tiger jump over a blue fence!

So how is eval () is vulnerable? Imagine a form consisting of the username input field value using eval($_GET[‘username’]) ,if the hacker type the following value in input field:

Username (“ hack@me.en ” , “ passwords ” , “ bin/cat/etc/passwd ”);

In the above shown username the server side the eval () function will execute the code and sends the passwords to the attacker.

Let consider a another example

```
<?php
$admin = “john”;
$new = “ $_GET[‘name’];
eval(‘$admin=’.$new.’:’);
?>
```

Here for eval function the arguments are passed from php code, so here the attacker cans additional commands. For example if the attacker enters following code:

www.site2u.com/eval.php?name=mike;system('find . -print | sed -e\'s;[^/]*/;|____;g;s;____|; |;g");

the additional code in URL run which executes system command in the server system ,which results in a directory contents in form of tree like structure.

To prevent eval () injection attack care filtering of user input should be carried out.

C. SQL Injection

1. What is it?

SQL injection attack is one of the many common type of web attack mechanism used by an attacker to steal data from the organizations. It is one of the application layer attacks using now days. It is a type of attack that takes the advantage of improper coding of the web applications that allows the attacker to inject the malicious code into the lets say search bar to allow them to access the data stored in your database. Indeed SQL injection arises because the fields available for user input allow the SQL statements to pass through it and query the database directly.

2. Goals of SQL-Injection attack

- Read the sensitive information from database.
- Alter the data in database.
- Gain the administrative powers and execute administrative queries in the database such as shutdown the database.
- Recover the content of a file which present in the database.
- In some case execute the operating system commands.

3. Most common types of SQL Injection attacks:

- Tautologies

The intension of the attacker to do this type attack is to bypass the authentication, identify the injectable parameters and extract the data

In this type attack, attacker injects the one or more conditional statements so that they always evaluate always true. Transforming the all conditional statements into a tautology causes all the rows in a table are returned. In general, to work a tautology based attack, the attacker not only consider the injectable parameters, but also a coding constraints which evaluates the query results.

Example: bypassing the login script

Query: select id from employee where user='$_POST[username]' AND pwd='$_post[password]';

The above query takes inputs from system users, suppose the user enters:

Username='a' OR '1=1'
Password='a' OR '1=1'

Query: Select id from employee where user='a' OR '1=1' AND pwd='a' OR '1=1';

The code injected in the conditional part ('a' OR '1=1') transfers the entire where clause into a tautology. The Database use the conditional statement to evaluate the each row whether it to return or not. The above query evaluate true for each and every row and return all of them. This would cause the user as authenticated user by the first row data in the result set.

- Logically incorrect queries:

This type of attack is done by attacker in order to know which type and the structure of the database of an application. It is a preliminary information gathering stage for the other attacks. The results produced for this type of attack is the default error page returned by application server this is overly descriptive. Indeed this helps the attacker to know the information about the schema of back-end database. When performing this type of attack, the attacker tries to

inject queries that cause syntax, type conversion, or logical error in the database. Syntax errors are used to identify the injectable parameters. Type errors are to know the data type of the certain columns or exact data. Logical errors sometimes show the table names.

Example: type conversion error that can reveal relevant data
Password: AND 'pin: "convert (int, (select top 1 name from sysobjects where xtype='u'))

Query: Select id from employee where user='' AND pwd= AND 'pin: "convert (int, (select top 1 name from sysobjects where xtype='u'));

The query tries to extract the first user table where xtype='u' from the database metadata table. The query tries to convert the table name into a integer, because it is not a legal operation the database throws an error, the attacker try to use this information to know the schema of database.

- Union query

In this type of attack, the attacker injects the vulnerable parameter to change the output produced for a given query. By this technique, the attacker can trick the database into returning data is different than the developer intended. In this type of attacker injects the statements of following form: UNION SELECT <rest of inject query> , here the attacker has all controls on the second query. This second query used to retrieve the data from the specified table. The results for the second query will be returned along with the first query.

Example:

Username:' UNION SELECT cardNumber from Creditdetails where acctNo=20032 - -"

Query: Select id from employee where user=' UNION SELECT cardNumber from Creditdetails where acctNo=20032 - -'AND pwd='';

Assume here no user with login equal '' .the original first query returns a null set but the second string returns the credit card number from the creditdetails table. Database takes this two results, unions them and returns them to the application.

- Piggy backed queries

In this type of attack, attacker tries to insert additional queries in the original query. In this the attacker don't modify the original query, but try to inject different queries that "piggy back" on the original query. As a result database receives multiple queries that all executed by the database. This attack is extremely harmful. On successful execution, attacker can execute any type of SQL command.

Password:";drop table manager --"

Query: Select id from employee where user='' AND pwd=";drop table manager --";

After completion of first query, the database recognizes the query delimiter (";") and executes the second injected query. Drops the manager table would likely destroy the valuable information.

- Stored procedures

SQL Injection of this type is done by executing the stored procedures present in database. Most of the databases have the standard set of stored procedure that extends the functionality of database and allows interacting with the operating system. So once an attacker knows the back-end database is using, SQL injection attacks prepared to execute procedure provided by that specific database. Here procedure are written in a specific scripting language, this languages may

contains the other vulnerabilities like buffer overflows. Attacker tries to run arbitrary code execution on server in order exploit these vulnerabilities.

- Inference

In this type of attack, the attacker modifies the query such that query executed based upon the answers to a logical questions that are true/false about the data values in the database. Usually this attack is done on the applications with enough security when injection has successful; there is no feedback from database error message. In this situation attacker injects commands into the database and observes how it is responding to the commands. By careful observation attacker not only knows the certain vulnerable parameters and information about values in database.

There are two types of attacks based on inference

- Blind injection

In blind injection is get from the behaviour of the page by asking logical questions which have true/false as answer. If the injected query evaluates as true, the site continue to functions normally. If that evaluates as false, there is no error page. But the page differs significantly from normal page.

- Timing attack

In this type attack, an attacker gains the information from database by observing timing delay in between the responses time of database. Attacker inject the query in the form of if / then statement. By measuring the response time of database, the attacker can infer which type of attack to take.

- Alternate encoding

I. In this type of attack the injected code is modified in order to avoid detection by defensive code practice and also from a automated prevention techniques. Generally this will not used directly as attack, this alternate coding is used in conjunction with the other attacks.[6]

III. Cross-Site Scripting (Xss)

a. XSS vulnerabilities

Cross-site scripting attacks are typically due to sloppy software written for web servers. A website is said to have XSS vulnerability if it inadvertently includes malicious scripts crafted by an attacker in pages returned by it. The malicious code is created by the clever design of an attacker, not the website developer

Social networking sites are more prone to this attack. Here the attacker injects the malicious script in to a page contributed by him. When other user's subscribers to this site download the page, they will also get this malicious script.

The malicious code, which is generally javascript, runs on the user browser; assume here javascript is enabled in user's browser. The script may read the cookies contains login details, credit card details on the user's browser and redirect them to site which chosen by an attacker

Such an attack could be avoided if the web server programmed to filter out the malicious input from the clients. So for example, it could simply reject the input containing the special characters and tags such as <script>.however attacker can have way to bypass this filtration. One is to use the alternate character encoding methods such as hexadecimal escape codes. For example, the opening and closing braces can represent in multiple ways shown below.

<	=	&lt	=	&#60	=	%3C
>	=	&gt	=	&#62	=	%3E

So an alternate representation for <script> is &ltSCRIPT&gt

The above showed attack, in which attacker's code persist in the website, is referred to as a persistent cross-site script attack. Here the attacker doesn't make permanent changes but still obtain the valuable information from the victim. Next we discuss how non-persistent cross-site scripting attacks are designed.

The key to identify the cross-site scripting vulnerability is to study the responses of web application for different inputs request by the user. Often the input received from the user is echoed back by the web server. To see how the attacker exploit the "echoing back" of user input, here we must understood how the input reached to web server and how this server responded.

Consider a website of simple retail store, let say www.store.com , which provides a search facility for the customers. It has a small text box in that user enter the product name they want. When the user strike the ENTER button or clicks on the GO button, the browser sends the query to web server in the following manner:

http://www.store.com/Search.php?product="Barbie Doll"

The above "extended URL" is in fact, a request to the web server invokes a program called search.php using the parameter "product" and the parameter value "Barbie Doll". The web server checks in the database and if the related information found it responds with that page. If the information related to Barbie Doll is not found then it will responds with "Product Barbie Doll not found…".

If the attacker instead enter the <SCRIPT>alert('Hi…') </SCRIPT> in the search bar, the URL generated will be

http://www.store.com/Search.php?product="<SCRIPT>alert('Hi...') </SCRIPT>"

On receiving the above request the search.php will search in database for the match for <SCRIPT>alert('Fire!') </SCRIPT>. It will not find the product with this name. it return the page with the embedded message.

Product "<SCRIPT>alert('Hi…') </SCRIPT>" not found…

Assume that here the users enabled the javascript on his browser; the above script will be executed. Here the attacker does this type of attack to examine the web server. Here he also knows that

i. www.store.com does not performing any validation of users input other than simply checking in the database. It doesn't parse user input to check for html tags and javascript tags.
ii. It echoes the invalid user input back to the user browser. By this information attacker makes the URL such as

http://www.store.com/Search.php?product="<SCRIPT>document.location='www.hack.com/cookie.php?'%20+document.cookie</SCRIPT>"

The attacker embeds this URL in an email and sends this to a victim. By looking this URL with attractive header and body victim clicks on the URL. Assume www.store.com has stored login details in the cookie during the previous login attempts. When the victim clicks on the above link, the is read and the details are dispatched to the attacker's website.

b. Overcoming Cross-site Scripting (XSS)

The most institutive technique to overcome and prevent the cross site script is to have the filtering and validation of user inputs. One strategy is to make a blacklist of all user inputs that should be filtered out. Maintain a list of inputs which should be filtered out such as braces, special symbols, angular braces etc. however this will be easily defeated by attacker who skip the filter by encoding the above characters by a Unicode and other representations.

A better way to handle is whitelist approach, in this approach specify the all the user expected inputs precisely. The search criteria must pass through this if the input is not satisfies these criteria that should be ruled out. This method is accomplished by the use of a regular expression. However writing regular expression for all permissible string may not be straight forward. Writing regular expression for email is straight forward than writing for a password because later may include special symbols like quotes, angular braces.

IV. Conclusion and Future Work

PHP is increasingly becoming the standard technology for designing websites and web applications. Like any other technology, it also has security holes and vulnerabilities. This paper discussed some of the major vulnerabilities found in PHP. It described the attacks that could be performed on PHP with examples. Remedy measures and precautions have also been suggested to prevent these attacks. A lot of scope is present to further exploit PHP vulnerabilities and also security could be improved further. In particular, Injection attacks and Cross-side Scripting attacks have been found to have serious threat to security and more sophisticated measures could be developed.

REFERENCES

[1] Jayamsakthi Shanmugam, Dr.M.Ponnavaikko "A solution to block Cross Site Scripting Vulnerabilities based on Service Oriented Architecture" 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)

[2] OWASP Top 10-2013
The Ten Most Critical Web Application Security Risks

[3] Golemtechnologies. Shell-Injection, 2012.
https://www.golemtechnologies.com/articles/shell-injection, accessed Jan. 2012.

[4] OWASP. Comment_injection_attack, 2011
https://www.owasp.org/index.php/Comment_Injection_Attack, accessed Jan. 2012.

[5] Eval () Vunerability & Exploitation
http://www.exploit-db.com/papers/13694

[6] SQL Injection In-Depth:Attacks and Prevention Methods
http://www.cleverlogic.net/articles/sql-injection-depth-attacks-and-prevention-methods

[7] Ettore Merlo, Dominic Letarte, Giuliano Antoniol "Automated Protection PHP application against SQL-injection Attacks" 11th European Conference on Software Maintenance and Reengineering (CSMR'07)

Biography

Venkatesh Yerram is pursuing his post graduation from School of Information Technology, JNTU Hyderabad. He has completed B.Tech from CM Engineering College. His research area interests include Information Security, Computer Networks, Operating Systems, Data Mining.

Dr G.Venkat Rami Reddy is presently Associate Professor in Computer Science and Engineering at school of Information Technology. He is more than 11 years of experience in Teaching, and Software Development. His areas of interests are: image Processing, Computer Networks, and Analysis of Algorithms, Data mining, Operating Systems and Web technologies