Curso de Especialización de Ciberseguridad en Entornos de Tecnología de la Información (CETI)

Puesta en Producción Segura

UD02. Pruebas de SQL Injection.
Tarea Online.

JUAN ANTONIO GARCIA MUELAS

INDICE

1.- Descripción de la tarea.

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.

¿Qué te pedimos que hagas?

✓ **Apartado 1: Responde a las siguientes cuestiones**

Buscar información sobre la vulnerabilidad **CVE-2023-39417** y rellenar la siguiente tabla

Breve descripción	Vulnerabilidad de inyección SQL en PostgreSQL si se usa @extowner@, @extschema@ o @extschema:...@, donde un atacante con privilegios CREATE de nivel de base de datos puede ejecutar código arbitrario como superusuario.
Impacto	Alto. Por el nivel de acceso a modificar el sistema.
Productos y versiones vulnerables	cpe:2.3:a:postgresql:postgresql:*:*:*:*:*:*:*:* Desde 11.0 Hasta 11.20; Desde 12.0 Hasta 12.15; Desde 13.0 Hasta 13.12; Desde 14.0 Hasta 14.8; Desde 15.0 Hasta 15.3; cpe:2.3:a:redhat:software_collections:-:*:*:*:*:*:*:* cpe:2.3:o:redhat:enterprise_linux:8.0:*:*:*:*:*:*:* cpe:2.3:o:redhat:enterprise_linux:9.0:*:*:*:*:*:*:* cpe:2.3:o:debian:debian_linux:8.0:*:*:*:*:*:*:*
Posibles soluciones	PostgreSQL publica la actualización de paquetes en su sitio: https://www.postgresql.org/support/security/CVE-2023-39417/ Debian recomienda actualizar los paquetes con la versión 13.13-0+deb11u1. https://www.debian.org/security/2023/dsa-5554 Red Hat también emite una actualización de paquetes mediante parche (actualizado el 28-11-2023), tanto para Red Hat Enterprise Linux 8.0 como 9.0. https://access.redhat.com/errata/RHSA-2023:7545

Buscar información del requisito **ASVS v4.0.3-5.3.4** (ASVS **versión 4.0.3 capítulo 5, apartado 3, requisito 4)** y rellena la siguiente tabla

Breve descripción	Verifique que la selección de datos o las consultas de base de datos (por ejemplo, SQL, HQL, ORM, NoSQL) utilizan consultas parametrizadas, ORM, marcos de entidades o están protegidas de los ataques de inyección de base de datos.
Niveles a los que debe aplicarse	L1, L2, L3
Categoría CWE	89

✓ Apartado 2: Crea el entorno de pruebas

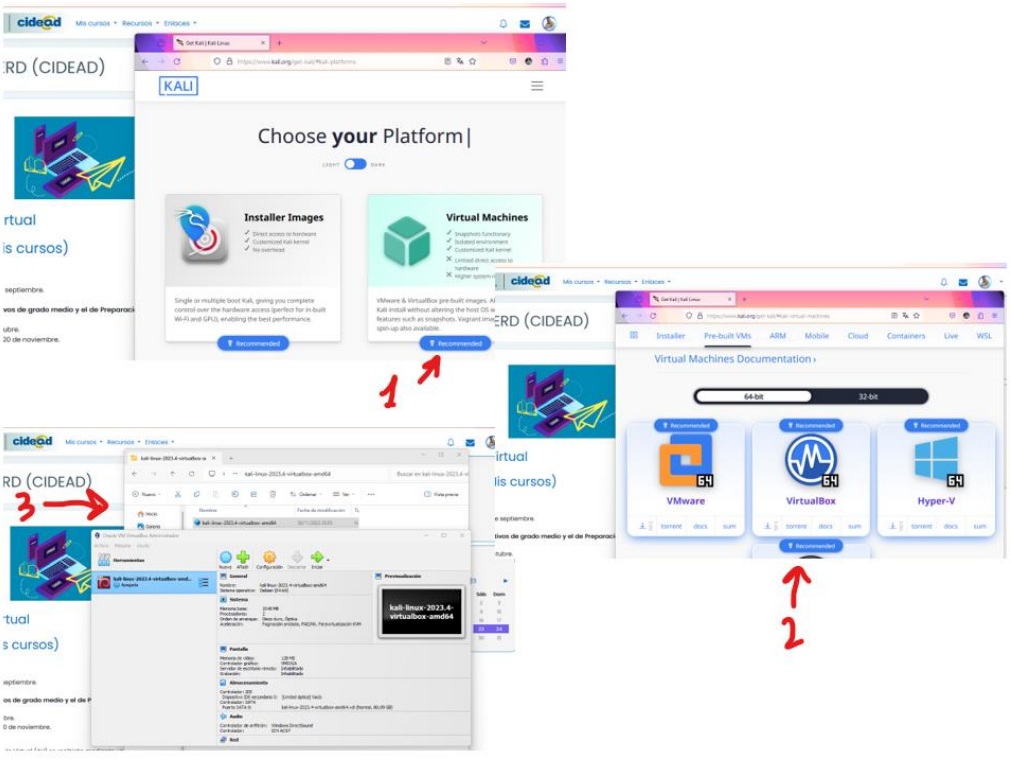
Hoy en día la mayoría de los equipos de desarrollo utilizan contenedores para montar los entornos de desarrollo. En la página web del proyecto de DVWA (https://github.com/digininja/DVWA) explican como instalar la aplicación con Docker, que es una de las herramientas más habituales para trabajar con contenedores.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

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

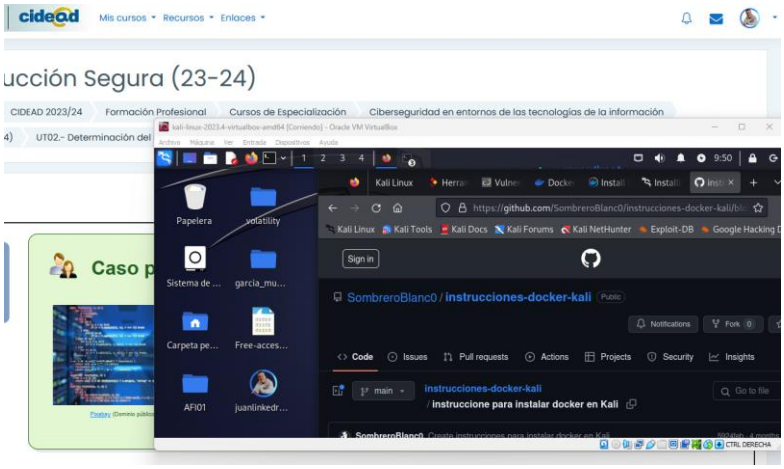
   Voy a instalar Docker en la Imagen de Kali Linux que estoy utilizando para las tareas.

   Es una máquina genérica descargable desde la propia web de Kali Linux (facilitaré los enlaces al final).

   Sólo hay que seleccionar donde vamos a utilizar la imagen y tras descargarla y descomprimirla, nos abrirá directamente en VirtualBox.

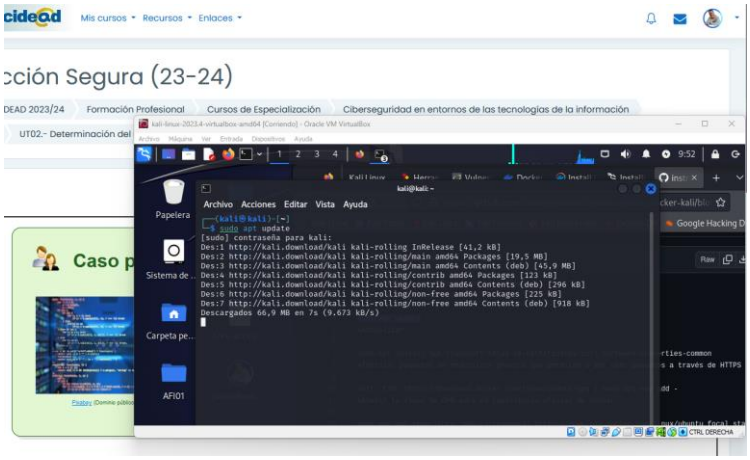


Aprovecharé el repositorio del usuario **"SombreroBlanc0"** que nos indica los pasos detallados para instalar Docker y compose en esta distribución.

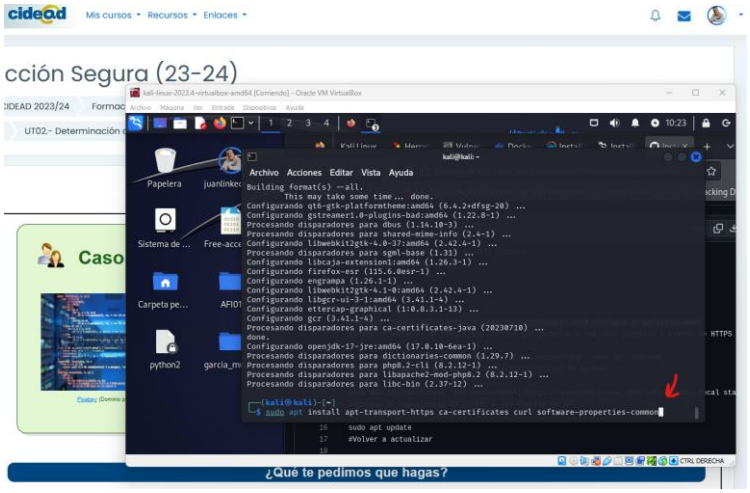


Lo primero es asegurar que contamos con la lista de paquetes actualizada.
`sudo apt update`

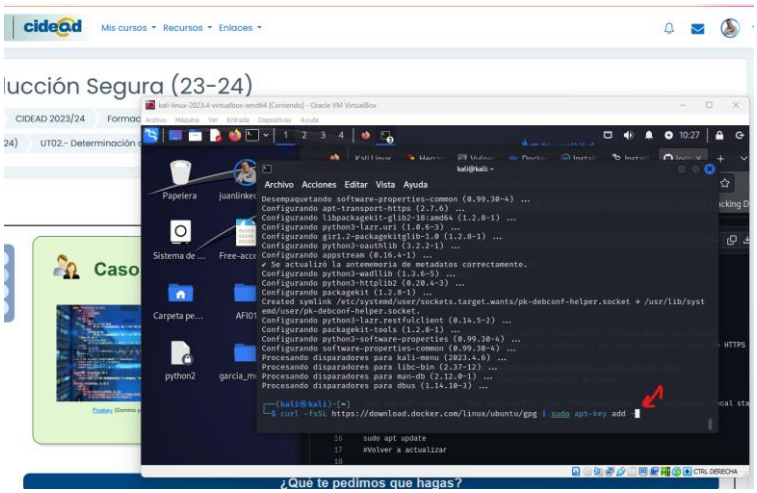
Y si es necesario:
`sudo apt upgrade`



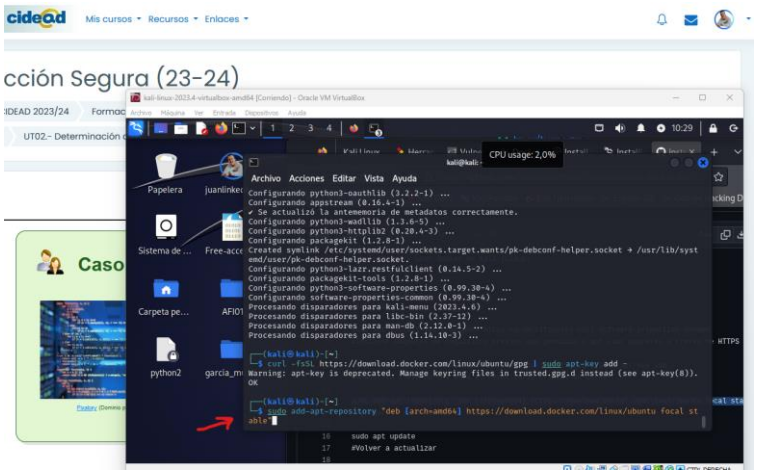
Tras ese paso, instalamos los paquetes de requisitos previos para paquetes a través de https.
`sudo apt install apt-transport-https ca-certificates curl software-properties-common`



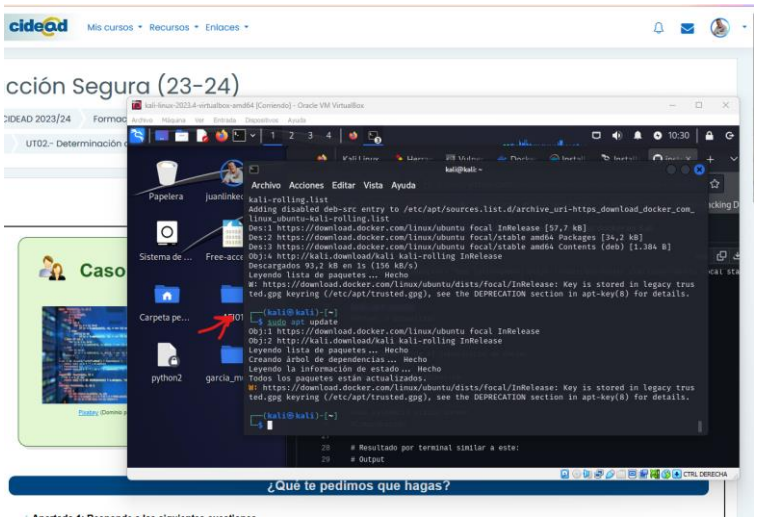
Añadimos la clave de GPG para el repositorio oficial de Docker.
`curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`



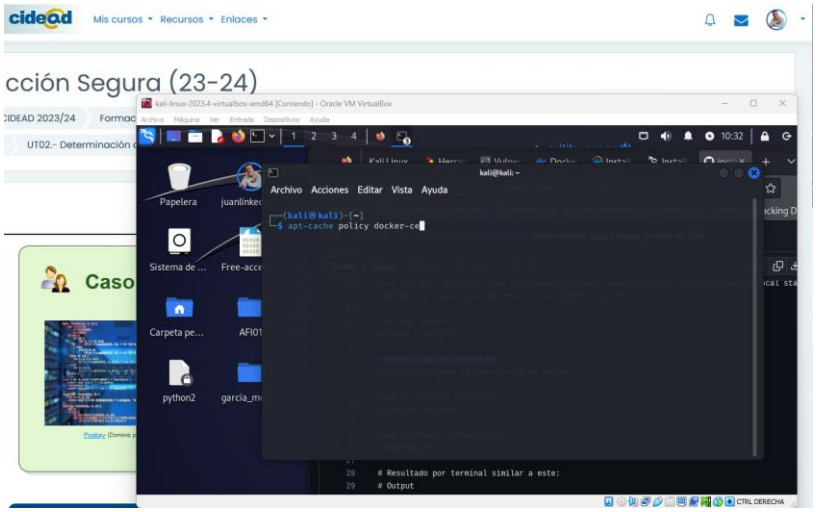
Agregamos el repositorio a las fuente de apt
`sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"`



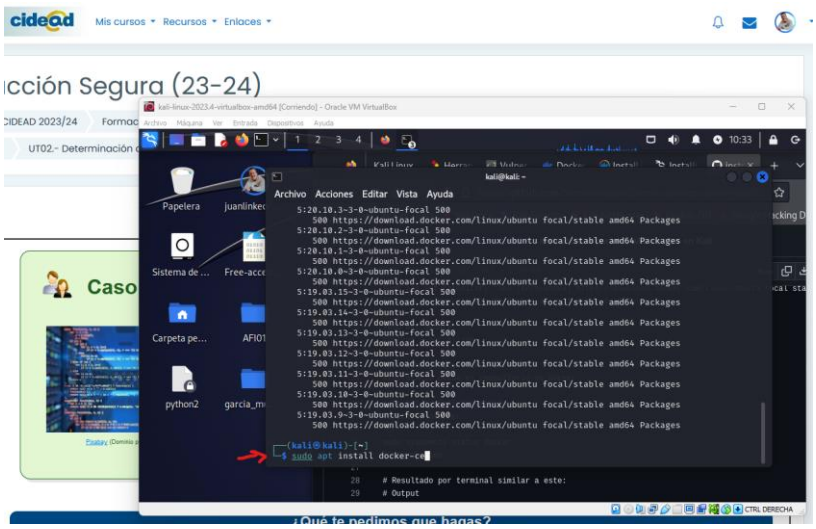
Volvemos a actualizar
`sudo apt update`



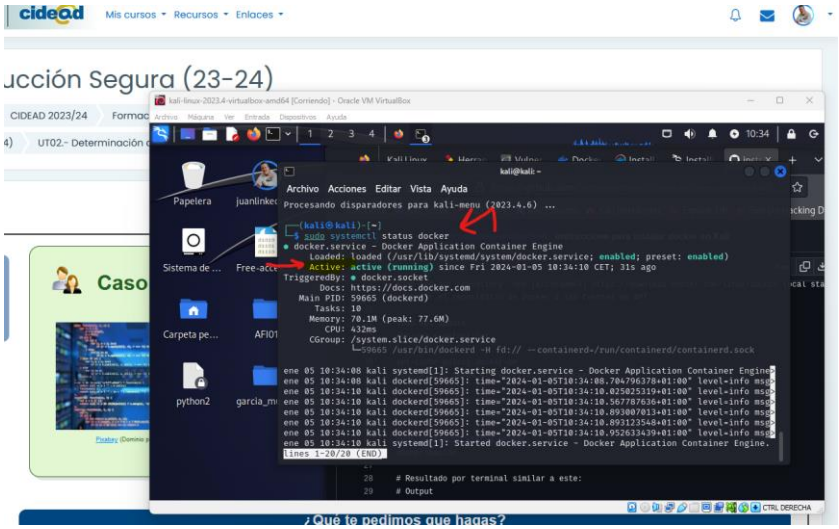
Instalamos desde el repositorio de Docker
`apt-cache policy docker-ce`



`sudo apt install docker-ce`

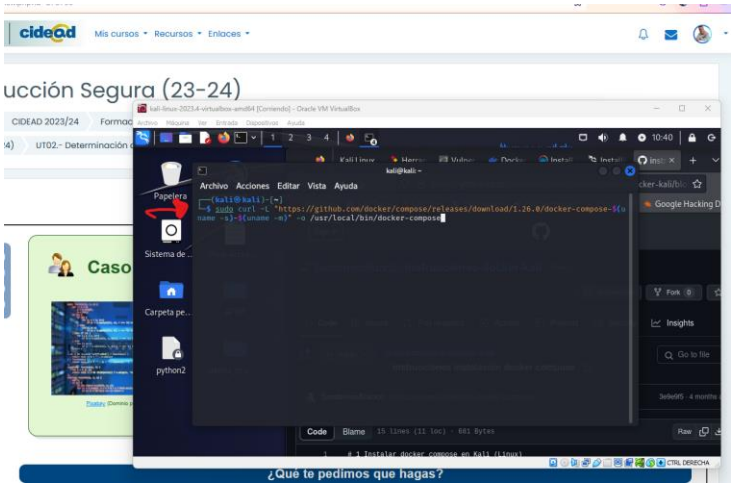


Comprobamos la instalación
`sudo systemctl status docker`

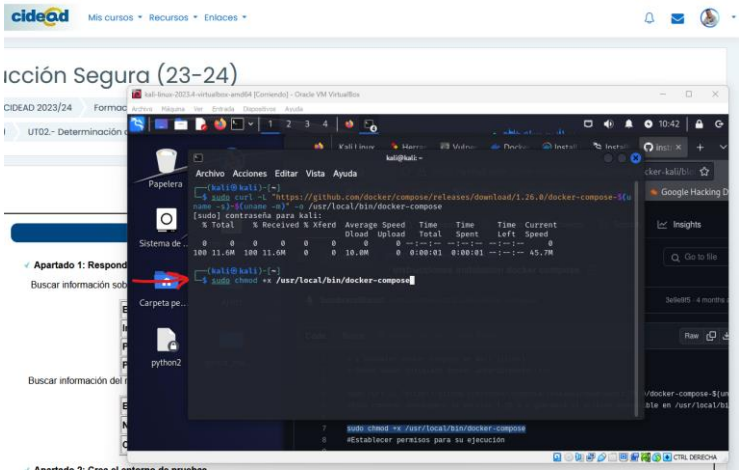


Vamos a docker compose:

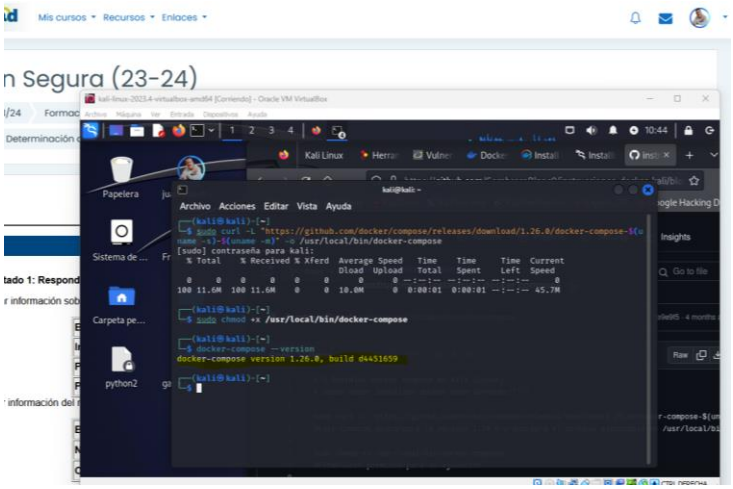
Guardamos el ejecutable para que sea accesible globalmente
`sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compos`



Establecemos los permisos para ejecutarlo
`sudo chmod +x /usr/local/bin/docker-compose`



Comprobamos la instalación
`docker-compose --version`

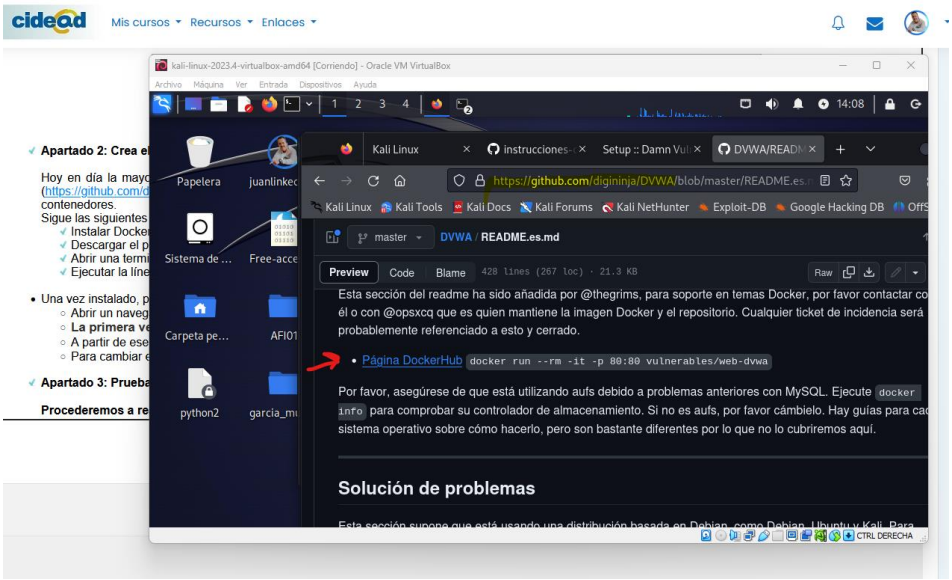


2. Descargar el proyecto DVWA y descomprimirlo
3. Abrir una terminal y cambiar al directorio donde se ha descomprimido.
4. Ejecutar la línea de comandos: **docker compose up -d**

➢ Una vez instalado, para acceder a DVWA:

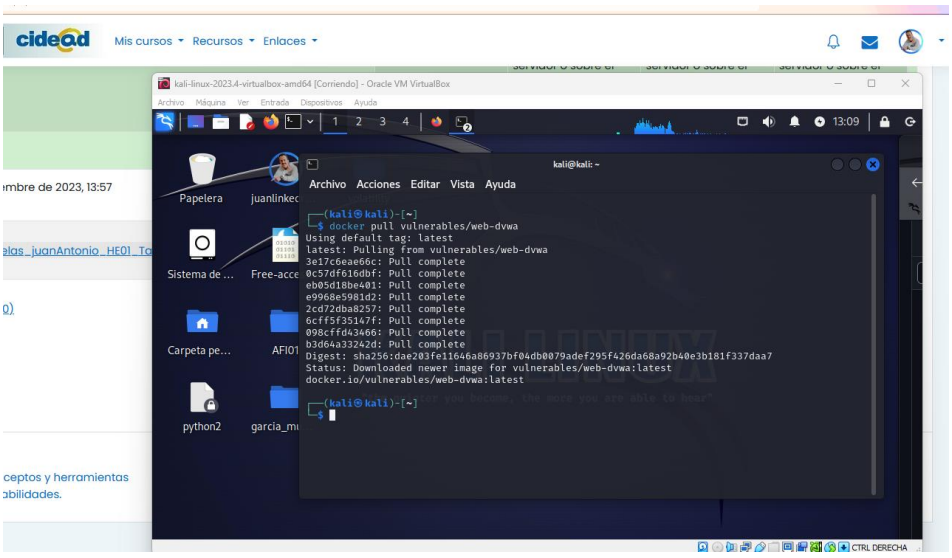
- Abrir un navegador con la dirección **http://localhost:4280 El usuario es admin y la clave es admin**
- **La primera vez que se abre la aplicación es necesario pulsar en el botón Create/Reset Database**
- A partir de ese momento el usuario es **admin** y la clave es **password**
- Para cambiar el nivel de seguridad debes ir a **DVWA Security**

Para esta parte que comprenden los puntos 2-4, voy a seguir las instrucciones que marca para Docker en la web del proyecto (enlaces al final de la tarea), donde han creado un repositorio con la imagen a clonar.



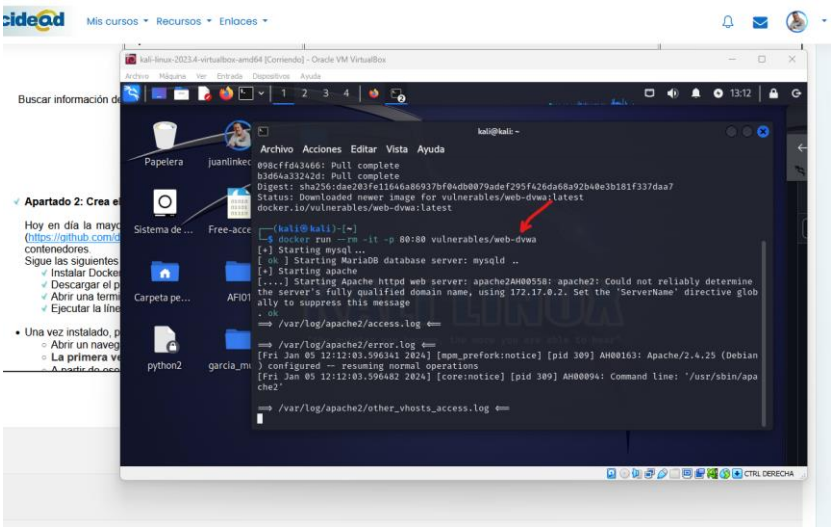
Hacemos un pull del contenedor desde el repo en Docker

`docker pull vulnerables/web-dvwa`

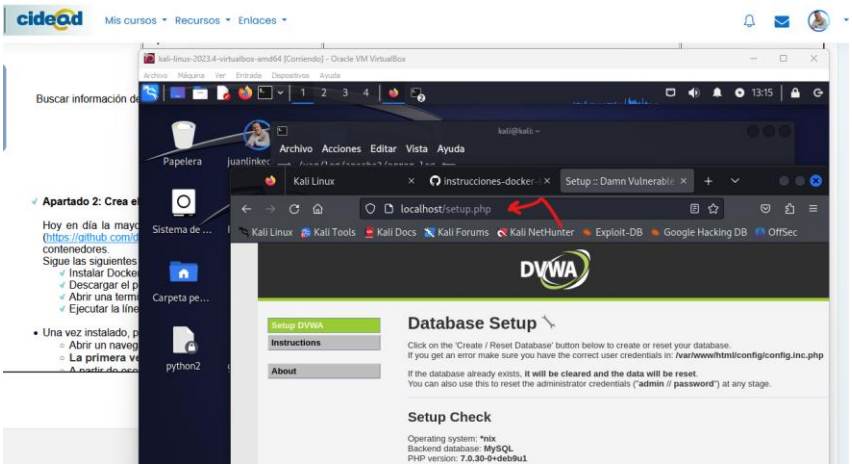


Corremos la imagen del proyecto.

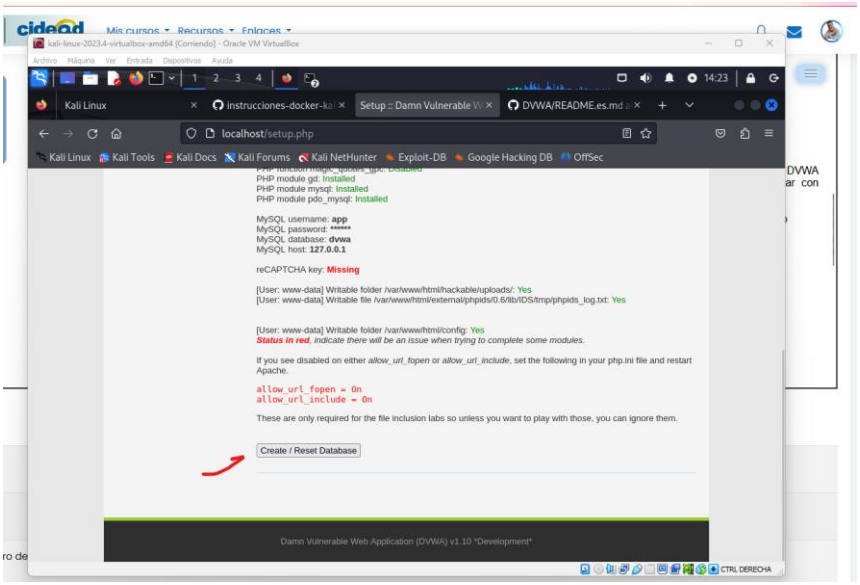
`docker run --rm -it -p 80:80 vulnerables/web-dvwa`



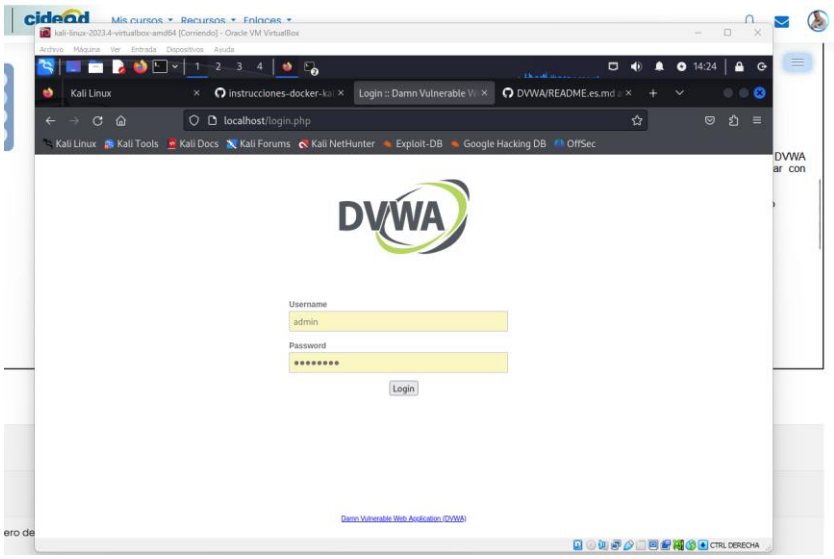
Si abrimos el navegador, podemos acceder:



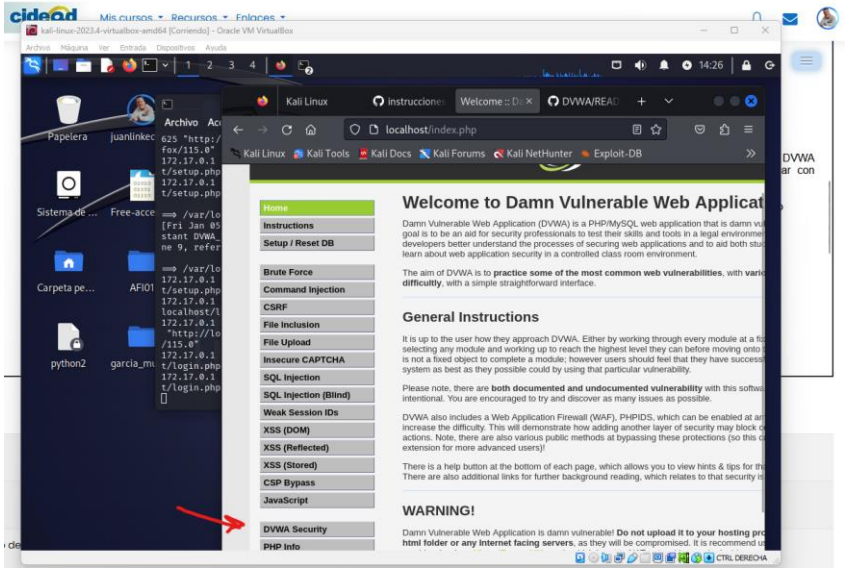
Bajamos para acceder a Create/Reset Database



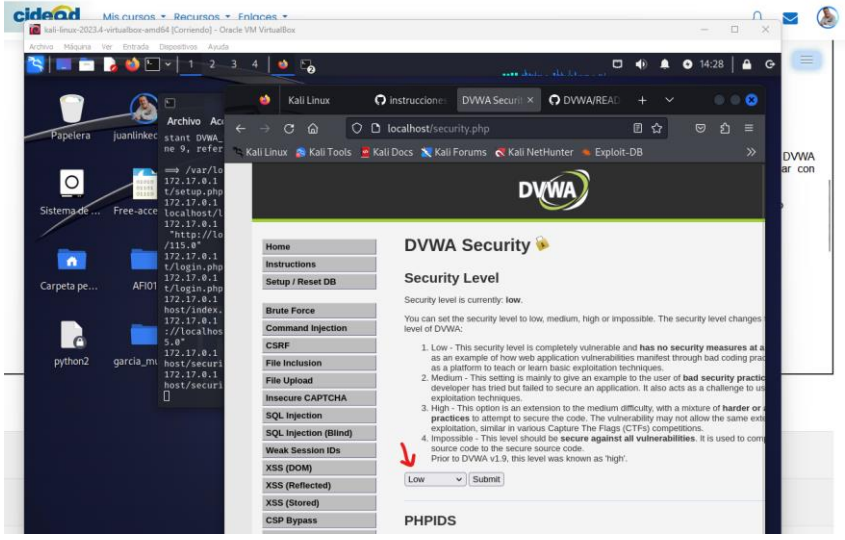
Accedemos con los datos por defecto.



Desde esta vista tenemos en el menú lateral un acceso a la seguridad.



Reducimos el nivel de seguridad a Low para poder hacer pruebas a continuación.

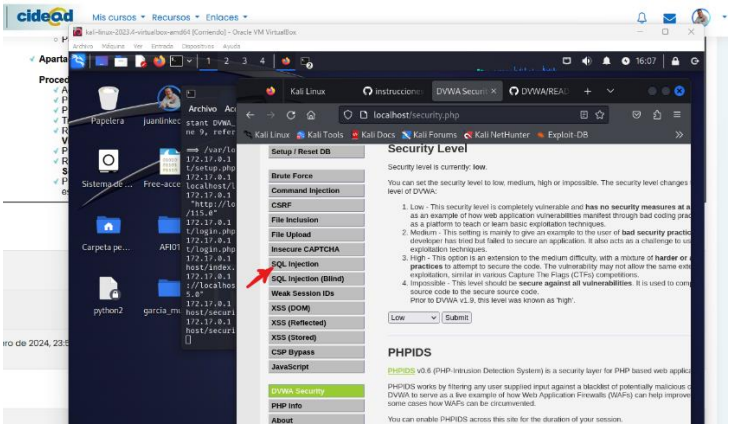


✓ Apartado 3: Pruebas de vulnerabilidades.

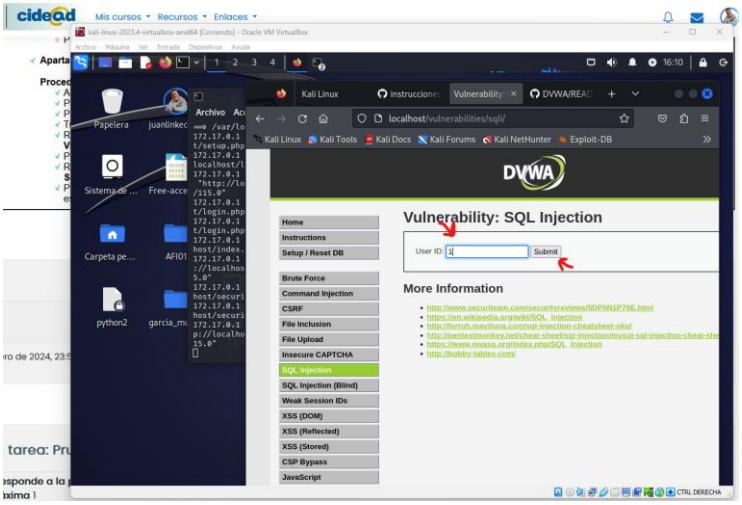
Procederemos a realizar distintas pruebas de SQL Injection

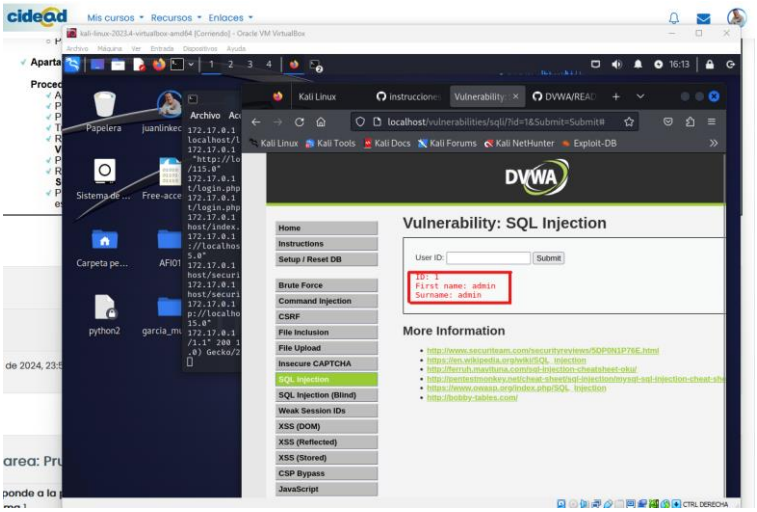
1. Accederemos en modo **Low**
2. Pulsar en el menú lateral **SQL Injection**

Accedemos al menú lateral tras seleccionar el modo Low.

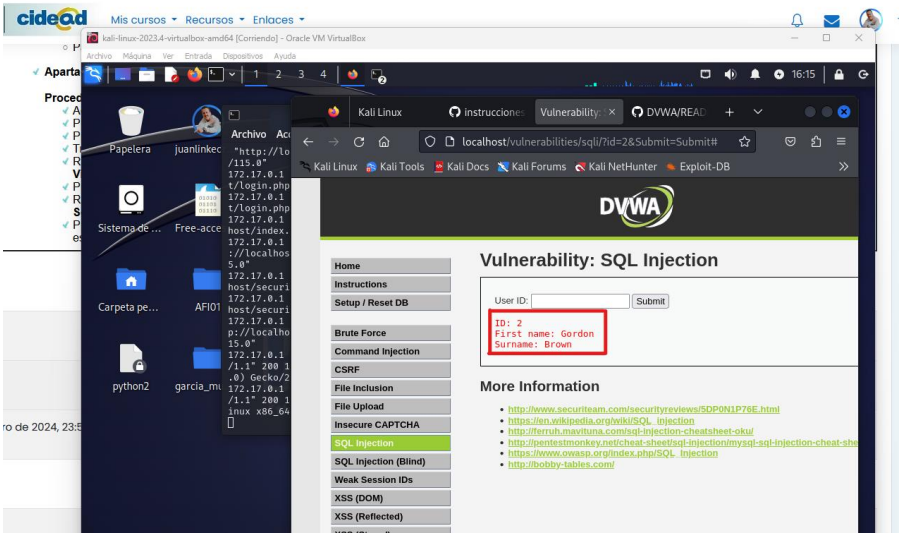


3. Primero comprobaremos cómo se comporta nuestro aplicativo cuando le damos IDs de usuarios (puedes probar con 1, 2, 3...)

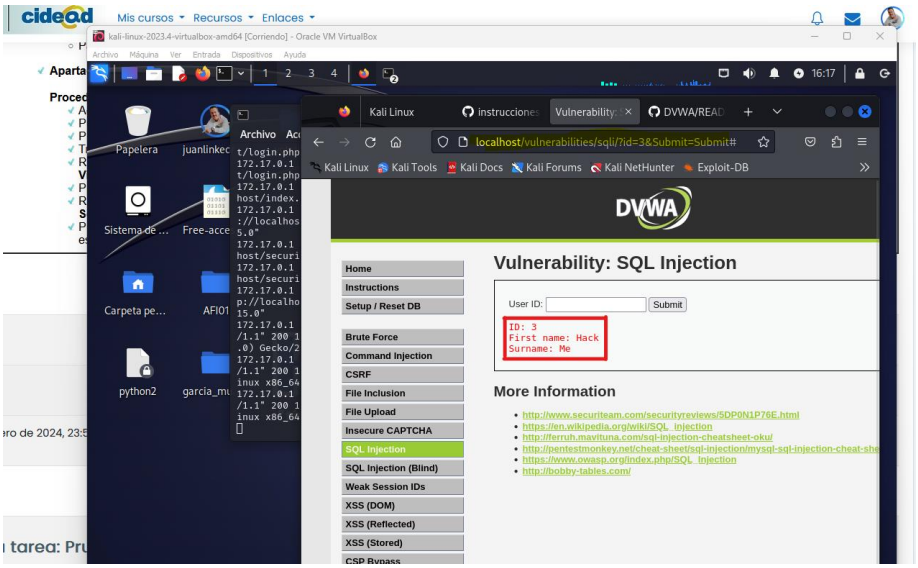




Podemos ver como muestra los datos de los IDs solicitados.

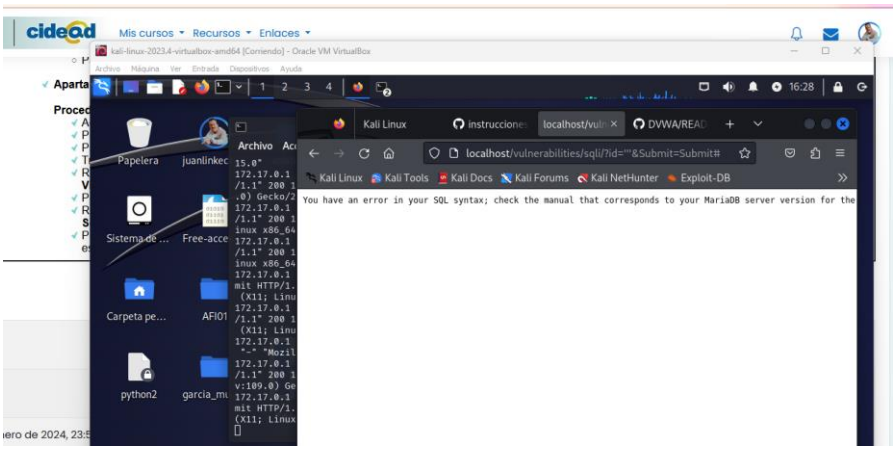


Incluso podemos ver la consulta en la propia barra de direcciones, al ser una petición **GET**.

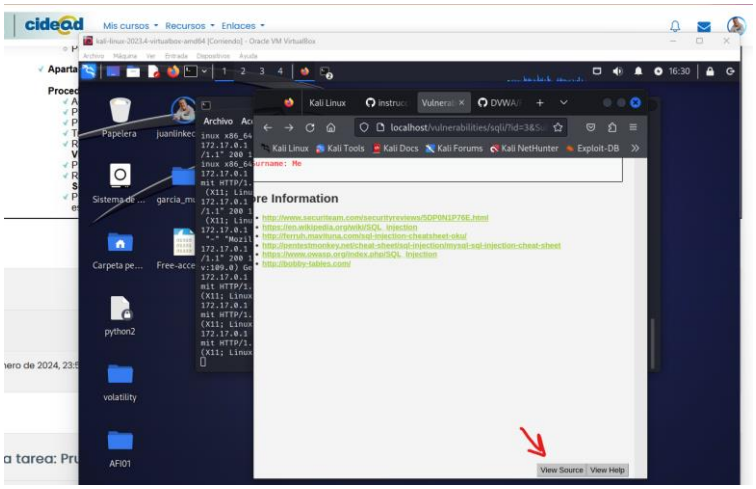


4. Trataremos de ver qué sucede cuando le damos resultados no esperados, por ejemplo, tres comillas simples '''

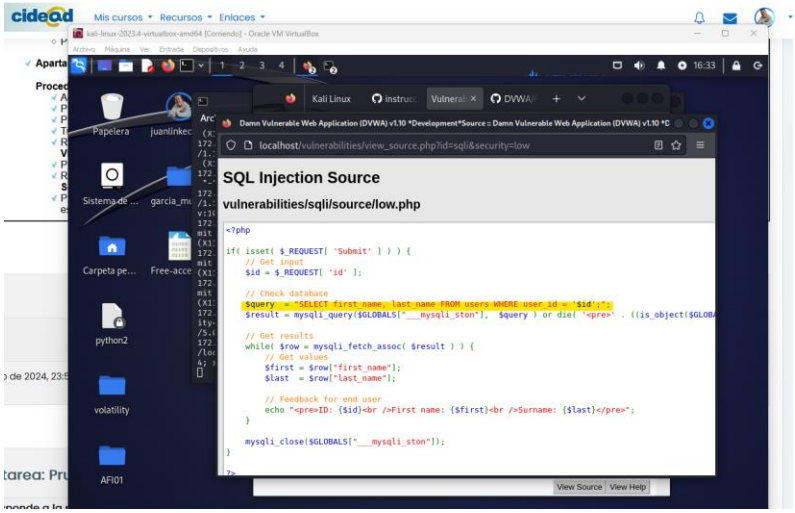
Nos arrojará un error al modificar el encapsulado en la query.



5. 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.

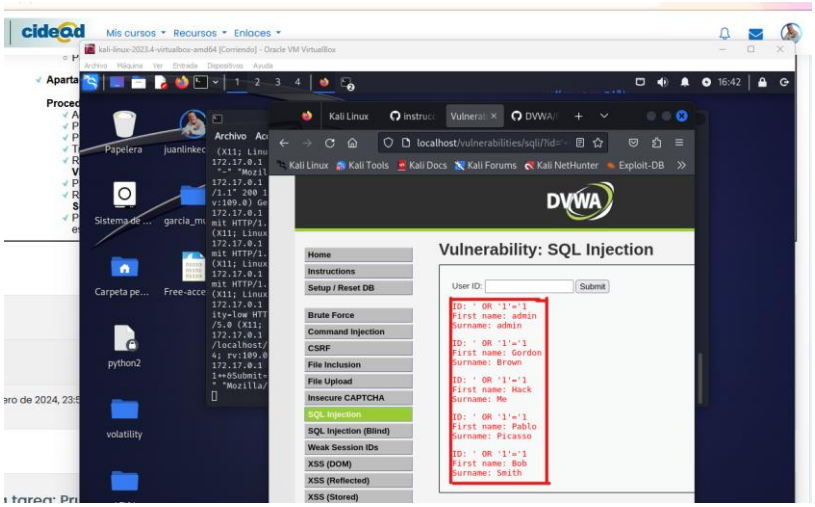


Ha intentado recoger el resultado de la consulta, y no ha recibido por el **`submit`** un valor numérico, sino las comillas y por eso ha fallado.



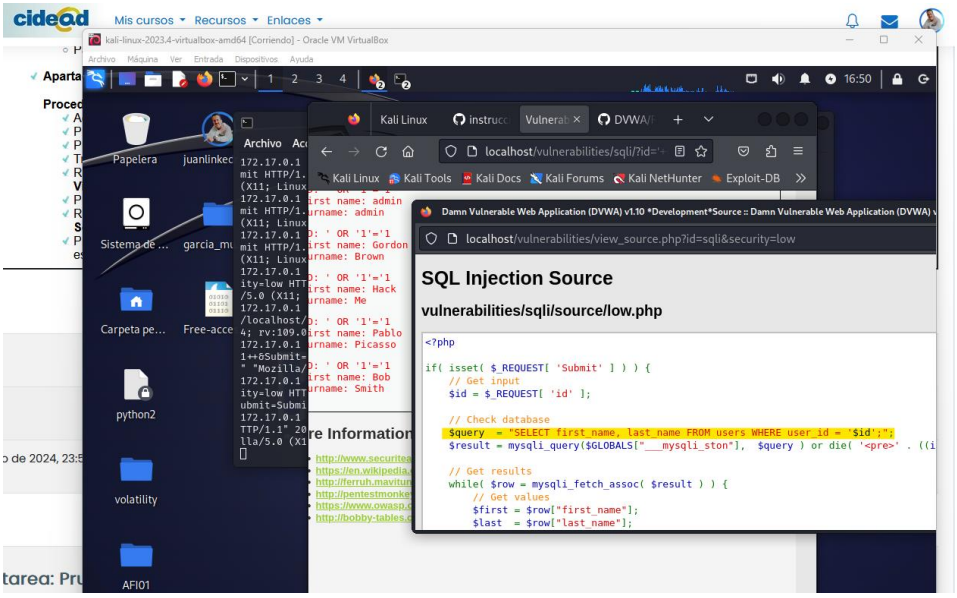
6. Probaremos con otros IDs como: **' OR '1'='1** (muy importante poner bien las comillas)

Modificamos la lógica y nos muestra todos los datos encontrados.



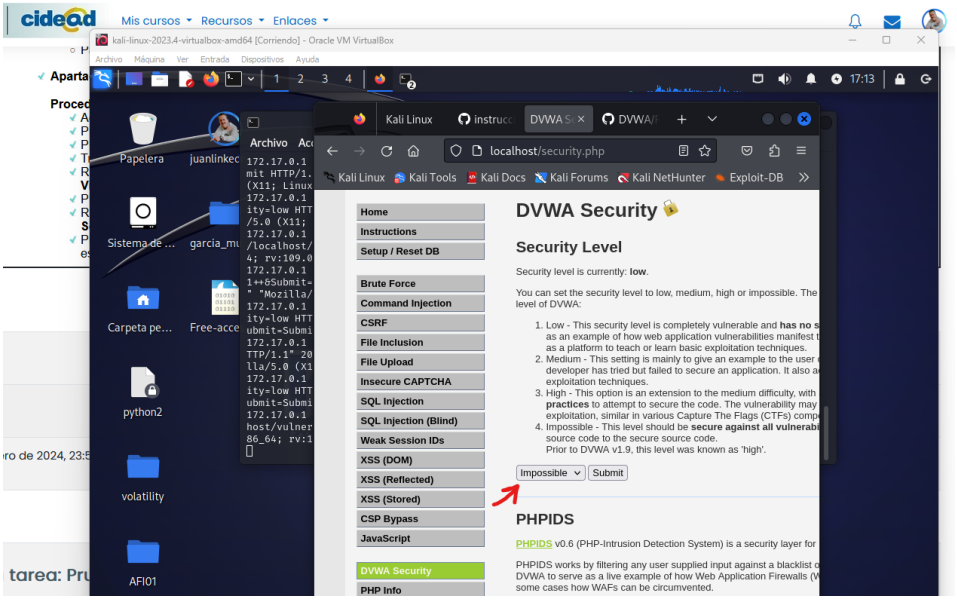
7. Revisaremos el código del aplicativo web para entender qué ha sucedido y que consulta (query) se ha realizado. Para ver el código pulsar el **botón View Source** que hay al final de la página.

   Ha intentado esta query, donde modifica con **' OR '1'='1** , el orden de las comillas y por esto consigue hacer la inyección de código mediante esta consulta y devuelve los datos encontrados.

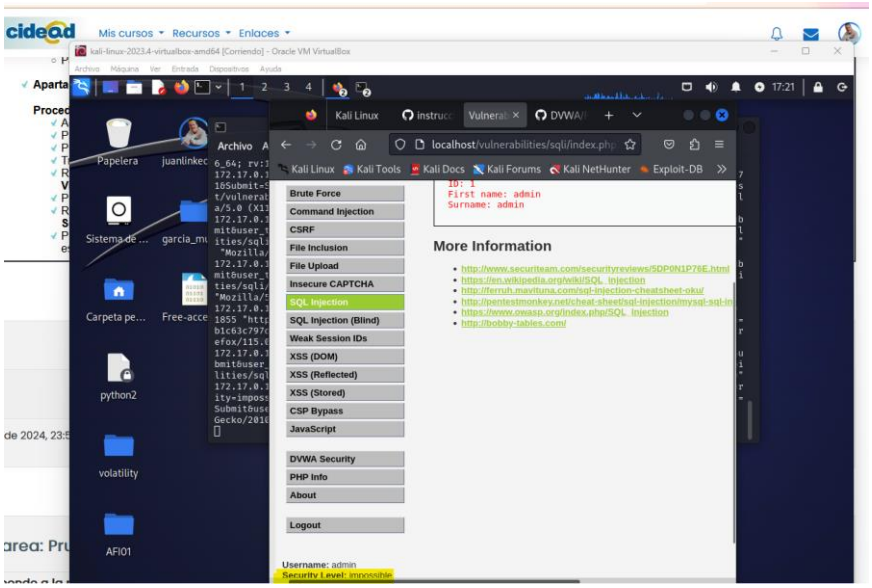


8. Probaremos a cambiar el modo de seguridad de DVWA a **Impossible** (dónde no debería haber vulnerabilidades) y volveremos a realizar todos los pasos de este apartado: probar con un id de usuario, probar con tres comillas, probar con ' OR '1'='1 y revisar el código del aplicativo.

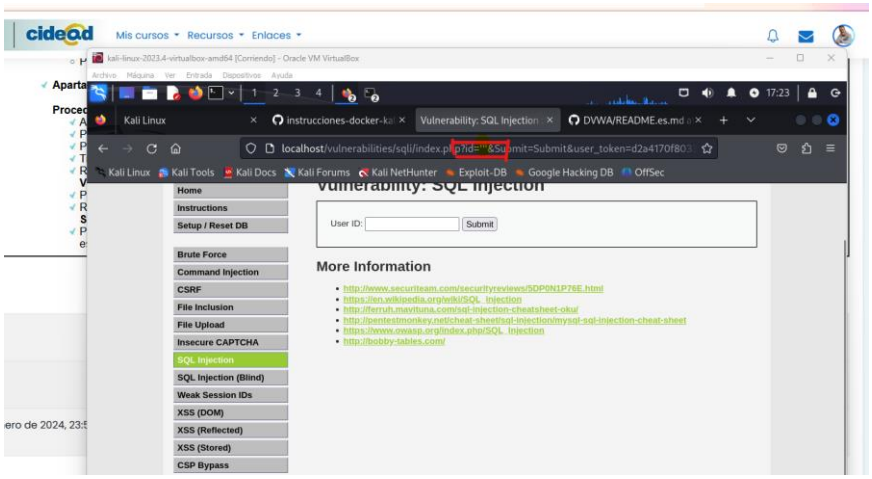
   Modifico la seguridad a **`Impossible`** y comparo:

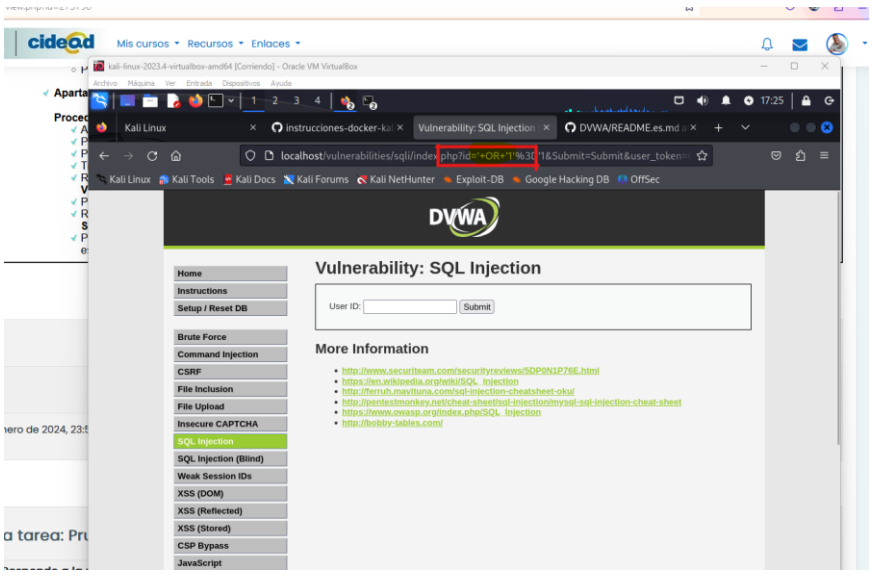


Con solo el ID, muestra los datos, incluso en este modo (Como puede verse en la parte inferior).



Con las comillas o el **OR**, no muestra a priori nada, aunque se puede ver en la barra de direcciones que ha intentado la consulta.





✓ Apartado 4: Preguntas finales

➢ ¿Cómo se ha comportado el aplicativo web cuando ha recibido 3 comillas simples en el modo **Low**? ¿Qué consulta crees que se ha intentado lanzar en la base de datos?

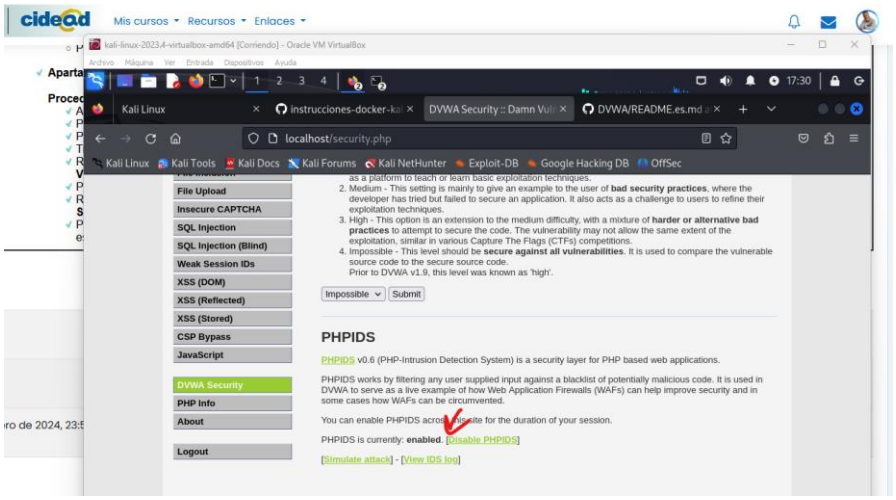
Toma por **`GET`** el id que se envía. Con el parámetro **`getid`**, ha intentado recoger con una función **`mysql_query`** el resultado de esta consulta, y no ha recibido por el **`submit`** un valor numérico, como en el primer ejemplo, sino las comillas, que rompen la consulta (**`''''''`**) y por eso ha fallado, pasando al "**`die`**" para que muestre el error a través de una función **`mysql_error`**.

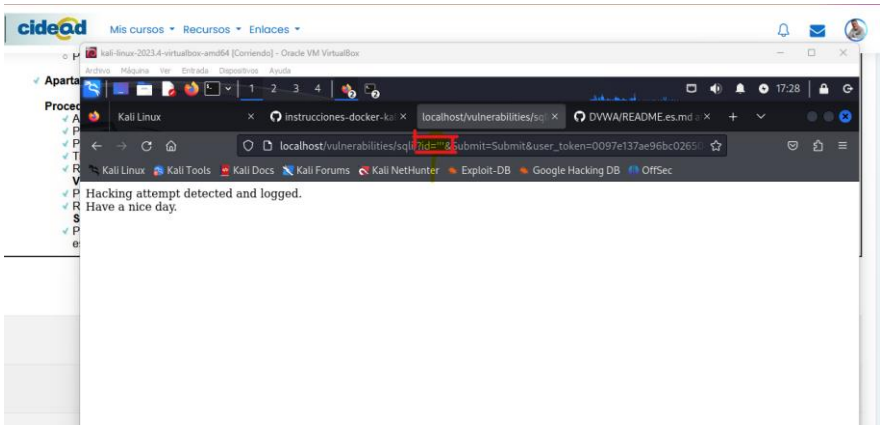
`"SELECT first_name, last_name FROM users WHERE user_id = '''';"`

➢ ¿Cómo se ha comportado el aplicativo web cuando ha recibido 3 comillas simples en el modo **Impossible**? ¿Qué consulta crees que se ha intentado lanzar en la base de datos?

Aparte de la pista que nos daba la barra de direcciones, cuesta ver si funciona correctamente en este modo.

Para ayudar, he activado en la vista de Seguridad el **`PHPIDS`**, y que así nos arroje al menos un mensaje.



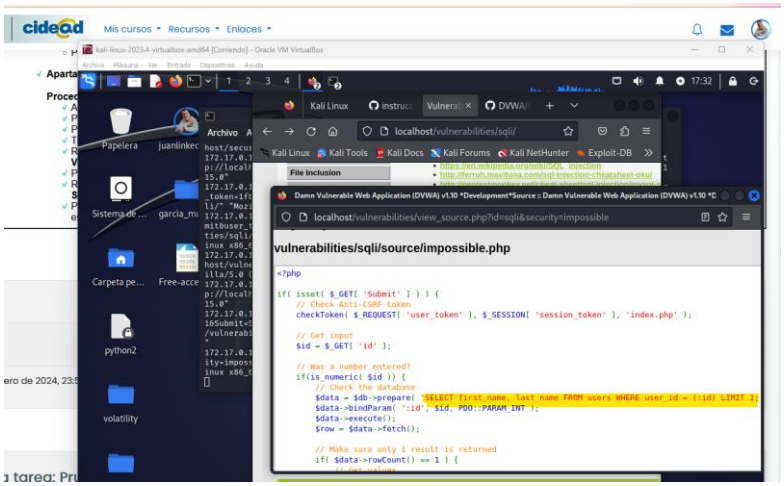


La consulta en modo **`Impossible`** cambia considerablemente. Mucho más robusta.

Tiene protección contra **`CSRF`** mediante tokens de usuario y de sesión.

Valida que el **`ID`** es numérico y realiza una consulta preparada, donde recoge el dato asignado y limita los registros para que sea válida si solo recibe uno.

En base a eso, prepara la respuesta.



La consulta no llega a prepararse, pues no entra siquiera en el condicional que requiere de un Id numérico, y por eso falla.

- ¿Cómo se ha comportado el aplicativo web cuando ha recibido ' OR '1'='1 en el modo **Low**? ¿Qué consulta crees que se ha lanzado en la base de datos?

El funcionamiento del aplicativo es similar al caso anterior de las comillas en modo **Low**.
Sin embargo, en este caso ha intentado ejecutar la query, donde modifica con **' OR '1'='1** , el orden de las comillas y por esto consigue hacer la inyección de código mediante esta consulta final.
"SELECT first_name, last_name FROM users WHERE user_id = '' OR '1'='1';"
Al utilizar la condición **OR**, no recibe un ID pero el **1=1** nos devolverá un true, por lo que devuelve los datos encontrados.

- ¿Cómo se ha comportado el aplicativo web cuando ha recibido ' OR '1'='1 en el modo **Impossible**? ¿Qué consulta crees que se ha lanzado en la base de datos?

En este caso, aunque el funcionamiento es muy similar al comentado en el punto anterior con la consulta preparada, va a recibir un primer valor numérico recogido en el parámetro **$id**, e intentará crear una consulta:
'SELECT first_name, last_name FROM users WHERE user_id = (1) LIMIT 1;'

Sin embargo, será en el siguiente paso, donde cotejará que solo puede recibir un resultado mediante la función **rowCount()**, cuando impedirá recoger y mostrar datos.

- ¿Qué diferencia ves en el código PHP cuando está en modo **Low** y cuándo está en modo **Impossible**? ¿Cómo crees que afecta ese código a las vulnerabilidades que estábamos explotando?

Creo que queda explicado en los puntos anteriores.

En el modo **Low**, vemos como si el registro que mandamos es correcto, no hay barreras que protejan nuestro código, pero en **Impossible**, hay sanitización de los parámetros que llegan, y la consulta y salida de datos es más segura.

Podemos resumirlo en que la seguridad en las consultas se incrementa según modificamos ese nivel, llegando a encapsular el intento de inyección de código en la propia consulta y evitando así cualquier problema de seguridad asociado a ello.

Webgrafía

https://www.kali.org/get-kali/#kali-platforms

https://github.com/SombreroBlanc0/instrucciones-docker-kali/blob/main/instruccione%20para%20instalar%20docker%20en%20Kali

https://www.youtube.com/watch?v=ODlDHiFZw-8

https://hackpuntes.com/instalacion-de-docker-en-kali-linux/

https://github.com/digininja/DVWA/blob/master/README.es.md

https://www.youtube.com/watch?v=WkyDxNJkgQ4

https://hub.docker.com/r/vulnerables/web-dvwa/