International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 01 | January - 2023 Impact Factor: 7.185 ISSN: 2582-3930

Systematic Review of Common Web-Application Vulnerabilities

Arhant Bararia[1], Ms. Vandana Choudhary [2]

[1] Information Technology scholar at Maharaja Agrasen Institute of Technology, Delhi-India
[2] Assistant Professor, Department of IT, Maharaja Agrasen Institute of Technology, Delhi-India

---***---

Abstract - Organizations are increasingly worried about web application vulnerabilities because they can be used by cybercriminals to access private data without authorization. In this review, we examine the current state of the vulnerabilities in the web application layer. We begin by discussing the different types of vulnerabilities that can affect web applications, including cross-site scripting (XSS), brute force, SQL injection, and cross-site request forgery (CSRF) attacks. And then for each vulnerability, we discuss the various approaches that have been developed to detect and prevent these vulnerabilities. Finally, we discuss the challenges and limitations of current approaches and suggest directions for future research.

Key Words: web application, vulnerabilities, cyber-security, SQL injection, XSS, CSRF. .

1. INTRODUCTION

[1]. Application software that is hosted on a server and made accessible through the internet is a web application. Web apps are extremely risky from a security standpoint since they fully depend on the open internet. The fact that web applications are comprised of several layers, including the client layer, the application layer, and the data layer, makes it challenging to ensure their security. Since security must be guaranteed at all tiers of the web application, the security mechanism must be created accordingly.
Web application vulnerabilities are weak points or faults that an attacker could use to obtain access without authorization, steal sensitive information, or carry out other malicious activities. Numerous factors, such as insufficient input validation, unsafe coding techniques, and inappropriate configuration of the web application or its supporting infrastructure, might lead to these vulnerabilities. [4] There are many different types of web application vulnerabilities, some of the most common ones and the ones we are going to discuss include:

- Password Guessing Attack: A crucial step in gaining access to a web application is authentication. By guessing the password, the attacker can gain access to the system or application in a password guessing attack. Because of the availability of numerous automated programs like Cain and Abel, John the ripper, Hashcat, Hydra, etc., passwords are now fairly simple to crack. Two methods of password guessing attacks exist:
    - Brute Force Attack
    - Dictionary Attack
- SQL injection: This vulnerability occurs when an attacker can execute malicious SQL statements by injecting them into a web application's input fields. This can allow the attacker to access or modify sensitive data stored in the database.
- Cross-Site Scripting (XSS): This flaw enables an attacker to insert malicious code (like JavaScript) into a web page, which is then executed by unwary site visitors. Sensitive information can be stolen with this, such as login credentials, or to perform other malicious actions.
- Cross-Site Request Forgery (CSRF): In order to send a request to a server of a web application without the user's knowledge or consent, an attacker must deceive the user. This attack can be used to carry out tasks on the user's behalf, like making financial transfers or changing private data.

Organizations should regularly check their web apps for vulnerabilities because there are numerous varieties of web application vulnerabilities and implement appropriate controls to prevent attacks. We restrict our discussion of the aforementioned vulnerabilities to those in this review study.

2. PASSWORD GUESSING ATTACK

"Password cracking" is the process of attempting to decipher or obtain a password that has been saved or communicated by a computer system. One common method of doing this is to try different combinations of characters until the correct password is found. [2]People often choose weak passwords, such as simple words found in dictionaries, personal information such as family names, or predictable patterns such as alternating vowels and consonants (usually examined fewer than six or seven characters) or recognizable patterns (such as leetspeak's alternating vowels and consonants, which turns "password" into "p@55w0rd").).
One way to increase the chances of successfully cracking a password is to create a list of words that are specifically targeted towards the person or system being attacked, using information such as company websites or personal social media profiles. As a final option, it is feasible to use a technique known as a brute force attack to test every possible password combination. Even if this approach has a chance of working, the time it takes to use it grows with the length of the password and the number of password combinations does as well.
Brute force attacks can be particularly effective against web applications that have weak password policies, as they allow an attacker to try a large number of different combinations without being detected. To prevent brute force attacks, it is important for web application administrators to enforce strong password policies and to use measures such as rate limiting and CAPTCHAs to prevent automated login attempts. Additionally, it is a good idea to use two-factor authentication,

which requires the user to provide an additional form of verification in addition to their password.

2.1 PREVENTING BRUTE-FORCE ATTACKS

[3] Here are some recommendations for avoiding brute-force attacks:

1. **Use a strong password.**
   The simplest and most efficient strategy to prevent brute force attacks is to have a strong password policy. For any online application or public server, you should make complicated passwords that cannot be guessed yet are also manageable to remember. 30% of recycled or changed passwords can be cracked in 10 attempts.
2. **Limit login attempts.**
   Most websites allow an unlimited number of logins, but you can use plugins to limit login attempts or block brute-force attacks. Someone's IP address can be blocked from accessing that website for a while if they attempt to log in more frequently than the number of permitted logins.
3. **Use two-factor authentication (2FA):**
   Two-factor authentication (2FA) is a security measure that requires users to verify their identity before being granted access to their accounts. For example, you might be required to enter your login credentials and a code sent to your mobile phone in order to access your account.
4. **Use CAPTCHAs.**
   The acronym CAPTCHA stands for Completely Automated Public Turing Test to Tell Computers and Humans Apart. In general, CAPTCHAs are challenging for automated computer programs to solve but simple for humans.
5. **Use a unique login URL**
   Another challenging and time-consuming step for an attacker is creating distinctive login URLs for various user groups. You are not required to end brute force attacks. It might, however, discourage unhindered attacks.
6. **Web Application Firewall (WAF)**
   The Web Application Firewall (WAF) offers sufficient defense against brute force attacks that try to obtain access to your system without authorization. In general, it limits the number of requests made while in transit from a source to a set of URLs.

3. SQL INJECTION

By inserting malicious code into a SQL statement, an attacker can use the SQL injection attack (SQLIA) type to modify a website's database. This can be done through a variety of means, such as through a website's form field or a URL parameter.
Website's database is typically accessed through a SQL statement that is constructed by the website's code. This statement might contain user input, such as a search query or a login username and password. If an attacker is able to inject their code into this statement, they can potentially access sensitive data, modify it, or even delete it.

One common way that attackers can inject their code is by including it in a form field or URL parameter that is not properly sanitized. For example, if a website has a search form that allows users to search for products, an attacker could enter malicious code in the search field that is then passed to the database as part of the SQL statement.
To protect against SQL injection attacks, it is important to sanitize all user input and use prepared statements when constructing SQL statements. It is also a good idea to use an input validation library or framework to help ensure that user input is safe.

3.1 TECHNIQUES OF SQL INJECTION ATTACKS

This section reviews the [5] techniques of SQL injection:

1. **Tautologies**: SQL database derivation is the main objective of this attack. By requiring that the condition statement return all true values, it inserts a query into the database. This provides an attacker with all of the table's information. Finding the username and password of the users recorded in the database is simple after access to the table has been achieved.

   Example:

   ```
   SELECT * FROM user WHERE username =
   "admin" AND password = ' ' OR 1=1
   #"
   ```

2. **Logical Incorrect Queries**: Attackers can leverage the error messages returned by SQL databases to learn more about the error's root cause. The error messages often show information about the table, column, and circumstances that led to the error.

   Example: [6]

   ```
   SELECT *  FROM creditCardUsers
   WHERE login='administrator' AND
   password = convert (int,(select top
   1 name from sysobjects where
   xtype='u'))
   ```

   In this attack error generated is :

   ```
   " SQL Server's Microsoft OLE DB
   Provider (0x80040E07) There was a
   problem converting the nvarchar
   value "CreditCardsUsers" to an int
   column."
   ```

   There are 2 things the error message reveals:
   i. The server is running a SQL database.
   ii. Password is an integer.

3. **Union Query**: Attackers who utilize code injection and manipulation to retrieve data from the table columns can compromise database tables. The urge to take advantage of security flaws is frequently the driving force behind this kind of attack.

   Example: [6] Someone could attempt to inject the text:

```
'UNION SELECT card_number from
DEBIT CARDS where acNo=889005 –
```

Inserting above in the user field produces following query:

```
SELECT * FROM users WHERE user=''
UNION SELECT cardNumber from
CreditCards where acNo=889005 --
AND pass=''
```

For the account "88905", the database returns the value "cardNumber" in that column. The program receives the combined result of these two queries from the database. The value for "cardNumber" is frequently shown together with the account details in this procedure.

4. **Stored Procedure**: Stored procedures are a way of carrying out SQL commands directly from your application. They're usually done by injecting SQL function calls into the database. This attack's objectives include performing denial of service, privilege escalation, and remote command execution. [7,8] Many developers mistakenly believe that by employing stored procedures while creating Web applications, they are protected from SQL injection attacks. In actuality, stored procedures might be subject to attacks in the same way as regular programs.

   Example: [6]
   To determine whether the user's credentials were successfully authenticated, stored procedure `IS_AUTHENICATED` produces a boolean value.
   The only requirement for a SQL injection attack is the injection of the vector `SHUTDOWN;—` into the username or password fields.
   Following the usual execution of the first query, a malicious second query is run that shuts down the database.

5. **Piggy-Backed Queries**: In this case, the attacker manipulates the data utilizing the DELETE, INSERT, and UPDATE clauses after maliciously injecting ordinary queries to exploit them.

   Example:

```
 SELECT * FROM employees WHERE
username='raman' AND pass = '';
DROP TABLE users — `
```

   The database would recognize the query delimiter (;) following the first query and would then carry out the DROP TABLE query.

6. **Alternate Encodings:** The attackers use special characters like ASCII, Unicode, and hexadecimals to bypass filters that developers have put in place to protect their systems.
   This attack aims to bypass filters and defenses put in place by the web application's creator.

   Example: [7]

   This is similarly an example of a piggy-backed query, however, to get around a filter, DROP TABLE is encoded in hexadecimal.

```
SELECT * FROM employees WHERE
user='adminUser' ;
exec(char(0x44524f50205441424c45))
– AND pass=''
```

3.2 PREVENTION OF SQL INJECTION ATTACKS

To stop SQL injection attacks, a number of methods are available. Some of these techniques are just good programming practices, while others are automated frameworks. The benefits and limitations of each strategy are discussed in this section.

1. Defensive Coding Practices: Incorrect input validation leads to SQL injection vulnerabilities, which are best avoided by using good coding techniques.
   a) Input examination: SQL injection attacks can be prevented by checking the input parameters for validity. For numeric inputs, this can be done by rejecting any input that does not consist of only digits.
   b) Encoding of inputs: Using functions to decrypt strings in a way that the database interprets all meta-characters as regular characters after being specially encoded.
   c) Identification of all input sources: All sources of information used by developers when building an application must be examined. This means that developers must check all the information they receive, both from inside and outside the company.

2. Black Box Testing: [9] WAVES is a black-box technique that Huang and colleagues suggest using to check web applications for SQL injection vulnerabilities. The method employs a web crawler to find all potential SQLIA injection locations in a web application. By employing machine learning to direct the testing, this method enhances penetration testing methods. Like any penetration testing method, it cannot ensure that the testing will be finished.

3. Combined Static and Dynamic Analysis: [10] A method for verifying the type validity of SQL queries that are created dynamically is JDBC-Checker. This method can be used to stop attacks that rely on type mismatches in a dynamically created query string, but it is not intended to detect and prevent general SQLIAs. This method's main flaw is that it can only be used for tautology detection and prevention. [11] AMNESIA is a model-based approach that combines

runtime monitoring with static analysis. Before any queries are submitted to the database, AMNESIA intercepts them all and compares them to the statically created models. The database identifies queries that break the model as SQLIAs and forbids them from running. This method's main drawback is that the effectiveness of its static analysis for creating query models depends on how accurate it is. This stage could become less accurate and provide false positives or false negatives depending on the sort of code obfuscation or query development technique used.

4. Intrusion Detection System: [12] An intrusion detection system (IDS) should be used to prevent SQL injection attacks, according to esteemed colleagues at Valeur. The machine learning method used to train this IDS system uses a set of common application queries. The method creates models of the normal queries and then keeps track of the program while it is running to find queries that deviate from the model.

4. CROSS-SITE SCRIPTING (XSS)

Cross-Site Scripting (XSS) is a form of code injection that attackers can use to exploit web browsers and gain access to sensitive data. XSS attacks are typically launched on the client side, but they can also be exploited on the web server side through attacks that inject malicious JavaScript payloads. XSS attacks on web applications can be exploited by crafting and injecting a malicious JavaScript payload that seems benign and then executing it within the trust zone of the web application.

4.1 XSS PAYLOAD INJECTION PROCESS

[13] Usually, malicious JavaScript code is injected into websites that the victim visits and gets infected. This is only conceivable if the online application allows user input since an attacker can include a malicious JavaScript string that will be rendered as code on the victim's browser. Since the victim assumes that the post will include only text, the attacker can include a malicious script:

<script>alert(''XSS Attack'')<\script>

As a result, when a victim accesses the website, their browser will now display the warning "XSS Attack" when they click on this most recent post.

4.2 TECHNIQUES OF XSS ATTACKS.

An Internet user's web browser can run JavaScript code without necessarily performing bad deeds. In actuality, it runs in a very constrained environment with very limited access to the user's login information and a few files on the user's operating system.
[14] However, when you take into account the subsequent information, the potential for JavaScript to be malevolent becomes more apparent.

- Some sensitive user data, including cookies, can be accessed by JavaScript. JavaScript is now potentially a dangerous source as a result of this.
- JavaScript may use XMLHttpRequest and other techniques to send HTTP requests with any content to any destination.
- JavaScript is capable of modifying the contents of tags, attributes, and elements on the current page in addition to other HTML manipulations.

[14] A website's cookies can be accessed by an attacker using cookie theft, and those cookies can be exploited to retrieve sensitive data like session IDs.
[14] Keylogging poses a significant security risk. A keystroke event listener can be registered by an attacker, who can then broadcast every keystroke the user makes to his own server, potentially collecting sensitive data.
Attackers might stealthily obtain your personal information by using phishing. They can modify the DOM of the website by introducing a fake login form and setting the form's action property to point at their own server. The attacker can exploit the information the user submits to gain control of their account. [14]
Jakob Kallin [14] states that there are 3 different forms of XSS attacks:

- Persistent XSS attack / Stored XSS
- Non Persistent XSS / Reflected XSS
- DOM Based XSS

1. **Persistent XSS:** An attacker might inject malicious code into a vulnerable website or web application, which would then be stored in server and served to additional users who visit the website, leading to a persistent XSS (stored XSS) attack. Persistent XSS is a type of vulnerability that can be particularly harmful because it targets websites that allow users to share content. This could include forums, blogs, email servers, or other websites where users can share information.

.

[15] If persistent XSS assaults are not adequately addressed, they can have a devastating and long-lasting impact on your applications. Similar to a pandemic, once it's established on a victim's website, anyone who visits it may become ill. The visitor doesn't have to click on a malicious link in order for the payload to be executed (as is the case with Non-Persistent XSS). All they need to do is go to an infected website.

2. **Non-Persistent XSS**: The victim of a reflected XSS attack is duped into sending a malicious string along with their request to a website. The response that is sent back to the recipient then contains this malicious string.
   Reflected XSS is a type of attack in which the attacker tricks the victim into sending a request that includes a malicious string. This type of attack is difficult to perform, as the victim would need to willingly send a request that includes the malicious string.
   In fact, there are at least two typical methods for getting a victim to conduct an XSS assault back at himself:

• If the user wants to fool a specific individual into visiting a harmful URL, they can send them one.
• The attacker can post a malicious URL on his website or a social media platform, wait for people to click it, and then use it to target a huge number of people.

Both of these techniques can be more effective when a URL shortening service is used to hide the malicious string from users who might otherwise be able to recognize it.

3. **DOM-based XSS:** An XSS attack known as DOM-based XSS occurs when the legal JavaScript on the page is first performed before the malicious string is actually parsed by the victim's browser.

The way DOM-based XSS operates is as follows:

- The attacker makes a URL that contains a malicious string and transmits it to the victim.
- The victim is misled into requesting a website for a URL.
- The response from the website did not contain the harmful string.
- The browser runs the script in the response, which results in the malicious script being put into the page.
- The attacker added the malicious script to the website, which transmitted the victim's cookies to the attacker's server.

4.3 PREVENTING XSS ATTACK

Input sanitization is the process of removing anything that could potentially harm the application. This can be done by removing white-space characters, removing characters that are not allowed in the application, or removing characters that are harmful if entered incorrectly. The process of making sure the input is correct and complies with the application's criteria is known as input validation. This can be done by checking for specific values, ensuring that the characters are in the correct order, or checking for errors.

There are two essentially distinct approaches to safe input handling that a web developer can take:

- User input is converted through the process of encoding so that the browser can read it without misinterpreting it as code.
- The process of validation makes sure that user input is secure and free of dangerous directives.

Both methods of preventing XSS involve checking to make sure that the data you are submitting is not malicious.

5. CROSS-SITE REQUEST FORGERY

A cyber-attack known as a cross-site request forgery (CSRF) tricked victims into unintentionally acting on the attacker's behalf. CSRF attacks take advantage of a weakness in web applications' security that makes it impossible to distinguish between a user's genuine request and one made during an authenticated session.

Social engineering tactics are frequently used in CSRF attacks to persuade the victim user to visit a page or click a link that contains a malicious request. The link causes the victim's browser to send a fraudulent request to the intended website.
The majority of websites automatically include session information in browser requests, such as a valid token or login information that the website connects to the user. The website considers the new malicious request as a genuine request from the user and executes it if the authenticated user is already engaged in an active session with the target website.

5.1 DIFFERENCE BETWEEN CSRF AND XSS

[16] Attacks like CSRF and XSS happen when a user uses their browser to access a web application and the web application trusts the user. Attackers may take advantage of this trust to carry out illegal operations on the user's behalf.

Attackers can run malicious scripts, access responses, and send sensitive follow-up data to destinations of their choice because of the two-way nature of XSS attacks. In contrast, CSRF is a one-way attack method. As a result, an attacker can only send her HTTP queries; he or she cannot receive responses to those requests.
While XSS attacks do not call for an authorized user to be in an active session, CSRF attacks do. When a user logs in, XSS attacks can store and send payloads.
The activities a user can do, such as clicking on a malicious link or going to the hacker's website, are the only ones that fall under the purview of CSRF attacks. Contrarily, XSS attacks broaden the attack surface by enabling the execution of malicious scripts to carry out any action the attacker chooses.
A XSS attack stores malicious code on a website, while a CSRF attack stores malicious code on a third-party website that the affected user is redirected to.

5.2 CSRF ATTACK PREVENTION

Here are some techniques [17] that can be implemented in web applications to make them less vulnerable to CSRF:

1. **REST:** REST(Representational state transfer) is a way of organizing your web requests so that they are easy to read and understand, and can handle large amounts of traffic. Following a RESTful design will help make your code easier to work with and faster

2. **Anti-forgery tokens:** Use of POST, PUT, PATCH, and DELETE requests to keep the website secure. An anti-forgery token is added to each request to make sure that only trusted sources can make requests in order to protect these endpoints. An anti-forgery token will be written out to a hidden HTML field in every server response. The client authenticates requests sent to the server using this token. The server is now aware that the request came from a reliable source.

3. **Set correct cookie attributes**: Websites can add a persistent state using cookies. Typically, this is

employed to store session data, authorize users, and more. It's also a simple approach to reveal weaknesses, though. Cookies have a variety of attributes that control their behavior. Cross-site request forgery attacks can be prevented with the use of Chrome's SameSite property. With the help of SameSite, you can control which websites can access your cookies, ensuring that only dependable parties have access to your information.

4. **Additional Authentication**: To protect website's critical data and sensitive actions, authentication is recommended before proceeding. This could be a one-time password, a simple CAPTCHA, or validating passwords.

6. CONCLUSION

In conclusion, this review paper has provided an overview of the most common types of web application vulnerabilities and the tools and techniques that can be used to identify and mitigate them. We have seen that SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF) are among the most prevalent types of vulnerabilities, and that code review, penetration testing, and secure coding practices are key strategies for preventing and addressing these vulnerabilities.
It is clear that web application vulnerabilities can have serious consequences, including data breaches and financial losses. Therefore, it is essential that organizations prioritize the security of their web applications and take steps to prevent and mitigate vulnerabilities. This includes regularly updating and patching applications, as well as implementing effective security measures such as input validation and authentication controls.
Future research in this field should focus on developing new and more effective methods for identifying and mitigating web application vulnerabilities, as well as on the identification of emerging threats. Additionally, there is a need for more comprehensive and up-to-date guidelines and best practices for web application security. By addressing these issues, we can work towards a more secure and reliable online environment for users and organizations alike.

ACKNOWLEDGEMENT

This paper and the research behind it would not have been possible without the exceptional support of my supervisor, Ms. Vandana Choudhary. Her enthusiasm, knowledge and exacting attention to detail have been an inspiration and kept my work on track.

REFERENCES

1. Dhingra, Tanvi. "A Review on Web Application Security - IJCSET." *IJCSET*, IJCSET, May 2015, https://ijcset.net/docs/Volumes/volume5issue5/ijcset2015050508.pdf.

2. digininja. "DVWA/DVWA_v1.3.Pdf at Master · Digininja/DVWA · Github." *DVWA Documentation*, Sept. 2015, https://github.com/digininja/DVWA/blob/master/docs/DVWA_v1.3.pdf

3. Descalso, Alessandra. "How to Prevent Brute Force Attacks." *Intelligent Technical Solutions*, Intelligent Technical Solutions, 29 Nov. 2022, https://www.itsasap.com/blog/how-to-prevent-brute-force-attacks.

4. OWASP. "Owasp Top Ten." *OWASP Top Ten | OWASP Foundation*, Open Web Application Security Project, https://owasp.org/www-project-top-ten/.

5. Lawal, M. A., Abu Bakar Md Sultan, and Ayanloye O. Shakiru. "Systematic literature review on SQL injection attack." International Journal of Soft Computing 11.1 (2016): 26-35.

6. Halfond, William G., Jeremy Viegas, and Alessandro Orso. "A classification of SQL-injection attacks and countermeasures." Proceedings of the IEEE international symposium on secure software engineering. Vol. 1. IEEE, 2006.

7. M. Howard and D. LeBlanc. Writing Secure Code. Microsoft Press, Redmond, Washington, second edition, 2003.

8. C. A. Mackay. SQL Injection Attacks and Some Tips on How to Prevent Them. Technical report, The Code Project, January 2005. http://www.codeproject.com/cs/database/SqlInjectionAttacks.asp

9. Y. Huang, S. Huang, T. Lin, and C. Tsai. Web Application Security Assessment by Fault Injection and Behavior Monitoring. In Proceedings of the 11th International World Wide Web Conference (WWW 03), May 2003

10. C. Gould, Z. Su, and P. Devanbu. JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications. In Proceedings of the 26th International Conference on Software Engineering (ICSE 04) – Formal Demos, pages 697–698, 2004.

11. W. G. Halfond and A. Orso. Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks. In Proceedings of the Third International ICSE Workshop on Dynamic Analysis (WODA 2005), pages 22–28, St. Louis, MO, USA, May 2005

12. F. Valeur, D. Mutz, and G. Vigna. A Learning-Based Approach to the Detection of SQL Attacks. In Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), Vienna, Austria, July 2005

13. Gupta, S., & Gupta, B. B. (2015). Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. International Journal of System Assurance Engineering and Management, 8(S1), 512–530. doi:10.1007/s13198-015-0376-0

14. Kallin, Jakob, and Irene Lobo Valbuna. "Excess XSS." *Excess XSS: A Comprehensive Tutorial on Cross-Site Scripting*, https://excess-xss.com/.

15. Maric, Nadim. "What Is Persistent (Stored) XSS and How It Works." *Bright Security*, 3 Dec. 2021, https://brightsec.com/blog/cross-site-scripting-persistent/.

16. Sengupta, Sudip. "XSS vs CRSF - the Differences Fully Explained." *Crashtest Security*, 24 Nov. 2022, https://crashtest-security.com/xss-vs-csrf-difference/.

17. Tischler, Natalie, and Jill Newberry Queenan. "Preventing CSRF Attacks." *Veracode*, 16 Feb. 2016, https://www.veracode.com/blog/secure-development/preventing-csrf-attacks