



TAREA 02

**DETERMINACIÓN DEL
NIVEL SEGURIDAD DE
REQUERIDO POR
APLICACIONES**

PUESTA EN PRODUCCIÓN SEGURA

ALBA MOREJÓN GARCÍA

2024/2025

CETI - Ciberseguridad en Entornos de las Tecnologías de la Información

Caso práctico

Julián está preocupado porque necesita crear un aplicativo web que interactuara con una base de datos y sabe que uno de los principales riesgos es la entrada de datos de usuarios ya que podrían provocar acceso a información que no debiera de la base de datos. Ha buscado información sobre vulnerabilidades de este tipo y ha decidido que debe aprender cómo desarrollar de forma más segura.

Para ello crea el entorno de pruebas Damn Vulnerable Web Application

(<https://github.com/digininja/DVWA>) donde podrá configurar distintos niveles de seguridad, ver cómo se refleja en el código y probar distintos tipos de ataque y ver de qué manera se comporta el aplicativo y sobre todo ver que recibe la base de datos.

Va a probar ataques de tipo Injection o inyección que es uno de los principales riesgos identificados en OWASP.

Apartado 1: Responde a las siguientes cuestiones

Vulnerabilidad CVE-2023-39417

- Breve descripción e impacto

La vulnerabilidad CVE-2023-39417 es un tipo de inyección SQL en una base de datos PostgreSQL. Esta vulnerabilidad fue descubierta y reportada en agosto del 2023, permite a un atacante ejecutar código SQL no autorizado en una base de datos, lo que puede llevar a una extracción de datos sensibles, modificación de datos y ejecución de código no autorizado.

Se encuentra en ciertas extensiones no empaquetadas, de confianza que se pueden instalar en las bases de datos de tipo PostgreSQL. Estas extensiones contienen expresiones vulnerables como @extowner@ o @extschema@, que no están adecuadamente protegidas dentro de la base de datos. Un atacante puede aprovechar esta vulnerabilidad si tiene permisos para crear objetos en la base de datos, utilizando estas expresiones vulnerables dentro de una estructura especial de comillas (' , “ o \$) para engañar al sistema y ejecutar comandos no autorizados esto le permite al atacante controlar la base de datos y ejecutar código con privilegios elevados.

- Productos y versiones vulnerables

PostgreSQL 11, versiones de la 11.0 a la 11.21

PostgreSQL 12, versiones de la 12.0 a la 12.16

PostgreSQL 13, versiones de la 13.0 a la 13.12

PostgreSQL 14, versiones de la 14.0 a la 14.9

PostgreSQL 15, versiones de la 15.0 a la 15.4

También se identificaron vulnerabilidades en sistemas basados en PostgreSQL como Red Hat Enterprise Linux y Debian.

- Posibles soluciones

Para mitigar la vulnerabilidad CVE-2023-39417, se recomienda:

- Actualizar PostgreSQL a una de las versiones no vulnerables. Las versiones seguras son 11.21, 12.16, 13.12, 14.9, 15.4 en adelante.
- Revisar y eliminar las extensiones no empaquetadas que puedan estar afectadas.
- Revisar y limitar los permisos de los usuarios de la base de datos para evitar que atacantes con privilegios “CREATE” puedan explotar esta vulnerabilidad.

Requisito ASVS v4.0.3-5.3.4 (ASVS v4.0.3, cap 5, apartado 3, requisito 4)

- Breve descripción

El requisito ASVS v4.0.3-5.3.4 del OWASP Application Security Verification Standard (ASVS), asegura que todos los flujos de lógica de negocio de alto valor con la autenticación, la gestión de sesiones y el control de acceso, sean seguros y resistentes a condiciones de carrera (time of check o time of use). Esto significa que las aplicaciones deben ser diseñadas para manejar correctamente las operaciones concurrentes y evitar que los atacantes puedan explotar estas vulnerabilidades. Las condiciones de carrera ocurren cuando múltiples procesos intentan acceder a un recurso compartido al mismo tiempo y la ejecución de estos procesos no está

adecuadamente sincronizada, esto puede llevar a resultados inesperados y peligrosos ya que el orden en el que se ejecutan las operaciones puede afectar al comportamiento del sistema.

- Niveles a los que debe aplicarse

Este requisito debe aplicarse a todos los niveles de verificación definidos por el OWASP:

N1 (básico)

N2 (intermedio)

N3 (avanzado)

Las condiciones de carreras son vulnerabilidades que pueden existir en cualquier tipo de aplicación independientemente de su complejidad o nivel de seguridad requerido. Si no se verifica y maneja adecuadamente estas vulnerabilidades pueden permitir a los atacantes explotar fallos en la sincronización de procesos, lo que puede llevar a accesos no autorizados, pérdida de datos y otros problemas de seguridad. Por esta razón, es vital que la verificación de resistencia a condiciones de carrera se aplique a todos los niveles asegurando que todas las aplicaciones estén protegidas.

- Categoría CWE

CWE (Common Weakness Enumeration), es una lista categorizada de debilidades de seguridad conocidas que pueden ser explotadas por atacantes para comprometer la seguridad de un sistema, el requisito ASVS está directamente relacionado con la categoría CWE-362 (Race Condition).

La condición de carrera ocurre cuando la salida de un proceso depende de la secuencia o el tiempo de otros eventos, es decir, cuando múltiples procesos acceden a un recurso compartido de manera no sincronizada, puede resultar en comportamiento inesperado o vulnerable. Esta debilidad puede ser explotada por atacantes para ejecutar acciones no autorizadas.

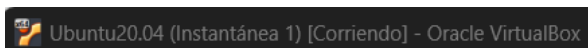
El requisito de ASVS se centra en asegurar que las aplicaciones web no sean susceptibles a estas condiciones de carrera, especialmente en flujos de lógica de negocio críticos como la autenticación, la gestión de sesiones y el control de acceso. A través de prácticas de diseño y verificación robusta se busca evitar que las condiciones de carrera introduzcan vulnerabilidades explotables.

Apartado 2: Crea el entorno de pruebas

Hoy en día la mayoría de los equipos de desarrollo utilizan contenedores para montar los entornos de desarrollo. En la página web del proyecto de DVWA (<https://github.com/digininja/DVWA>) explican como instalar la aplicación con Docker, que es una de las herramientas más habituales para trabajar con contenedores. Sigue las siguientes instrucciones:

- **Instalar Docker y si es necesario docker compose (en algunas versiones ya viene con docker). En la página oficial de Docker explican cómo instalarlo**
- **Descargar el proyecto DVWA y descomprimirlo**
- **Abrir una terminal y cambiar al directorio donde se ha descomprimido.**
- **Ejecutar la línea de comandos: docker compose up -d**
- **Una vez instalado, para acceder a DVWA: Abrir un navegador con la dirección <http://localhost:4280>, el usuario es admin y la clave es admin**
- **La primera vez que se abre la aplicación es necesario pulsar en el botón Create/Reset Database**
- **A partir de ese momento el usuario es admin y la clave es password**
- **Para cambiar el nivel de seguridad debes ir a DVWA Security**

Vamos a utilizar una máquina Ubuntu 20.04

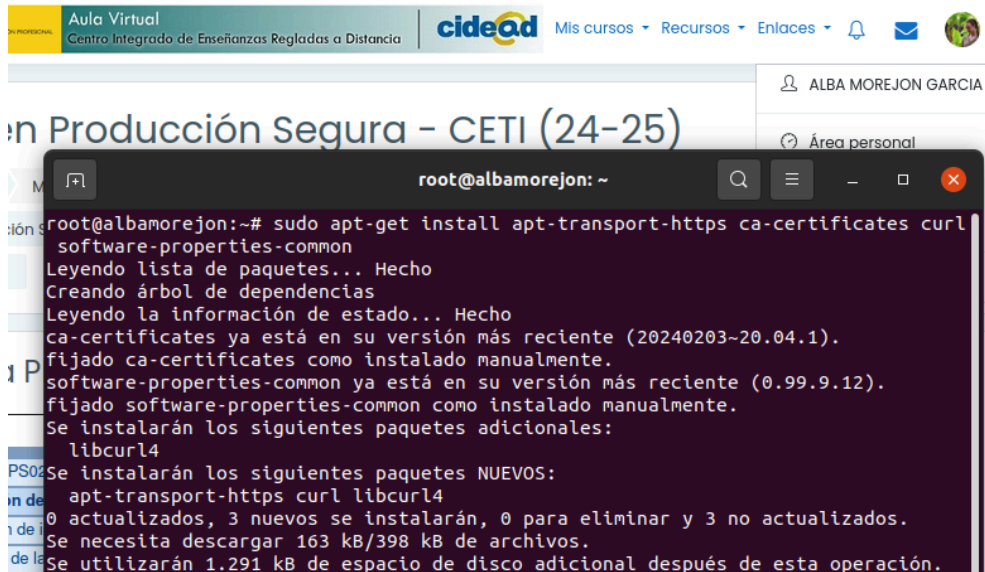


- **Instalar Docker** y si es necesario docker compose (en algunas versiones ya viene con docker).
En la página oficial de Docker explican cómo instalarlo.

1. Configuramos el repositorio

Instalamos paquetes necesarios con el comando

”sudo apt-get install apt-transport-https ca-certificates curl software-properties-common”.



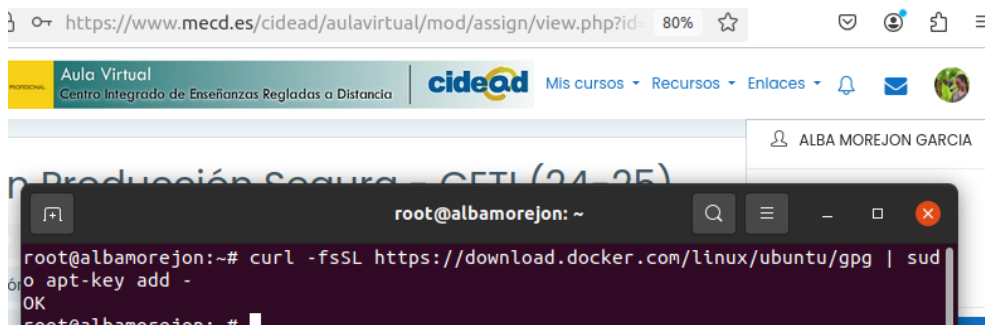
```

root@albamorejon: ~
root@albamorejon:~# sudo apt-get install apt-transport-https ca-certificates curl
software-properties-common
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
ca-certificates ya está en su versión más reciente (20240203~20.04.1).
fijado ca-certificates como instalado manualmente.
software-properties-common ya está en su versión más reciente (0.99.9.12).
fijado software-properties-common como instalado manualmente.
Se instalarán los siguientes paquetes adicionales:
libcurl4
Se instalarán los siguientes paquetes NUEVOS:
apt-transport-https curl libcurl4
0 actualizados, 3 nuevos se instalarán, 0 para eliminar y 3 no actualizados.
Se necesita descargar 163 kB/398 kB de archivos.
Se utilizarán 1.291 kB de espacio de disco adicional después de esta operación.

```

Agregamos clave gpg de docker

“curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo apt-key add -”.



```

root@albamorejon: ~
root@albamorejon:~# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
OK
root@albamorejon:~#

```

Agregamos el repositorio de docker a APT

“sudo add-apt-repository “deb [arch=amd64] <https://download.docker.com/linux/ubuntu> \$(lsb_release -cs) stable”.



```

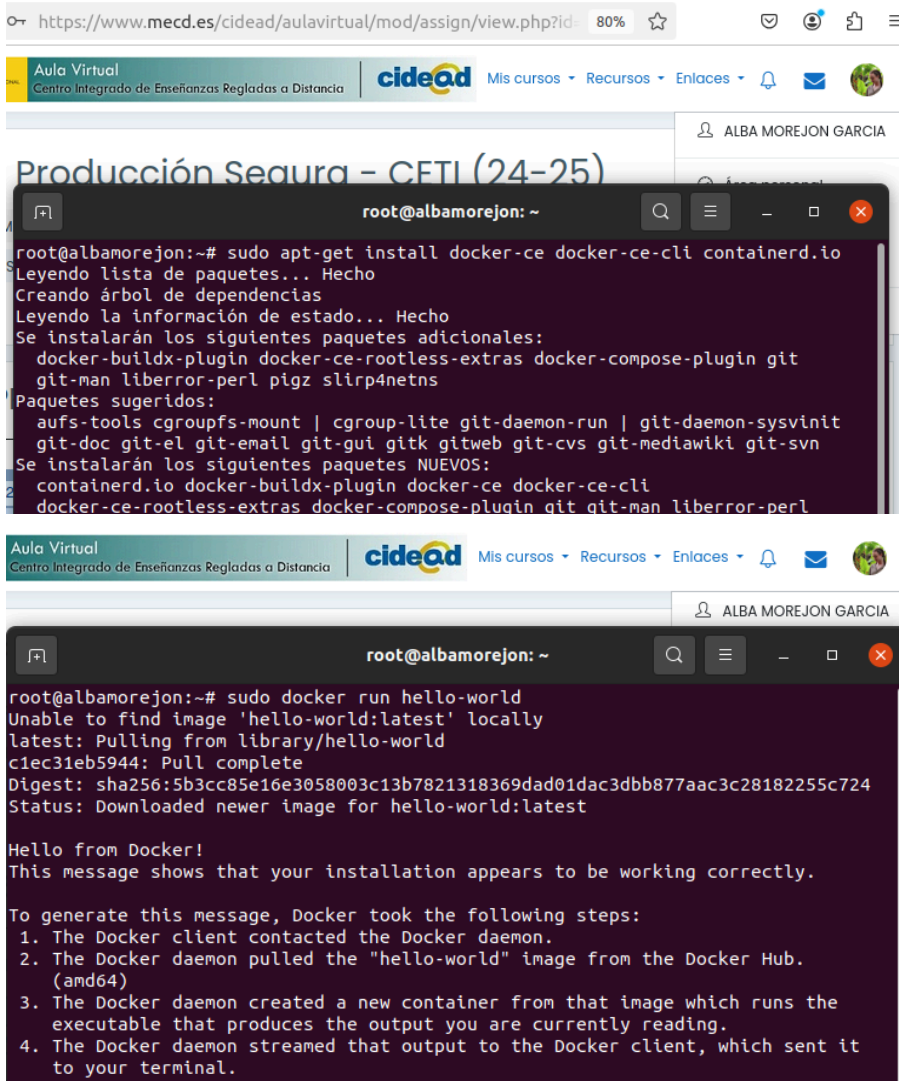
root@albamorejon: ~
root@albamorejon:~# sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
Obj:1 http://es.archive.ubuntu.com/ubuntu focal InRelease
Obj:2 http://es.archive.ubuntu.com/ubuntu focal-updates InRelease
Obj:3 http://es.archive.ubuntu.com/ubuntu focal-backports InRelease
Obj:4 http://security.ubuntu.com/ubuntu focal-security InRelease
Des:5 https://download.docker.com/linux/ubuntu focal InRelease [57,7 kB]
Des:6 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [53,3

```

2. Instalar Docker

Hacemos “sudo apt update”.

Instalamos Docker Engine “sudo apt-get install docker-ce docker-ce-cli containerd.io” y verificamos que este instalado con “sudo docker run hello-world”.



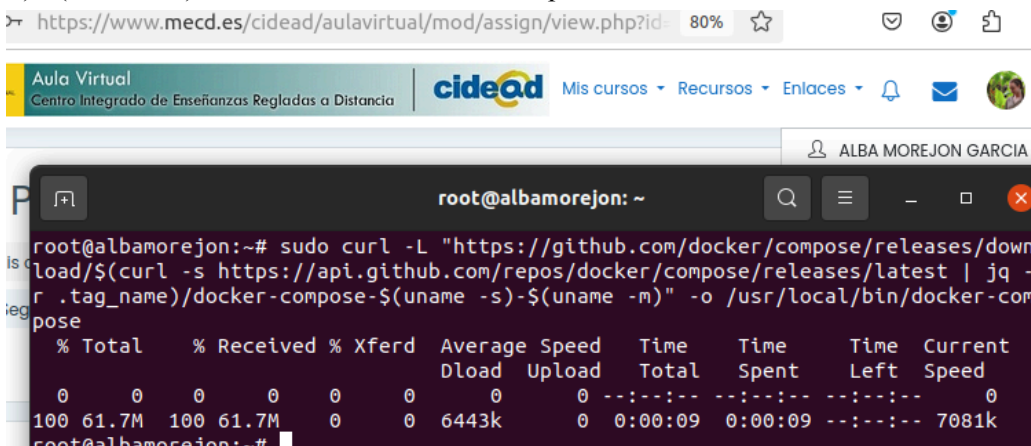
```
root@albamorejon: ~  
root@albamorejon:~# sudo apt-get install docker-ce docker-ce-cli containerd.io  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes adicionales:  
  docker-buildx-plugin docker-ce-rootless-extras docker-compose-plugin git  
  git-man liberror-perl pigz slurp4netns  
Paquetes sugeridos:  
  aufs-tools cgroupfs-mount | cgroup-lite git-daemon-run | git-daemon-sysvinit  
  git-doc git-el git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn  
Se instalarán los siguientes paquetes NUEVOS:  
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli  
  docker-ce-rootless-extras docker-compose-plugin git git-man liberror-perl  
  
root@albamorejon:~# sudo docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
c1ec31eb5944: Pull complete  
Digest: sha256:5b3cc85e16e3058003c13b7821318369dad01dac3d8bb877aac3c28182255c724  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
   (amd64)  
3. The Docker daemon created a new container from that image which runs the  
   executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
   to your terminal.
```

3. Instalar Docker Compose

Instalamos primero el paquete jq con el comando “sudo apt install jq”.

Descargamos Docker Compose con el comando

“sudo curl -L “https://github.com/docker/compose/releases/download/\$(curl -s https://api.github.com/repos/docker/compose/releases/latest | jq -r .tag_name)/docker-compose-\$(uname -s)-\$(uname -m)” -o /usr/local/bin/docker-compose”.



```
root@albamorejon: ~  
root@albamorejon:~# sudo curl -L "https://github.com/docker/compose/releases/download/$(curl -s https://api.github.com/repos/docker/compose/releases/latest | jq -r .tag_name)/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current  
             Dload  Upload  Total   Spent    Left     Speed  
0 0 0 0 0 0 0 0 0:00:00 0:00:00 0:00:00 0  
100 61.7M 100 61.7M 0 0 6443k 0 0:00:09 0:00:09 --:--:-- 7081k  
root@albamorejon:~#
```


Damos permisos de ejecución “sudo chmod +x /usr/local/bin/docker-compose” y verificamos la versión.

```
Aula Virtual Centro Integrado de Enseñanzas Regladas a Distancia cidead Mis cursos Recursos Enlaces ALBA MOREJON GARCIA
root@albamorejon: ~
root@albamorejon:~# sudo chmod +x /usr/local/bin/docker-compose
root@albamorejon:~# docker-compose --version
Docker Compose version v2.32.2
root@albamorejon:~#
```

- **Descargar el proyecto DVWA y descomprimirlo.**

Procedemos a descargar el archivo .zip de github (<https://github.com/digininja/DVWA>) y descomprimirlo en la carpeta DVWA.

```
Aula Virtual Centro Integrado de Enseñanzas Regladas a Distancia cidead Mis cursos Recursos Enlaces ALBA MOREJON GARCIA
Producción Segura – CETI (24-25)
root@albamorejon: /home/albamorejon/Documentos/DVWA-master# ls
about.php      dvwa           phpinfo.php    README.md      security.php
CHANGELOG.md   external       php.ini        README.pl.md   security.txt
compose.yml    favicon.ico    README.ar.md   README.pt.md   setup.php
config         hackable       README.es.md   README.tr.md   tests
COPYING.txt    index.php      README.fa.md   README.vi.md   vulnerabilities
database       instructions.php README.fr.md   README.zh.md
Dockerfile     login.php      README.id.md   robots.txt
docs           logout.php     README.ko.md   SECURITY.md
root@albamorejon: /home/albamorejon/Documentos/DVWA-master#
```

- **Ejecutar la línea de comandos: docker compose up -d**

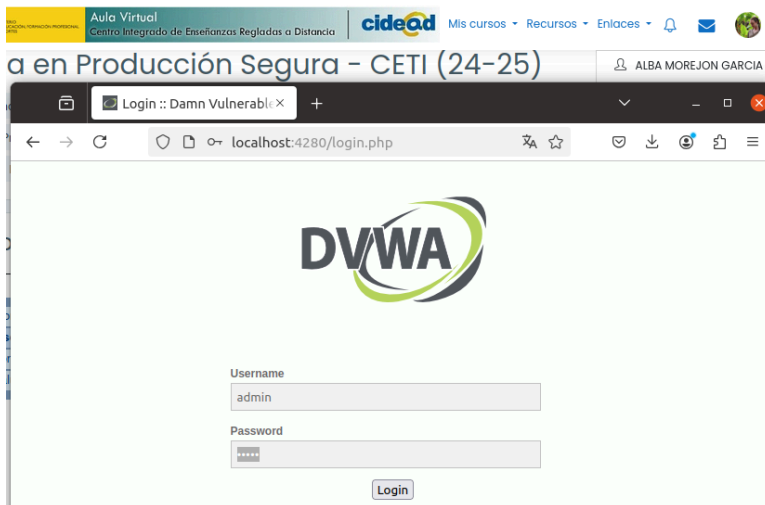
Ejecutamos “docker-compose up -d” que crea y contenedores para el servicio web y el servicio db.

```
Aula Virtual Centro Integrado de Enseñanzas Regladas a Distancia cidead Mis cursos Recursos Enlaces ALBA MOREJON GARCIA
Producción Segura – CETI (24-25)
root@albamorejon: /home/albamorejon/Documentos/DVWA-master# docker-compose up -d
[+] Running 8/19
  dvwa [██████████] Pulling                               17.4s
  db [██████████] Pulling                                 17.4s

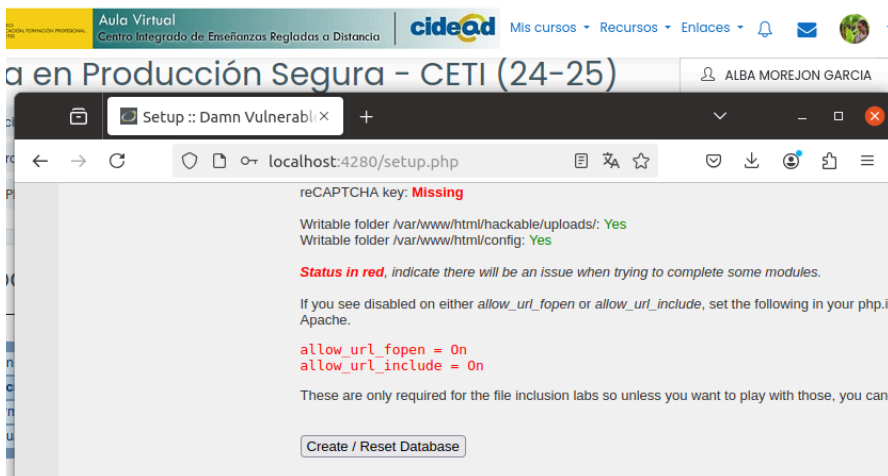
Aula Virtual Centro Integrado de Enseñanzas Regladas a Distancia cidead Mis cursos Recursos Enlaces ALBA MOREJON GARCIA
roducción Segura – CETI (24-25)
root@albamorejon: /home/albamorejon/Documentos/DVWA-master#
[+] Running 4/4
✓ Network dvwa-master_dvwa      Created           1.2s
✓ Volume "dvwa-master_dvwa"     Created           0.2s
✓ Container dvwa-master-db-1    Started           6.2s
✓ Container dvwa-master-dvwa-1  Started           5.6s
```

- Una vez instalado, para acceder a DVWA: Abrir un navegador con la dirección **http://localhost:4280**, el usuario es **admin** y la clave es **admin**

Copiamos la dirección “http://localhost:4280” en un navegador.
admin:admin



- La primera vez que se abre la aplicación es necesario pulsar en el botón **Create/Reset Database**
- Pulsamos en el botón **Create / Reset Database**

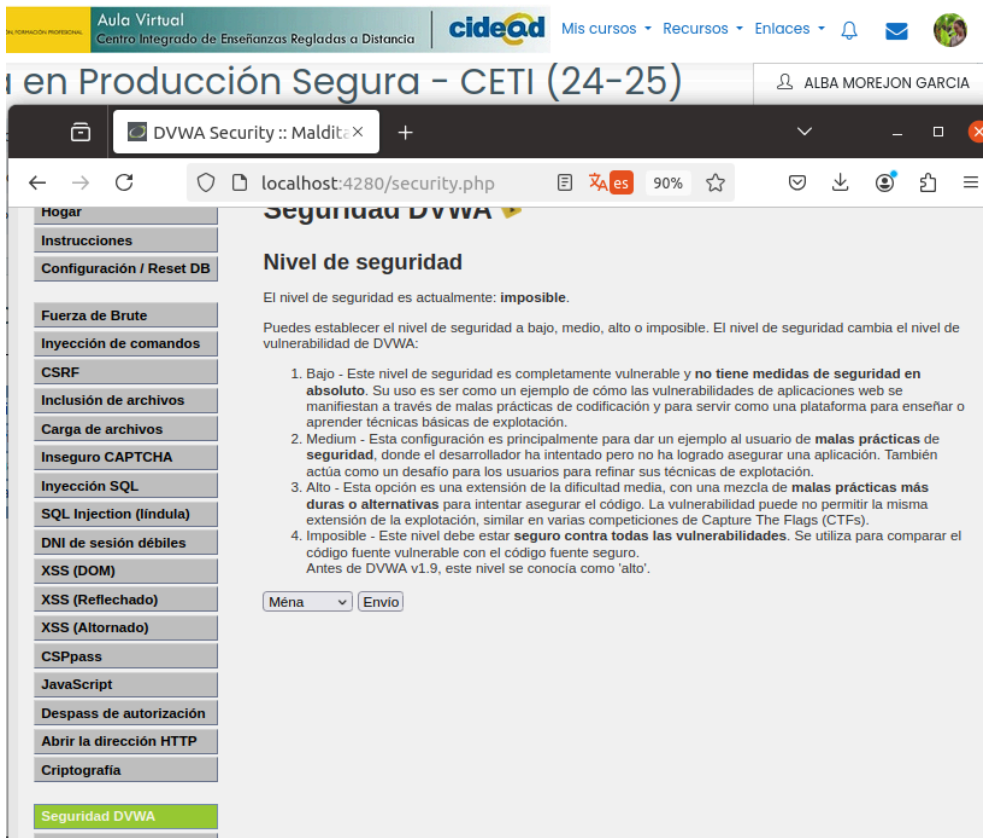


- A partir de ese momento el usuario es **admin** y la clave es **password**
- Nos llevará al login e introducimos las credenciales **admin:password**



- **Para cambiar el nivel de seguridad debes ir a DVWA Security**

En el apartado Security DVWA nos permite cambiar el nivel de seguridad



Apartado 3: Pruebas de vulnerabilidades.

Procederemos a realizar distintas pruebas de SQL Injection

- **Accederemos en modo Low**
- **Pulsar en el menú lateral SQL Injection**
- **Primero comprobaremos cómo se comporta nuestro aplicativo cuando le damos IDs de usuarios (puedes probar con 1, 2, 3..)**
- **Trataremos de ver qué sucede cuando le damos resultados no esperados, por ejemplo tres comillas simples '''**
- **Revisaremos el código del aplicativo web para entender qué ha sucedido y que consulta (query) se ha intentado realizar. Para ver el código pulsar el botón View Source que hay al final de la página.**
- **Probaremos con otros IDs como: ' OR '1'='1 (muy importante poner bien las comillas)**
- **Revisaremos el código del aplicativo web para entender qué ha sucedido y que consulta (query) se ha realizado. Para ver el código pulsar el botón View Source que hay al final de la página.**
- **Probaremos a cambiar el modo de seguridad de DVWA a Impossible (dónde no debería haber vulnerabilidades) y volveremos a realizar todos los pasos de este apartado: probar con un id de usuario, probar con tres comillas, probar con ' OR '1'='1 y revisar el código del aplicativo.**

- **Accederemos en modo Low**

Establecemos el nivel de seguridad en Low



- **Pulsar en el menú lateral SQL Injection**
- **Primero comprobaremos cómo se comporta nuestro aplicativo cuando le damos IDs de usuarios (puedes probar con 1, 2, 3..)**



- **Trataremos de ver qué sucede cuando le damos resultados no esperados, por ejemplo tres comillas simples '''**

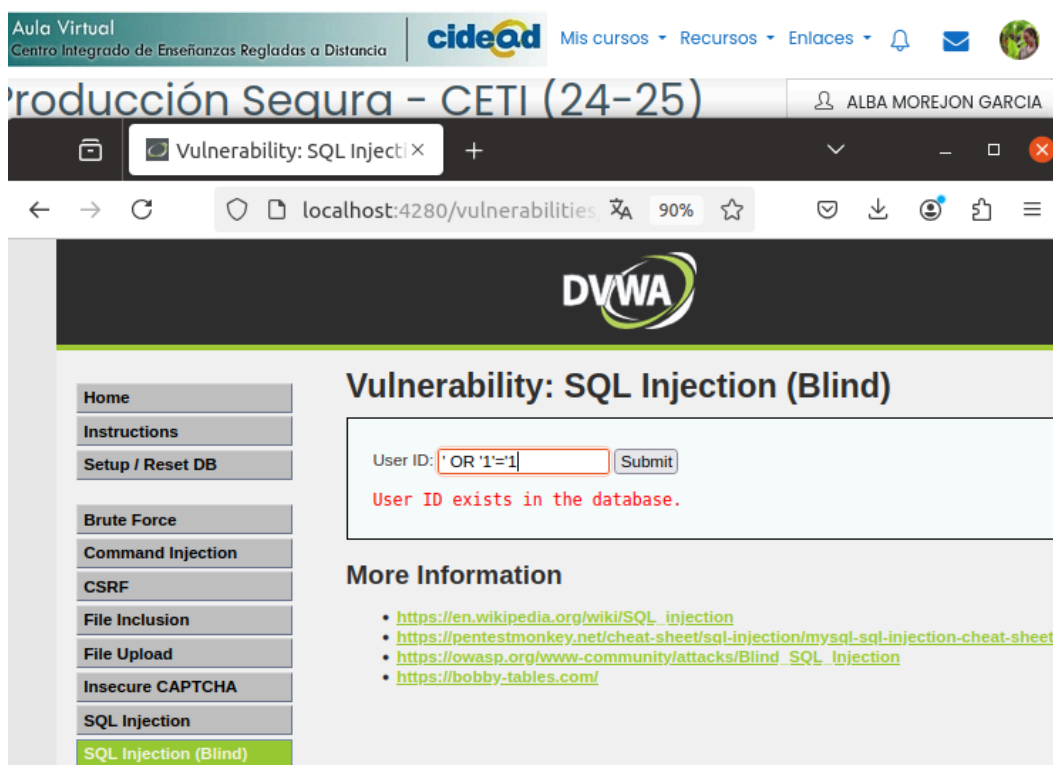


- Revisaremos el código del aplicativo web para entender qué ha sucedido y que consulta (query) se ha intentado realizar. Para ver el código pulsar el botón View Source que hay al final de la página.

```
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
try {
    $result = mysqli_query($GLOBALS["___mysqli_ston"], $query); // Removed 'or die' to suppress mysql
errors
} catch (Exception $e) {
    print "There was an error.";
    exit;
}
```

Se muestra ese mensaje porque el código PHP está diseñado para excepciones y errores en la consulta SQL, al introducir ''' como User ID se ha causado un error de sintaxis en la consulta SQL, lo que muestra el mensaje de error por ser una excepción.

- Probaremos con otros IDs como: ' OR '1'=1 (muy importante poner bien las comillas)



- Revisaremos el código del aplicativo web para entender qué ha sucedido y que consulta (query) se ha realizado. Para ver el código pulsar el botón View Source que hay al final de la página.

Al meter ese usuario se hace la siguiente consulta a la base de datos:

```
"SELECT first_name, last_name FROM users WHERE user_id = " OR '1'=1";
```

Debido a que lo introducido es una condición que siempre es verdadera, la consulta devuelve todos los registros de la tabla "users" y debido a que el valor \$exists será mayor a 1, se devuelve el mensaje mostrado.

```
if ($exists) {
    // Feedback for end user
    echo '<pre>User ID exists in the database.</pre>';
}
```

La inyección SQL introducida manipula la consulta para que siempre sea verdadera, permitiendo el acceso a todos los registros de la tabla.

- Probaremos a cambiar el modo de seguridad de DVWA a Impossible (dónde no debería haber vulnerabilidades) y volveremos a realizar todos los pasos de este apartado: probar con un id de usuario, probar con tres comillas, probar con ' OR '1'='1 y revisar el código del aplicativo.

Introduciendo el valor 1, devuelve “User Id exist in the database”

El código comprobará que el valor introducido sea numérico y buscará si existe una fila en la tabla users que coincida con el dato, si encuentra una fila mostrará un mensaje de que si existe y si no, mostrará otro diciendo que no.

Comprobar valor numérico:

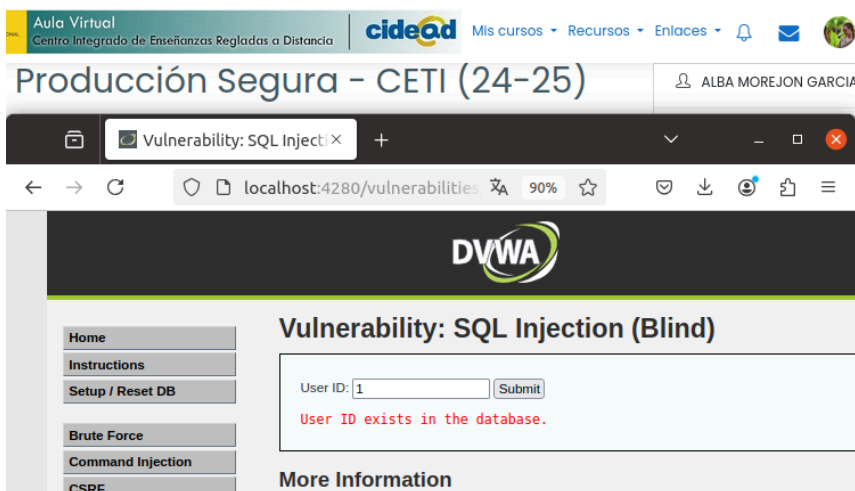
```
if(is_numeric( $id )) {
    $id = intval( $id );
```

Buscar en la base de datos:

```
$data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );
$data->bindParam( ':id', $id, PDO::PARAM_INT );
$data->execute();
$exists = $data->rowCount();
```

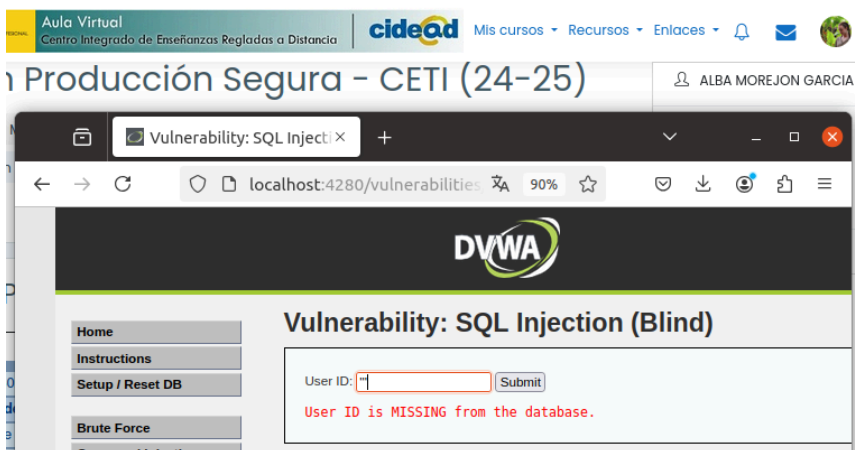
Mostrar resultado:

```
if( $exists ) {
    echo '<pre>User ID exists in the database.</pre>';
} else {
    header( $_SERVER[ 'SERVER_PROTOCOL' ] . ' 404 Not Found' );
    echo '<pre>User ID is MISSING from the database.</pre>';
}
```



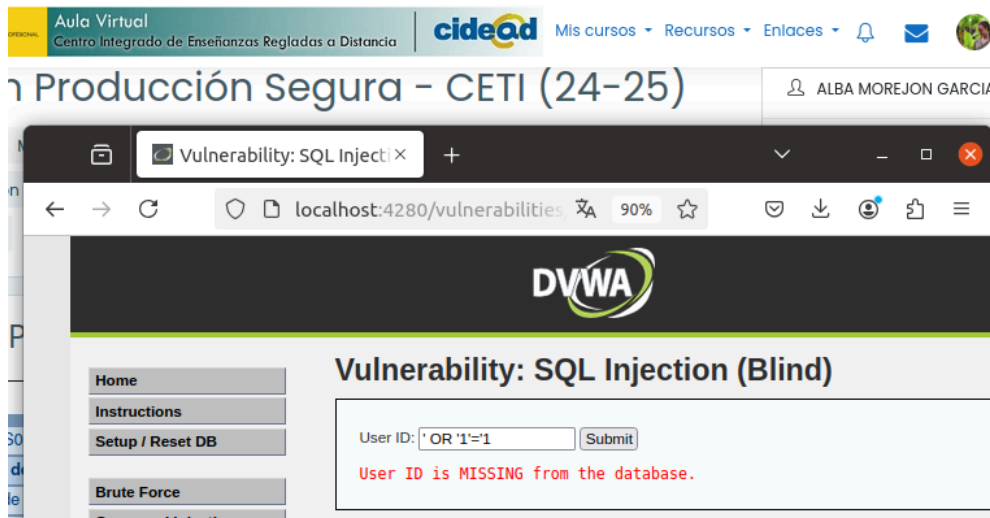
Introduciendo el valor ””, devuelve “User Id is MISSING from the database”

Como hemos explicado antes, al verificar que el valor sea numérico el input no pasa la verificación, lo que impide que se ejecute la consulta SQL porque no se establece ninguna variable \$exists.



Introduciendo el valor ' OR '1'=1, devuelve "User Id is MISSING from the database"

Pasa lo mismo que el anterior, en la sección donde el código comprueba que el valor introducido sea numérico, no se produce la condición, por eso da error, y el código no sigue ejecutándose, dando como resultado el mensaje de error.



Apartado 4: Preguntas finales

1. Indica cómo se ha comportado el aplicativo web:

- En el caso 3 comillas simples en el modo Low

Resulta un error de sintaxis SQL, ya que no lo reconoce como un valor válido y se mostrará el mensaje de que ha habido un error.

- En el caso ' OR '1'=1 en el modo Low

Debido a la vulnerabilidad de inyección SQL, el aplicativo mostrará que existe ese User ID en la tabla. La consulta devolverá todas las filas de la tabla, haciendo que la variable \$exist se establezca como true y haciendo que devuelva el mensaje de que si existe en la base de datos.

- En el caso 3 comillas simples en el modo Impossible

Al estar diseñado de la forma más segura, comprobará si el valor introducido cumple la condición de ser un número, al no cumplir la condición, fallará y no se ejecuta la consulta.

- En el caso ' OR '1'=1 en el modo Impossible

Como el valor introducido no cumple la condición, no es numérico, no se establece la variable necesaria para continuar (\$exist) y el código no procede a la consulta, devolviendo un error.

2. Indica qué consulta exacta (el código SQL) crees que se ha intentado/se ha lanzado en la base de datos:

- En el caso 3 comillas simples en el modo Low

SELECT first_name, last_name FROM users WHERE user_id = ''';".

- En el caso ' OR '1'=1 en el modo Low

"SELECT first_name, last_name FROM users WHERE user_id = ' OR '1'=1 '";".

- En el caso 3 comillas simples en el modo Impossible

La consulta SQL no llega a la base de datos, porque da el error antes y por tanto no sigue con la ejecución del código.

- En el caso ' OR '1'=1 en el modo Impossible

La consulta SQL no llega a ejecutarse en la base de datos, porque al ser un valor diferente a numérico, da un error y por tanto no sigue con la ejecución del código.

3.Realiza una comparativa del código php del modo Low y del modo Impossible ¿Qué diferencias ves?

LOW:

```
<?php
if( isset( $_GET[ 'Submit' ] ) ) {
    $id = $_GET[ 'id' ]; // Get input
    $exists = false;
    switch ( $_DVWA[ 'SQLI_DB' ] ) {
        case MYSQL:
            // Check database
            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
            try {
                $result = mysqli_query( $GLOBALS[ "__mysqli_ston" ], $query ); // Removed 'or die' to suppress mysql errors
            } catch (Exception $e) {
                print "There was an error.";
                exit;
            }
            $exists = false;
            if ( $result !== false ) {
                try {
                    $exists = (mysqli_num_rows( $result ) > 0);
                } catch (Exception $e) {
                    $exists = false;
                }
            }
            ((is_null( $___mysqli_res = mysqli_close( $GLOBALS[ "__mysqli_ston" ] ))) ? false : $___mysqli_res);
            break;
        case SQLITE:
            global $sqlite_db_connection;
            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
            try {
                $results = $sqlite_db_connection->query( $query );
                $row = $results->fetchArray();
                $exists = $row !== false;
            } catch (Exception $e) {
                $exists = false;
            }
            break;
    }
    if ( $exists ) {
        echo "<pre>User ID exists in the database.</pre>"; // Feedback for end user
    } else {
        // User wasn't found, so the page wasn't!
        header( $_SERVER[ 'SERVER_PROTOCOL' ] . ' 404 Not Found' );
        echo "<pre>User ID is MISSING from the database.</pre>"; // Feedback for end user
    }
}
?>
```

IMPOSSIBLE

```
<?php
if( isset( $_GET[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );
    $exists = false;
    $id = $_GET[ 'id' ]; // Get input
    if( is_numeric( $id ) ) { // Was a number entered?
        $id = intval( $id );
        switch ( $_DVWA[ 'SQLI_DB' ] ) {
            case MYSQL:
                // Check the database
                $data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );
                $data->bindParam( ':id', $id, PDO::PARAM_INT );
                $data->execute();
                $exists = $data->rowCount();
                break;
            case SQLITE:
                global $sqlite_db_connection;
                $stmt = $sqlite_db_connection->prepare( 'SELECT COUNT(first_name) AS numrows FROM users WHERE user_id = :id
LIMIT 1;' );
                $stmt->bindValue( ':id', $id, SQLITE3_INTEGER );
```

```

$result = $stmt->execute();
$result->finalize();
if ($result !== false) {
    // There is no way to get the number of rows returned
    // This checks the number of columns (not rows) just as a precaution, but it won't stop someone dumping multiple rows
    and viewing them one at a time.
    $num_columns = $result->numColumns();
    if ($num_columns == 1) {
        $row = $result->fetchArray();
        $numrows = $row[ 'numrows' ];
        $exists = ($numrows == 1);
    }
    break;
}
}
if ($exists) { // Get results
    // Feedback for end user
    echo '<pre>User ID exists in the database.</pre>';
} else {
    // User wasn't found, so the page wasn't!
    header( $_SERVER[ 'SERVER_PROTOCOL' ] . ' 404 Not Found' );
    // Feedback for end user
    echo '<pre>User ID is MISSING from the database.</pre>';
}
}
generateSessionToken(); // Generate Anti-CSRF token
?>

```

La diferencia clave entre ambos códigos es que en el código en modo Impossible se verifica el token Anti-CSRF con `checkToken` y se establece una verificación para que el valor introducido sea un número. En la consulta SQL a la base de datos, el código en modo Low introduce directamente el id en la cadena y en el modo Impossible, se utiliza consultas ya preparadas lo que impide las posibles inyecciones y en el resultado, el modelo más básico da el resultado directamente, mientras que el más complejo verifica el número de columnas devueltas.

En conclusión, en el modo Low se utiliza un código más vulnerable a ataques de inyección porque no valida adecuadamente el User ID y construye las consultas SQL directamente con los datos del usuario. En cambio, el modo Impossible verifica que el User ID sea numérico, utiliza consultas preparadas y añade verificación Anti-CSRF, esto hace que sea mucho más robusto y seguro frente a posibles ataques.

4.Revisa la entrada de OWASP para este tipo de vulnerabilidad

(https://owasp.org/www-community/attacks/SQL_Injection).

¿Cómo crees que afecta el código php del nivel low y del nivel impossible a la vulnerabilidad que estábamos explotando?

La entrada de OWASP sobre inyecciones SQL explica que este tipo de ataques ocurre cuando un atacante inserta una consulta SQL a través de los datos de entrada de un cliente a la aplicación. Esto puede permitir al atacante leer datos sensibles, modificar datos, ejecutar operaciones administrativas...

En el modo low, el código PHP es vulnerable a este tipo de inyecciones porque no valida adecuadamente el `user_id` antes de usarlo en la consulta SQL y construye las consultas concatenando directamente el valor proporcionado por el usuario, lo que permite al atacante manipular la consulta. Como es en el caso de “‘ OR '1'='1” esto devolvería todas las filas de la tabla “user” permitiendo al atacante acceder a datos sensibles.

En el modo impossible el código PHP está protegido contra inyecciones porque verifica el `user_id` sea numérico antes de usarlo en la consulta y utiliza consultas preparadas con parámetros, lo que evita que el valor proporcionado se interprete como parte del código SQL, esto asegura que el `user_id` se trate como un valor de datos y no como código SQL.

En resumen, el modo low es vulnerable a inyecciones SQL debido a la falta de validación, mientras que el modo impossible es seguro porque valida la entrada y utiliza consultas preparadas.