

CSCI262

LAB 2 - SQL INJECTION & PASSWORD CRACKING

Description

This lab will instruct you on how the SQL engine is vulnerable to SQL code injection. You will know how to start the docker system to do the SQL lab. Then, you will study a dictionary attack, which is one of the brute-force attacking techniques. In this lab, we instruct you to implement dictionary attacks using python code. After doing this lab, you will know how the popular ethical hacking tools like SQLmap, Hydra, and Hashcat works.

Learning Outcomes

By doing this hands-on configuration, you should be able to:

1. Investigate how SQL injection attacks execute
2. Apply knowledge of SQL injection attacks to protect web database
3. Working with files in Linux system
4. Run the python code from Linux system
5. Implement Dictionary Attack
6. Finish 2 main tasks

SQL injection & Password Cracking

Table of Contents

Description	1
Learning Outcomes.....	1
1. Task 1 – SQL Injection.....	3
1.1. What is SQL Injection?.....	3
1.2. SQL Injection Based on 1=1 is always True	4
1.3. SQL Injection Based on ""="" is always True	5
1.4. Running docker files for SQL Injection tasks	6
Step 1: Turn off all running containers	7
Step 2: Change working directory to Desktop/Labsetup-SQLInjection	7
Step 3: Build and start docker files	7
Step 4: Opening the website: www.seed-server.com	8
TASK 1:.....	8
Stop the running container	9
2. Task 2 - Dictionary Attack	11
2.1. Terms	11
2.2. Dictionary attack using hashed	11
2.3. Hands-on Demonstration	15
TASK 2.....	20
2.4. Further Tasks	20
Reference:	21

Login Information:

We will use the following account to log in to the virtual machine:

- ✓ Account ID: **seed**
- ✓ Password : **dees**

SQL injection & Password Cracking

1. Task 1 – SQL Injection

Interact with the SQL database, with one command, we can get the data that we want. All we need is to create the SELECT command with the condition after the WHERE keyword. However, the SQL vulnerability is from this condition. Attackers will take advantage of this vulnerability to inject the SQL code to get the data. So, what is the SQL vulnerability? What is SQL injection? Why is it so dangerous? How to exploit SQL injection? And how to prevent this kind of attack.

All answers to these questions will be revealed in this task.

1.1. What is SQL Injection?

According to Cloudflare, *Structured Query Language (SQL*) Injection is a code injection technique used to modify or retrieve data from SQL databases.*

<https://www.cloudflare.com/en-gb/learning/security/threats/sql-injection/>

From the definition, we can understand that to execute an SQL injection attack, an attacker needs to insert a piece of code into the server to modify or get data from the SQL database.

Therefore, SQL injection attack has several features: (1) SQL injection is a code injection technique that might destroy your database; (2) SQL injection is one of the most common web hacking techniques; (3) SQL injection is the placement of malicious code in SQL statements, via web page input.

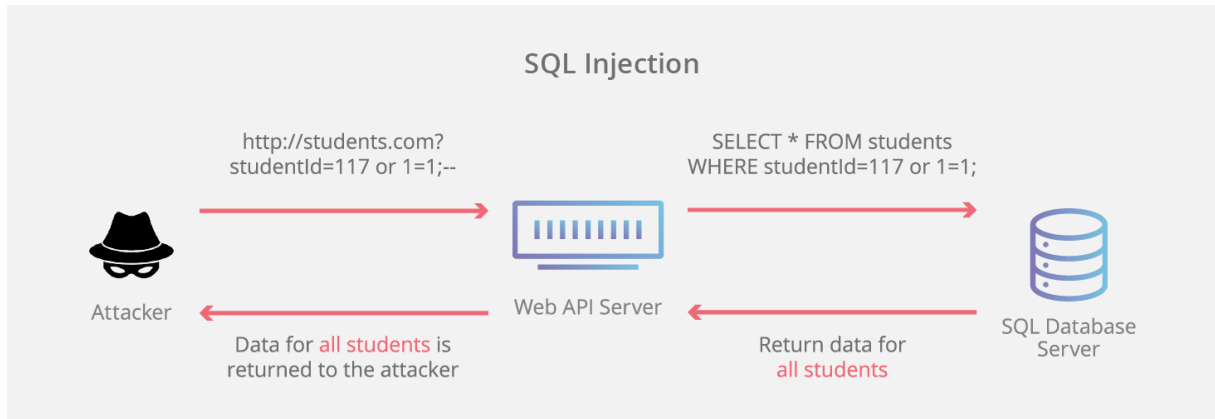
https://www.w3schools.com/sql/sql_injection.asp

According to OWASP, the SQL injection vulnerability is one of the most dangerous data confidentiality issues and integrity in web applications. It has been listed in the OWASP Top 10 list of the most common and widely exploited vulnerabilities since its inception. Read the SQL injection vulnerability history for a more detailed explanation of how the SQL Injection vulnerability originated. <https://www.netsparker.com/blog/web-security/sql-injection-vulnerability/>

SQL injection & Password Cracking

SQL in Web Pages

SQL injection usually occurs when the server asks a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement. This statement can trick the SQL database into getting data. Studying the SQL injection attack is the way to explore how to fool the SQL engine.



(Source: <https://www.cloudflare.com/en-gb/learning/security/threats/sql-injection/>)

Look at the following example, which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

Example

```
txtStdId = getRequestString("StudentId");  
txtSQL = "SELECT * FROM Students WHERE StudentId = " + txtStdId;
```

Now we will investigate two main ways to execute SQL injection attacks.

1.2. SQL Injection Based on 1=1 is always True

Look at the example above again. The code's original purpose was to create an SQL statement to select a student with a given student id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId: **117 OR 1=1**

SQL injection & Password Cracking

Then, the SQL statement will look like this:

```
SELECT * FROM Students WHERE StudentId = 117 OR 1=1;
```

The SQL above is valid and will return ALL rows from the "Students" table since OR 1=1 is always TRUE.

Does the example above look dangerous? What if the "Students" table contains names and passwords?

The SQL statement above is much the same as this:

```
SELECT StudentId, Name, Password FROM Students WHERE StudentId = 117 or 1=1;
```

A hacker might access all the user names and passwords in a database by inserting 117 OR 1=1 into the input field.

It does not matter it is 117 or other values. We all know that wherein the SQL statement is the signal telling that followed by a condition. SQL engine is the machine. It is not smart enough as a human to differentiate what exact user id and the true condition. SQL engine needs a true condition to execute the select command. It is the SQL vulnerability that attackers can exploit with any kinds of SQL database.

Now, you should be quite clear that to trick SQL engine, all the tasks attackers want to do is **make the condition true** after the WHERE keyword.

From the security perspective, the defender should think of how to monitor input so that attackers cannot trick the SQL engine by “making the condition true”

1.3. SQL Injection Based on ""="" is always True

Here is an example of user login on a website:

Username: John Doe

Password: myPass

Example

SQL injection & Password Cracking

```
uName = getQueryString("username");  
uPass = getQueryString("userpassword");  
sql = 'SELECT * FROM Users WHERE Name =' + uName + ' AND Pass =' + uPass + ''
```

Result

```
SELECT * FROM Users WHERE Name ="John Doe" AND Pass ="myPass"
```

A hacker might get access to user names and passwords in a database by merely inserting " OR ""=" into the user name or password text box:

User Name:

```
" or ""="
```

Password:

```
" or ""="
```

The code at the server will create a valid SQL statement like this:

Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

The SQL above is valid and will return all rows from the "Users" table, since OR ""="" is always TRUE

These above two techniques are basic methods to “make the condition true”.

There are many more exciting methods to execute SQL injections. I suggest you read this link to know almost all techniques to exploit SQL injection attacks with different kinds of database for your benefit.

<https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>

Now, we will practice with the cyberlab VM.

1.4. Running docker files for SQL Injection tasks

SQL injection & Password Cracking

To do the task, we need to access a website called www.seed-server.com. To build this website, we need to run the docker files in the directory Labsetup-SQLInjection on the Desktop.

Step 1: Turn off all running containers

Notes: before doing the tasks, please make sure that you have done these tasks in Lab 1. If you have not done it, please follow lab 1 instructions.

Step 2: Change working directory to Desktop/Labsetup-SQLInjection

Open a terminal window, then execute the command as follows.

```
seed@VM: ~/.../Labsetup-SQLInjection
[01/19/22] seed@VM:~$ cd Desktop/Labsetup-SQLInjection/
[01/19/22] seed@VM:~/.../Labsetup-SQLInjection$
```

Step 3: Build and start docker files

To build the docker files for SQL Injection tasks, we use the command **dcbuild**

```
[01/19/22] seed@VM:~/.../Labsetup-SQLInjection$ dcbuild
Building www
Step 1/5 : FROM handsonsecurity/seed-server:apache-php
--> 2365d0ed3ad9
Step 2/5 : ARG WWWDir=/var/www/SQL_Injection
--> Using cache
--> 2d421d753457
Step 3/5 : COPY Code $WWWDir
--> Using cache
--> bbff491c3663
Step 4/5 : COPY apache_sql_injection.conf /etc/apache2/sites-available
--> Using cache
--> e2d1b4a541e1
Step 5/5 : RUN a2ensite apache sql injection.conf

Step 7/7 : COPY sqllab_users.sql /docker-entrypoint-initdb.d
--> Using cache
--> 1dcbc1a128d0

Successfully built 1dcbc1a128d0
Successfully tagged seed-image-mysql-sqli:latest
[01/19/22] seed@VM:~/.../Labsetup-SQLInjection$
```

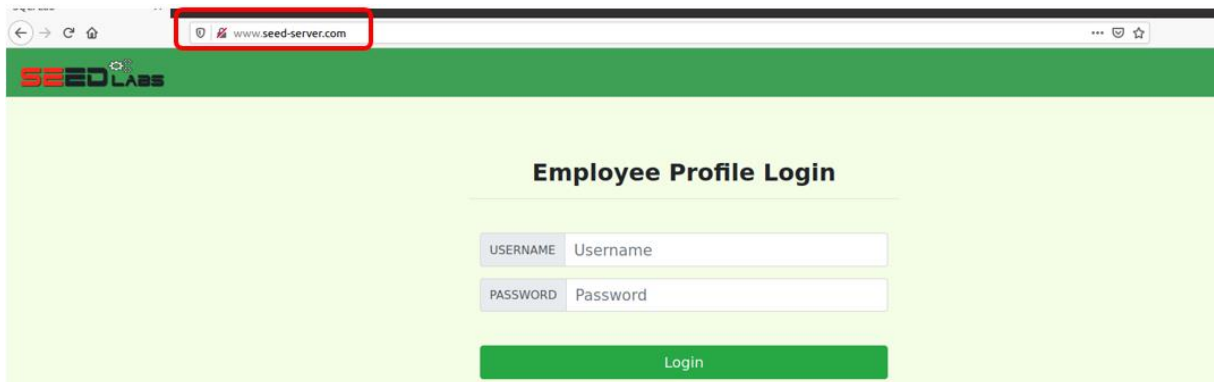
After building the docker file successfully, we need to start the container using **dcup** command.

SQL injection & Password Cracking

```
[01/19/22]seed@VM:~/../Labsetup-SQLInjection$ dcup
Starting 93e46667f003_mysql-10.9.0.6 ... done
Starting 63b074f5cbd6_www-10.9.0.5 ... done
Attaching to 93e46667f003_mysql-10.9.0.6, 63b074f5cbd6_www-10.9.0.5
93e46667f003_mysql-10.9.0.6 | 2022-01-19 12:41:06+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1de
bian10 started.
93e46667f003_mysql-10.9.0.6 | 2022-01-19 12:41:07+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
93e46667f003_mysql-10.9.0.6 | 2022-01-19 12:41:07+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1de
bian10 started.
63b074f5cbd6_www-10.9.0.5 | * Starting Apache httpd web server apache2 AH00558: apache2: Could not reliably deter
mine the server's fully qualified domain name, using 10.9.0.5. Set the 'ServerName' directive globally to suppress this me
ssage
63b074f5cbd6_www-10.9.0.5 | *
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.426074Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.22)
starting as process 1
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.452032Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has starte
d.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.720948Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.865307Z 0 [System] [MY-011323] [Server] X Plugin ready for connections.
Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqld.sock
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.943574Z 0 [Warning] [MY-013414] [Server] Server SSL certificate doesn't
verify: certificate is not yet valid
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.944138Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self s
igned.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.944488Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to
support TLS. Encrypted connections are now supported for this channel.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.946388Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pi
d-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.986218Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for conn
nections. Version: '8.0.22' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.
```

Step 4: Opening the website: www.seed-server.com

We need to open Firefox to access this website.



Now, we have successfully set up the docker file for running the www.seed-server.com website. Let start practicing SQL Injection tasks below.

TASK 1:

Task 1.1: Check if the database is built with MySQL or not to know which syntax should be appropriate for attacking.

SQL injection & Password Cracking

Hint: Verify the database type using a special character(s) to be part of possible database types' syntax. Reminding that SQL, MySQL, MariaDB... using single quotation (') or double quotations (") in the command syntax.

Task 1.2: Attempt to log in to the system with known username without knowing its password or unknown username and password

Hint: log in with an unknown username and password that always make a true Select SQL statement

Task 1.3: Edit a user's salary and nickname to 80,000 and ABC with known username: Alice and salary field named 'Salary'

Hint: editing "salary" field of a user's profile using the Update SQL statement.

Task 1.4: Retrieve information of the user with salary > 80000.

Hint: log in with the required condition using the Select SQL statement.

NOTE: For each sub-task done, please take the screenshot give your explanation and observation, showing that you complete the sub-task successfully.

After finishing all the sub-tasks, please stop all running dockers by doing the guide below:

Stop the running container

Go to the terminal window where the docker files of SQL Injection are running, press **Ctrl + c** to stop the running process

SQL injection & Password Cracking

```

[01/19/22]seed@VM: ~/.../Labsetup-SQLInjection$ dcup
Starting 93e46667f003_mysql-10.9.0.6 ... done
Starting 63b074f5cbd6_www-10.9.0.5 ... done
Attaching to 93e46667f003_mysql-10.9.0.6, 63b074f5cbd6_www-10.9.0.5
93e46667f003_mysql-10.9.0.6 | 2022-01-19 12:41:06+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1de
bian10 started.
93e46667f003_mysql-10.9.0.6 | 2022-01-19 12:41:07+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
93e46667f003_mysql-10.9.0.6 | 2022-01-19 12:41:07+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1de
bian10 started.
63b074f5cbd6_www-10.9.0.5 | * Starting Apache httpd web server apache2          AH00558: apache2: Could not reliably deter
mine the server's fully qualified domain name, using 10.9.0.5. Set the 'ServerName' directive globally to suppress this me
ssage
63b074f5cbd6_www-10.9.0.5 | *
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.426074Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.22)
starting as process 1
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.452032Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has starte
d.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.720948Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.865307Z 0 [System] [MY-011323] [Server] X Plugin ready for connections.
Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqld.sock
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.943574Z 0 [Warning] [MY-013414] [Server] Server SSL certificate doesn't
verify: certificate is not yet valid
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.944138Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self s
igned.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.944488Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to
support TLS. Encrypted connections are now supported for this channel.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.946388Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pi
d-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.986218Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for conn
ections. Version: '8.0.22' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.

```

Press **Ctrl + c**, you can see the result as follows

```

igned.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.944488Z 0 [System] [MY-013602] [Server] Channel mysql_main co
support TLS. Encrypted connections are now supported for this channel.
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.946388Z 0 [Warning] [MY-011810] [Server] Insecure configurati
d-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different dir
93e46667f003_mysql-10.9.0.6 | 2022-01-19T12:41:07.986218Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: rea
ections. Version: '8.0.22' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.
^CGracefully stopping... (press Ctrl+C again to force)
Stopping 93e46667f003_mysql-10.9.0.6 ... done
Stopping 63b074f5cbd6_www-10.9.0.5 ... done
[01/19/22]seed@VM: ~/.../Labsetup-SQLInjection$ █

```

BONUS task: Write the code from the back-end server to prevent all SQL injection attacks. Hint: please refer to the link to the SEED project from reference (end of this document) to have some suggestions.

2. Task 2 - Dictionary Attack

A dictionary attack is one of the brute-force attacking techniques. In this lab, we instruct you on how to implement dictionary attacks by using python code. You will be working with files in Linux system, run the python code from cyberlab VM, and implement Dictionary Attack.

2.1. Terms

Before exploring the dictionary attack, we go through several related concepts.

Password is a string of characters used to verify the identity of a user during the authentication process. <https://searchsecurity.techtarget.com/definition/password>

Password Cracking The process of attempting to guess or crack passwords to gain access to a computer system or network.

https://www.webopedia.com/TERM/P/password_cracking.html

A brute-force attack: is the process that an attacker submits many passwords or passphrases with the hope of eventually guessing correctly.

https://en.wikipedia.org/wiki/Brute-force_attack

2.2. Dictionary attack using hashed

The dictionary Attack method is one kind of brute-force attack. It also tries all possible guesses. However, your guesses here are stored in the dictionary file.

Problem description:

Imagine that you are an attacker. You have attacked one server and get the user database containing, looking like this:

SQL injection & Password Cracking

User	Password
Willy	0144e99a8cd56304aff370c875cbeceb5
Christ	646213f71947354898a96c0f78658fd6
Ngoc	892f46f3724a16427ab1916deb7cf9d1

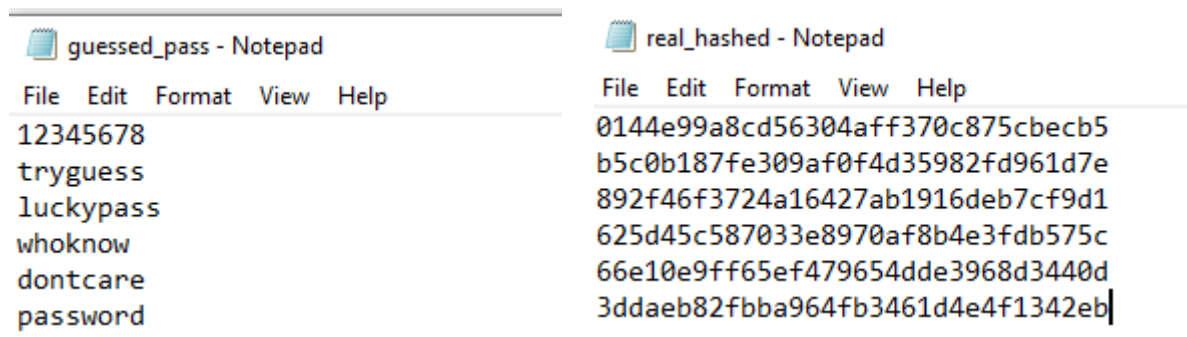
Password in MD5 hashes

If you know the plain text password, you can get access to the critical system. However, the password is not plain text. As you can see in the above picture, it is md5 hashed Hexa code. In the Linux system, you can see the MD5 of the user password in /etc/shadow file.

For dictionary attacks, the critical file that you have is the list of common passwords. You can see the below link to refer to the list of files containing the common password. Till now, there is a file that includes ten million passwords.

<https://github.com/danielmiessler/SecLists/tree/master/Passwords/Common-Credentials>

In this lab, we give you `guessed_pass.txt` and `real_hashed.txt` files for testing the lab. In reality, the hacker can use ten million passwords or more to attack. The guess and real hash files look like the below picture.



SQL injection & Password Cracking

Your task:

Write a Python program using the dictionary attack method to crack the real hashed code to find the plain text of the password.

Analyse:

The hashed code is popular now. It is not only stored in the company database but also transmitted over the Internet. When the attacker gets the hashed code for one user's password, it is time to crack the hashed code to plain text.

After getting the user database with hash code, directly thinking, you want to decrypt it to the plain text, then you can get access to the system. However, the answer is tough to decrypt this hash. There is another working-around way to do this. That is a dictionary attack. Instead of trying every plain text to guess the real password correctly, this method just compares the real hashed code with every hashed code from guessed plain password dictionary. A bit abstract, now we go to steps to do and code for this method. First, we call hashed code from the database we get is "real hashed code". We need to find the password of this real hashed code. We call it "real password".

Solution - 3 steps (Algorithm)

First, get the guessed plain password file. It contains the most popular password in plain text. Go to GitHub and get that file. Each line in this file is a password that people used it. We call this "guessed password" In this demonstration, we give this `guessed_pass.txt` file as above.

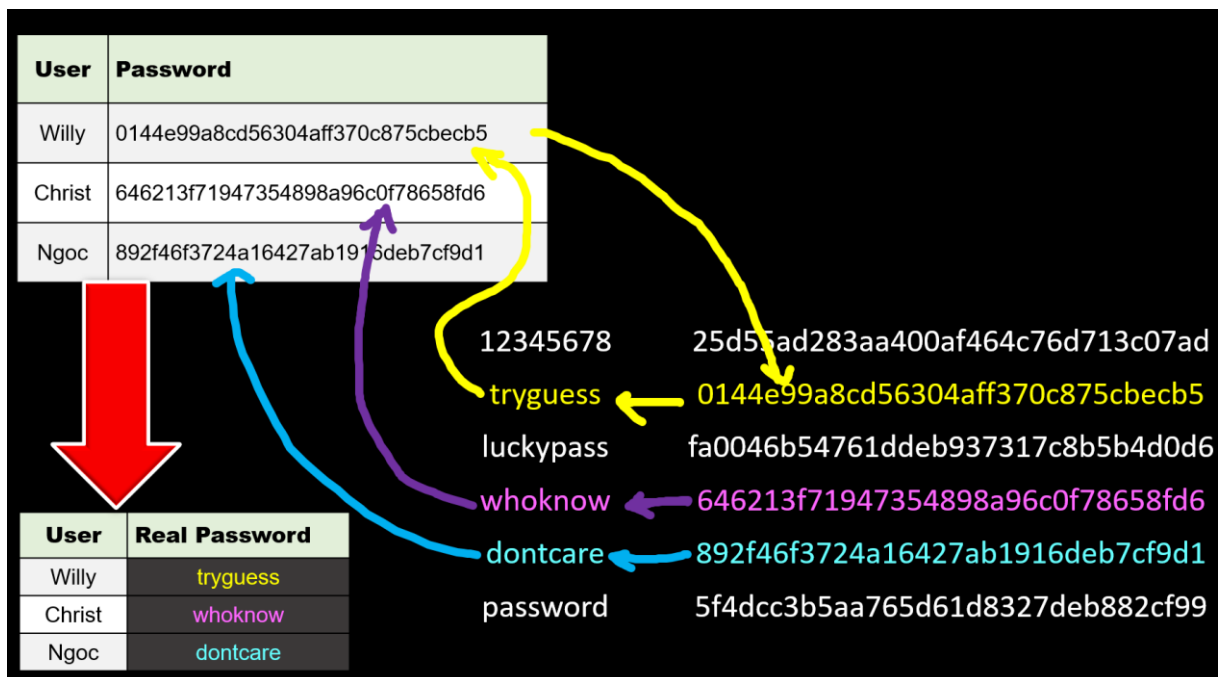
Second, transfer to hashed code. Each password will be altered to hashed code. We call this "guessed hashed code"

Third, this guessed hashed code will be compared to real hashed code that we need to crack. If it is the same, then stopping the program and the real password of the real hashed code is the guessed password of the guessed hashed code.

SQL injection & Password Cracking

For example, we have real hashed code like this "0144e99a8cd56304aff370c875cbeeb5". We need to find the real password of this hashed code. Now from the above file, we write the code to change each guessed password to hashed. Then comparing each hashed with "0144e99a8cd56304aff370c875cbeeb5". If it is the same, come back to find the guessed password make the same hashed with real hashed. In this case, the guessed password is tryguess. This is also the real password of the real hashed. It is simple but effective.

The below picture shows the process for testing each password



This is the code, we also provide this file to you:

```
import hashlib

flag = 0
# paste the tested hashed code in the double quotation
real_hash = ""
try:
    guessed_pass = open("guessed_pass.txt", "r")
except:
    print("Guessed password file is not found")
```

SQL injection & Password Cracking

```
quit()
for g_pass in guessed_pass:
    guessed_hash = hashlib.md5(g_pass.encode('utf-
8')).strip().hexdigest()
    if guessed_hash == real_hash:
        print("Congratulations!")
        print("The real password of md5 hash is found" + real_hash)
        print("It is " + g_pass)
        # flag to 1 if password we found in the list
        flag = 1
        break
# if no match from the file the flag still 0
if flag == 0:
    # password is not in the list
    print("The plain password is not found in the guessed password file")
```

2.3. Hands-on Demonstration

Step 1: Get the zip file (Task2_Dict_Att.zip) from the public cloud link below or from Moodle site of CSCI262, then unzip.

<https://cloudstor.aarnet.edu.au/plus/s/Xvc9WBvGGJKvKcp>

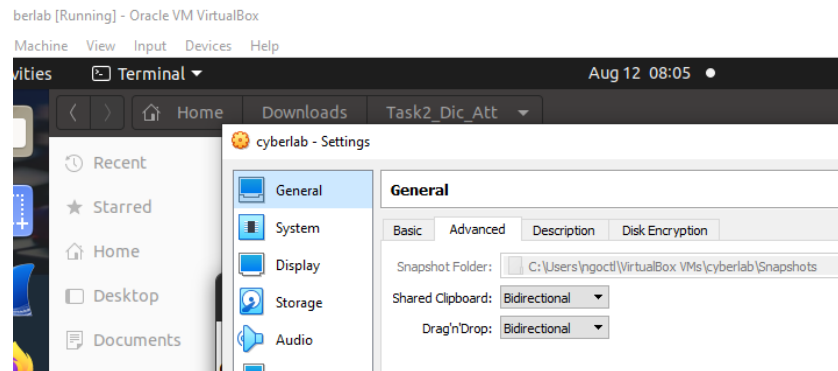
The folder provided from Moodle contains three files:

- crack.py contains the python code to crack password
- guessed_pass.txt contains the dictionary of plain guessed password
- real_hashed.txt contains the hashed password for testing

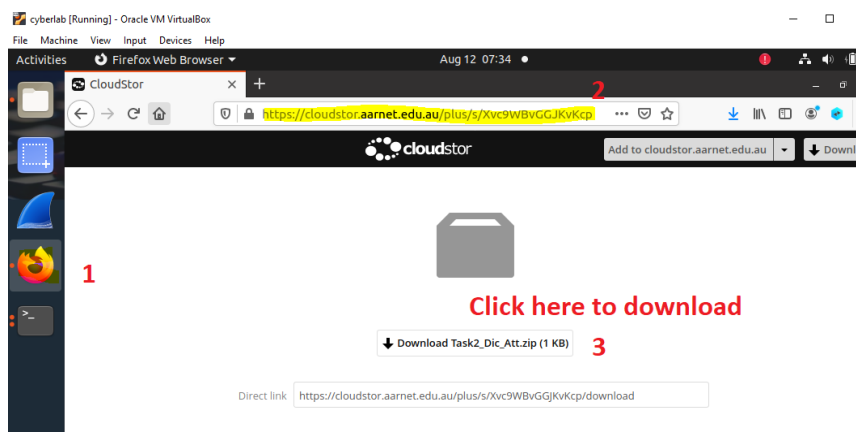
First, how to download the zip file from the above link.

From your cyberlab VM, open Firefox, then copy the above link and paste to the firefox (make sure your VM with bidirectional mode from the setting).

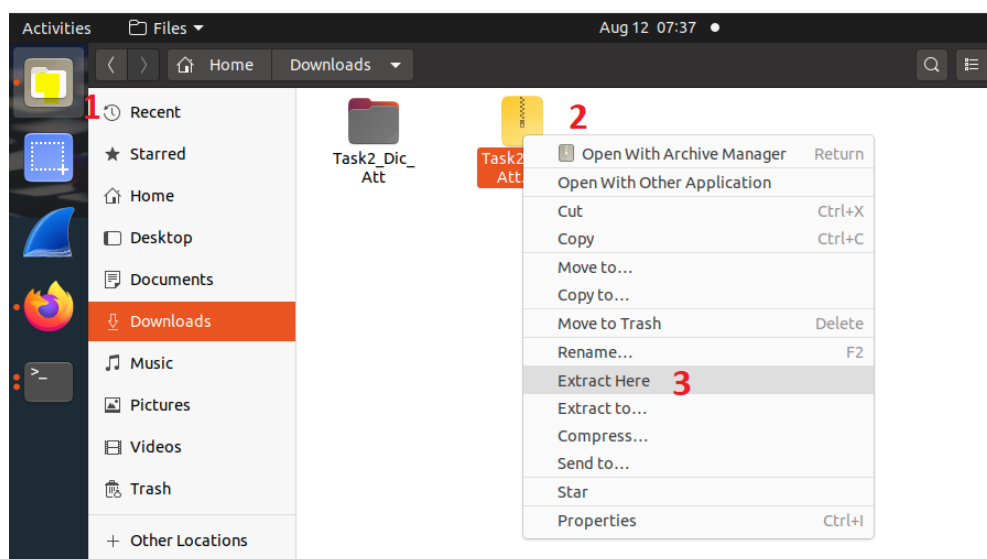
SQL injection & Password Cracking



Then, you will see a file to download from Firefox.



It will store in the download folder. Click on the first top left icon to navigate the download folder, then right click to the zip file and unzip it.

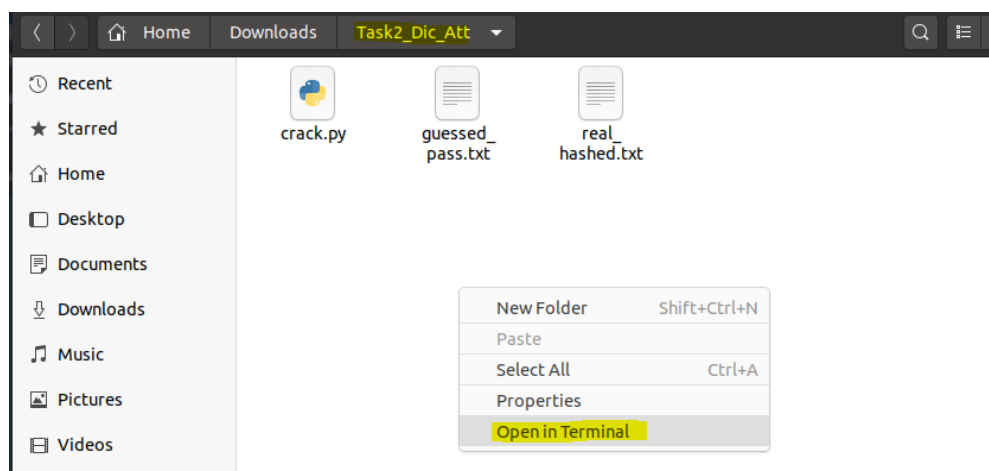


SQL injection & Password Cracking

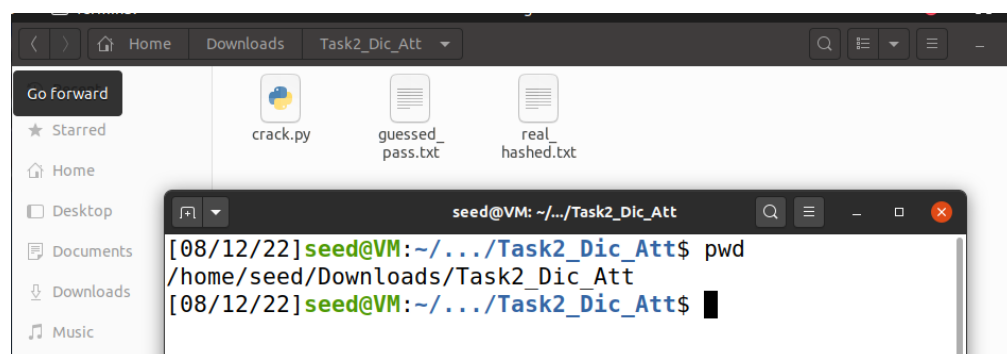
Step 2: Open the terminal from the folder.

One of the easy ways to navigate the working folder on Ubuntu is to open the terminal right from the folder.

First, open the unzip folder (Task2_Dic_Att), right-click on any empty space then select open terminal.



Ubuntu will open the terminal with the path to this directory, like below:



Now, you can practice dictionary attacks.

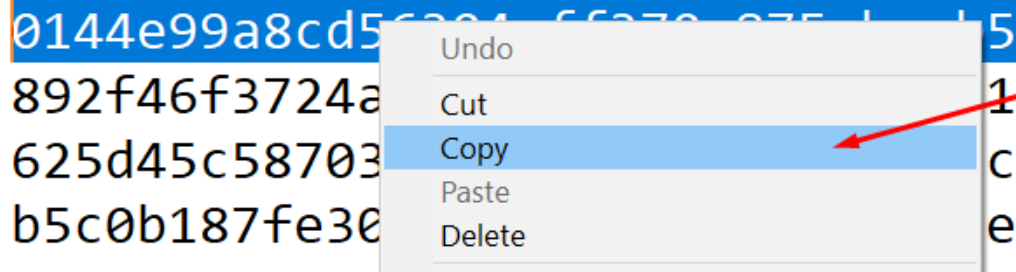
Step 3: Copy hashed code (from `real_hashed.txt`) to python code (to `crack.py`)

Open `real_hashed.txt` and copy the first line of hashed code

SQL injection & Password Cracking

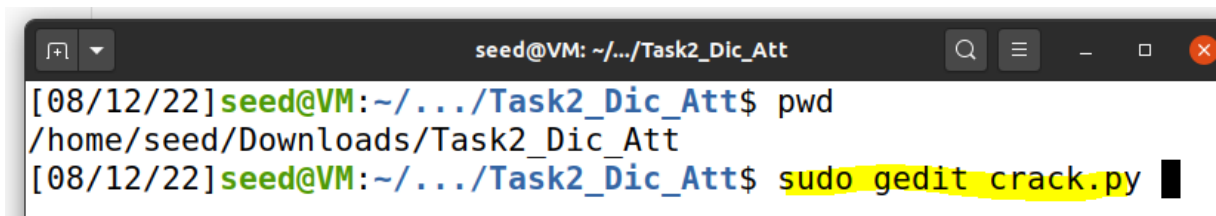
real_hashed.txt - Notepad

File Edit Format View Help




Open the code crack.py and paste the hashed code for real_hash variable

To do that, you can use **gedit** with **sudo** right. So you can write to the file.



```
seed@VM: ~/.../Task2_Dic_Att
[08/12/22] seed@VM:~/.../Task2_Dic_Att$ pwd
/home/seed/Downloads/Task2_Dic_Att
[08/12/22] seed@VM:~/.../Task2_Dic_Att$ sudo gedit crack.py
```

After open crack.py file, you paste the real_hashed code as the picture below, then save the file, then click “X” symbol to close the file.



```
1 import hashlib
2
3 flag = 0
4 # paste the tested hashed code in the double quotation
5 real_hash = "0144e99a8cd56304aff370c875cbecb5"
6 try:
7     guessed_pass = open("guessed_pass.txt", "r")
8 except:
9     print("Guessed password file is not found")
10    quit()
11 for g_pass in guessed_pass:
12     guessed_hash = hashlib.md5(g_pass.encode('utf-8').strip()).hexdigest()
13     if guessed_hash == real_hash:
14         print("Congratulations!")
15         print("The real password of md5 hash is found" + real_hash)
16         print("It is " + g_pass)
17         # flag to 1 if password we found in the list
18         flag = 1
19         break
20 # if no match from the file the flag still 0
21 if flag == 0:
22     # password is not in the list
23     print("The plain password/passphrase is not found in the guessed password file")
```

Comeback to the terminal and keep going

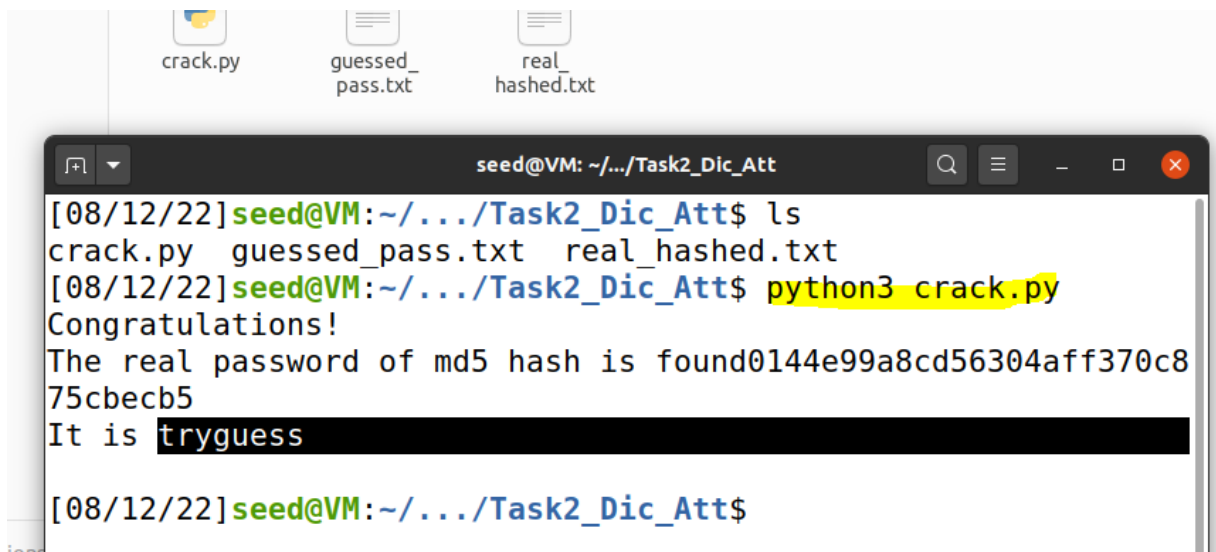
SQL injection & Password Cracking

Step 4: Run crack.py to test this hashed code match one of real passwords from guessed_pass.txt or not.

Using this command to run crack.py

```
python3 crack.py
```

Boom!, it is done. The password is “**tryguess**” as the below picture.



If the crack.py does not run, it may be the problem from executing the file.

So, you can use command chmod command to give the executable right for this python file

```
chmod +x crack.py
```

After that, you can run the python code again.

If the screen display “Congratulations”, you have found the real password.

SQL injection & Password Cracking

TASK 2

Test with the rest of real hashed passwords to find each one has been found or not. It should be from the second real hashed one from `real_hased.txt` file. You need test with one hashed to do the task.

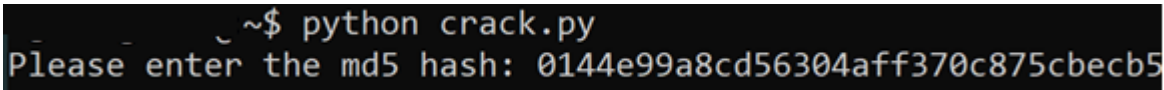
Take the screenshot of the final step (run `crack.py`), you can explain your observation.

2.4. Further Tasks

There are several ways to improve the above code through these tasks:

Task 1:

Instead of copy each real hashed code for testing, you write the code so you can input real hashed code from the command line like the below picture.



```
~$ python crack.py
Please enter the md5 hash: 0144e99a8cd56304aff370c875cbeeb5
```

Task 2:

You can write the code to read each real hashed in the real hashed password file. Then you process each line of real hashed.

Task 3:

You can count the time for each testing.

Task 4:

If you run with a file containing 10 million common passwords or even more, improve the compute is the problem. One suggestion is that you can make all

SQL injection & Password Cracking

your CPU cores involve in your computing to find the real password. For example, your CPU has 4 cores. So, you can order all 4 cores to compute parallelly.

Your task is to write the code to divide the task for each core CPU, counting the time or each core CPU the observe how well your solution is.

Enjoy your happy lab!

Reference:

1. https://seedsecuritylabs.org/Labs_20.04/Web/Web_SQL_Injection/
2. <https://github.com/danielmiessler/SecLists/tree/master/Passwords/Common-Credentials>