ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

Classification of Web Application Vulnerabilities

Savita B. Chavan
M.Tech Computer Engineering
V.J.T.I, Mumbai, India

Dr. B. B. Meshram
Head of Computer Technology Department
V.J.T.I, Mumbai, India

Abstract*— Web applications provide access to increasing amounts of information, some of which is confidential. From an application perspective, vulnerability identification is absolutely critical and often over looked as a source of risk. The outlines of this paper is to classify the attacks and weaknesses that can lead to the compromise of a website, its data, or its users In this paper web application vulnerabilities are classified on the basis of requirement analysis, design, implementation and deployment of web application.

***Index Terms*— Cross Site Scripting, SQL Injection, Brute Force Attack, Buffer Overflow, Session Fixation, Insufficient Authentication, Application Misconfiguration, Session Expiration, Content Spoofing, Command Injection, Path Traversal.**

I. INTRODUCTION

Over the past few years, a clear trend has emerged within the information security landscape; web applications are under attack. "Web applications continue to be a prime vector of attack for criminals, and the trend shows no sign of abating; attackers increasingly shun network attacks for cross-site scripting, SQL injection, and many other infiltration techniques aimed at the application layer." Web application vulnerabilities can be attributed to many things including poor input validation, insecure session management, improperly configured system settings and flaws in operating systems and web server software. Certainly writing secure code is the most effective method for minimizing web application vulnerabilities. However, writing secure code is much easier said than done and involves several key issues.Web security vulnerabilities continually impact the risk of a web site. When any web security vulnerability is identified, performing the attack requires using at least one of several application attack techniques. These techniques are commonly referred to as the class of attack (the way security vulnerability is taken advantage of). Many of these types of attack have recognizable names such as Buffer Overflows, SQL Injection, and Cross-site Scripting. As a baseline, the class of attack is the method the Web Application vulnerabilities Classification will use to explain and organize the threats to a website [3].

II. BACKGROUND

Over the last several years, the web security industry has adopted dozens of confusing and esoteric terms describing vulnerability research. Terms such as Cross-site Scripting, Parameter Tampering and Cookie Poisoning have all been given inconsistent names and double meanings attempting to describe their impact. For example, when a web site is vulnerable to Cross-site Scripting, the security issue can result in the theft of a user's cookie. Once the cookie has been compromised, this enables someone to perform a session hijacking and take over the user's online account. To take advantage of the vulnerability, an attacker uses data input manipulation by way of URL parameter tampering. This previous attack description is confusing and can be described using all manner of technical jargon. This complex and interchangeable vocabulary causes frustration and disagreement in open forums, even when the participants agree on the core concepts [2].

III. CLASSIFICATION OF WEB APPLICATION VULNERABILITIES

Web application vulnerabilities classification will compile and distill the known unique classes of attack, which have presented a threat to web sites in the past. Each class of attack will be given a standard name and explained with thorough this paper discussing the key points. The classes are describing as follows [3]:

A. Requirement Analysis

Covers vulnerabilities that are likely to be introduced due to a lack of mitigations specified in the software requirements, or due to a poorly /improperly defined requirement.

1. Broken Access Control

Access control, sometimes called authorization, is how a web application grants access to content and functions to some users and not others. These checks are performed after authentication, and govern what 'authorized'

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

users are allowed to do. Access control sounds like a simple problem but is insidiously difficult to implement correctly. A web application's access control model is closely tied to the content and functions that the site provides. In addition, the users may fall into a number of groups or roles with different abilities or privileges [4].

- Access unauthorized content - ability to view additional reports or to perform some additional task(s)
- Access unauthorized account(s) - ability to access another user(s) account that may be able to perform additional functions
- If the vulnerability grants access to an unauthorized individual, the potential impact could be devastating. Especially, if this exploitation allows that individual to access administrative interfaces essentially giving the hacker they 'keys to the castle,' so to say.
- Accessing restricted files
- Compliance and/or legal ramifications

2. Abuse of Functionality

Abuse of Functionality is an attack technique that uses a web sites own features and functionality to attack it or others. Abuse of Functionality can be described as the abuse of an applications intended functionality to perform an undesirable outcome. These attacks have varied results such as consuming resources, circumventing access controls, or leaking information. The potential and level of abuse will vary from web site to web site and application to application [3]. Examples of Abuse of Functionality are [2]:

- Abusing Password-Recovery Flows
- Abusing functionality to make unrestricted proxy requests

3. Improper Error Handling Attacks

Improper error handling happens when web applications do not limit the amount of information they return to their users. A classic example of improper error handling is when an application doesn't sanitize SQL error messages that are returned to the user. Upon receiving a SQL error message an attacker will immediately identify a place for identifying injection flaws. Although preventing error messages from reaching users will not prevent vulnerabilities from occurring, it does make it difficult for an attacker to accomplish his goal and it is also an industry best practice. Example [3]:

In the following example, sensitive information might be printed depending on the exception that occurs.

```
try {
/.../
}
catch (Exception e) {
System.out.println(e);
}
```

If an exception related to SQL is handled by the catch, then the output might contain sensitive information such as SQL query structure or private information. If this output is redirected to a web user, this may represent a security problem.

B. Design

Covers vulnerabilities that are likely to be introduced due to a lack of mitigations specified in the software design, or due to a poorly /improperly defined design [3].

1. Brute Force

A Brute Force attack is an automated process of trial and error used to guess a person's username, password, credit-card number or cryptographic key. Simple brute-force attack may have a dictionary of all words or commonly used passwords and cycle through those words until it gains access to the account. A more complex brute-force attack involves trying every key combination in an effort to find the correct password that will unlock the encryption. Due to the number of possible combinations of letters, numbers and symbols, a brute force attack can take a long time to complete [1].

Example [1]:

Username = Jon

Passwords = smith, michael-jordan, [pet names], [birthdays], [car names]

Usernames = Jon, Dan, Ed, Sara, Barbara,

Password = 12345678

2. Cross-Site Request Forgery

A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf [9]. CSRF attacks are effective in a number of situations, including:

- The victim has an active session on the target site.
- The victim is authenticated via HTTP auth on the target site.
- The victim is on the same local network as the target site.

3. Information Leakage

Information Leakage is when a web site reveals sensitive data, such as developer comments or error messages, which may aid an attacker in exploiting the system. Sensitive information may be present within HTML comments, error messages, source code, or simply left in plain sight [1]. There are many ways a web site can be coaxed into revealing this type of information. While leakage does not necessarily represent a breach in security, it does give an attacker useful guidance for future exploitation. Leakage of sensitive information may carry various levels of risk and should be limited whenever possible.

Example [3]: There are three general categories of Information Leakage: Insufficient censorship of application content, improper server configurations, or Dangerous application behavior.

Developer comments left in page responses:

```
<TABLE border="0" cellPadding="0" cellSpacing="0" height="59" width="591">
        <TBODY>
        <TR>
         <!--If the image files fail to load, check/restart 192.168.0.110 -->
         <TD bgColor="#ffffff" colSpan="5" height="17" width="587">
         </TD>
         </TR>
```

Here we see a comment left by the development/QA personnel indicating what one should do if the image files do not show up. The information being disclosed is the internal IP address of the content server that is mentioned explicitly in the code, "192.168.0.110".

4. Insufficient Authentication

Insufficient Authentication occurs when a web site permits an attacker to access sensitive content or functionality without having to properly authenticate. Web-based administration tools are a good example of web sites providing access to sensitive functionality. Depending on the specific online resource, these web applications should not be directly accessible without requiring the user to properly verify their identity.

Example [3]: Many web applications have been designed with administrative functionality located directly off of the root directory (/admin/). This directory is usually never linked from anywhere on the web site, but can still be accessed using a standard web browser. The user or developer never expected anyone to view this page because it is not linked, so enforcing authentication is many times overlooked. If attackers were to simply visit this page, they would obtain complete administrative access to the web site.

C. Implementation

Covers vulnerabilities that are likely to be introduced due to a poor choice of implementation.

1. Buffer Overflow

The attacker's goal is almost always to control the target process' execution. This is accomplished by identifying a function pointer in memory that can be modified, directly or indirectly, using the overflow. When such a pointer is used by the program to direct program execution through a jump or call instruction, the attacker-supplied instruction location will be used, thereby allowing the attacker to control the process.

Example [3]:

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

```
void bad_function(char *input)
{  char dest_buffer[32];
   strcpy(dest_buffer, input);
printf ("The first command-line argument is %s.\n",dest_buffer);
}
int main (int argc, char *argv [])
{if (argc > 1)
{bad_function (argv [1]);
}
else
{printf  ("No command-line argument was given.\n");
}
return 0;
}
```

In this example, the first command-line argument, argv [1], is passed to bad_function. Here, it is copied to dest_buffer, which has a size of 32 bytes allocated on the stack. If the command-line argument is greater than 31 bytes in length, then the length of the string plus its null terminator will exceed the size ofdest_buffer. The exact behavior at this point is undefined. In practice, it will depend on the compiler used and the contents of the command-line argument; suffice it to say that a string of 40 "A" characters will almost certainly crash the process.

2. Content Spoofing

Content Spoofing is an attack technique that allows an attacker to inject a malicious payload that is later misrepresented as legitimate content of a web application [3].

1. Text Only Content Spoofing

2. Markup Reflected Content Spoofing

Example [2]:

http://foo.example/news?id=123&title=Company+y+stock+goes+up+5+percent+on+news+of+sale

The "title" parameter in this example specifies the content that will appear in the HTML body for the news entries. If an attacker where to replace this content with something more sinister they might be able to falsify statements on the destination website.

Example [2]:

http://foo.example/news?id=123title=Company+y+filing+for+bankrupcy+due+to+insider+corruption,+investors+urged+to+sell+by+finance+analyists

Upon visiting this link the user would believe the content being displayed as legitimate. In this example the falsified content is directly reflected back on the same page, however it is possible this payload may persist and be displayed on a future page visited by that user.

3. Credential/ Session Prediction

Credential/Session Prediction is a method of hijacking or impersonating a web site user. Deducing or guessing the unique value that identifies a particular session or user accomplishes the attack. Also known as Session Hijacking, the consequences could allow attackers the ability to issue web site requests with the compromised user's privileges.

Example [3**]:**

Many web sites attempt to generate session IDs using proprietary algorithms. These custom methodologies might generation session IDs by simply incrementing static numbers. Or there could be more complex procedures such as factoring in time and other computer specific variables.

The session ID is then stored in a cookie, hidden form-field, or URL. If an attacker can determine the algorithm used to generate the session ID, an attack can be mounted as follows:

- Attacker connects to the web application acquiring the current session ID.
- Attacker calculates or Brute Forces the next session ID.
- Attacker switches the current value in the cookie/hidden form-field/URL and assumes the identity of the next user.

4. Cross-Site Scripting

Cross-site scripting is an attack against web applications. It occurs when an web application accepts input containing malicious script through the application. This input is then sent as part of the response to the client as a reflected script which can lead to phishing attack, and the malicious script can be injected in the database as stored form so that whenever the page is accessed the injected script is executed like<Script> alert (document. cookie)</script>[9].

Example [9]:

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

The following sample code illustrates the XSS vulnerability.

```
Iterator iter = request.getParameterNames ();
While (iter.hasNext ()) {
String paramName = (String) iter.next ();
out.println (paramName +request.getParameter (paramName));
}
```

5. Denial of Service

Denial of Service (DoS) is an attack technique with the intent of preventing a web site from serving normal user activity. DoS attacks, which are easily normally applied to the network layer, are also possible at the application layer. These malicious attacks can succeed by starving a system of critical resources, vulnerability exploit, or abuse of functionality.

- **DoS targeting a specific user:** An intruder will repeatedly attempt to login to a web site as some user, purposely doing so with an invalid password. This process will eventually lock out the user.
- **DoS targeting the Database server:** An intruder will use SQL injection techniques to modify the database so that the system becomes unusable (e.g., deleting all data, deleting all usernames etc.)
- **DoS targeting the Web server:** An intruder will use Buffer Overflow techniques to send a specially crafted request that will crashes the web server process and the system will normally be inaccessible to normal user activity.

6. Injecting OS Command

Executing System/Operating System Commands through the application are the sources of Command injection Attacks. Java has the provision to execute system commands with the Runtime. Exec () method. Improper validation of the exec method arguments leads to vulnerability in the source code and attacker can pass malicious commands through the exec argument and gain control of the system.

The following sample code illustrates the Command injection vulnerability [9].

```
Class Exec class
{public static void main (String args [])
{ Runtime rt = Runtime.getRuntime ();
  Process proc = rt.exec ("cmd.exe /C") ;}}
```

7. Path Traversal

The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal.The most basic Path Traversal attack uses the "../" special-character sequence to alter the resource location requested in the URL. Although most popular web servers will prevent this technique from escaping the web document root, alternate encodings of the "../" sequence may help bypass the security filters. These method variations include valid and invalid Unicode-encoding ("..%u2216" or "..%c0%af") of the forward slash character, backslash characters ("..\") on Windows-based servers, URL encoded characters "%2e%2e%2f"), and double URL encoding ("..%255c") of the backslash character [4].

Example [3]:

Path Traversal attacks against a web server

http://example/../../../../../etc/passwd

http://example/..%255c..%255c..%255cboot.ini

http://example/..%u2216..%u2216someother/file

Path Traversal attacks against a web application

Original: http://example/foo.cgi?home=index.htm

Attack: http://example/foo.cgi?home=foo.cgi

In the above example, the web application reveals the source code of the foo.cgi file because the value of the home variable was used as content. Notice that in this case the attacker does not need to submit any invalid characters or any path traversal characters for the attack to succeed.

8. SQL Injections

SQL queries use invalidated user input as vulnerable script. In an SQL injection attack, an attacker sends malicious input through the application interface like login page and these queries are executed in the backend database and attacker can gain control over a database or perform DOS attacks like Database shutdown. The

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

following example illustrates JSP login form where the attacker can enter malicious commands like "XP_cmdshell" or "*; OR 1=1—"etc [1].
The exampleis as follows [10]:

```
String Uname = request.getParameter ("User");
String Pword =request.getParameter (Password");
String s = "SELECT * FROM Table WHERE Username = ' " +
UName + " " AND Password = ' " + Pword " ' ";
Statement stmt = connection.createStatement ();
ResultSet rs = stmt.executeQuery (s);
If (rs.next ()) {
UID = rs.getInt (1)
User = rs.getString (2)
}
PrintWriter writer= response.getWriter ();
writer.println ("User Name: "+ User);
}
```

In the above example [9], the SQL query is created using the username and password transferred to the server without validating the input which is vulnerable to a SQL injection attack. To mitigate the risk of SQL injection is to use only stored procedures or parameterized database calls. Using statement all the queries will be treated as a string but not commands to be executed by the database Structured.

D. Deployment

Covers vulnerabilities that are likely to be introduced due to poor deployment procedures, or bad application/server configurations [3].

1. Insufficient Session Expiration

Insufficient Session Expiration occurs when a Web application permits an attacker to reuse old session credentials or session IDs for authorization. Insufficient Session Expiration increases a Web site's exposure to attacks that steal or reuse user's session identifiers. Session expiration is comprised of two timeout types:

- **Inactivity:** an inactivity timeout is the amount of idle time allowed before the session is invalidated
- **Absolute**: An absolute timeout is defined by the total amount of time a session can be valid without re-authentication

Example [2]:
In a shared computing environment (more than one person has unrestricted physical access to a computer), Insufficient Session
Expiration can be exploited to view another user's web activity. If a web site's logout function merely sends the victim to the site's home page without ending the session, another user could go through the browser's page history and view pages accessed by the victim. Since the victim's session ID has not been expired, the attacker would be able to see the victim's session without being required to supply authentication credentials.

2. Application Misconfiguration

Application Misconfiguration attacks exploit configuration weaknesses found in web applications. Many applications come with unnecessary and unsafe features, such as debug and QA features, enabled by default. These features may provide a means for a hacker to bypass authentication methods and gain access to sensitive information, perhaps with elevated privileges.
Likewise, default installations may include well-known usernames and passwords, hard-coded backdoor accounts, special access mechanisms, and incorrect permissions set for files accessible through web servers. Default samples may be accessible in production environments. Application-based configuration files that are not properly locked down may reveal clear text connection strings to the database, and default settings in configuration files may not have been set with security in mind. All of these misconfigurations may lead to unauthorized access to sensitive information.

1) Example [3]:
The php.ini file includes the expose_php variable that is enabled by default, as follows:
expose_php = 'on'
This default setting causes the application server to reveal in the server header that a specific version of PHP is being used to process requests. The information revealed may be used to formulate an attack that is specific to the PHP version found.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

3. Session Fixation

Session Fixation is an attack technique that forces a user's session ID to an explicit value. Depending on the functionality of the target web site, a number of techniques can be utilized to "fix" the session ID value. These techniques range from Cross-site Scripting exploits to peppering the web site with previously made HTTP requests. After a user's session ID has been fixed, the attacker will wait for that user to login. Once the user does so, the attacker uses the predefined session ID value to assume the same online identity.

Example [3]:

The Session Fixation attack is normally a three step process:

1 **Session set-up:** The attacker sets up a "trap-session" for the target web site and obtains that session's ID. Or, the attacker may select an arbitrary session ID used in the attack. In some cases, the established trap session value must be maintained (kept alive) with repeated web site contact.

2 **Session fixation:** The attacker introduces the trap session value into the user's browser and fixes the user's session ID.

3 **Session entrance:** The attacker waits until the user logs into the target web site. When the user does so, the fixed session ID value will be used and the attacker may take over.

Fixing a user's session ID value can be achieved with the following techniques:

1) **Issuing a new session ID cookie value using a client-side script*:** Cross-site scripting vulnerability present on any web site in the domain can be used to modify the current cookie value.

2) **Issuing a cookie using the META tag:** This method is similar to the previous one, but also effective when Cross-site Scripting countermeasures prevent the injection of HTML script tags and not Meta tags.

3) **Issuing a cookie using an HTTP response header:** The attacker forces either the target web site, or any other site in the domain, to issue a session ID cookie. This can be achieved in many ways:

- Breaking into a web server in the domain (e.g., a poorly maintained WAP server)
- Poisoning a user's DNS server, effectively adding the attacker's web server to the domain
- Setting up a malicious web server in the domain.
- Exploiting an HTTP Response Splitting attack.

IV. COUNTERMEASURE AND WEAKNESS OF WEB APPLICATION VULNERABILITIES

The table shows in below contents the classification, countermeasure and weakness of Web Application Vulnerabilities:

Table 1: Classification, Countermeasure and Weakness of Web Application Vulnerabilities

Classification	Vulnerabilities	Countermeasure	Weakness
Requirement analysis	Broken Access Control Attack	• Configure secure Web permissions. • Lock down files and folders with restricted NTFS permissions. • Use .NET Framework access control mechanisms within your ASP.NET applications	• Access to confidential or restricted data • Execution of unauthorized operations • Tampering
	Abuse of Functionality	• Applications should only perform their intended function • Applications should also verify all user-supplied input to ensure that proper parameters are being passed from the client.	• Password cracking • Unauthorized access
	Improper Error Handling	• Software development team shares a common approach to exception handling. • Disable or limit detailed error handling.	• Disclosure of sensitive system-level • Elevation of privilege
Design	Brute Force	• Use strong passwords • Store non-reversible password hashes in the user store.	• Password cracking • Elevation of privileges
	• Cross-Site Request Forgery	• Requiring a secret, user-specific token in all form submissions and side-effect URLs • Requiring the client to provide authentication data in the same HTTP Request • Checking the HTTP referralheader or checking the HTTP original header.	• An unauthorized action • Forcing the victim's browser to perform a hostile action.
	Information Leakage	• Use strong authorization. • Use strong encryption. • Secure communication links with protocols that provide message confidentiality.	• Access sensitive data in storage • Access user information

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

		• Do not store secrets (for example, passwords) in plaintext	
	2.5.1.4 Insufficient Authentication	• Use Strong Credentials • Handle Credentials Secretively • Prevent Misuse of the Password Change Function	• Password cracking • Elevation of privileges • Unauthorized access
Implementation	Buffer overflow	• Prevent use of dangerous functions: gets strcpy, etc. • Prevent return addresses from being overwritten • Prevent data supplied by the attacker from being executed • Runtime Protections against Buffer Overflows	• Tampering • Execution of unauthorized operations
	Content spoofing	• Use data hashing and signing. • Use digital signatures. • Use strong authorization. • Use tamper-resistant protocols across communication links. • Secure communication links with protocols that provide message integrity.	• Access sensitive data in storage • Network eavesdropping • Datatampering
	1. Credential/ session prediction	• Generate Strong Tokens • Protect Tokens Throughout Their Life Cycle • Per-Page Tokens • Log, Monitor, and Alert	• Session hijacking • session replay • man in the middle
	2. Cross-Site Scripting	• Check and validate all the form fields, hidden fields • Perform a security review of the code. • Find the script output to defeat XSS • Input fields should be limited to a maximum	• Stealing and continuing the session of the victim. • Manipulating files on the victim's. • Performing brute force password cracking
	Denial of Service	• Configure your applications, services, and operating system with denial of service • Stay current with patches and security updates. • Harden the TCP/IP stack against denial of service. • Use an IDS that can detect potential denial of service attacks	• Disclosure of sensitive system-level details • Elevation of privilege
	3. Injecting OS Command	• Restrict Permissions on OS Commands • Whitelist Allowed Characters • Filter out Command Directory Names	• Access to victim system • Access Sensitive data
	Path Traversal	• The application should check whether it contains either of the path traversal. • Use a hard-coded list of permissible file types and reject any request for a different • After performing all its filtering on the user-supplied filename	• File uploading and downloading
	SQL Injection	• Perform thorough input validation. • Use parameterized stored procedures for database • Use least privileged accounts to connect to the database	• Unauthorized access to database • Drop the table from database
Deployment	4. Insufficient Session Expiration	• Forcefully expire a session token after a predefined period of time that is appropriate • Forcefully expire a session token after a predefined period of inactivity • Forcefully expire a session token when the user actuates the log-out function	• Session hijacking and/or identity • Spoofing due to Capture of session ID
	Application Misconfiguration:	• Restricting access to specific directory • Blocking attacks based on predictable resources	• Unauthorized access to administration • Unauthorizedaccess to configuration
	Session Fixation	• Session set-up. • Session fixation. • Session entrance	• Predictable tokens, • insecure handling of tokens

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

V. CONCLUSION

In this paper classes of web application vulnerabilities are described which intrude the web application security. There are many different ways an attacker can break into a system and wreak havoc on a network or computer system. It is up to the web application coder to do their part in making sure the applications they design are not vulnerable to any know threats. The attacker can attack the web application if there is fault in design, implementation and deployment of web application.

REFERENCES

[1] "The Web Application Hackers Handbook" by Dafydd Stuttard and Marcus Pinto, WillyIndia Publication.

[2] "Improving web application Security" Threats and Countermeasure by Mark Curphey, Joel Scambray, and Erik Olson, Microsoft Publication.

[3] Web Application Security Consortium version 2.0.0 by webappsec.org. www.owasp.org.

[4] David Scott and Richard Sharp," Specifying and Enforcing Application-Level Web Security Policies", IEEE Transactions On Knowledge And Data Engineering, Vol. 15, No. 4, July/August 2003.

[5] Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari, and Roberto Zunino," Semantics-Based Design for Secure Web Services", IEEE Transactions On Software Engineering, Vol. 34, No. 1, January/February 2008.

[6] LievenDesmet, Pierre Verbaeten, Member, IEEE, Wouter Joosen, and Frank Piessens," Provable Protection against Web Application Vulnerabilities Related to Session Data Dependencies", IEEE Transactions On Software Engineering, Vol. 34, No. 1, January/February 2008.

[7] XiaoFeng Wang, Member, IEEE, and Michael K. Reiter, Senior Member, IEEE," Using Web-Referral Architectures to Mitigate Denial-of-Service Threats", IEEE Transactions On Dependable And Secure Computing, Vol. 7, No. 2, April-June 2010.

[8] Shynlie Simmons," Hacking Techniques: Web Application Security".

[9] "Security Code Review- Identifying Web Vulnerabilities"by Kiran Maraju.

[10] TejinderSingh,"Securing Web Applications And Finding Security Vulnerabilities In Java", IJREAS Volume 2, Issue 3 (March 2012) ISSN: 2249-3905.

[11] Padmaja K, "A Study on web Applications & Protection against Vulnerabilities", International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com Vol. 2, Issue 6, November- December 2012, pp.001-006.