Vidhyayana - ISSN 2454-8596
An International Multidisciplinary Peer-Reviewed E-Journal
www.vidhyayanaejournal.org
Indexed in: Crossref, ROAD & Google Scholar

32

Web Application Vulnerabilities: A Comprehensive Study of Attack Techniques and Countermeasures

Abhishek Jagtap

Master of Science in Computer Applications, Dr. Vishwanath Karad MIT World Peace University - Pune,

jagtapabhishek227@gmail.com

Anurag Yewale

Master of Science in Computer Applications, Dr. Vishwanath Karad MIT World Peace University - Pune,

yewalea7@gmail.com

Omkar Parve

Master of Science in Computer Applications, Dr. Vishwanath Karad MIT World Peace University - Pune,

omkarparve107@gmail.com

Vikas Magar

Assistant Professor, Dr. Vishwanath Karad MIT World Peace University - Pune,

vikas.magar@mitwpu.edu.in

Abstract-

The significance of secure computer systems is becoming more and more clear as more computer systems are used to automate corporate processes and store confidential material. This significance is made more apparent by the fact that applications and computer systems are scattered and accessed via unsecure connections, including the Internet. The Internet has become a crucial component for governments, corporations, financial institutions, and millions of users in their day-to-day lives. Computer networks enable a variety of operations that, if lost, would seriously impair the operation of these companies. As a result, cybersecurity-related challenges have evolved into national security-related issues. The challenge of safeguarding the Internet is difficult.

This paper presents certain recognized vulnerabilities of information security, classifies them, and evaluates safeguarding measures and methods for opposing the vulnerabilities

Index Terms- Broken Access Control, CSRF, Injection, LFI, OWASP Top-10, SQL Injection, XXE, Session Misconfigurationn

I. INTRODUCTION

TCP/IP (Transmission Control Protocol / Internet Protocol) is a particular collection of communication protocols that permits access to the Internet, a global network of interconnected computer networks, in a number of different methods. Currently, millions of consumers utilize the Internet regardless of time or national or geographic restrictions. [1]. The internet has facilitated the connection of people worldwide, resulting in a growing level of interconnectivity. The advancement and extension of the internet have generated numerous possibilities for individual engagement and commercial pursuits. Communication expenses have plummeted significantly, but more crucially, the range of feasible trade allies has dramatically expanded, producing significant benefits from the exchange. The widespread use of the Internet has made web pages and their users appealing to a variety of cyber offenses, such as data leaks, targeted phishing attempts, ransomware, and fraudulent tech support schemes. The count of web application attacks rose by 12.56% over the same period last year, reaching 62.8875 million per day, according to the State of Web Security Report for H1 2022[2] report. Integrity, confidentiality, and availability of web application are the three components that need to be secured. This paper will investigate the various assaults that undermine these three elements and the methodologies applied to exploit the vulnerability of the web application. Nonetheless, the primary emphasis will be on the most notable vulnerabilities of web applications: XSS, CSRF, SQL Injections, Session Management, Broken Access Control.

The Open Web Application Security Project (OWASP) is a nonprofit organization whose goal is to improve the security of software. Everyone can participate and contribute to online conversations, projects and other OWASP-related events, because they follow a "open community" model. The OWASP makes sure that all of its products, which include forums, events, and online tools and videos, are still free and simple to access through its website. [3]. The Open Web Application Security Project (OWASP) is a nonprofit organization that offers unbiased, cost-effective, and practical knowledge about software for computers and the Internet. A variety of international security experts are brought together by the project to share their understanding of vulnerabilities, threats, attacks, and mitigation strategies. The group is an open community dedicated to assisting businesses in the creation, acquisition, and upkeep of reliable applications. OWASP includes a variety of tools and programs aimed at enhancing application security throughout the sector.

II. Web Vulnerabilities

[1] Broken Access Control:

Broken Access Control (BAC) is a security vulnerability that is ranked as the most critical vulnerability in the Open Web Application Security Project (OWASP). This vulnerability can have serious consequences in web applications, such as privilege escalation, which may result in substantial financial loss and reputational harm to the organization The BAC vulnerability can be exploited in a number of methods including, improper use of code functions, improper configuration of sensitive data, weak user credential validation unsupervised exception handling, uncontrolled website redirection, etc. These vulnerabilities can lead to unauthorized access or upgraded access levels for intruders in a web system BAC Maps 34 CWEs with 318,487 total occurrences and 19,013 CVEs [5].

CWEs (Common Weakness Enumeration):

[A] CSRF (Cross Site Request Forgery):

A Cross-Site Request Forgery (CSRF) attack compels authenticated users to send a request to a Web application for which they are already authorized. When a malicious application or website has a link to a trusted website, this kind of attack is conducted. When the user opens the link, the attacker's website's malicious code leverages the user's browser and session cookies to authenticate the request and transmit it to the genuine site. If the user is logged in to the legitimate site, the request is granted, allowing the attacker to perform any action that the user is authorized to do on the site, such as making a purchase, changing a password, or deleting an account." Cross-Site Reference Forgery", "Confused Deputy", "XSRF", "session Riding", are other names for cross-site request forgery attacks. [4] The Online Security Threat Classification does not list CSRF attacks, and academic and technical writing rarely mentions them [6]. The same study by Imperva found that the average number of CSRF attacks per web application was 17 per month.

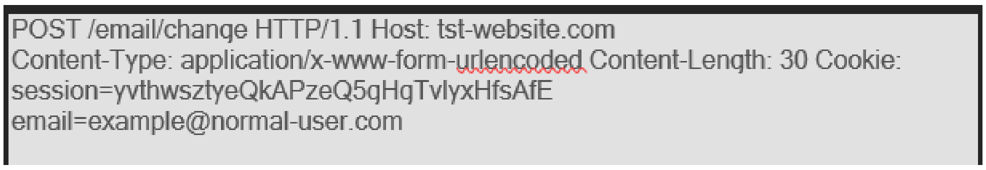
a) Overview of CSRF Attack:

Cross site request forgery [7,8] (abbreviated HTTP is a stateless protocol that is not able to recognize XSRF or CSRF, sometimes also called "Session Riding"), when a number of requests all belong to a particular user denotes a relatively new class of attack against web application users. Once a user has successfully authenticated, a CSRF attack takes advantage of HTTP protocol capabilities to send a session cookie along with each request to the server. This provides the server the ability to verify that the request comes from an authorized person. In order to execute such an attack, the attacker first looks at the pattern of the request, that contains the request type (such as GET or POST), parameter names, and parameter values. After thoroughly studying the URL pattern of the request, the attacker embeds the URL in HTML tags of web pages or emails. The attacker inserts the URL in emails or web pages' HTML tags after carefully examining the request's URL pattern. The attacker then induces the user who has provided authentication to carry out this request. The session cookie data is automatically included with the request by the browser because the user has already been authenticated, which the server accepts and executes.

Vidhyayana - ISSN 2454-8596
An International Multidisciplinary Peer-Reviewed E-Journal
www.vidhyayanaejournal.org
Indexed in: Crossref, ROAD & Google Scholar

b) Attack Scenario:

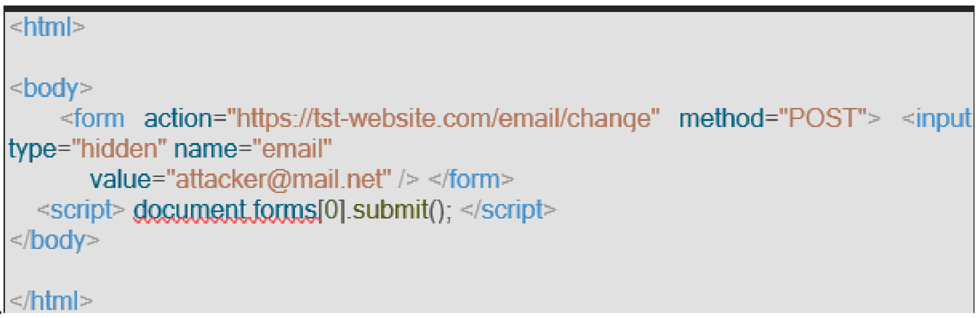
Let's assume a program has a functionality that enables users to update the email address of heir account. When a user does this action, they send an HTTP request looking something like this:



```
POST /email/change HTTP/1.1 Host: tst-website.com
Content-Type: application/x-www-form-urlencoded Content-Length: 30 Cookie:
session=yvthwsztyeQkAPzeQ5qHqTvlyxHfsAfE
email=example@normal-user.com
```

This satisfies the prerequisites for CSRF.

An attacker plans to alter the email address associated with a user's account. The hacker can often manipulate the system to reset the user's password and gain complete access to their account. The software relies on a session cookie to recognize the user who initiated the request. There are no additional tokens or program for keeping track of user sessions. The attacker is able to easily identify the necessary values for the request parameters in order to carry out the desired action. Once these requirements are fulfilled, the attacker can develop a website containing the provided HTML code



```
<html>

<body>
      <form  action="https://tst-website.com/email/change"  method="POST">  <input
type="hidden" name="email"
           value="attacker@mail.net" /> </form>
    <script> document.forms[0].submit(); </script>
</body>

</html>
```

If a victim user accesses the attacker's website, the following will take place:

An attacker's webpage will send a request to a website that has a vulnerability. If the user is logged in to the website, their browser will automatically attach their session cookie to the request (unless SameSite cookies are being used). The website will process the request as it normally would, treating the user who sent the request as the victim and allowing the attacker to change the victim's email address.

c) Mitigation:

1. Use only JSON APIs: JavaScript is utilized in AJAX calls, which are limited by CORS. When only accepting JSON data, it's impossible for a basic HTML form to submit data in that format. Therefore, using JSON exclusively prevents the aforementioned HTML form from being used.
2. Disable CORS: Disabling cross-origin requests is the first step in mitigating CSRF attacks.

   If you're going to enable CORS, limit it to the functions OPTIONS, HEAD, and GET because those aren't designed to have any unintended consequences.

   However, because it does not employ JavaScript, this does not prevent the aforementioned request (so CORS is not applicable).
3. GET should not have side effects: Verify that none of your GET requests alter any essential database data. This is a very beginner error to make, and it leaves your app open to more types of attacks than only CSRF ones.
4. Use CSRF Tokens: To secure a website from CSRF attacks, a server provides a token to the client, who then includes it in the form submission. If the token is invalid, the server rejects the request. If the website doesn't support Cross-Origin Resource Sharing (CORS), an attacker cannot obtain the CSRF token and is therefore unable to exploit the vulnerability. Obtaining the token requires using JavaScript, so the attacker must find a way to acquire it from the site.

[B] LFI (Local File Inclusion):

LFI is a kind of cyber security weakness in web applications that allows a user to integrate various files located on the server machine hosting the web application. To exploit this vulnerability, the attacker targets the Dynamic File Inclusion mechanism implemented in the affected web application. The vulnerability results from the incorrect usage of programming functions/methods and the failure to validate user-selected parameters.[9] LFI attacks reveal the config file of the system, password, or database connection file too. LFI causes Denial of Service, code execution (server-side/client-side) and the leaking out of sensitive information or source code (DoS).

Overview of LFI attack:

We typically utilize LFI to access the below-mentioned files.

/etc/group

/etc/security/passwd

/etc/security/environ

/etc/security/limits

/etc/passwd

/etc/shadow

[1] Basic Local File Inclusion

Attackers can get sensitive information from a server by using LFI to their advantage.

An illustration of how LFI might be leveraged by attackers to extract sensitive information is if the application's code reveals the name of a file in the URL.

```
https://test.com/?module=contacts_test.php
```

An attacker can alter the URL to seem as follows:

```
https://test.com/?module=/etc/passwd
```

[2] Null Byte Injection

In the absence of suitable filtering, the server will show the sensitive information of the /etc/passwd file.

A control character from the reserved character sets that is used to signify the end of a string is referred to as a "null character," "null terminator," or "null byte." The null byte's purpose is to render any subsequent character irrelevant. Usually, a null byte is inserted at the end of a URL in the form of %00. The following is an illustration:

```
https://test.com/preview.php?file=../../../../../passwd%00
```

Mitigation:

1. Utilize secured, confirmed whitelist files only, and disregard everything else.
2. Instead of including files on a web server that could potentially be hacked or abused, databases are advised. Using a database offers better security measures and reduces the risk of compromise.
3. To enhance security, it is recommended that you store the path to your file in a secure database and allocate a unique Identity to each path. This approach ensures that users can only access and modify the files associated with their ID, without directly viewing or changing the file path.

[II] Injection

Web applications are vulnerable to injections, which have been a persistent and harmful threat for a long time. These attacks can cause serious damage, such as stealing data, loss of data, loss of credibility, denial of service, and complete network compromise. Typically, injections occur when user data lacks proper validation. Attackers exploit various attack vectors in injection attacks by providing untrusted program data.This information is then interpreted by a program as a request or instruction, which alters the way the program runs. Injection attacks can take many different forms and pose a variety of dangers, such as using shell commands to call operating system and external programmes. SQL injection attacks, which involve calling backend databases using SQL, are also a common type of injection attack. Scripts written in Perl, Python, or any other languages may be able to be embedded and run as a result of poorly designed programmes. Any program that employs an interpreter can potentially be vulnerable to input attacks. Injection Maps 33 CWEs with 274,228 total occurrences and 32,078 CVEs [10].

Types of Injection Attacks:

[1] SQL Injection:

Web applications are highly threatened by SQL injection vulnerabilities, which are considered one of the most severe risks. [11]. In 2022, SQL Injection was identified as the primary cause of critical vulnerabilities in web applications worldwide [12]. Web applications that exhibit susceptibility to SQL injection could potentially provide an opportunity for a perpetrator to gain absolute control over the databases supporting them. Considering that these databases typically harbor sensitive user or customer data, any ensuing security breaches could involve theft of identity, disclosure of confidential information, and fraudulent activities. On certain occasions, assailants may also leverage an SQL injection vulnerability to commandeer and damage the server system that operates the web application.

Types of SQL Injection:

[A] In Band SQL Injection:

If the attacker is able to both carry out the attack and receive its outcomes through the same communication channel, it is known as an in-band attack.

[a] Error Based:

Error-based injections provide valuable information about a database, which can be useful to network administrators and developers. However, on the application side, it is essential to limit the impact of these errors to ensure that they do not pose a security threat.

Example: If the server responds to this specific URL with a SQL error, it means the server made an insecure connection to the database.

Following the discovery of this flaw, an attacker can use specific SQL commands to manipulate or harm the database.

```
https://www.test.com/category.php?id=-11'
```

[b] Union based:

The UNION operator is used to merge the results of multiple SELECT statements into a single output when those statements generate different outputs. This technique is known as injection and it is used to extract more data from the database.

Example: The example that follows shows how an attacker can use this injection attack to find out how many columns there are.

```
https://test.com/category.php?id=5 ‘UNION+SELECT+NULL,NULL,NULL—
```

[B] Out of Band SQL Injection:

Out-of-band SQL injection is the term for when an attacker is unable to conduct an attack and obtain the results using the same channel. In such a scenario, the database server has the capability to transmit information to the attacker who can use DNS or HTTP requests.

[C] Inferential SQL Injection:

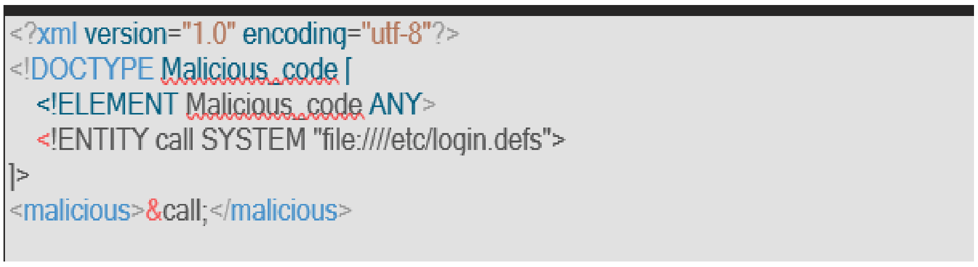
Inferential SQL Injection is a type of SQLi that is slower than in-band SQLi but still just as dangerous. In this type of attack, the attacker cannot see the results of the attack in real-time because no data is transferred through the web application. This is why it is often called "blind SQL Injection". To execute this type of attack, the attacker sends payloads to the web application, analyzes the response, and monitors the behavior of the database server to gain knowledge about the database structure.

[2] XXE (XML Injection):

XXE, a vulnerability in web security, enables a harmful user to disrupt an application's handling of XML data. This flaw often allows the attacker to access files located on the application server's filesystem and interact with any back-end or external systems that the application can reach. The attacker can modify the XML's syntax, content, or commands before it is processed by the end system, due to the software's failure to properly neutralize certain elements used in XML [13]. Organizations may suffer losses as a result of XML external entity injection (XXE), which can also expose sensitive information, engage in server-side request forgery (SSRF) attacks, and scan ports from the perspective of the parser.

Example of XXE Attack:

On a vulnerable Linux system, the code below will display the contents of the login.defs file, which describes login settings:



```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Malicious_code [
   <!ELEMENT Malicious_code ANY>
   <!ENTITY call SYSTEM "file:///etc/login.defs">
]>
<malicious>&call;</malicious>
```

Mitigation

1]Before trusting XML data, sanitize it.

2] If feasible, completely disable DTD and external entity processing.

3] Regularly update XML libraries.

4]Restrict connections that egress from the applications (only trusty connections)

[III] Security Misconfiguration

As per OWASP [3], security misconfiguration has been identified as one of the top ten risk, ranking fifth in 2021, that could result in various risks. These dangers could include everything from an entire system breach to unauthorized access to system data or functionality. The situation is made worse by the fact that a single, potentially insecure environment is commonly utilized to host multiple web applications in a shared technology context, such as a shared web server. Furthermore, an insecure configuration instance may be duplicated, posing additional risks. Security misconfiguration leads to information disclosure and can compromise the application “Availability”, “Integrity”, “Confidentiality”. This common vulnerability is caused by human error or neglect, such as leaving default configurations/credentials in place, failing to secure storage, improper error handling, and leveraging known vulnerable software versions and protocols. Experienced cybercriminals tend to view misconfigurations as a simple and vulnerable target. They can easily identify misconfigured web servers, cloud instances, and applications, which then become exploitable. As a result, this can cause significant damage and result in severe data leakage problems for companies. Security Misconfiguration Maps 20 CWEs with 208,387 total occurrences and 789 CVEs [15].

There are various forms of security misconfiguration, and they can occur at different levels, including the network, operating system, web application, and database.

[A] Default Configurations & Credentials

Default passwords and settings are preconfigured usernames and passwords that are frequently used as default login credentials for a specific device or application. These defaults are typically provided by the manufacturer or developer and are intended to serve as a jumping-off point for users to create their own custom usernames and passwords. Many users, however, may fail to change these defaults, making the device or application vulnerable to attack. Because default settings and passwords are frequently well-known and widely disseminated online, they pose a security risk and make it simple for hackers to access systems without authorization. Attackers may employ automated tools like “John the Ripper” [16] to scan the Internet for devices or applications that are still using default credentials and exploit them.

a) Impact:

The consequences of default passwords and settings can be severe. Attackers can gain access to sensitive information, delete or modify files, install malware or other malicious software, and use the compromised system as a launching point for further attacks. Default credentials can also be used to pivot to other systems within the same network, leading to a larger-scale breach.

b) Mitigation:

To mitigate this vulnerability, it is critical to change default passwords and settings as soon as possible after initial setup to reduce the risks associated with them. This entails creating a strong, one-of-a-kind password and username that are neither easily guessable nor publicly available. Maintaining software and firmware with the most recent security patches is essential because many updates fix known issues with default settings.

[B] Weak Encryption

Information is encoded using encryption so that it cannot be read without the proper decryption key. Weak encryption, which permits unauthorized access to or manipulation of encrypted data, refers to the use of encryption algorithms or protocols that are simple to compromise or break.

A number of factors, such as the use of dated encryption protocols or keys that are too short or easily guessed, can result in weak encryption. The use of deprecated SSL and early versions of TLS (Transport Layer Security) protocols, for example, can be considered weak encryption because they are vulnerable to various attacks such as POODLE [17], BEAST [18], and CRIME [19]. Using short or predictable encryption keys can also result in weak encryption. For example, modern computers can easily break the RSA-512 encryption key, which is considered too short, using brute-force attacks. Similarly, using common passwords as encryption keys, or employing weak algorithms such as MD5 or SHA1, can result in insecure encryption.

a) **Impact:**

Weak encryption can have serious repercussions because it makes it possible for hackers to read, modify, or steal sensitive data, including passwords, credit card numbers, and other private information. Because attackers can use ineffective encryption to obtain unauthorized access and control, it can also result in the compromise of entire systems or networks.

b) **Mitigation:**

Organizations should use modern, strong encryption algorithms and protocols, such as AES (Advanced Encryption Standard) [20], RSA-2048 or higher, and TLS 1.3 or higher, to avoid weak encryption. They should also make certain that encryption keys are generated at random, are long enough, and are kept secure. Furthermore, organisations should regularly monitor their systems and applications for any vulnerabilities or weaknesses that attackers could exploit.

[C] Outdated Software:

Outdated software is software that no longer receives updates or patches from the developer or vendor. Software that is out-of-date may have bugs or known security flaws that attackers can use to break into a system, steal data, or interfere with services. The causes of software becoming obsolete can vary. For example, the software might no longer be supported by its creator or vendor, or it might be incompatible with newer hardware or operating systems. In some cases, software simply reaches the end of its lifecycle and is no longer actively developed or maintained.

a) Impact:

Outdated software can pose a serious security risk to businesses. Attackers can gain access to sensitive data, install malware or ransomware, and take control of systems by exploiting vulnerabilities in outdated software. Furthermore, outdated software can cause system crashes or downtime, resulting in lost productivity and revenue.

b) Mitigation:

Organizations should update their software and firmware on a regular basis to reduce the risks associated with outdated software. This includes installing patches and security updates as soon as they are released, as well as replacing software that is no longer supported or maintained. To make sure that updates are applied consistently and quickly across all systems, organisations should think about implementing an automated patch management system.

[D] Misconfigured HTTP headers:

HTTP headers are pieces of data sent between a web server and a web client (such as a browser) when they communicate using the HTTP protocol. HTTP headers can include a range of information, such as the type of material being delivered, the language of the content, as well as cache and authentication instructions. Misconfigured HTTP headers refer to a situation in which the headers are not correctly configured, exposing the system to security risks. Misconfigured HTTP headers include the following:

a) Lack of Content Security Policy (CSP): A Content Security Policy (CSP) is a header that tells the browser which content sources can be loaded. Failure to implement CSP can result in the execution of malicious scripts or the loading of untrusted content [21].
b) Missing or misconfigured HTTP Strict Transport Security (HSTS): HSTS is a browser header that causes the browser to use HTTPS rather than HTTP, which can aid in the prevention of man-in-the-middle attacks. Failing to implement or incorrectly configure HSTS can leave the system vulnerable to assaults [22].
c) Improperly configured cross-origin resource sharing (CORS): CORS is a header that allows web pages to request resources from domains other than their own. If CORS is not properly configured, it can expose the system to cross-site scripting (XSS) attacks or data theft [23].
d) Insecure Cookies: Cookies are pieces of information that are sent between the client and server to keep track of user sessions. If cookies are not configured securely, they can be intercepted by attackers and used to hijack user sessions.

Mitigation:

To reduce the risks associated with misconfigured HTTP headers, organizations should implement secure coding practices, update software and firmware on a regular basis, and conduct security audits and testing. Furthermore, organizations should review and properly configure HTTP headers to ensure that they are correctly set up and do not leave any vulnerabilities open to exploitation.

III. CONCLUSION

Web application security is an essential component of modern computing. The increased reliance on web-based applications for communication, financial transactions, and data storage has increased the potential for malicious attacks. The study looked into various aspects of web application security, such as common vulnerabilities, attacks, and countermeasures. According to the findings, web application security should be a top priority for both organizations and developers. A security breach can have serious consequences, including data loss, financial losses, legal liabilities, damage to reputation. As the use of web-based applications grows, it is critical for developers and organizations to stay current on emerging threats and best practices in security. By putting the right security measures in place, businesses can protect sensitive data, ensure business continuity, adhere to regulations, and shield themselves from cyberattacks.

Web application attacks pose a significant risk to organizations, people, and society at large. The growing reliance on web-based applications, combined with the evolving nature of attacks, has made it difficult to adequately secure web applications. The impact of various web application attacks, including SQL injection, cross-site scripting, and local file inclusion, on web application security was examined in this research paper. We've also covered a few measures that can be taken to lower the risk of web application attacks, including using web application firewalls, secure coding practices, regular vulnerability assessments, and penetration testing. Web application attacks are clearly a constantly evolving threat, and developers, security professionals, and end users must remain vigilant in order to stay ahead of attackers. Businesses must prioritize web application security and invest in the resources and expertise required to build and maintain secure web applications. Failure to do so can lead to data breaches, reputational harm, and financial loss. With the continued growth of web-based applications, it is critical to develop a thorough understanding of web application security and to put in place effective security measures to prevent web application attacks.

REFERENCES

[1] Isern, G. Internet Security Attacks at the Basic LevelsACM SIGOPS Operating Systems Review, 32(2):4 15,2002.

[2] https://www.radware.com/getattachment/ba8a3263-703b-4cc7-a5d0-741dc00e9273/H1-2022-Threat-Analysis-Report_2022_Report-V2.pdf.aspx

[3] https://owasp.org/www-project-top-ten/

[4] Xiaoli Lin, Pavol Zavarsky, Ron Ruhl, Dale Lindskog, “Threat Modeling for CSRF Attacks”, International Conference on Computational Science and Engineering, 2009

[5] https://owasp.org/Top10/A01_2021-Broken_Access_Control/

[6] http://www.webappsec.org/projects/threat/

[7] T. Schreiber. Session Riding: A Widespread Vulnerability in that our solution will prove useful in protecting vulnerable Today's Web Applications. http: //www. securenet.web applications. de/papers/Session_Riding.pdf,2001.

[8] P. W. Cross-Site Request Forgeries. http: //www.securityfocus. com/archive/l/1913 90, 2001.

[9] Begum, Afsana & Hassan, Md Maruf & Bhuiyan, Touhid & Sharif, Md Hasan. (2016). RFI and SQLi based local file inclusion vulnerabilities in web applications of Bangladesh. 21-25. 10.1109/IWCI.2016.7860332.

[10] https://owasp.org/Top10/A03_2021-Injection/

[11] Aucsmith. Creating and Maintaining Software that Resists Malicious Attack.

[12] https://www.statista.com/statistics/806081/worldwide-application-vulnerability-taxonomy/

[13] https://cwe.mitre.org/data/definitions/91.html

[14] https://www.hackerone.com/knowledge-center/xxe-complete-guide-impact-examples-and-prevention

[15] https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

[16] https://www.kali.org/tools/john/

[17] https://www.cisa.gov/news-events/alerts/2014/10/17/ssl-30-protocol-vulnerability-and-poodle-attack

[18] https://www.wallarm.com/what/what-is-a-beast-attack

[19] https://www.acunetix.com/vulnerabilities/web/crime-ssl-tls-attack/

[20] Akkar, ML., Giraud, C. (2001). An Implementation of DES and AES, Secure against Some Attacks. In: Koç, Ç.K., Naccache, D., Paar, C. (eds) Cryptographic Hardware and Embedded Systems — CHES 2001. CHES 2001. Lecture Notes in Computer Science, vol 2162. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-44709-1_26

[21] https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

[22] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

[23] https://developer.mozilla.org/en-US/docs/Glossary/CORS