APUNTES 02

DETERMINACIÓN DEL NIVEL DE SEGURIDAD REQUERIDO POR APLICACIONES

PUESTA EN PRODUCCIÓN SEGURA

ALBA MOREJÓN GARCÍA

2024/2025

Ciberseguridad en Entornos de las Tecnologías de la Información

ÍNDICE

1. Fuentes abiertas para desarrollo seguro.
2. Listas de riesgos de seguridad habituales.
   2.1. Vulnerabilidades y exposiciones comunes
3. Comprobaciones de seguridad a nivel de aplicación.
   3.1. Niveles de seguridad ASVS
4. Requisitos de verificación necesarios asociados al nivel de seguridad establecido.

El desarrollo de software ha evolucionado mucho durante los últimos años, han surgido fundaciones como OWASP cuyo objetivo es la mejora en la seguridad del software mediante la identificación de las principales vulnerabilidades existentes y como poder afrontarlas.
También ha evolucionado mucho el desarrollo de software a nivel de exposición, ya que a día de hoy muchas de las líneas de código de las aplicaciones que vemos en internet están disponibles en repositorios públicos, lo cual aporta grandes beneficios como veremos más adelante pero también añade una exposición adicional al código, que cómo veíamos en unidades anteriores hay una gran relación entre el grado de exposición y el número de vulnerabilidades.

1.- FUENTES ABIERTAS PARA DESARROLLO SEGURO

Cuando hablamos de software de fuentes abiertas nos referimos a aquel software que podemos encontrar de fácil acceso, publicado y por lo general libre (el código fuente se pone a disposición del público para ser usado y modificado conforme los usuarios o desarrolladores puedan aportar).
Durante los últimos años ha habido una evolución clara, ya que hasta ahora el software que usábamos tanto a nivel doméstico como profesional era software cerrado o propietario, es decir, aquel en el que no se permite el acceso al código fuente.
Estos últimos años se ha demostrado que el software de fuentes abiertas aporta una serie de ventajas tanto a usuarios finales como a empresas y corporaciones de forma clara:

- Estabilidad: al ser el acceso libre, los usuarios y programadores expertos pueden revisar el código y ayudar a mejorarlo, mejorando por tanto la estabilidad del software. Si consideramos que hay sistemas operativos abiertos hace que estos sistemas tengan comunidades detrás que los mejoren día tras día. Mayor estabilidad es también software más seguro, que coteje mejor los datos con los que trabaja la aplicación, por tanto afecta tanto al usuario final como las empresas desarrolladoras.
- Coste: cuando hay una comunidad tan importante de usuarios detrás del código, los costes de desarrollo del mismo se reducen, ya que la comunidad de usuarios interactuando con el código reduce los costes de desarrollo, depuración, testeo, etc.
- Favorece la innovación colectiva ya que incentiva la acción de la comunidad de usuarios detrás del código, revisando, mejorándolo.
- Mejoras: muchos usuarios cuando interactúan con el código lo adaptan y modifican dándole nuevos usos y capacidades, por lo tanto al nivel el software original es depurado y potenciado.

La idea y el movimiento detrás del software de fuentes abiertas ha sido aceptada, aplicada, promovida y difundida por todo el mundo. Haciendo que muchas empresas de desarrollo de software lo hayan incorporado como parte de su ADN y por tanto el paradigma a nivel de desarrollo de software ha cambiado de forma radical. Como hemos comentado, al final este proceso no solamente beneficia a los usuarios finales y empresas sino también

2.- LISTAS DE RIESGOS DE SEGURIDAD HABITUALES

El riesgo asociado con el uso de software se puede describir conceptualmente de la siguiente manera: un agente de amenaza (threat agent, en inglés) interactúa con un sistema (system), que puede tener un vulnerabilidad (vulnerability) que puede ser explotada (exploited) para causar un impacto (impact). Un ejemplo podría ser un ladrón (agente de amenaza) pasea por un gimnasio revisando las taquillas de los clientes (el sistema) en busca de taquillas sin candado (la vulnerabilidad) y cuando encuentran una, abre la puerta sin candado (el exploit) y toma lo que sea que haya dentro (el impacto).
Toda aplicación/sistema puede tener fallos que pueden ser explotados, pero detectar y corregir los mismos no es tarea simple. Existen diferentes organismos/empresas que se dedican a detectar y plantear soluciones a las vulnerabilidades que se van detectando. Algunas de las más importantes son:

- NVD (La base de datos nacional de vulnerabilidades de EE.UU): realiza la recopilación de datos de gestión de vulnerabilidades basados en estándares operados por el gobierno federal. La y el , junto con otros expertos mundiales, mantienen el sistema de categorías CWE que cataloga las diversas vulnerabilidades.
- OWASP (Proyecto Abierto de Seguridad de Aplicaciones Web), de la que veremos diversos proyectos en profundidad, se encarga de determinar y combatir las causas de que el software no sea seguro.

En España tenemos el INCIBE. Como indica en su página web: "Trabaja para afianzar la confianza digital, elevar la ciberseguridad y la resiliencia y contribuir al mercado digital de manera que se impulse el uso seguro del ciberespacio en España". Te recomiendo visitar su página web en profundidad.

OWASP Foundation

La Fundación OWASP es un organismo sin ánimo de lucro, cuyo único objetivo es trabajar en mejorar la seguridad del software. A través de proyectos de software de código abierto (y fuentes abiertas) liderados por la comunidad, decenas de miles de miembros, conferencias educativas y cientos de cursos anuales de capacitación líderes, la Fundación OWASP es uno de los principales motores en la mejora del software en entornos web.
Uno de los proyectos desarrollados por esta organización es el Top10 de riesgos a nivel software en entornos web y el Top10 de riesgos para entornos móviles. El proceso para confeccionar esta lista no es un proceso sencillo ya que se tienen en cuenta y se evalúan distintos parámetros cómo las vulnerabilidades, ataques conocidos, impactos, manejo de los datos, etc.

Top 10 vulnerabilidades web

En la última versión, de 2021, el Top 10 de vulnerabilidades web es (ver enlace):

A1 - Pérdida de Control Acceso
A2 - Fallos Criptográficos Corporación Mitre Instituto SANS
A3 - Inyección de código
A4 - Diseño Inseguro
A5 - Problemas de configuración a nivel de seguridad
A6 - Componentes vulnerables y obsoletos
A7 - Fallos de Autenticación e Identificación
A8 - Fallos de Software y de Integridad de los Datos
A9 - Fallos en el registro (logging en inglés) y Monitorización de la Seguridad
A10 - Falsificación de peticiones en el servidor (Server-Side Request Forgery en inglés)

Top 10 vulnerabilidades móviles

En cuanto a las aplicaciones móviles, la última versión de 2023 que está a punto de publicarse es:

M1: Uso inadecuado de credenciales
M2: Seguridad inadecuada de la cadena de suministro (Inadequate Supply Chain Security en inglés)
M3: Autenticación/autorización insegura
M4: Insuficiente validación de entrada/salida
M5: Comunicación insegura
M6: Controles de privacidad inadecuados
M7: Protecciones binarias insuficientes
M8: Configuración errónea de la seguridad (Security Misconfiguration en inglés)
M9: Almacenamiento de datos inseguro
M10: Criptografía insuficiente

2.1.- Vulnerabilidades y exposiciones comunes

Las Vulnerabilidades y exposiciones comunes (CVE por sus siglas en inglés), es una lista de información registrada sobre vulnerabilidades de seguridad conocidas, en la que cada referencia tiene:

- Un número de identificación CVE-ID.
- Descripción de la vulnerabilidad.
- Qué versiones del software están afectadas.
- Posible solución al fallo (si existe) o como configurar para mitigar la vulnerabilidad.
- Referencias a publicaciones o entradas de foros o blog donde se ha hecho pública la vulnerabilidad o se demuestra su explotación.

Además, suele también mostrarse un enlace directo a la información de la base de datos de vulnerabilidades (NVD) del NIST, en la que pueden conseguirse más detalles de la vulnerabilidad y su valoración.
El CVE-ID ofrece una nomenclatura estándar para identificación de la vulnerabilidad de forma inequívoca que es usada en la mayoría de repositorios de vulnerabilidades Normalmente, el formato es CVE-YYYYNNNN, donde YYYY es el año y NNNN un número generado aleatoriamente

Debes conocer

La lista CVE es definida y mantenida por The MITRE Corporation con fondos de la National Cyber Security Division del gobierno de los Estados Unidos de América. Forma parte del llamado Security Content Automation Protocol.

3.- COMPROBACIONES DE SEGURIDAD A NIVEL DE APLICACIÓN

La fundación OWASP, como hemos visto, no solamente ayuda a disponer de un software más seguro mediante la identificación de los principales riesgos a los que estamos expuestos, sino que también ha arrancado otros proyectos, entre los que destacan una serie de guías y herramientas para la construcción y mantenimiento de aplicaciones seguras. Por ejemplo:

- La guía de desarrollo (en ingles, Development Guide) muestra cómo diseñar y construir una aplicación segura.
- La guía de verificación de seguridad de la aplicación o ASVS muestra cómo verificar la seguridad de la misma.
- La guía de pruebas o WSTG muestra cómo verificar la seguridad de una aplicación web.

El proyecto del Estándar de Verificación de Seguridad de Aplicaciones (ASVS de sus siglas en inglés) de OWASP proporciona información para poder probar los controles técnicos de seguridad de las aplicaciones web y también proporciona una lista de requisitos para un desarrollo correcto y seguro, minimizando el impacto de los riesgos anteriormente detectados. ASVS tiene dos objetivos principales:

- Ayudar a las organizaciones en el desarrollo y mantenimiento de aplicaciones seguras.
- Permitir la alineación entre las necesidades y ofertas de los servicios de seguridad, proveedores de herramientas de seguridad y consumidores.

Los requisitos se desarrollaron con los siguientes objetivos en mente:

- Usarlo como métrica, proporcionando a los desarrolladores y propietarios de aplicaciones un criterio con el que evaluar el grado de confianza que se puede depositar en sus aplicaciones web.
- Utilizarlo como guía, proporcionando orientación a los desarrolladores de controles de seguridad sobre qué incorporar en los controles de seguridad para satisfacer los requisitos de seguridad de las aplicaciones.
- Usarlo durante la compra de software como requisito, proporcionando una base para especificar los requisitos de verificación de seguridad de la aplicación en los contratos.

Debes conocer
OWASP desarrolla el MASVS o Estándar de Verificación de Seguridad de Aplicaciones Móviles, que es un equivalente al ASVS pero para aplicaciones móviles. Dicho proyecto ha proporcionado tradicionalmente tres niveles de verificación (L1, L2 y R), pero en la última actualización de 2023 han sido reformulados como "perfiles de pruebas de seguridad" y trasladados al OWASP MASTG (en inglés Mobile Application Security Testing Guide).

3.1.- NIVELES DE SEGURIDAD ASVS OWASP

El ASVS versión 4.0.3 define 3 niveles de Seguridad:

ASVS Nivel 1 (Opportunistic):

- Deben cumplirlo todas las aplicaciones.
- Contiene 136 controles.
- Una aplicación alcanza ASVS Nivel 1 si logra defenderse contra vulnerabilidades de seguridad de aplicaciones que son fáciles de descubrir, incluido el Top 10 de OWASP y otras listas de comprobación similares.

ASVS Nivel 2 (Standard)

- Deben cumplirlo las aplicaciones que contienen datos sensibles, y es recomendable para todas las aplicaciones.
- Contiene 267 controles
- Una aplicación alcanza ASVS Nivel 2 (o Estándar) si se defiende adecuadamente contra la mayoría de los riesgos asociados con el software hoy en día.

ASVS Nivel 3 (Advanced).

- Deben cumplirlo aplicaciones críticas por el alto valor de las transacciones (por ejemplo, la bolsa), o por tratar datos especialmente protegidos (por ejemplo, datos médicos) o cualquier aplicación que requiera el más alto nivel de confianza.
- Contiene 286 controles.
- Una aplicación alcanza ASVS Nivel 3 (o Avanzado) si se defiende adecuadamente contra vulnerabilidades avanzadas de seguridad de aplicaciones y también demuestra principios de buen diseño de seguridad.

4.- REQUISITOS DE VERIFICACIÓN NECESARIOS ASOCIADOS AL NIVEL DE SEGURIDAD ESTABLECIDO

Como hemos visto cada nivel ASVS versión 4.0.3 contiene una lista de requisitos de seguridad. Cada uno de estos requisitos también se puede asignar a características y capacidades específicas de seguridad que los desarrolladores deben incorporar al software.

Cada requisito tiene un identificador en el formato (cada elemento es un número):

v<version>-<capitulo>.<seccion>.<requisito>

Por ejemplo: v4.0.2-1.11.3 corresponde a Versión de ASVS 4.0.2

- El capítulo 1 es de Arquitectura.
- La sección 11 del capítulo 1: 1.11. es la sección Requisitos de arquitectura de la lógica de negocio del capítulo Arquitectura.
- El requisito 3 de la sección 11 del capítulo 1, 1.11.3 de esta norma es: Verificar que todos los flujos de lógica de negocio de alto valor, incluyendo la autenticación, la gestión de la sesión y el control de acceso son seguros para los hilos y resistentes a las condiciones de carrera de tiempo de verificación y tiempo de uso.

El listado de controles generales en ASVS versión 4.0.3 es:

V1. Arquitectura, diseño y modelado de amenazas
V2. Autenticación
V3. Gestión de sesiones
V4. Control de acceso
V5. Validación, sanitización y codificación
V6. Criptografía en el almacenamiento
V7. Gestión y registro de errores
V8. Protección de datos
V9. Comunicaciones
V10. Código Malicioso
V11. Lógica de negocio
V12. Archivos y recursos
V13. Servicios Web y API
V14. Configuración

Dentro de cada uno de estos capítulos encontraremos secciones con aspectos más específicos dentro de ese control. Es importante que entiendas no solamente los controles en sí, sino que nivel se necesita cumplir, según el tipo de aplicación que se esté desarrollando. Por otra parte, el nivel 1 se considera el mínimo que debería de cumplir cualquier aplicación, por lo que es importante que te familiarices con este nivel.

Autoevaluación I

Identifica si las siguientes frases son verdaderas o falsas

1- OWASP es una organización que busca obtener beneficios económicos mejorando el software.

a)Verdadero

b)Falso

2- El primer riesgo detectado por OWASP es un diseño inseguro

a)Verdadero

b)Falso

3- OWASP valora vulnerabilidades y sobre todo el acceso a los datos que pudieran provocar.

a)Verdadero

b)Falso

4- Para la elaboración de la lista de OWASP intervienen miles de programadores, empresas y expertos en seguridad y desarrollo.

a)Verdadero

b)Falso

TEST

1- ¿Qué ventajas ofrece el software de fuentes abiertas?:

a) Modelo claro de desarrollo seguro.
b) Reducción de tiempos de acceso al dato.
c) Sostenibilidad.
d) Estabilidad, Seguridad, Colaboración.

2- Podríamos definir el objetivo de OWASP como:

a) Proporcionar una guía de referencia identificando riesgos y controles para las aplicaciones.
b) Ayudar en la comprobación del software.
c) Comprobar aplicaciones web y móviles para mejorar la seguridad del software.
d) Comprobar aplicaciones web para mejorar la seguridad del software.

3- EI TOP10 de OWASP tiene el objetivo de identificar riesgos. ¿Verdadero o falso?

a) Verdadero
b) Falso

4- El objetivo de OWASP es:

a) Trabajar en mejorar la seguridad del software.
b) Evaluar software.
c) Evaluar software y aplicaciones.
d) Evaluar aplicaciones.

5- Cuando el código es accesible los usuarios pueden acceder a él, mejorándolo en muchos casos.

a) Verdadero
b) Falso

6- El nivel 2 del ASVS está dirigido a:

a) Todo tipo de software.
b) Aplicaciones que manejen datos sensibles.
c) Software propietario.
d) Entornos móviles con transacciones económicas.

7- El fallo de Denegación de Servicio está dentro del TOP7 de OWASP. ¿Verdadero o falso?

a) Verdadero
b) Falso

8- OWASP ofrece un Top de riesgos para móviles y para web. ¿Verdadero o falso?

a) Verdadero
b) Falso

9- Los niveles de aplicación definidos en el ASVS tienen en cuenta el dato manejado y el entorno de la aplicación. ¿Verdadero o Falso?

a) Verdadero
b) Falso

10-Los datos personales no condicionan los niveles del ASVS.

a) Verdadero
b) Falso

Respuestas:

Autoevaluación I: 1 b), 2 b), 3 a), 4 a)

TEST: 1 d), 2 a), 3 a), 4 a), 5 a), 6 b), 7 b), 8 a), 9 a), 10 b)

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 us). 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.

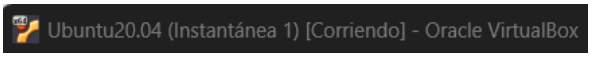
El requisito de ASVS se centra en asegurar que las aplicaciones web no sean susceptibles a estas condiciones de carrera, especialmente en flujos de lógica de negocio críticos como la autenticación, la gestión de sesiones y el control de acceso. A través de prácticas de diseño y verificación robusta se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

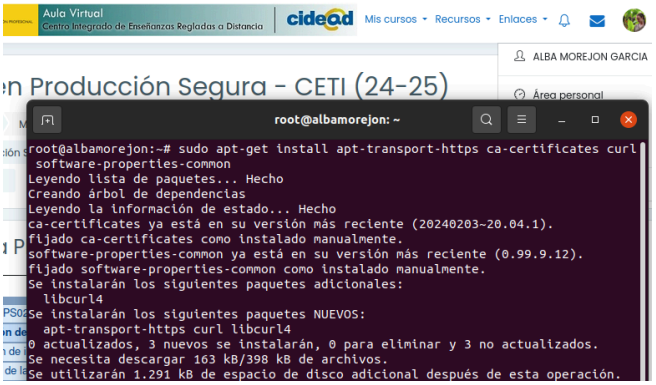
Vamos a utilizar una máquina Ubuntu 20.04



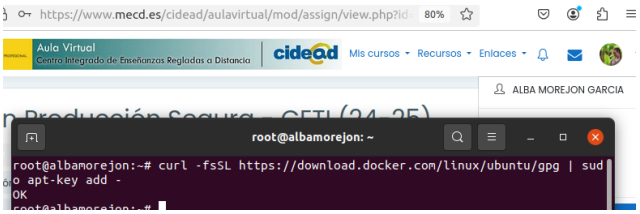
- **Instalar Docker y si es necesario docker compose (en algunas versiones ya viene con docker). En la página oficial de Docker explican cómo instalarlo.**

1. Configuramos el repositorio

Instalamos paquetes necesarios con el comando
”sudo apt-get install apt-transport-https ca-certificates curl software-properties-common”.



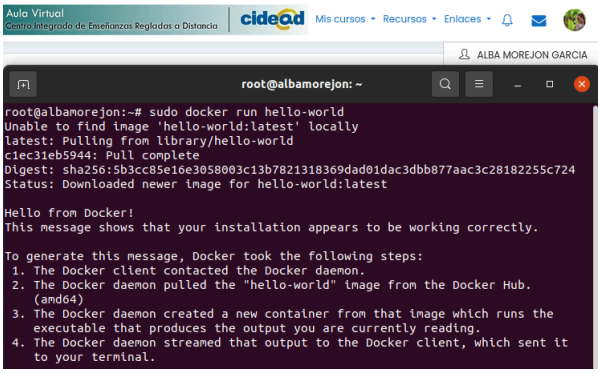
Agregamos clave gpg de docker
“curl -fsSL https://download.docker.com/linux/ubunut/gpg | sudo apt-ky add -”.



Agregamos el repositorio de docker a APT
“sudo add-apt-repository “deb [arch=amd64] https://download.docker.com/linux/ubunut $(lsb_release -cs) stable”.

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”.

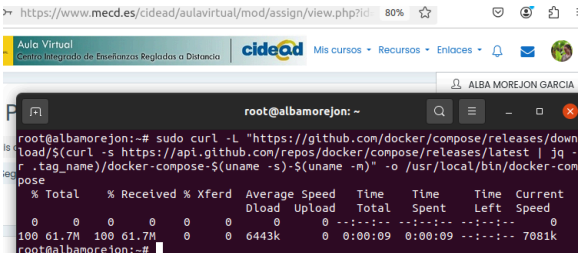


3. Instalar Docker Compose

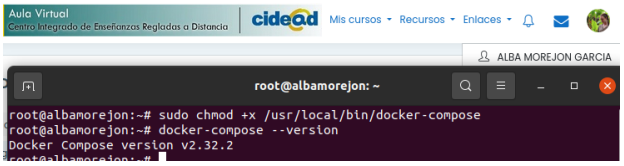
Instalamos primero el paquete jp con el comando “sudo apt install jq”.

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”.

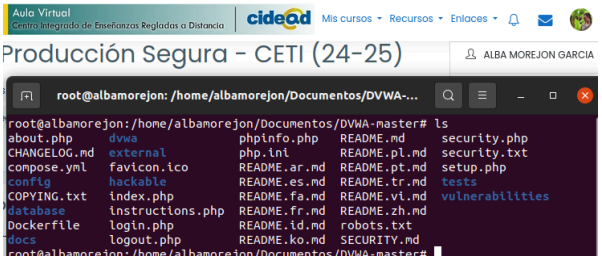


Damos permisos de ejecución “sudo chmod +x /usr/local/bin/docker-composer” y verificamos la versión.



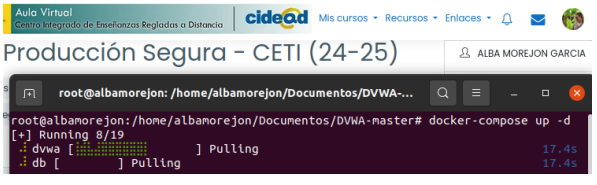
- **Descargar el proyecto DVWA y descomprimirlo.**

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



- **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.

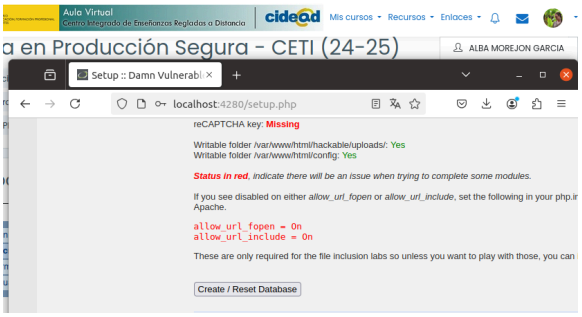


- **Una vez instalado, para acceder a DVWA: Abrir un navegador con la dirección http://localhost:4280, el usuario es admin y la clave es admin**

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

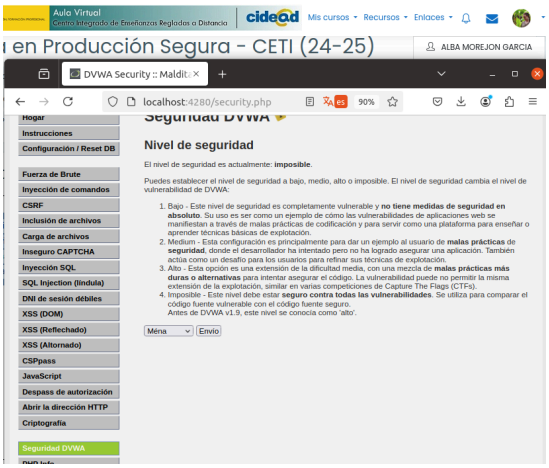


- **A partir de ese momento el usuario es admin y la clave es password**

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**

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';

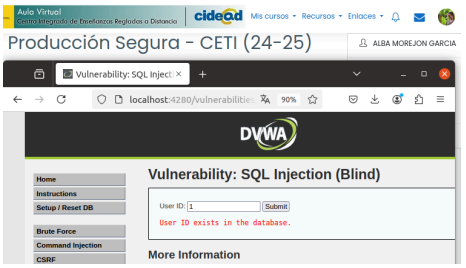
Debido a que lo introducido es una condición que siempre es verdadera, la consulta devuelve todos los registros de la tabla "users" y debido a que el valor $exist será mayor a 1, se devuelve el mensaje mostrado.

```
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:

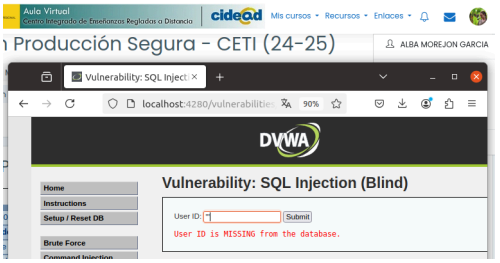
```
if(is_numeric( $id )) {
    $id = intval ($id);
```

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();
```

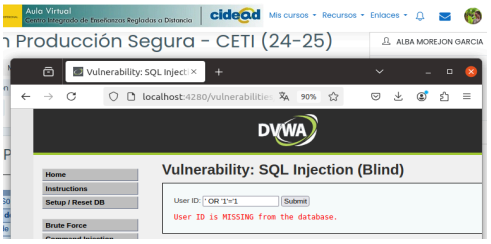
Mostrar resultado:

```
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 cumpla 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?

La diferencias clave entre ambos códigos es que en el código en modo Impossible se verifica el token Anti-CSPF 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 la 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.