Web Application Vulnerabilities and Best Practices: A Comprehensive Analysis

Pradeep B Tarakar
Department of MCA
Dayananda Sagar College of Engineering Kumaraswamy layout, Bangalore, India.

Dr. Srinivasan V
Asst professor
Department of MCA
Dayananda Sagar College of Engineering Kumaraswamy layout, Bangalore, India.

1. Abstract:

Web applications are integral to our digital lives, but they are also prone to numerous security vulnerabilities that can lead to data breaches, unauthorized access, and other malicious activities. This research paper aims to provide a comprehensive analysis of common web application vulnerabilities and propose best practices for mitigating these risks. The study examines a wide range of vulnerabilities, including SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and insecure direct object references. By exploring real-world examples and conducting in-depth analysis, this research aims to raise awareness and provide practical recommendations for developers, security professionals, and organizations to enhance the security of their web applications.

2. Introduction:

Every year, there are more and more web-based programs available. The number of domain names was approximately 367 million as of the first quarter of 2020, despite the fact that there are no statistics on the total number of web applications currently in use worldwide. From one perspective, each of these domains could be seen as a static or dynamic web application.

The domain of software project management faces ongoing challenges with risk management and risk assessment. 78 people took part in Demir's [1] survey on the difficulties in project management.

According to the findings, there were issues with security and risk control in about one out of every four projects. Since many enterprise web applications are integrated within medium- and large-sized businesses, web-based application architecture has become widely used in these organizations.

2.1: Background and significance of web application security:

In the modern era web applications have become an integral part of our lives. From online banking to healthcare management, web applications enable users to perform a wide range of tasks conveniently and efficiently. However, the conveniency comes with many risks. Web applications are exposed wide variety of security vulnerabilities. Web applications often handle sensitive user data, including protected, these data can be compromised, resulting in identity theft, financial loss, and reputational damage to both individuals and organizations.

2.2 Research Objectives and Scope:

The current paper's goal is to present an in-depth evaluation of web applications vulnerabilities and propose best practices for mitigating these risks. By examining common vulnerabilities and their associated impact, the purpose of the article is to increase awareness of the value of web application security among a variety of qualified individuals, including software developers, cybersecurity specialists, and companies..

The research will focus on identifying and understanding prevalent web application vulnerabilities, such as SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and insecure direct object references. It will delve into the techniques employed by attackers to exploit these vulnerabilities and the potential consequences for individuals and organizations. Furthermore, this research paper aims to provide practical recommendations in the form of best practices for enhancing web application security. These best practices including secure coding practices, robust authentication and authorization mechanisms, input validation techniques, secure session management, and regular security assessments.

3. Literature Survey:

3.1 XSS:

In the paper [1] discuss about different types of XSS vulnerabilities and suggest some ways to prevent them. The proposed types and prevention methods are re-Valuated in this research paper.

In another paper [2] discuss about the security vulnerabilities that might cause due to generic input validation. And also propose methodologies to find these vulnerabilities.

3.2 CSRF:

The following papers are related to CSRF attack. The paper [3] focuses on banking system and aid in formulating suggestions for banks and users to make future online banking transactions safer.

[5] in this paper authors proposed and implemented a new automated tool for the identification and mitigation of CSRF vulnerability.

3.3 IDOR (Insecure Direct Object References):

Penetration testing on Web Application using IDOR method: Insecure Direct Object References (IDOR), a penetration testing technique, are used in this paper's case study to evaluate the online application's flaws and vulnerabilities. The URL in question belongs to the application.

3.4 SQL injection:

The paper [9] conducts experiment using three extensively adopted open source vulnerable benchmarks. The method proposed by this paper indicates 26% reduce in the number of SQLi attempts.

4. Proposed Methodology

4.1 Key Findings:

Vulnerability in Cyber Security is a flaw in internal controls, system procedures, or information systems of an organization. Cybercriminals may target these weaknesses and use these areas of weakness to their advantage. Some of the most common and dangerous attacks include:

4.1.1 XSS Attacks (Cross-Site Scripting)

XSS which is abbreviated to cross-site-scripting, is the most common exploitation in web applications. It can control victims to attack other targeted servers and takeover sessions from users. It can also edit, study, and erase company information from applications running on the internet.

A hacking group called Bantown penetrated the 2 million-user online community LiveJournal in 2006 using the recently reported XSS flaws. In order to deceive visitors into clicking, the hacker created a lot of URLs that were infected with malware. Users' cookies have the potential to be obtained by the attacker when victims clicked these URLs, and he or she could then utilize those cookies to log into the victims' accounts [1].

Types:

1. **DOM-based**, also called Type-0 XSS. The exploitation happens on the client side. The attacker sends a legit-looking link that has malicious script in it. The victim clicks on it, makes a request to the actual server, and gets back the response. Once the response is received, the malicious script executes and sends sensitive data to the attacker without the knowledge of the victim.
2. **Stored-XSS:** also called Type-1 XSS. where an attacker sends the server a request. There is harmful code in the request. The XSS vulnerable server blindly processes it and stores it in a database. Thereafter, Every request received by the server is served with a malicious script. The script then sends the sensitive information of the victims to the attacker. It commonly occurs in forums and blog sites.
3. **Relected XSS:** known as Type-2 XSS. The procedure is comparable to DOM-based XSS; however, the difference is that the victim sends the malicious script sent by the attacker with the link back to the server. The identical procedure then continues.

Demo:

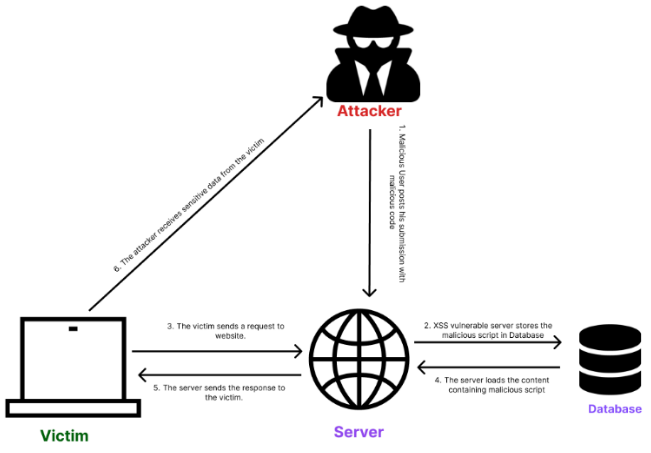

Attacker
1. Malicious User posts his submission with malicious code
6. The attacker receives sensitive data from the victim
3. The victim sends a request to website.
5. The server sends the response to the victim.
2. XSS vulnerable server stores the malicious script in Database
4. The server loads the content containing malicious script
Victim
Server
Database

Fig 1: Stored XSS

4.1.2 CSRF (Cross-Site request forgery)

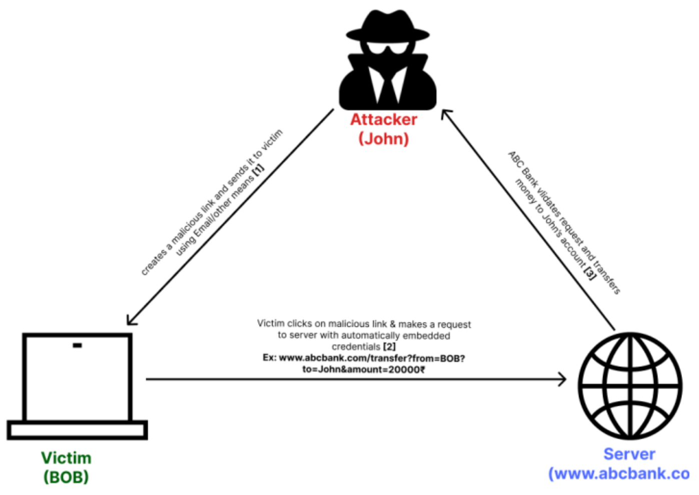
Cross-Site Request Forgery (CSRF) is an attack or method used to carry out unauthorized operations on a web-based application where people have been logged in now as members of the public. A CSRF attack can be carried out with just a little assistance from social engineering techniques (chat, visiting a rogue website, or sending a link via email). CSRF attacks primarily target requests that can alter the status of an application. There is no method for the attacker to inspect the response to the faked requests, and while establishing the target, the attacker will not take any of the data. [4]

This attack focuses on luring the victim to submit a malicious request to an authentic server (like a bank server) to gain financial or other advantages. Since the browser automatically includes credentials associated with the site, such as session cookies and Windows domain credentials, Therefore, whether a user is signed in right now and a session has not expired yet, the server will consider it an authentic request. Due to the absence of a mechanism on the server that can distinguish between false and legitimate requests. [4]

<a href="http://www.abcbank.com/transfer?from=victim?to=attacker&amount=$100">Click Me! </a>

GET http://www.abcbank.com/transfer?from=victim?to=attacker&amount=$100

HTTP 1.1


Attacker
(John)
creates a malicious link and sends it to victim using Email/other means [1]
ABC Bank vlidates request and transfers money to John's account [3]
Victim clicks on malicious link & makes a request to server with automatically embedded credentials [2]
Ex: www.abcbank.com/transfer?from=BOB?to=John&amount=20000₹
Victim
(BOB)
Server
(www.abcbank.co

4.1.3 IDOR (Insecure Direct Object References)

The lack of a complete protective strategy for sensitive data or resources is represented by the IDOR, which highlights design defects in the system. A developer exposes a reference to a database key, file, directory, or other internal implementation object as a URL or form parameter when doing a direct object reference. The attackers may use such references to gain access to protected data in the absence of a security check or additional security measures. Because of the wrong amount of administrative access to the system data caused by the lack of authentication level checks, the unsecured direct object reference simply represents this. This occurs when developers offer data items through a web application with the presumption that users would always abide by the rules of the program.[2]

Case: Let's consider a bank web application provides financial data report. The web page is designed in such a way that the user will not check others financial data. The url looks something like this:

http://www.vulnerableBank.com/accounts/viewDetail?id=0010

Now, if a user is knowledgeable enough to understand the URL he/she can simply change the id value to access others financial report resulting IDOR attack. In worst case he/she may access other URL's and change user details.

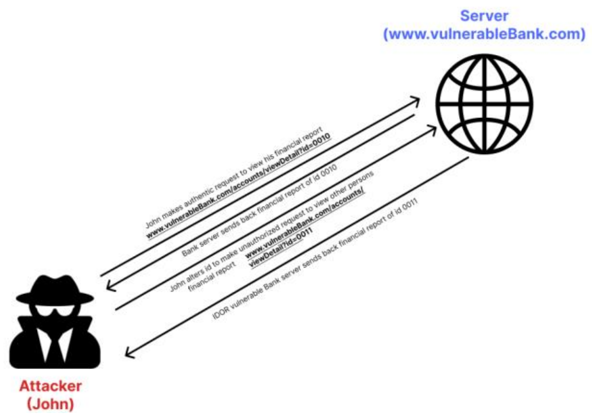

Server
(www.vulnerableBank.com)
John makes authentic request to view his financial report
www.vulnerableBank.com/accounts/viewDetail?id=0010
Bank server sends back financial report of id 0010
John alters id to make unauthorized request to view other persons financial report
www.vulnerableBank.com/accounts/viewDetail?id=0011
IDOR vulnerable Bank server sends back financial report of id 0011
Attacker
(John)

Fig: IDOR in action

4.1.4 SQL Injection

In an SQL injection attack, the attacker tries to put in a malicious string which the database will interpret and execute. The string can be a malicious SQL query to get the information back from the database or to escalate the privilege.

Over a nine-month period, the Web application attack report recorded an increase in various Web attacks of about 17% on average [35]. It is noted that website attacks have become more complicated and much longer in duration (44% greater than they were in past studies), as well as that a typical web-based application might encounter over 26 attempts in a minute.

Another security analysis revealed that at least 8% of the Web services provided by organizations like Microsoft and Google had various security flaws.

Popular Types: Tautology, piggy-backed queries, stored procedures, union query and inference based attacks.

Ex1: Consider a company server that stores employee details. To get a employee detail we use url something like:

http://www.comp.com/employee/employee.esp?id=1002

Attacker might use URL as shown below to get all employee details

http://www.comp.com/employee/employee.esp?id=1002 or 1=1

As a result, SQL query might look something like

```
SELECT id, name, address, phone
FROM Employee
WHERE id = 999 OR 1=1
```

Database executes and since 1=1 is true it returns all employees details.

Ex2: Considering same scenario attacker might use below url to delete all records

http://www.comp.com/employee/employee.esp?id=1002; DROP TABLE EMPLOYEE

As a result, SQL query might look something like

```
SELECT id, name, address, phone
FROM Employee
WHERE id = 999; DROP TABLE employee
```

4.2 Best Practices Proposed

4.2.1 Mitigating Vulnerabilities:

1. Developing secure code

Software Engineers of mission-critical Web-based systems ought to use defense-in-depth programming techniques, taking into account each safety measure has a chance of failing, to generate code free from vulnerabilities. During implementation, it is crucial to use a strategy that relies on multiple layers of defensive mechanisms since sometimes a single precaution or defense is not enough to prevent security flaws.

There are 3 distinct lines of defense: Input validation,

Hotspot protection, and output validation.

2. Input Validation: Only after the intended software wrongly checks its input data does the majority of security problems arise. Applications must consequently treat all inputs—including information from unreliable sources—as hazardous until otherwise proven.

As a first line of defense, input validation reduces the input domain of an application by operating solely on the values supplied by the user. This type of defense concentrates on either stopping the software from running when a user enters a value that isn't allowed in the domain in question or demanding that the input parameter fall inside a given acceptable domain. Starting with normalizing the inputs to a standard character set and encoding is recommended for Web applications. The program must then apply filtering techniques to the normalized inputs and discard any that contain beyond the domain parameters. By using this kind of technique, Web applications that do input validation using positive pattern matching or positive validation might prevent several issues.

3. Hotspot Protection: To counteract the shortcomings of input validation, a second layer of protection is required. Each attack type focuses on a hotspot, or a specific group of statements in the application's code that are vulnerable to a particular class of flaws. Only important areas are protected by this supplementary defense, for instance by making sure the information actually utilized in these lines falls inside their input area. This contrasts with generic input validation, where the application validates or changes inputs within the context of the entire Web application.

The majority of SQL injection attacks, which use quote marks, both single and double, are one specific example. These characters can be used within a SQL expression using mechanisms provided by some computer languages, but only to delimit values in the statement.4

However, these methods have two fundamental issues. First, these mechanisms can be gotten over by using more complex injection techniques, including quoting and escaping characters together. Second, adding escape characters lengthens the string and raises its chance of exceeding the database's maximum length, which may result in data truncation.

The best method to prevent injection vulnerabilities is to correctly use parameterized commands.1 In this instance, the developer establishes the instructions' structure by utilizing placeholders to represent the Combining quotation marks with escaping characters helps get around these defenses. Second, adding escape characters lengthens the string and raises its chance of exceeding the database's maximum length, which may result in data truncation.

The best method to prevent injection vulnerabilities is to correctly use parameterized commands. Under this instance, the developer defines the commands' structure by utilizing placeholders to denote the variable values of the instructions.

Using the command-line translator the appropriate values appropriately without messing with as a result of the hierarchy of commands afterwards, when the software attaches the appropriate inputs to the command. The most well-known application of this concept is parameterized inquiries, often known as database prepared statements. The structure of a prepared statement is stored in the database when it is created by an application.

4.2.2 Defending Attacks

The three primary etiological techniques utilized to defend web applications from injection attacks are as follows: Policy Enforcement, Instruction Set Randomization, and Parse-Tree Validation.

1. **Parse-Tree Validation**

Parse-tree validation's main principle is to compare the source code's intended execution's abstracted syntactic framework with its tree representation to ensure that it is written as intended. Trees that diverge indicate that the application is possibly being attacked.

2. **Policy Enforcement**

XSS and CSRF attacks are prevented with this strategy. Developers that implement an architecture that imposes policies need to set up particular server level security rules. Syntax-specific preferences, matching patterns, or JavaScript plugins can all be used to convey policies. The limits are then put into effect via a server-side gateway that blocks server responses or a user's browsers at runtime. Incorporating rules directly into JavaScript or HTML code to regulate user behavior is another way to enforce policies.

In order to look through the HTML of server answers and identify scripts, BrowserShield functions as an intermediary on the server-side (Point StB). It then edits them into secure alternatives to shield web users from exploits built using known browser flaws. ConScript, CoreScript, and the Phung et al. architecture introduce new primitive functions to JavaScript, offering safe barriers against potentially harmful JavaScript methods. Every time, policy enforcement happens on the client side, which is handled by the browser's JavaScript engine (Point UB). This would prevent XSS attacks from succeeding, which combine seemingly innocent elements into harmful texts by employing popular methods like write and eval.

3. Symptomatic

Tracking Toxicity:

Using a taint tracking method, such as a parameter set by a field in a web form, untrustworthy ("tainted") information is found and its spread through the software is tracked. A variable is labeled as untrusted if it is used within an expression which sets another one, and so on. The scheme could function in line with the use of any of these factors in a potentially dangerous behavior (for instance, sending the data to a vulnerable "sink," such a database, file, or the network).

Some computer languages, like Perl and Ruby, offer a way to track taint. This feature would stop Perl from executing any code that could be used in a SQL injection attack. (Think about the use of a contaminated variable in a request) and would end with a failure.

5. Results and Discussion

With the use of the above-mentioned methodologies and defense mechanisms we can certainly draw a good chance of mitigating common security attacks like XSS, CSRF, IDOR and SQL injection attacks.

The suggested technique offers a methodical and exacting way to conduct an extensive study of web application weaknesses and best practices. The study article uses this methodology in an effort to offer insightful analysis, suggestions, and directives for enhancing online application security. The methodology's important findings are highlighted in the results and discussion section, highlighting the completeness and reliability of the research strategy.

6. Conclusion

In order to better understand these crucial security issues and offer useful suggestions for enhancing online application security, the research paper conducted a thorough analysis on web application vulnerabilities and best practices. This work has accomplished its goals and made significant contributions to the field of online application security using a systematic process that includes data collecting, analysis methods, validation, and synthesis.

SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and unsafe direct object references were among the common online application vulnerabilities identified by the research. These flaws represent serious threats to the privacy, security, and accessibility of sensitive user data and may have dire repercussions like unauthorized access, data breaches, and monetary losses.

By conducting a thorough literature review, the study identified key methodologies, techniques, and best practices proposed by researchers and practitioners to mitigate these vulnerabilities. The analysis involved a comparative examination of the selected literature, which revealed patterns, trends, and emerging issues in web application security. The development of a conceptual framework facilitated the organization and categorization of the identified vulnerabilities and best practices, providing a structured approach for analysis and presentation.

The validation and evaluation process involved expert review and real-world case studies, which served to validate the research findings and recommendations. Experts in the field of web application security provided valuable insights and feedback, enhancing the accuracy and applicability of the analysis. Real-world case studies highlighted the impact of vulnerabilities on organizations and demonstrated the effectiveness of the identified best practices in mitigating risks.

7. References:

1. Miao Liu; Boyu Zhang; Wenbin Chen; Xunlai Zhang. A survey of exploitation and Detection methods of XSS Vulnerabilities. https://ieeexplore.ieee.org/document/8935148/

2. Debasish Das; Utpal Sharma; D. K. Bhattacharyya Detection of XSS attack under multiple scenarios. https://ieeexplore.ieee.org/document/8187802/

3.Gifty Buah; scholastica Memusia; John Munyi; Timothy Brown; Rober A. Sowah. Vulnerability Analaysis of Online Banking sites to CSRF attacks. https://ieeexplore.ieee.org/document/9681978/

4.Muhammad Zulkhairi Zakaria; Rashidah Kadir. Risk Assessment of Web Application Penetration Testing on CSRF attacks and server-side includes(SSI) injections. https://ieeexplore.ieee.org/document/9617554/

5. W.H. Rankothge; S M. N. Randeniya. Identification and mitigation Tool for CSRF. https://ieeexplore.ieee.org/document/9357029/

6. Pratibha Yadav; Chadresh D. Parekh. A report on CSRF security challenges and prevention techniques. https://ieeexplore.ieee.org/document/8275852/

7. Putu Agus Eka Pratama; Alvin Maulana Rhusuli. Penetration testing on Web Application using IDOR method. https://ieeexplore.ieee.org/document/9915074/

8. Identification and Illustration of Insecure Direct Object References and their Countermeasures https://www.researchgate.net/profile/Ajay-Shrestha-5/publication/275043190_Identification_and_Illustration_of_Insecure_Direct_Object_References_and_their_Countermeasures/links/56a8763008aeded22e37a0ec/Identification-and-Illustration-of-Insecure-Direct-Object-References-and-their-Countermeasures.pdf

9. Long Zhang; Donghong Zhang; Chenghong wang; Jing Zhao; Zhenyu Zhang. The art of SQL injection

on vulnerability discovery https://ieeexplore.ieee.org/document/8716725

10. Xin Xie; Chunhui Ren; Yusheng Fu; Jie Xu; Jinhong Guo. SQL injection detection for web applications based on Elastic Pooling CNN https://ieeexplore.ieee.org/document/8877739/

11. Leena Jacob, Virginia Mary Nadar, Madhumita Chatterjee. Web Application Security: A Survey https://ijcsit.com/docs/Volume%207/vol7issue1/ijcsit2016070196.pdf

12. Nuno Antunes; Marco Vieira. Defending against web application Vulnerabilities https://ieeexplore.ieee.org/document/5999632

13. Dimitris Mitropoulos. Defending Against web application Attacks: Approaches, Challenges and Implications.
https://ieeexplore.ieee.org/document/7865911