State-of-the-Art Survey on Web Vulnerabilities, Threat Vectors, and Countermeasures

Jasleen Kaur and Urvashi Garg

1 Introduction

The primary reason behind any web attack is insufficient security or design flaws in the web application, thereby, allowing hackers to enter into the system and steal confidential data such as username, passwords, transaction details, session Ids/token, and database-related information. According to a survey, PHP (78.5%) and JavaScript (94.7%) are the most commonly used server-side and client-side programming languages, respectively [1].

A cyber-criminal would first analyse the website for a vulnerability using online tools such as vulnerability scanners or botnets. Vulnerabilities such as virus-infected administrator's system, weak password, out of date security patches, browser plugins, and permissive coding practices may give a chance to the hacker to enter the system and steal the data. Moreover, a recent survey was conducted on the usage of world-wide web in which the authors depicted that most common vulnerabilities are found at application level, which is layer 7 according to the OSI network model [2], and 93% of data breaches occur due to human error while designing and developing the web application [3]. For an instance, neglect of data validation could give a clear path to attacker to deceive the web server into running unsafe commands [4]. In 2019, an EDGESCAN organisation generated a vulnerability statistics report in which it claimed that 19% of all vulnerabilities were associated with layer 7, and the rest 81% of vulnerabilities were linked with network layer. SQL injection was significant at 5.55%, XSS at 14.69%, and other injection attacks such as OS, CRLF, and JavaScript were significant at 8.18%. According to 2020 Cyber security report, approximately 93% of the files, which were shared through web in India, were found to be malicious [5], and 64% of the organisations in India are believed to be impacted by the information disclosure vulnerability. Despite the extensive research

J. Kaur (✉) · U. Garg
CSE Department, Chandigarh University, Mohali, India

R. Agrawal et al. (eds.), *Cyber Security in Intelligent Computing and Communications*, Studies in Computational Intelligence 1007,
https://doi.org/10.1007/978-981-16-8012-0_1

Table 1 Top 10 common vulnerabilities and exploits (CVE)

CVE Number	Vulnerability name
CVE1	Injection
CVE2	Broken authentication
CVE3	Sensitive data exposure
CVE4	XML external entities (XXE)
CVE5	Broken access control
CVE6	Security misconfigurations
CVE7	Cross site scripting (XSS)
CVE8	Insecure deserialisation
CVE9	Using components with known vulnerabilities
CVE10	Insufficient logging and monitoring

being done in developing new tools and protocols to detect, prevent, and mitigate the web attacks, still numerous websites are non-immune to the web attacks. This clearly depicts the need to detect the software-related vulnerability in order to prevent web security exploitation by the hacker. Following are the top 10 vulnerabilities in 2020 [6] according to OWASP (Open Web Application Security Project) [7] (Table 1).

Figure 1 demonstrates an attacker interrupting normal communication between client and server, getting successful in bypassing the system, and modifying the crucial data. The attack is viable due to the nature of HTTP and HTTPS protocol. In case of HTTPS, two connections are built up: one SSL connection is created between the client and the hacker, second SSL connection is created between the hacker and web server where the cybercriminal splits the TCP connection between the client and web server. In 2019, SSL labs claimed that 1.2% of HTTPS servers are still vulnerable to attack. For example, DROWN attack vulnerability can be easily carried out on websites using HTTPS and SSL/TLS services [8]. The misconfiguration and

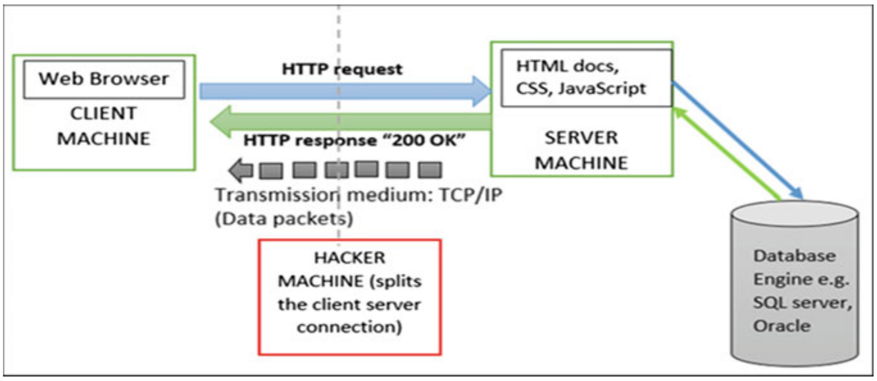

Web Browser
CLIENT MACHINE
HTTP request
HTTP response "200 OK"
Transmission medium: TCP/IP (Data packets)
HTML docs, CSS, JavaScript
SERVER MACHINE
HACKER MACHINE (splits the client server connection)
Database Engine e.g. SQL server, Oracle

Fig. 1 Scenario of attack on a web application

inappropriate default setting allow the attacker to decrypt TLS connection between client and server.

2 Literature Survey

2.1 SQL Injection Vulnerability Attack and Prevention

This web attack was first discovered in 1998 by a security researcher, Jeff Forristal, and is still at the top list since 2003. Antivirus programs are ineffective at handling SQLI attack. Any company that operates its website on SQL database is prone to this attack if it does not have sufficient input validations in its web forms. As a result, anyone can insert malicious SQL commands into the input string of a web form, web cookie, or a page request (browser), and can retrieve, modify, and delete the data present in the database putting data integrity, authentication, authorisation, and confidentiality at risk. In 2012, a researcher claimed that 97% of data breaches occur due to SQLi. Surprisingly, health industry is the most attacked industry and with maximum number of data breaches due to SQLi attack. The attack is done on data-driven applications as the behaviour of these applications generally depends upon the data input. Therefore, this attack is quite easy to execute. However, lack of awareness and implementation of security protocols by the organisations leads to data leaks resulting from SQLI attack. The attack could be carried out with one of the following objectives [9]:

- to identify injectable parameters
- to extract/retrieve data
- to add/modify data
- to perform DOS
- to evade detection
- to bypass authentication
- to execute remote commands
- to perform privilege escalation.

SQL injection attack has two stages [10]:

i. Injection attack stage 1
ii. Injection attack stage 2.

Stage 1 is known as reconnaissance. At this stage, the attacker passes random unexpected values to the arguments and observes how application responds. Stage 2 is known as actual attack. At this stage, the attacker provides carefully-crafted input values that will be interpreted as part of SQL commands rather than merely data. The database then executes the SQL commands as altered by the attacker. SQLi can be categorised into the following four types as illustrated in Fig. 2:

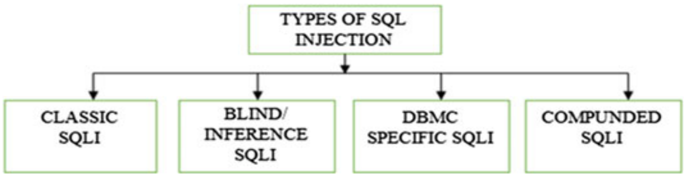


Fig. 2 Types of SQL injection vulnerability attacks [9]

2.1.1 CLASSIC SQL Injection Attack

It occurs generally when the user input is not filtered and escaped correctly in the web form. Owing to this, attacker sends batch commands to the database server, and in return receives specific output based on the input statements [11]. As a result, he can control application's entire database as illegitimate admin user. The input may include SELECT commands, which can download entire database including users' personal information such as unique identification, phone number or credit/debit card numbers. The attacker could also use INCLUDE or UPDATE commands to create new user accounts or alter the existing ones.

For instance, following Fig. 3, integer value '1' is passed to the web submission form (DVWA), where the security level was set as low, which returned the first name and surname for user id '1'. Similarly, it will return the values for user id 2 or 3. This means that website is vulnerable to SQLI attack. Moreover, the URL also depicts the id number. The URL is:

http://localhost/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#

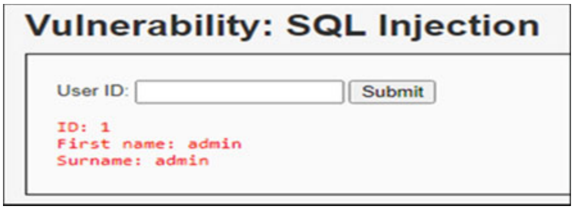
If the id number is changed in URL itself, the results will be displayed for that particular id. For example, if we change the id from 1 to 2 and press enter, the database will return the first name and surname for id 2.

We can also extract all first names and surnames by passing the string %' or '0'='0 in the input form. This will return the information for all five records present in the database (Fig. 4).

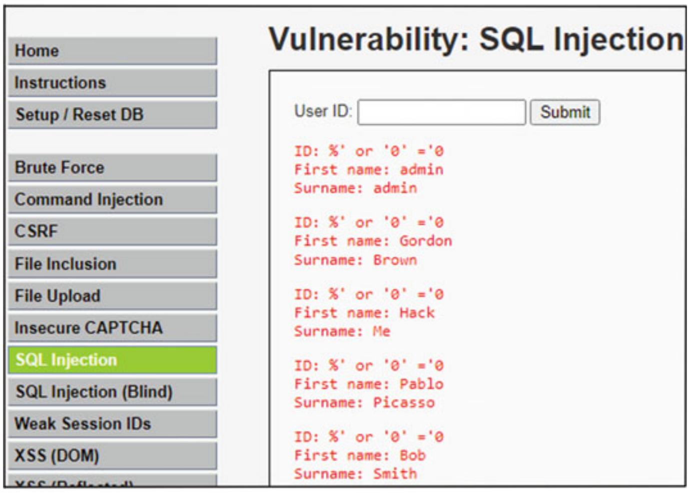
Classic SQLI can be implemented by one of the following techniques [1, 9, 12]:

i. Tautologies
ii. UNION SQL Queries
iii. Piggy-backed SQL queries
iv. Alternate encoding

Fig. 3 Extracting the values of argument 'id'



```
ID: 1
First name: admin
Surname: admin
```



```
ID: %' or '0' ='0
First name: admin
Surname: admin

ID: %' or '0' ='0
First name: Gordon
Surname: Brown

ID: %' or '0' ='0
First name: Hack
Surname: Me

ID: %' or '0' ='0
First name: Pablo
Surname: Picasso

ID: %' or '0' ='0
First name: Bob
Surname: Smith
```

Fig. 4 Extracting the values for all records

v. Illogical Queries
vi. Stored Procedures.

USING TAUTOLOGY

Tautology means an expression or a logical statement, which is always true. This means that attacker can use such SQL statements, which will always be true, and hence results in executing the queries at the database server. The attack is carried with one of the following objectives:

- to extract/retrieve data
- to bypass authentication
- to identify injectable arguments.

The attack is implemented by using conditional expressions using OR operator.

USING UNION COMMAND

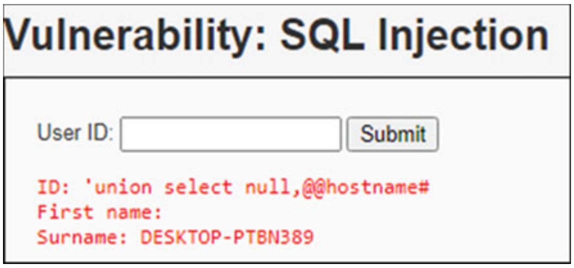
UNION SQL attack is operated especially to determine the database version or the information about the number of rows and columns. Just like the former uses OR operator, this attack uses UNION operator where UNION is used to merge two SELECT statements.

The attack is carried out with one of the following objectives:

- to extract/retrieve data
- to bypass authentication.

By default, most of the databases such as MySQL stores the database-related information like name with version, number of tuples, etc. and can display the database

Fig. 5 Using union query to retrieve the hostname



```
ID: 'union select null,@@hostname#
First name:
Surname: DESKTOP-PTBN389
```

version while generating error messages for incorrect queries [13]. Such misconfiguration could allow attackers to compromise database for future attacks. For example, the following UNION SQL query will end up in extracting the information of all records with database version as the last one.

%’ or 0=0 union select null, version() #

Attacker could also use the union statement to extract the hostname (Fig. 5) using the following command:

‘union select null, @@hostname#

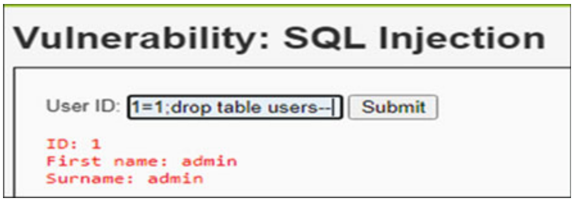
The attacker could also pass union queries to extract the details of information schema, location of database files, and even read files located on the remote system. Therefore, in order to avoid such type of attacks, it is always recommended to use prepared statements in conjunction with GET statement.

USING PIGGY-BACKED STATEMENTS

As the name implies, piggy-backed statement means to add one statement at the end of another statement to make it a single command by using semicolon. The database that would be vulnerable to such attack could allow multiple statements to be treated as a single statement if and only if the former statement comes out to be valid and true. For example, following Fig. 6 demonstrates inserting second query using semicolon after the first true query.

This vulnerability can be misused by the attacker to execute remote commands such as dropping the tables or to shut-down the entire system using command SHUTDOWN;–. As a result, he can effortlessly implement ‘Denial of service (DoS)’ attack.

Fig. 6 Sample example of piggybacked SQLI attack



```
1=1;drop table users--
ID: 1
First name: admin
Surname: admin
```

USING ALTERNATE ENCODINGS

An attacker may use special encoding techniques in order to prevent detection of malicious code by the software [14]. For instance, he may use ghost characters to bypass the filters as Web or FTP server fails to detect the extra characters. These characters are the extra characters, which do not have any effect on the API layer, hence, will automatically get stripped off from input string. Following is the list of 'improper handling of encoding' vulnerabilities [15] that could allow attacker to do further damage:

i. Using char() of ASCII [1]
ii. Using ghost characters
iii. Passing special characters using % in the URL (URL encoding due to insufficient filtering on the URL)
iv. Repetition of encoding or Double encoding
v. Encoding IP/web address
vi. Adding NULL bytes in the input
vii. Using Unicode/UTF-8 encoding technique
viii. Using NULL terminator by post-fixing the data to avoid filter.

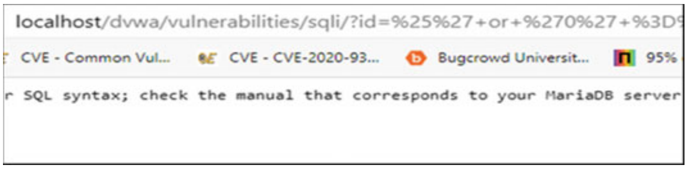
This type of attack is difficult to implement as the developer needs to check the validation and proper sanitisation for all of the above-mentioned encodings including URLs, IP address, and input.

USING ILLOGICAL QUERIES

As the name suggests, a threat actor can pass incorrect SQL statements in order to collect critical information about the database just from the error or log messages, which could display errors related to syntax of code, logical error, or type mismatch error. This could lead to exposure of injectable arguments/parameters to the attacker. Due to this reason, this type of attack is also sometimes referred to as error-based injection [1]. For example, in the following Fig. 7, after inserting incorrect query, server return name of database in the error message.

USING STORED PROCEDURES

Stored procedures are the compound statements that contain a set of multiple SQL statements as a group, which further gets saved in a data dictionary of RDBMS [16]. This group is given a specific unique name. This provides flexibility to call these set of statements from multiple programs using a single name (just like we


localhost/dvwa/vulnerabilities/sqli/?id=%25%27+or+%270%27+%3D

```
r SQL syntax; check the manual that corresponds to your MariaDB server
```

Fig. 7 Sample example of error-based SQLI attack

call functions). As a result, they provide various benefits such as handling runtime errors, data validation, provide mechanism for access control, etc. There is a common myth among most of the developers that stored procedures are always safe. However, they are completely not, if dynamic SQL inside the stored procedure is not handled properly. What I mean to say is, if the dynamic query used inside the stored procedures is created by concatenating the user input values instead of formal parameters, then it is at high risk. For example, the following first statement illustrates the bad example of dynamic SQL.

sb.command.Append("Name="+inputName.value+, ",");

Good example:

sb.command.Append("Name=@Name");

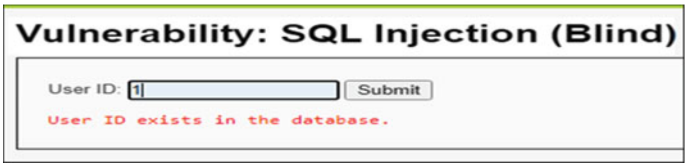
2.1.2 BLIND SQL Injection Attack

As the name suggests, in this type of attack, the results of SQL injection are hidden from the attacker, therefore, it becomes quite difficult for the attacker to extract data in one attempt [9, 11]. The attacker performs number of attempts before reaching to final successful request. It is also known as inference injection attack. For example, let us take the same example that we took in CLASSIC SQL injection attack. If we pass a true value, i.e. 1, then instead of getting the actual results such as value for first name and last name, we will get out mentioned in Fig. 8.

Blind SQLI attack has two types:

i. Time-based blind SQL injection attack
ii. Content-based SQL injection attack.

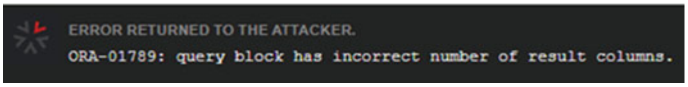
Sometimes, it is also referred to as conditional response as the attacker sends a malicious code with some conditions to the server and checks the response. In most cases, the queries are crafted as Boolean values, i.e. true or false. If the response happens to be true, the injectable parameters can be detected, else attacker can try another malicious set. It could also be the response rate of HTTP request [17]. In the first type, the attacker sets a time limit in the code and analyses the response received from the web server. Whereas, in the content-type, it is done depending upon the content generated by the query. In order to check whether the website is vulnerable to BLIND SQLI attack (stage 1: reconnaissance), attacker could use


Vulnerability: SQL Injection (Blind)

User ID: 1

User ID exists in the database.

Fig. 8 Sample example of SQL blind injection attack



```
ERROR RETURNED TO THE ATTACKER.
ORA-01789: query block has incorrect number of result columns.
```

Fig. 9 Sample error message

online vulnerability tool such as SQLMap (could be even used by researchers for teaching and learning process).

2.1.3 DBMS Specific SQL Injection Attack

This type of attack is done using two techniques: DB fingerprinting and DB mapping. DB fingerprinting means executing illogical queries in order to extract database-related information such as analysing error messages, inserting query to know DB version, ascertaining table names, information schema, and number of rows and columns, etc.. The type of error message generated by the database will vary depending upon the type of back-end database used. For example, the following error messages tell about the incorrect number of columns, so attacker can easily modify the input to obtain the correct result (Fig. 9).

The attacker could also construct a query to retrieve the exact version of database using inference testing as discussed earlier. By mapping the database using online tools, hackers can easily access the application's data layer.

2.1.4 COMPOUNDED SQL İnjection Attack

Compounded SQLI attack means that the attacker can use another attack in conjunction with SQLI attack. For instance, the following attacks can be executed by the attacker after performing SQLI attack.

i. XSS attack,
ii. insufficient authentication attack,
iii. DDoS attack, and
iv. DNS hijacking attack.

Finally, SQLI attack can be prevented only by considering the above-mentioned vulnerabilities while developing a website as firewalls, antivirus programs, and SSL are ineffective in preventing such attacks. Therefore, developer must consider the following points in order to avoid SQL injection attack on web application:

i. Using prepare() function (prepared statements)
ii. Including user input validation statements such as removing the extra special character or string such as –,;, ‘, SHUTDOWN, DROP, or DELETE (from web URL, web form or cookie) while receiving input from the user as such characters could be used to bypass the web filters.

iii. Treating received input from the user as a string instead of a command.
iv. Always keep in check of permission scheme of database, and doing regular checks of all system files for any modification to the system.
v. Configuring the database error messages so that critical information do not get exposed to someone who do not have access rights.

2.2 Broken Authentıcatıon and Sessıon Management Vulnerabılıty Attack and Preventıon

Since HTTP is a stateless protocol, some kind of protocol is required that can keep track of the activities of a particular user using the website and is passed as an argument in the GET or POST query. This is achieved by providing session ID or token to a user when he visits any website [18]. This session id is used to identify that user during the information exchange (HTTP request and HTTP response). The time span of the sessions is kept as short as possible for security purposes. If sessions are not handled properly during the website development, the attacker could use or steal any logged-in user's session id and can obtain the potential privileges. Session ID is usually generated as a random long string so that it becomes difficult for the user to guess the next one [18].

Generally, sessions can be maintained either on server side or on client side depending upon the web application's requirements. While storing session on server is a highly complex process and may result in an increase in latency time, the users' credentials are generally claimed to be much safer as the users' data are not exposed, and the cookie size is kept small. On the other hand, due to the complex nature of server-side session management, most developers prefer storing the session inside the authentication cookie on the client side. However, this common technique generally poses higher risks if the data integrity, authenticity, and confidentiality are not guaranteed.

The website could be vulnerable to session fixation attacks if the sessions and authentication are not handled properly while designing or developing the website. The following Fig. 10 demonstrates the session fixation attack.

Any website is vulnerable to broken authentication and session fixation attack if the following points are not considered:

i. Permits the use of weak password
ii. Permits the multiple failed login attempts
iii. Session ID is visible in the URL
iv. Multi-factor authentication is missing
v. Session ID is not refreshed during the activity
vi. Session id still persists in memory even when the user has logged out, especially when user sign-in using SSO (Single Sign ON that means signing-in by trusting the third party such as login through Google or Facebook)
vii. Using unencrypted communication channel for sending password or session ids/tokens.

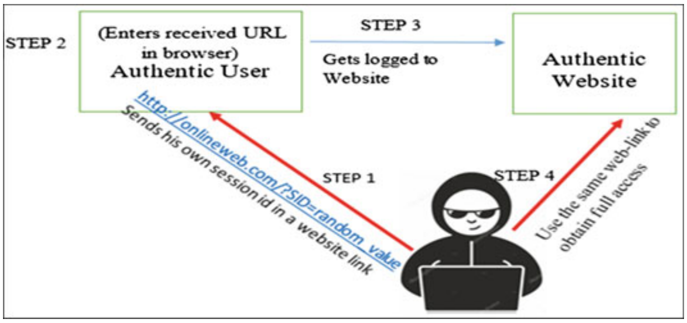

STEP 2
(Enters received URL in browser)
Authentic User
STEP 3
Gets logged to Website
Authentic Website
http://onlineweb.com/?SID=random_value
Sends his own session id in a website link
STEP 1
STEP 4
Use the same web-link to obtain full access

Fig. 10 Sample scenario of session fixation attack

viii. Using weak account recovery algorithms.

Md. Maruf Hassan et al. [15] executed a case study on weak authentication and session management vulnerability in Bangladesh and found out that a total of 267 public (72%) and private (28%) organisations were vulnerable to this attack, i.e. approximately 56% websites among their sample. The intruder can obtain the session id of targeted user using online tools such as Google dork, eat my cookie, or cookie manager.

Following points must be considered to prevent this type of attack:

i. Ensure strong password by adding validation checks.
ii. Limit the failed login attempts and alert the concerned user and the admin regarding the brute-force attempt.
iii. Ensure the shortest life span of each session ID.
iv. Sessions must be shared over the encrypted channels.
v. Use strong hashing and salting algorithm to store passwords in the database such as SHA256 [19].
vi. Better use POST method instead of GET method as it is more secure because it never expose the user's data either through web URL or server logs.
vii. Use strong hashing function to encrypt password.
viii. It is necessary to test all the platforms (such as Google or Facebook) used where sessions are being shared through URL.
ix. Include the option of asking old password while the processing request of changing the password.
x. Ensure the data are not cached in the browser, i.e. back button must not show the previous result in case of banking websites.
xi. Web application firewalls can be used to validate the sessions.
xii. Use SSL certificate.

2.3 Cross-Site Scripting Attack and Prevention

Two-third of all web apps are found to be vulnerable to cross-site scripting attack, also known as XSS attack. The term was first introduced in November 1999 when a group of security researchers heard about the injection of malicious scripts and image tags into the HTML pages of some dynamic websites. After 2 months, in February 2000, they published a report demonstrating the XSS vulnerability. For your knowledge, it was named XSS instead of its short form CSS only to avoid name ambiguity for Cascading Style sheets (CSS). The malicious script is executed on the client side, usually on user's browser. As a result, the communication between the user and vulnerable website is compromised. If a dynamic website is vulnerable to SQLI attack or broken authentication and session management attack, then there is a higher risk that the website will be vulnerable to XSS attack as well. Just like SQLI attack is targeted for SQL-based applications by passing SQL queries, XSS attack is targeted to HTML pages where the intruder injects the malicious code into HTML web pages. It is the most popular technique used by cyber-criminals to steal sessions or to attack a company's entire social network. According to Wikipedia, the most prominent websites such as Facebook, Twitter, and YouTube had also suffered from this attack in the past. The following steps explain the general scenario of attack:

Step 1: Attacker finds a vulnerable website, which allows the injection of untrusted malicious code into its webpage. For example, inserting false advertisements on the web page, displaying false content on the website.

Step 2: Attacker inserts malicious client-side JavaScript/ActiveX/VBScript/HTML code on the web application. This code is either sent to the victim's web browser or the web server depending upon the type of XSS attack.

Step 3: User clicks on the malicious link either while visiting the website or accessing service from the web server.

Step 4: Attacker has access to private credentials or details of the victim through a vulnerable website by bypassing the SOP (Same Origin Policy).

M. Liu et al. [18] conducted a survey on XSS attacks on their local vulnerable test website. The paper illustrated the various risks associated with XSS vulnerability. The risks include phishing attacks, exploitation of user's session id or token id, DoS and DDoS attacks, stealing client's web browser screenshot, and risk of XSS worms on click malicious link.

Germán E. Rodríguez et al. [16] conducted a survey on mitigation of XSS attacks and discovered that 40% of attacks are implemented using XSS technique. The following table illustrates the use of most common attacks with their percentage according to [16] (Table 2).

XSS attacks can be categorised into the following types [18]:

i. Server-side vulnerability
   - Persistent XSS
   - Non-Persistent XSS

ii. Client-side vulnerability

Table 2 Percentage of occurrence of attacks

Vulnerability attack	Percentage of occurrence (%)
XSS attack	40
SQLI attack	24
Inclusion of local files	4
DDoS attack	3

- DOM-based XSS

Persistent XSS is also known as Stored XSS. In this attack, the malicious script is added directly on the website (especially forms, blogs or comment sections), therefore, it is also known as direct/second-order/type-1/stored XSS attack as the script gets stored on the web server. So, whenever user visits that website, the malicious code gets executed, and hence, it is said to be more harmful than other two types.

If website is vulnerable to this attack, then attacker can execute phishing attack and key-logger attack. In former attack, the credentials of the user are compromised. In later, attacker is able to capture the keystrokes of the user for the vulnerable web page. Attacker can also construct a script to take screenshot of the web page by injecting that script on the website. As a result, personal data or bank balance of victim can be easily exploited. Bind-XSS is one of the types of Persistent XSS attacks. The following steps explain the scenario of stored XSS attack:

Step 1: Attacker posts a message containing malicious script on a form/blog.

Step 2: The script gets stored in the server's database.

Step 3: Victim visits the webpage with malicious content and requests a service.

Step 4: The website displays the content containing the malicious code.

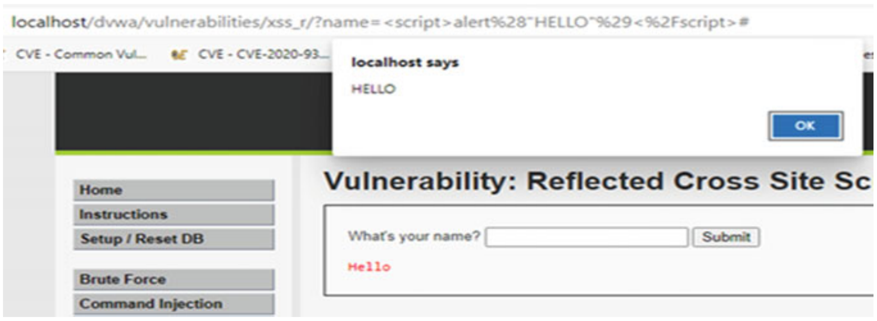
Step 5: Attacker gets complete control over the victim's system.

Non-Persistent XSS attack is also referred as type-II or reflected XSS attack where reflected means that the results of malicious query are visible to the attacker. The attacker crafts the malicious link in such a way that it appears to be from a trusted source. When the victim clicks on the malicious link, web server sends a response including the malicious script to the user. For example, the following figure demonstrates reflected XSS attack when a script query: <script> alert("HELLO") </script> is entered in the name box, it is reflected in the URL (Fig. 11).

DOM-based/type-0 XSS attack is a client-side vulnerability attack where DOM is abbreviated as Document Object Model. DOM is an object model for every HTML webpage. It includes the properties of the HTML page, which allows it to change its' content. So, when DOM-based XSS attack is executed, the JavaScript code is embedded in the client-side program, which allows it to modify the content of DOM and can also change the values of objects' properties, while the user visits the page without malicious link. Since the malicious code executes on victim's computer, server-side detection algorithm would fail to detect this type of attack.

Following points should be considered to prevent XSS attacks:

i. Execution of JavaScript code can be prevented by setting the cookie to HTTPOnly flag.


localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert%28"HELLO"%29<%2Fscript>#

localhost says

HELLO

Vulnerability: Reflected Cross Site Sc

What's your name?

Hello

Fig. 11 Example of reflected XSS vulnerability

ii. Invalid requests can be redirected.
iii. Simultaneous multiple logins to the same account must be detected and session must be declared invalid.
iv. Escaping schemes could be used.
v. Appropriate response header must be used.
vi. Detection should be done at both client side and server side.

3 Conclusıon and Future Scope

In conclusion, with the rise of internet technology, it has become crucial to protect one's data and privacy, where the hackers could use numerous online tools to catch just one vulnerability in website, and if found can put the users' integrity, confidentiality, and authenticity at risk. SQL injection, XSS attack, and broke authentication attacks could put users' privacy at risk. It is suggested to use the artificial intelligence-based detection method to detect a web vulnerability.

Since it has become extremely crucial to protect online resources from being exposed to hackers as new attacks are being carried out every day by hackers, web attacks could be defeated by integrating the detection and prevention techniques using machine learning algorithms.

References

1. Ami, P.V., and Malav S.C., Top five dangerous security risks over web application. Int. J. Emerg. Trends Technol. Comput. Sci. **2**(1), 41–43 (2013)
2. A.M. Shabut, K.T. Lwin, M.A. Hossain, Cyber attacks, countermeasures, and protection schemes—a state of the art survey, in *2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*, Chengdu (2016), pp. 37–44
3. K. Nirmal, B. Janet, R. Kumar, Web application vulnerabilities—the hacker's treasure, in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, Coimbatore (2018), pp. 58–62

4. Hackers target 1 Indian firm over 1,500 times a week, January (2020), https://www.livemint.com/technology/tech-news/hackers-target-1-indian-firm-over-1-500-times-a-week-11580011583037.html
5. 95% of HTTPS servers vulnerable to trivial MITM attacks, March (2016), https://news.netcraft.com/archives/2016/03/17/95-of-https-servers-vulnerable-to-trivial-mitm-attacks.html
6. OSWAP top ten security risks and vulnerabilities (2020), https://sucuri.net/guides/owasp-top-10-security-vulnerabilities
7. Y. Wang, D. Wang, W. Zhao, Y. Liu, Detecting SQL vulnerability attack based on the dynamic and static analysis technology, in *2015 IEEE 39th Annual Computer Software and Applications Conference, Taichung* (2015), pp. 604–607
8. SQL injection, March (2020), https://en.wikipedia.org/wiki/SQL_injection
9. S.A. Faker, M.A. Muslim, H.S. Dachlan, A systematic literature review on sql injection attacks techniques and common exploited vulnerabilities. Int. J. Comput. Eng. Inf. Technol. **9**, 284–291 (2017)
10. L. Ma, D. Zhao, Y. Gao, C. Zhao, Research on SQL ınjection attack and prevention technology based on web, in *2019 International Conference on Computer Network, Electronic and Automation (ICCNEA)*, Xi'an, China (2019), pp. 176–179
11. CWE: common weakness enumeration, https://cwe.mitre.org/data/definitions/173.html
12. J. Ombagi, Time-based blind SQL ınjection via HTTP headers: fuzzing and exploitation, in *Conference: 2017 Strathmore Research Symposium*, At Nairobi, Kenya (2017)
13. A.K. Dalai, S.K. Jena, Neutralizing SQL ınjection attack using server side code modification in web applications. Secur. Commun. Netw. Hindawi **2017**, 12 pages (2017)
14. O.B. Al-Khurafi, M.A. Al-Ahmad, Survey of web application vulnerability attacks, in *2015 4th International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, Kuala Lumpur (2015), pp. 154–158
15. M.M. Hassan, S. Nipa, M. Akter, R. Haque, F. Deepa, M.M. Rahman, M.A. Siddiqui, M.H. Sharif, Broken authentication and session management vulnerability: a case study of web application. Int. J. Simul. Syst. Sci. Technol. **19**, 6.1–6.11 (2018)
16. G.E. Rodríguez, J.G. Torres, P. Flores, D.E. Benavides, Cross-site scripting (XSS) attacks and mitigation: a survey. Comput. Netw. J. Elsevier **166**, 22 pages (2020)
17. A. Shrivastava, S. Choudhary, A. Kumar, XSS vulnerability assessment and prevention in web application, in *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, Dehradun (2016), pp. 850–853
18. M. Liu, B. Zhang, W. Chen, X. Zhang, A survey of exploitation and detection methods of XSS vulnerabilities. IEEE Access **7**, 182004–182016 (2019)
19. Sunardi, I. Riadi, P.A. Raharja, Vulnerability analysis of E-voting application using open web appli-cation security project (OWASP) framework. Int. J. Adv. Comput. Sci. Appl. (IJACSA) **10**(11) (2019)