IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 9, June 2022
Impact Factor: 6.252

Loop Holes in Web Based Security

Kashiram Jeevan Vengurlekar
Student, Master of Computer Application
Late Bhausaheb Hiray S S Trust's Institute of Computer Application, Mumbai, India

Abstract: *Loopholes in web-based security is very important topic to learn and understand during creating and running an web based application, as there are many different was to breach an application some of them are since the 1st ever site was made, such as SQL Injection, Broken Authentication & Session Management, XSS, Security Misconfiguration, CSRF and many more, where there is many different ways to manage those type of vulnerability like sanitization of data before accepting it, proper validation of data, proper handling and configuring of web-based application.*

Keywords: *SQL Injections, Cross Site Scripting (XSS), Broken Authentication & Session Management, Insecure Direct Object References, Security Misconfiguration, Cross-Site Request Forgery (CSRF)*

I. INTRODUCTION

The most important part of web-based applications is security, where there are different types of attacks such as users' data stealing, password hacking, session hijacking, etc. As the growth of building a web application began at the same time breaching of application started. The popularity of web-based applications attracts hackers and invites towards them. In today's time secure and maintaining a web-based application is very difficult and challenging as compare to old time. Where the loopholes of an web application plays an important role, in breaching an security of an web-based application.

As more and more application are develop to provide much better security and critical service they become hotspot for attacker to attack. Where most of the application deal with back-end database system, which might store some sensitive information like, financial related, government related, health related or other sensitive information which might cause a big impact on a single person or a community if it might get on wrong hands. As it might lead to severe economic losses, ethical and legal consequences. A breach report from Verizon shows that web applications now reign supreme in both the number of breaches and the amount of data compromised.

A web based application in mostly hosted on a very complex infrastructure which have some kind of security already present on it, but even though it's been compromised with low level of setting in insufficient security Misconfiguration or incomplete knowledge of a particular system, which leads to the threat to security.

Where software are divided into two groups of category Open Source and Closed Source. Most of may think that close source is much secure that open source, but it's actually it's much complicated that we think, as both have their own benefits and flaws which will further discuss in this research paper.

II. THE MOST COMMON LOOP HOLES IN WEB-BASED SECURITYARE

1. SQL Injections
2. Cross Site Scripting(XSS)
3. Broken Authentication & Session Management
4. Insecure Direct Object References
5. Security Misconfiguration
6. Cross-Site Request Forgery(CSRF)

2.1 SQL Injections

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. An SQL injection attack have to be perform using insertion or "injection" of a SQL query via the input data from the client to the web based application. If an SQL Injection is done properly than one can get all type of data which is present on that database very easily, or one can easily modify the data e.g. edit, update or delete the data present on database of

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 9, June 2022
Impact Factor: 6.252

that web based application. One can even execute administration operations like shutdown the DBMS or even one can even run commands that can roll-back the dataset of DBMS and also recover the files which was deleted before and many more. These types of commands are inserted into data-plane input in order to effect the execution of predefined SQL commands. Some known SQL database such as MySQL, Oracle, SQL Server, or others. An unauthorized person can access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more through SQL Injection. The SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application vulnerabilities.

A. How and Why Is an SQL Injection Attack Performed?

To make an SQL Injection attack, an attacker must first find vulnerable user inputs within the web page or web application. There are many option or ways how an attacker can attack, few of them are like Cookies, Server Variables. An attacker can easily modify cookies to attack a web application's database query. Cookies store client state information locally, and web applications commonly load cookies and process that information. A malicious user, or malware, can modify cookies to inject SQL into the back-end database which leads to a classic SQL Injection Vulnerability, and also server variables can also be used as a SQL injection attack vector. A variable can containing arbitrary SQL can inject that code into the database if the web application fails to sanitize those inputs as well.

Type of SQL attack which could be bifurcate as Classic SQL Injection & Blind SQL Injection.

How a Classic SQL Injection Work global$var;

```
$title = $var->get_var("select post_title from " .
$wpdb->posts ." where ID=" .
$_GET['id']);
//Do something with title, but don't echo.
```

The above code is an example of a SQL injection (SQLi) vulnerability. It is an SQLi vulnerability because the user input in $_GET[‘id’] is sent directly to the database without sanitization or escaping. This allows an attacker to send commands directly to the database.

The database output is then sent directly back to the user’s browser. Because the output is sent to the browser, this makes the vulnerability a classic SQLi vulnerability, as opposed to a blind SQL injection vulnerability, which is discussed below.

Using this vulnerability an attacker can send commands directly to the database. These include SELECT commands to download your entire database including any user personally identifiable information (PII). In some cases it also includes INSERT and UPDATE commands to create new user accounts or modify existing user accounts.

How a Blind SQL Injection Works. global$var;

```
$title = $var->get_var("select post_title from " .
$wpdb->posts ." where ID=" .
$_GET['id']);
//Do something with title, but don't echo.
```

In the above example, raw unsanitized user input is sent directly to the database by concatenating the $_GET[‘id’] variable directly to the SQL query. To fix this vulnerability, you would simply use the prepare() method as above to sanitize and escape any database input.

The difference here is that the output is never sent to the browser. A blind SQLi vulnerability is just as serious as a regular SQLi vulnerability because an attacker can in some cases easily insert or update data in your database. The difference is that it becomes more difficult to extract data from the database because the attacker can’t see the output of the database because it is not written to the web browser.

B. Cross Site Scripting (XSS)

Cross Site Scripting is also known as XSS. It occurs when a vulnerability in an application enables an attacker to insert a malicious script—typically JavaScript—into the vulnerable website’s code. As the malicious script is added to the

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 9, June 2022
Impact Factor: 6.252

site, user's browser automatically runs the script when he or she visits the affected website. This type of XSS is designed to carry out variety of malicious acts, without user knowing that user's credential, data, or other important data like identity are been compromised.

In cross site scripting is an client side scripting attack where an attacker acts as a user where attacker writes a script on those places where user accept inputs like login pages, search boxes, comment fields sign-up forms, forms requesting name, address, phone, and credit card numbers and so on where a user can input any kind of data into it.

Cross-site Scripting may also be used to deface or damage a website instead of targeting the user. The attacker can use injected scripts to change the content of the website or even redirect the browser to another web page, for example, one that contains malicious code.

Cross-site Scripting (XSS) vulnerabilities are considerable less dangerous than for example SQL Injection vulnerabilities. Consequences of the ability to execute JavaScript on a web page may not seem harmful at first, because most of the web browsers run JavaScript in a very tightly controlled environment. JavaScript has limited access to the user's operating system and the user's files. However, JavaScript can still be dangerous if misused as part of malicious content.

Types of Cross-Site Scripting Attacks.

Stored (Persistent) Cross-Site Scripting

Stored cross-site scripting (also known as second-order or persistent XSS) where it appear when an application receives data from an untrusted source and includes that data within it's after there is an HTTP responses in an unsafe way. Suppose a website allows users to submit comments on blog posts, which are displayed to other users. Users submit comments using an HTTP request. Once the user has post the comment user receives an application's acknowledgement that the post has been successfully commented. Assuming the application doesn't perform any other processing of the data, an attacker can submit a malicious script. So now when any user will it visit the blog post this malicious script will also run and can get the users cookies data to attacker.

Reflected Cross-Site Scripting

Reflected cross-site scripting (or XSS) occurs when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way. Suppose a website has a search function which receives the user-supplied search term in a URL parameter:

https://insecure-website.com/search?term=gift

The application echoes the supplied search term in the response to this URL:

You searched for: gift

Assuming the application doesn't perform any other processing of the data, an attacker can construct an attack like this:

https://insecure-website.com/status?message=<script>/*+Bad+stuff+here...+*/</script>

This URL results in the following response:

You searched for: <script>/* Bad stuff here... */</script>

If another user of the application requests the attacker's URL, then the script supplied by the attacker will execute in the victim user's browser, in the context of their session with the application.

Self Cross-Site Scripting

Self Cross site scripting(XSS) is a vulnerability in web applications which gives the ability of executing JS as the same user and not to other users. Self cross-site scripting occurs when attackers exploit a vulnerability that requires extremely specific context and manual changes. The only one who can be a victim is yourself. These specific changes can include things like cookie values or setting your own information to a payload.

Self-ReflectedXSS occurs when a user is able to execute JS using the URL in his own account/browser only because of the existence of a variable(Ex: Token, userID,...etc) in the URL.

Self-StoredXSS occurs when a user is able to execute JS using the URL in his own account/browser only (because of the existence of HTTP session tokens).

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 9, June 2022
Impact Factor: 6.252

Blind Cross-Site Scripting

Blind XSS vulnerabilities are a variant of persistent XSS vulnerabilities. They occur when the attacker input is saved by the server and displayed in another part of the application or in another application. For example, an attacker injects a malicious payload into a contact/feedback page and when the administrator of the application is reviewing the feedback entries the attacker's payload will be loaded. The attacker input can be executed in a completely different application (for example an internal application where the administrator reviews the access logs or the application exceptions). These types of vulnerabilities are much harder to detect compared to other Reflected XSS vulnerabilities where the input is reflected immediately. In the case of Blind XSS, the attacker's input can be saved by the server and only executed after a long period of time when the administrator visits the vulnerable Dashboard page. It can take hours, days or even weeks until the payload is executed. Therefore, this type of vulnerabilities cannot be tested as the other type of XSS vulnerabilities.

DOM-Based Cross-Site Scripting

DOM XSS stands for Document Object Model-based Cross-site Scripting. This kind of XSS attack occurs when an application receives some client-side JavaScript that processes data from an unsafe, or untrusted source by writing the data to a potentially dangerous sink within the DOM instead of writing data in HTML which would present a regular XSS DOM-based cross-site scripting attacks occur when the server itself isn't the one vulnerable to XSS, but rather the JavaScript on the page is. As JavaScript is used to add interactivity to the page, arguments in the URL can be used to modify the page after it has been loaded. By modifying the DOM when it doesn't sanitize the values derived from the user, attackers can add malicious code to a page.

An example of DOM-based cross-site scripting attack would be when the website changes the language selection from the default one to one provided in the URL.

Broken Authentication & Session Management

According to the most recent OWASP Top 10 list, "Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently." Attackers are able to detect this type of broken authentication manually once they have got know that this is an broken authentication they try different types of automated tool to hack such type application or websites.

a) Threat Agents / Attack Vectors

Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.

b) Security Weakness

The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.

c) Impacts

Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.

d) OWASP lists a number of reasons why an application may be vulnerable,including:

- User authentication credentials aren't protected when stored using hashing orencryption.
- Credentials can be guessed or overwritten through weak account managementfunctions.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 9, June 2022
Impact Factor: 6.252

- Session IDs are exposed in the URL.
- Session IDs are vulnerable to session fixation attacks.
- SessionIDsdon'ttimeout,orusersessionsorauthenticationtokens,particularlysinglesign-ontokens,aren't properly invalidated during logout.
- Session IDs aren't rotated after a successful login.
- Passwords, session IDs, and other credentials are sent over unencrypted connections.

B) Insecure Direct Object References

Insecure Direct Object Reference (called IDOR from here) occurs when a application exposes a reference to an internal implementation object. Using this way, it reveals the real identifier and format/pattern used of the element in the storage backend side. The most common example of it (although is not limited to this one) is a record identifier in a storage system (database, file system and so on).

By exploiting Insecure Direct Object References, attackers can bypass authorization and access resources directly by modifying the value of a parameter used to directly point to an object (i.e. by modifying the user account ID in a URL string to access the information of other users) . The potentially accessed resources can be database entries belong to other users, files in the system, and more. The references pointing to these resources, which may be exploited by attackers, can be a database key or a directory of a file. If the application takes user supplied input and uses it to retrieve an object without performing sufficient authorization checks, this vulnerability is enabled.

Classic Case of Yahoo Suggestions vulnerability

In the case of Yahoo! Suggestions vulnerability, an attacker could modify the parameters in HTTP POST requests to delete 1.5 million records entered by Yahoo users. In the blog of Ibrahim Raafat, an Egyptian Cyber Security Analyst who report this vulnerability, he described how he found it: At first, he found he could delete his comments, so he opened Live HTTP Headers to check the content in the POST request, where he found out that he was able to delete other comment by simply changing the tid&cid of another user comment. So Just by simply modifying some HTTP headers request he was able to delete 1.5 million records entered by yahoo users.

E.g. prop=addressbook&fid=367443&crumb=Q4.PSLBfBe.&cid=1236547890&cmd= delete_comment

C) Security Misconfiguration

Security misconfiguration can be dangerous at times because it is easy to detect misconfigured web servers and applications and then exploit them. Whenever the implementation of security controls for a server or a web application fails or is met with errors, it is referred to as a security misconfiguration. Sometimes a safe environment of an organization built by several professionals (systems administrators, DBAs, or developers) is left with vulnerable gaps. These security loopholes then lead the organization to grave risks. The occurrence of failure of security safeguards can occur at any level of the application stack. From the platform of the web application to its web server and web application server; it also includes its database (containers or storage), framework, custom code, and pre-installed VMs. The perpetrators get to these vulnerabilities through unauthorized access to default accounts, rarely accessed web pages, not frequently updated applications, unprotected files and folders, directory listings, and so on. Once the system falls prey to the vulnerability, the sensitive data might get stolen or altered, and to recover from such a situation is a time-consuming and costly affair.

a) A few typical examples of security misconfiguration are listed below:

- Applications and products under production phase in debug mode
- Running unwanted services on the system
- No proper configuration for accessing the server resources and services
- Leaving default keys and passwords as it is
- Incorrect exception management—can disclose unauthorized data, including stack traces
- Using default accounts with default credentials

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 9, June 2022
Impact Factor: 6.252

b) Real-Life Damages by Security Misconfiguration. Case 1: Accidental S3 Data Leaks by AWS

The data of around 14 million Verizon subscribers were exposed on an unsecured Amazon S3 bucket. Under this massive data exposure of 2017, the phone numbers and account PINs of the customers were compromised. The data was accessible and downloadable to anyone who can get their hands on the right web address.

Case 2: Accenture Exposed 137 GB of Data

The misconfigured security aspect of servers hosted on Amazon's S3 storage led to 2018's compromise of highly sensitive data of Accenture. The Key Management System of Accenture was out in public and would have allowed an attacker to gain complete access to the encrypted data of the organization. The exposed servers contained various customer credentials and private keys to sign in, which were stored in plaintext.

c) Cross-Site Request Forgery(CSRF)

Cross-site request forgery (also known as CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other.

In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. For example, this might be to change the email address on their account, to change their password, or to make a funds transfer.

Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality.

D) How does CSRF work?

For a CSRF attack to be possible, three key conditions must be in place:

- **A relevant action.** There is an action within the application that the attacker has a reason to induce. This might be a privileged action (such as modifying permissions for other users) or any action on user-specific data (such as changing the user's own password).
- **Cookie-based session handling.** Performing the action involves is using one or more HTTP requests, and the application relies solely on session cookies to identify the user who has made the requests. There is no other mechanism in place for tracking sessions or validating user requests.
- **Noun predictable request parameters.** The requests that perform the action do not contain any parameters whose values the attacker cannot determine or guess. For example, when causing a user to change their password, the function is not vulnerable if an attacker needs to know the value of the existing password.

III. CONCLUSION

This paper provided a overall view of the loops holes that are present in Web Based Security. As there are many ways to break through the security of an application by various means as mention above. We as users need to be aware of such type of variability of the web-based application which could cause a loss in data, loss in identity, or some other way of security loss. This is why we should be always prepared for such kind of malicious activity and take a step forward and find all the loopholes which could cause our security ar risk in our application before we make it on production.

REFERENCES

[1]. DPLOOP: Detection & Prevention of Loopholes in Web Application Security.
[2]. Security Estimation Framework for Development of Secure Software and Cyber Attacks.
[3]. Idea: Towards Architecture-Centric Security Analysis of Software
[4]. A Survey on Web Application Security
[5]. Security of Open Source Software
[6]. https://www.acunetix.com/websitesecurity/sql-injection/

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 9, June 2022
Impact Factor: 6.252

[7]. https://www.oreilly.com/library/view/securing-node-applications/9781491982426/ch04.html
[8]. https://blog.eccouncil.org/most-common-cyber-vulnerabilities-part-5-security-misconfiguration/