IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727,
PP 65-68
www.iosrjournals.org

Web Application Security

Chinchu Thankachan[1], Dr. C. N. Kayte[2]
[1,2](Dept. Of Digital & Cyber Forensic, Govt. Institute Of Forensic Science, Aurangabad, India)

Abstract : *Web Applications are very common now a days. Almost all the services are provided through web applications too. So web applications are important and its security relies on server-side. For securing web applications its server-side must be secured. Web applications can be attacked through SQL injection, cross-site scripting, cookie theft, browser hijacking etc. This paper is going to deal with attacks like SQL injection, cross-site scripting, file inclusion vulnerabilities and cross site request forgery.*

Keywords: *Cross Site Request Forgery, Cross-Site Scripting, File Inclusion Vulnerability, SQL Injection, Web Application, Web Application Security*

I. Introduction

Early computer crimes often involved physical damage to computer systems and subversion of the long-distance telephone networks. They included direct damage to computer centres. Most of the attacks were done for taking revenge. And they cause monitory loss to the victim but no financial gain to the attackers. The most of these were done at the time of 1960s and 1970s. The incidents of physical abuse of computer systems did not stop as other forms of computer crime increased [2]. Now the trends have changed and most of the attacks are focused on web applications. Some of the reason attackers choose web based attacks are:

- Network or operating system software attacks are becoming difficult. The number of exploitable vulnerabilities at this layer is getting reduced to a great extent. Most of these software vendors deploy rigorous testing process to identify most of the vulnerabilities even before the release of the software. Once the software is released, these vendors release security fixes for any vulnerabilities detected or reported within an acceptable timeframe. This reduces the possibility of vulnerabilities kept unaddressed from the operating system vendors. Additionally, implementation of security controls such as segmentation using Firewalls, better patch management, controlled network access etc. are making it difficult for an attacker to perform effective penetration [3].
- While the operating system software are developed by well-known software vendors; the applications, specifically web applications are developed by not so well known vendors. Considering the uniqueness of the web applications, less focus, in terms of security, is given to the web applications. Not every web application development company has a solid security testing team and thus may not undergo proper security reviews of the application, also not to forget the freelance web developers sourced over the web. Additionally, the budgetary constraints of a web application development project also make it difficult to ensure the security aspects of the web application [3].
- In case of widely used software's, such as operating systems and some application frameworks, security researchers and end users have mechanisms and interest to report vulnerabilities. However; when it comes to web applications since most of them are custom built, not many security researchers take interest in finding vulnerabilities and/or reporting them to the companies. Most of the time, these vulnerabilities on the web applications are detected and exploited by the hackers. Often the businesses come to know about these vulnerabilities only after a compromise [3].
- Compromising a web application gives better financial benefit for an attacker. Compromising a card database or e-commerce application gives a financial opportunity and is a major incentive [3].
- Hackers consider exploiting a web application is easier than exploiting an underlying operating system. In addition, exploiting a web application gives higher incentives for an attacker [3].

World Wide Web has evolved from a system that delivers static pages to a platform that supports distributed applications, known as web applications and become one of the most prevalent technologies for information and service delivery over Internet. The increasing popularity of web application can be attributed to several factors, including remote accessibility, cross-platform compatibility, fast development, etc. As web applications are increasingly used to deliver security critical services, they become a valuable target for security attacks. Many web applications interact with back-end database systems, which may store sensitive information (e.g., financial, health), the compromise of web applications would result in breaching an enormous amount of information, leading to severe economic losses, ethical and legal consequences[4].

So it is important to secure the web application. The best method to secure the web application is to secure the server-side. This can be done by coding server-side properly. Security is not a one-time event. It is insufficient to secure your code just once. A secure coding initiative must deal with all stages of a program's lifecycle. Secure web applications are only possible when a secure SDLC is used. Secure programs are secure by design, during development, and by default [1].

II. Review From Literature

No language can prevent insecure code, although there are language features which could aid or hinder a security-conscious developer [4]. So we should have to find methods to eliminate the vulnerability in code that make the web application insecure. This paper is going to explain the vulnerabilities given below:

i. Cross-Site Scripting(XSS)
ii. SQL Injection
iii. Inclusion Vulnerabilities
iv. Cross Site Request Forgery (CSRF)

1.1 Vulnerabilities

2.1.1Cross-Site Scripting(XSS)

XSS usually affects web browser on the client side. Here are three distinct types of XSS attacks: the Persistent, Non-Persistent and DOM-base attack.

a) Persistent

Persistence XSS vulnerability is a more devastating variant of a cross site scripting flaw. It occurs when the data provided by the attacker is saved by the server, and then permanently displayed on normal pages returned to other users in the course of regular browsing without proper HTML escaping. Persistent XSS can be more significant than other types because the attacker malicious script is rendered automatically, without the need to individually target victims or lure them to the third party website [5].

b) Non-Persistent

In non-persistent type of XSS attack the code is injected and send back to the off server visitor. It is known as reflected XSS. Malicious code will get executed in a target browser which act as a victim and the payload is not stored anywhere else. It returns as a part of HTML response. Typically there are 3 steps research, social engineering and payload execution [5].

c) XSS DOM-BASE ATTACK

The Document Object Model is a convention for representing and working with objects in an HTML document (as well as in other document types). Basically all HTML documents have an associated DOM, consisting of objects representing the document properties from the point of view of the browser. Whenever a script is executed client-side, the browser provides the code with the DOM of the HTML page where the script runs, thus, offering access to various properties of the page and their values, populated by the browser from its perspective [5].

The success of this attack requires the victim to execute a malicious URL which may be crafted in such a manner to appear to be legitimate at first look. When visiting such a crafted URL, an attacker can effectively execute something malicious in the victim's browser. Some malicious JavaScript, for example, will be run in the context of the web site which possesses the XSS bug.

Here is a sample piece of code which is vulnerable to XSS attack:

```
<form action="search.php" method="GET" /> Welcome!! <p>Enter your name: <input type="text" name="name_1" /><br /> <input type="submit" value="Go" /></p><br> </form>
 <?php echo "<p>Your Name <br />"; echo ($_GET[name_1]); ?>
```

In this example, the value passed to the variable 'name_1' is not sanitized before echoing it back to the user. This can be exploited to execute any arbitrary script [1].

Here is some example exploit code:

http://victim_site/clean.php?name_1=<script>code</script> or
http://victim_site/clean.php?name_1=<script>alert(document.cookie);</script>

2.11SQL Injection

SQL injection attacks are initiated by manipulating the data input on a Web form such that fragments of SQL instructions are passed to the Web application. The Web application then combines these rogue SQL fragments with the proper SQL dynamically generated by the application, thus creating valid SQL requests.

These new, unanticipated requests cause the database to perform the task the attacker intends [6]. SQL injection is a very old approach but it's still popular among attackers. This technique allows an attacker to retrieve crucial information from a Web server's database. Depending on the application's security measures, the impact of this attack can vary from basic information disclosure to remote code execution and total system compromise.

Here is an example of vulnerable code in which the user-supplied input is directly used in a SQL query:

```
<form action="sql.php" method="POST" /> <p>Name: <input type="text" name="name" /><br /> <input type="submit" value="Add Comment" /></p> </form>
<?php $query = "SELECT * FROM users WHERE username = '{$_POST['username']}"; $result = mysql_query($query); ?>
```

The script will work normally when the username doesn't contain any malicious characters. In other words, when submitting a non-malicious username (steve) the query becomes:

```
$query = "SELECT * FROM users WHERE username = 'steve'";
```

However, a malicious SQL injection query will result in the following attempt:

```
$query = "SELECT * FROM users WHERE username = " or '1=1'";
```

As the "or" condition is always true, the mysql_query function returns records from the database [1].

2.1.1Remote and Local File Inclusion Vulnerabilities

RFI/LFI attacks enable hackers to execute malicious code and steal data through the manipulation of a company's web server. RFI and LFI attacks take advantage of vulnerable PHP Web application parameters by including a URL reference to remotely hosted malicious code, enabling remote execution. PHP is a programming language designed for Web development and is in use across more than 77% of applications on the Internet [7].

2.1.1Cross Site Request Forgery (CSRF/XSRF)

Cross site request forgery (abbreviated XSRF or CSRF, sometimes also called "Session Riding"), denotes a relatively new class of attack against web application users. By launching a successful XSRF attack against a user, an adversary is able to initiate arbitrary HTTP requests from that user to the vulnerable web application. Thus, if the victim is authenticated, a successful XSRF attack effectively bypasses the underlying authentication mechanism. Depending on the web application, the attacker could, for instance, post messages or send mails in the name of the victim, or even change the victim's login name and password. Furthermore, the damage caused by such attacks can be severe. In contrast to the well-known web security problems such as SQL injection and XSS, cross site request forgery (XSRF) appears to be a problem that is little known by web application developers and the academic community. As a result, only few mitigation solutions exist. Unfortunately, these solutions do not offer complete protection against XSRF or require significant modifications to each individual web application that should be protected [8].

2. Methodologies

This paper is going to deal with above said vulnerabilities using PHP.

3. Counter Measures

4.1. Cross-Site Scripting(XSS)

Cross-site scripting attack is a result of accepting unchecked input data and showing them in the browser. Suppose you have a comment form in your web application that allows users to enter data, and on successful submission it shows all the comments. The user could possibly enter a comment that contains malicious JavaScript code in it. When the form is submitted, the data is sent to the server and stored into database. Afterward the comment is fetched from database and shown in the web page and the JavaScript code will run. To protect the web application from these kinds of attacks, run the input data through **strip_tags()** to remove any tags present in it. When showing data in the browser, apply **htmlentities()** function on the data.

4.2. SQL Injection

The methods which are commonly used for preventing SQL injection are mysql_real_escape_string() and addslashes(). What mysql_real_escape_string() and addslashes() does is it adds an escape character, the backslash before certain potentially dangerous characters in a string passed to the functions inluding single quotes and doubles which are mainly used in SQL injection.

But the problem lies in the cases where you are expecting a number from user, then the query may be

```
$query = "SELECT * FROM users WHERE userid = $userid";
```

This can be easily cracked with

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

So we cannot say that these two methods are 100 % secure. But we could use **prepare()** to avoid SQL injection. The query can be write as :

```
$query="select * from userids where userid=? and password=?";
$stmt=$conn→prepare($query) ;
$stmt→bind_param("ss",$userid,$password);
```

How does it work is first an SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters which are labelled as " ?". Secondly the database parses, compiles, and performs query of optimization on the SQL statement template, and stores the result without expecting it. At a later time, the application binds the values to the parameters, and the database execute the statement.

4.3. Remote and Local File Inclusion Vulnerabilities

The best way to avoid this attack is to avoid use of arbitrary input data in a literal file include request. If it is important to include such user submitted input create an array which have the allowed file names.

4.4. Cross Site Request Forgery (CSRF)

In a Cross Site Request Forgery (CSRF) attack, the attacker tricks the victim into loading sensitive information or making a transaction without their knowledge. This mainly occurs in web applications that are badly coded to trigger business logic using GET requests.

Ideally, GET requests are Idempotent in nature. Idempotency means the same page can be accessed multiple times without causing any side effects. Therefore, GET requests should be used only for accessing information and not for performing transactions.

Here is a code which is vulnerable to CSRF attacks:

```
<?php
if (isset($_REQUEST["name"], $_REQUEST["amount"])) {
// process the request and transfer the amount from
// from the logged in user to the passed name.}
?>
```

Let's assume Bob wants to perform a CSRF attack on Alice, and constructs a URL like the following and sends it to Alice in an email:

```
<a href="http://example.com/process.php?name=Bob&amount=1000">Visit My WebSite</a>
```

If Alice clicks on this link, and is logged into the website already, this request will deduct $1000 from her account and transfer it to Bob's!

The solution is to process any function that changes the database state in POST request, and avoid using $_REQUEST. Use $_GET to retrieve GET parameters, and use $_POST to retrieve POST parameters.

III. Conclusion

Most of the companies nowadays use web applications to improve their business. Most of the services are provided through web applications that make clients happy. It is developer's duty to make the web application secure. Most of the attacks are due to improper handling of input data. Proper care should be taken while handling input data. PHP have a lot of features that help in this.

References

[1]. Ashwani Garg and Shekhar Singh, "A Review on Web Application Security Vulnerabilities", IJARCSSE, 2013
[2]. Denzyl P. Dayal, "Handbook Of Cyber Crimes"(Dominant Publishers & Distributors Pvt Ltd)
[3]. http://www.ciso.in/why-we-need-web-application-security/
[4]. Xiaowei Li and Yuan Xue, "A Survey on Web Application Security", Department of Electrical Engineering and Computer Science Vanderbilt University
[5]. Punam Thopate, Purva Bamm, Apeksha Kamble, Snehal Kunjir and Prof S.M.Chawre, "Cross Site Scripting Attack Detection & Prevention System", IJARCET, Volume 3 Issue 11, November 2014
[6]. SQL Injection Attack: Detection, www.dbnetworks.com
[7]. Hacker Intelligence Initiative, Monthly Trend Report #8, April 2012, Imperva.
[8]. Nenad Jovanovic, Engin Kirda, and Christopher Kruegel, "Preventing Cross Site Request Forgery Attacks", Secure Systems Lab, Technical University of Vienna