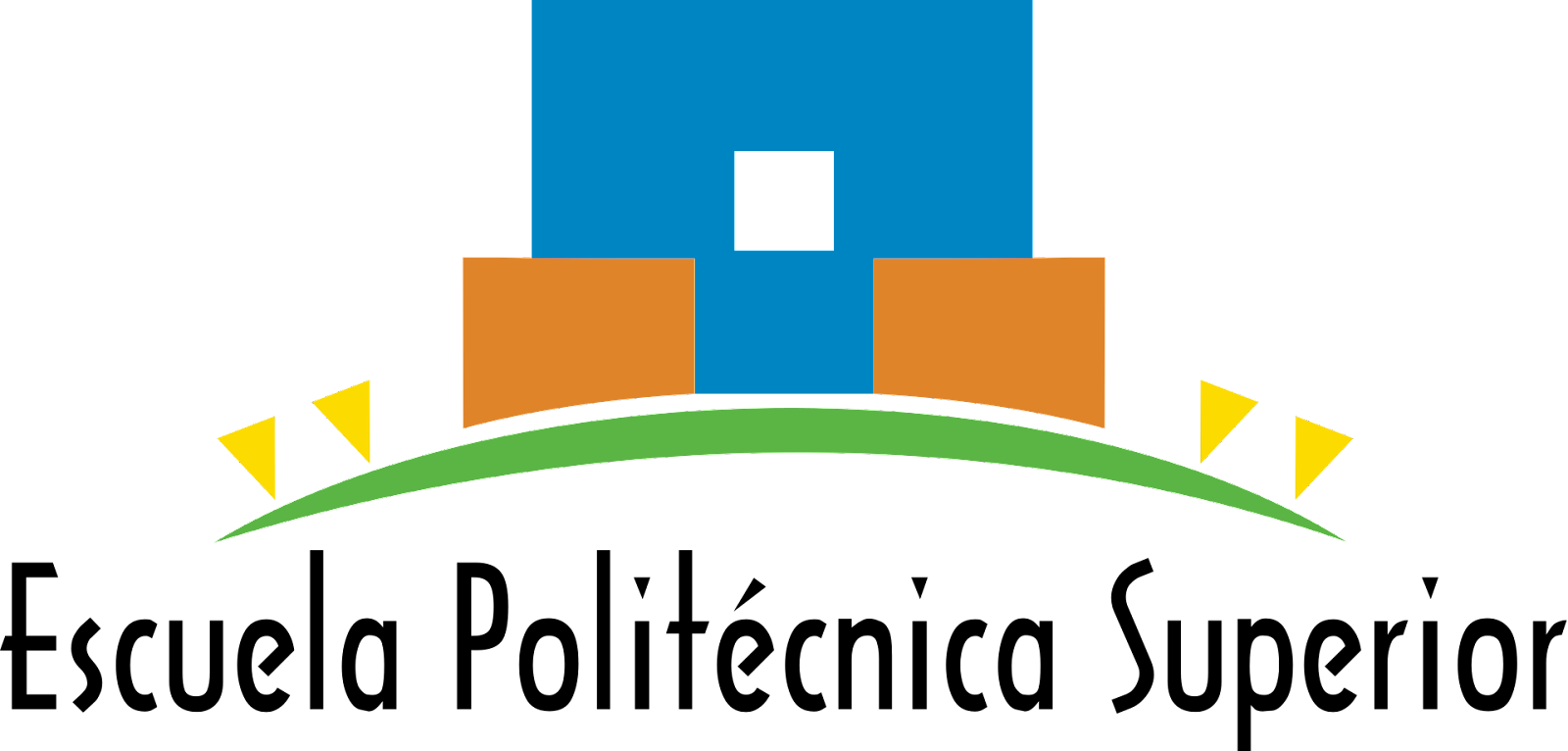
**UNIVERSIDAD AUTÓNOMA DE MADRID**



**Prácticas Ciberseguridad**

**Práctica 3 - Atacando una aplicación Web**

Miguel Ibáñez González

Óscar Navalón Navarro

10/05/2024

Grupo: 2462

Pareja: 05

**ÍNDICE**

[**● Introducción 2**](#_aw8ykr5wvcg3)

[**● XSS 2**](#_yuv8m1ectg7t)

[**● Acceso no autorizado 5**](#_jtq0c4hdk01v)

[**● Falta de control de acceso directo a recursos 7**](#_is54ei5ygsvx)

[**● Insecure Debug Enabled 9**](#_ujpx901h5q8d)

[**● Content Security Policy no está configurado 11**](#_pbqgakuky0ry)

[**● X-Content-Type-Options no está configurado 13**](#_vscmjqp23zee)

[**● Strict-Transport-Security no está configurado 14**](#_oizfha38uu3v)

[**● El indicador HttpOnly no está configurado en la cookie 15**](#_4cn47j164181)

[**● La bandera segura no está configurada en la cookie: csrftoken 16**](#_nwweqqinlx4g)

[**● Límite de tamaño de archivos no configurado 17**](#_jowzcyy1e3y5)

[**● Privilegios de administrador no configurados. 19**](#_xhmok04ilcbq)

[**● SQL INJECTION: 20**](#_28lx18m63bqw)

# 

# Introducción

El objetivo de esta práctica es aprender seguridad ofensiva para descubrir vulnerabilidades antes de que sean explotadas.

Vamos a comprobar si la aplicación de la compañía Tannen Corp es segura frente a atacantes.

Tenemos una versión candidata en formato de máquina virtual que vamos a lanzar en nuestro equipo y atacar para sacar todos los problemas a la luz.

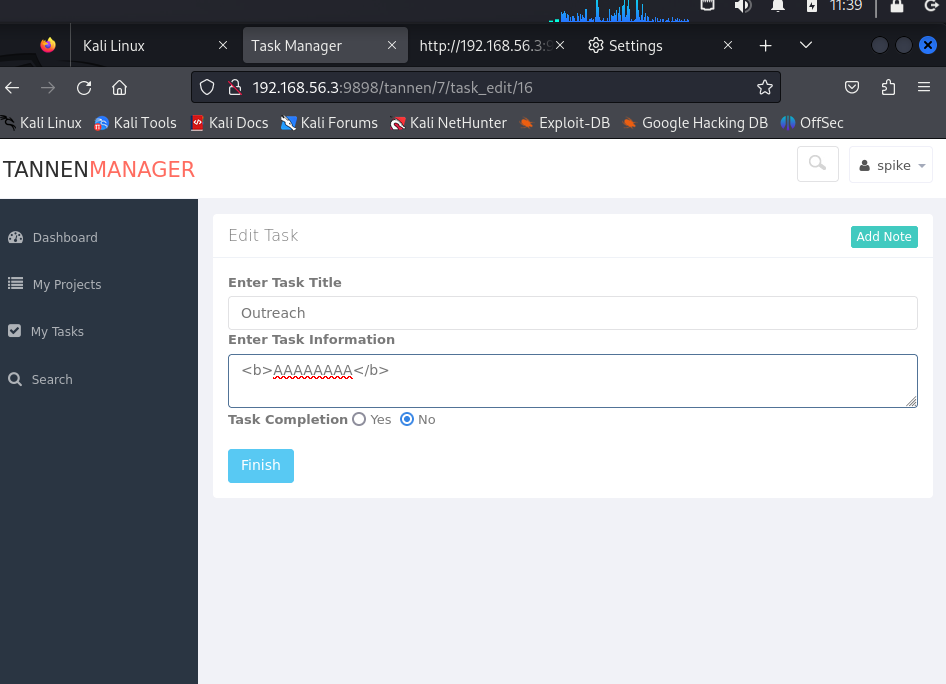
El objetivo es encontrar 10 vulnerabilidades susceptibles de ser atacadas, en nuestro caso hemos encontrado 12 vulnerabilidades susceptibles a ataques, para cada una de las vulnerabilidades, hemos realizado su explotación y hemos reportado los resultados obtenidos. También para cada una de ellas proponemos posibles soluciones entre las principales causas comunes de ese tipo de vulnerabilidad.

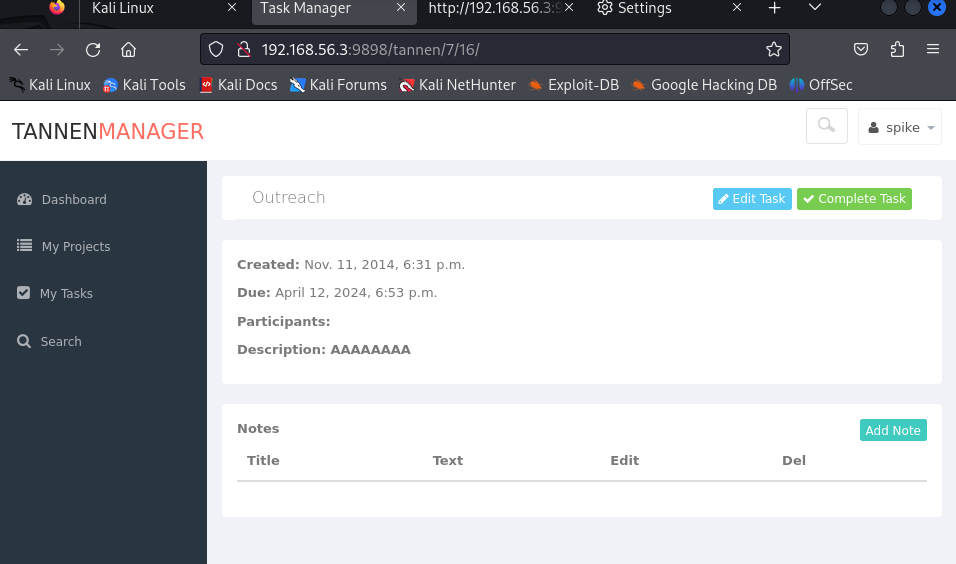
# XSS

XSS (Cross-Site Scripting) es una vulnerabilidad de seguridad común en aplicaciones web. Se produce cuando un atacante inyecta código malicioso en páginas web visitadas por otros usuarios.

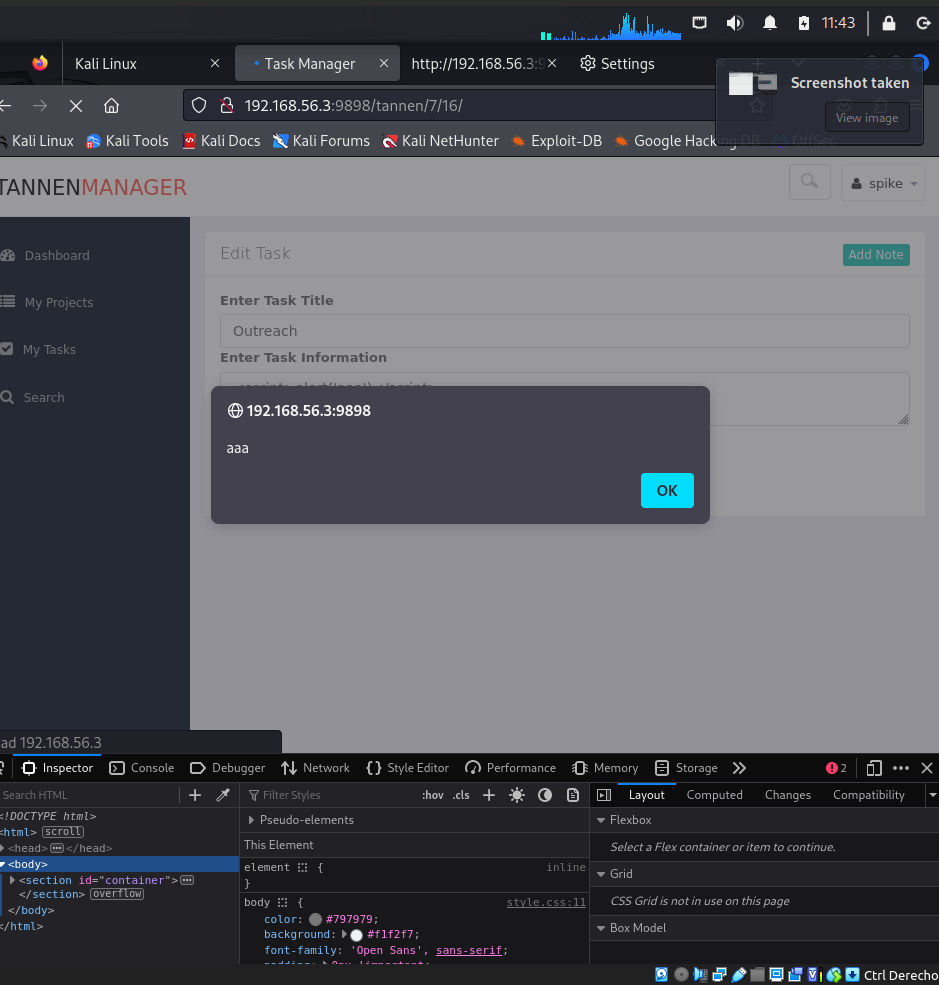
* Explotación de la vulnerabilidad

En primer lugar probamos a ver si la página es vulnerable a XSS poniendo un comentario en negrita con las etiquetas <b>.





Una vez visto que esto es posible, probamos el script alert (aaa), y como podemos comprobar que el sitio sí que es vulnerable a XSS.



Wapiti nos avisa también de la vulnerabilidad:

* Mitigar la vulnerabilidad:

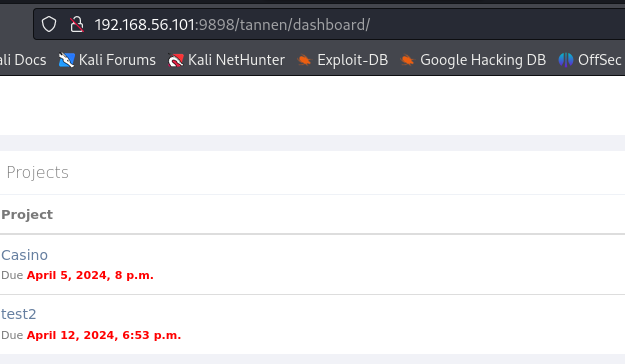
Ya hemos identificado que la página web es vulnerable a XSS mediante la inserción de etiquetas HTML como <b> y la ejecución de un script de alerta. Para solucionar este problema hay varias opciones y medidas que se deben tomar:

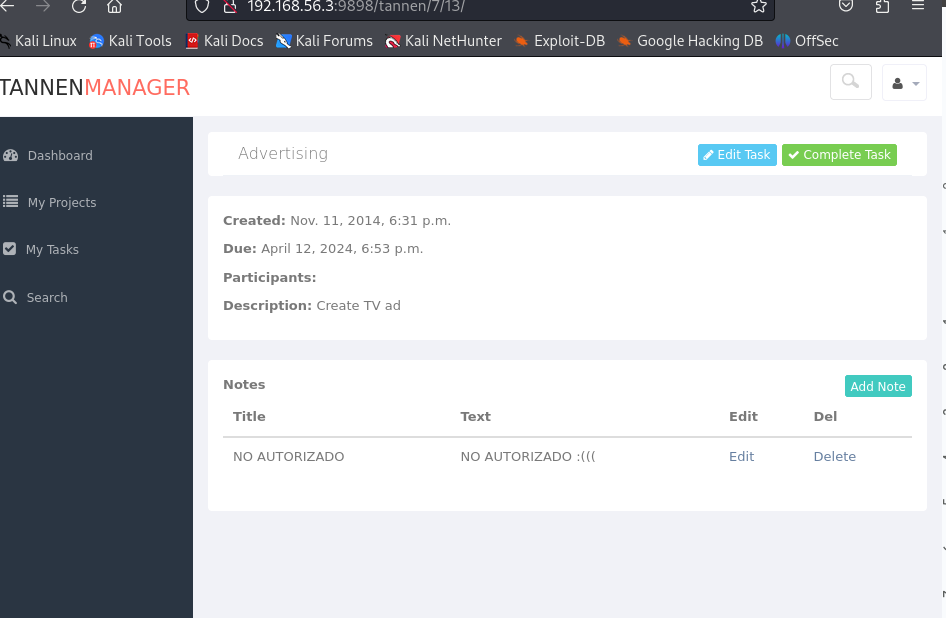
1. Validar entradas de usuario: hay que asegurarse de que todas las entradas de usuario (como campos de formularios, URL, parámetros de consulta, etc.) sean validados adecuadamente para eliminar cualquier código malicioso. Como lo pueden ser el uso de funciones como htmlspecialchars() en PHP o librerías similares en otros lenguajes.
2. Implementar encabezados de seguridad: se deben configurar encabezados de seguridad como Content Security Policy (CSP) para limitar qué recursos externos pueden ser cargados y ejecutados en el sitio web de Tannen Corp, lo que ayudará a prevenir la ejecución de scripts no autorizados.
3. Codificación adecuada de datos: hay que codificar correctamente todos los datos que se muestran dinámicamente en la página web para evitar la interpretación indebida de caracteres especiales como <, >, &, etc. Para ello se pueden usan funciones de codificación como encodeURIComponent().

# Acceso no autorizado

La vulnerabilidad de acceso no autorizado es muy peligrosa ya que puede ser explotada por cualquier persona que tenga acceso a la URL, lo que significa que no se requiere una credencial de usuario válida para acceder a ciertas partes del sitio web que deberían estar protegidas.

* Explotación de la vulnerabilidad

El problema en la página web es que se puede acceder a <http://192.168.56.101:9898/tannen/dashboard/> sin necesidad de loguearse.

A partir de aquí se puede explotar esta vulnerabilidad de muchas maneras ya que tienes acceso a todo sin necesidad de loguearse. Podemos editar los datos de las tareas y proyectos que estén publicados.

* Mitigar la vulnerabilidad

Para solucionar esta vulnerabilidad y garantizar que solo los usuarios autorizados tengan acceso a las áreas protegidas del sitio web, existen varias opciones:

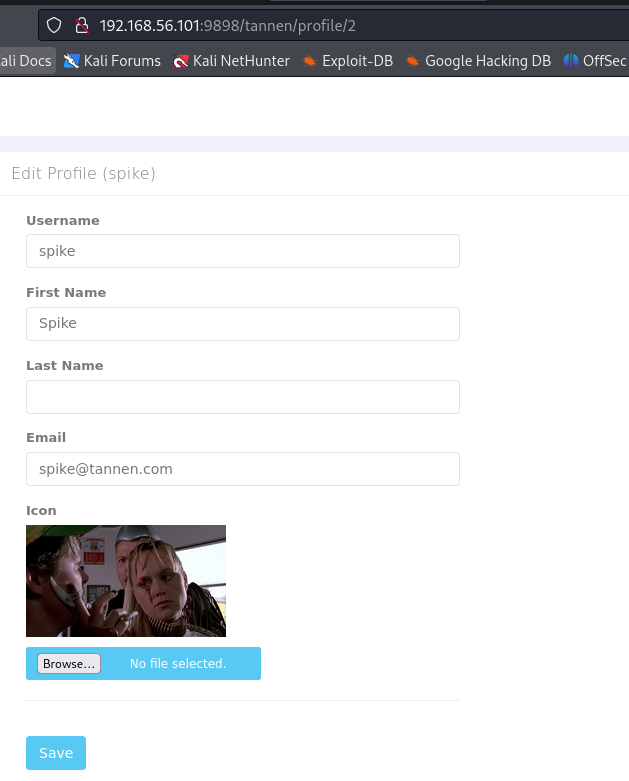
1. Implementar un sistema de autenticación: hay que asegurarse de que todas las páginas que requieran autenticación implementen un sistema de inicio de sesión seguro. Para ello se debe comprobar que los usuarios ingresen un nombre de usuario y una contraseña válidos antes de acceder al contenido protegido.
2. Configurar adecuadamente los controles de acceso: se deben utilizar mecanismos de control de acceso para restringir el acceso solo a usuarios autenticados y autorizados. Lo que implica que configurar los permisos y roles para determinar quién tiene acceso a qué recursos.
3. Proteger las rutas y URL sensibles: hay que asegurarse de que las rutas y URL que requieran autenticación estén protegidas adecuadamente para evitar el acceso no autorizado. Para ello se deben de configurar las reglas de acceso en el servidor web y en la aplicación para requerir autenticación antes de permitir el acceso a ciertas rutas.

# Falta de control de acceso directo a recursos

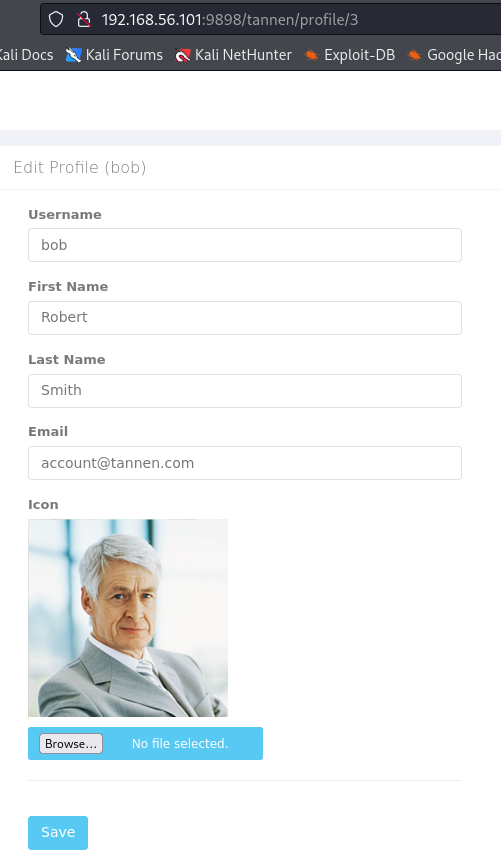
La vulnerabilidad de "falta de control de acceso directo a recursos" ocurre cuando la aplicación web no implementa adecuadamente los controles de acceso para proteger los recursos sensibles, como los perfiles de usuario, y permite el acceso a esos recursos simplemente manipulando la URL.

* Explotación de la vulnerabilidad

El problema en el caso de la página web de Tannen Corp es que una vez logueado con el usuario que se nos proporciona “spike / gang”, si nos metemos en el perfil nos aparece la url: <http://192.168.56.101:9898/tannen/profile/2>



Simplemente cambiando la url por [http://192.168.56.101:9898/tannen/profile/](http://192.168.56.101:9898/tannen/profile/2)3, podemos meternos en otro perfil al cual no deberíamos de tener acceso, y a partir de ahí, podemos explota la vulnerabilidad cambiando la contraseña de ese perfil y robandole la cuenta.



* Mitigar la vulnerabilidad

Para solucionar esta vulnerabilidad se pueden tomar varias medidas:

1. Implementar controles de acceso adecuados: hay que asegurarse de que la aplicación web implemente un sistema de control de acceso sólido que valide la autenticación y autorización de cada solicitud de acceso a recursos sensibles, como perfiles de usuario. Para ello se debe incluir una verificación de permisos y roles para determinar si un usuario tiene el derecho de acceder a un perfil específico.
2. Protege las rutas y URL sensibles: se debe configurar la aplicación para que valide la autenticación y autorización en el servidor antes de servir cualquier recurso sensible. Para ello podemos usar un middleware de seguridad o filtros en el servidor web para verificar la identidad del usuario y sus permisos antes de procesar la solicitud.
3. Utilizar identificadores opacos: en lugar de utilizar identificadores numéricos simples en las URL (como "/profile/2" y "/profile/3"), se deberían de usar identificadores opacos o tokens únicos que no sean predecibles ni fácilmente manipulables por los usuarios. Esto dificultará la suplantación de identidad y el acceso no autorizado a través de la URL.

# Insecure Debug Enabled

Configurar DEBUG en True en un entorno de producción de Django es una vulnerabilidad de seguridad importante. Cuando DEBUG está habilitado en Django, se activa el "modo de depuración", que puede revelar información sensible sobre la aplicación, incluyendo el código fuente, trazas de pila y mensajes de error detallados. Esta vulnerabilidad la podría aprovechar un atacante para obtener información privilegiada sobre la estructura interna de la aplicación. El modo de depuración está destinado para uso durante el desarrollo y no debe estar habilitado en un entorno de producción. En producción, siempre se debe establecer DEBUG en False. Como se puede observar en la imagen TEMPLATE\_DEBUG es True en la configuración del django, lo cual permite ver parte de código fuente.

* Explotación de la vulnerabilidad

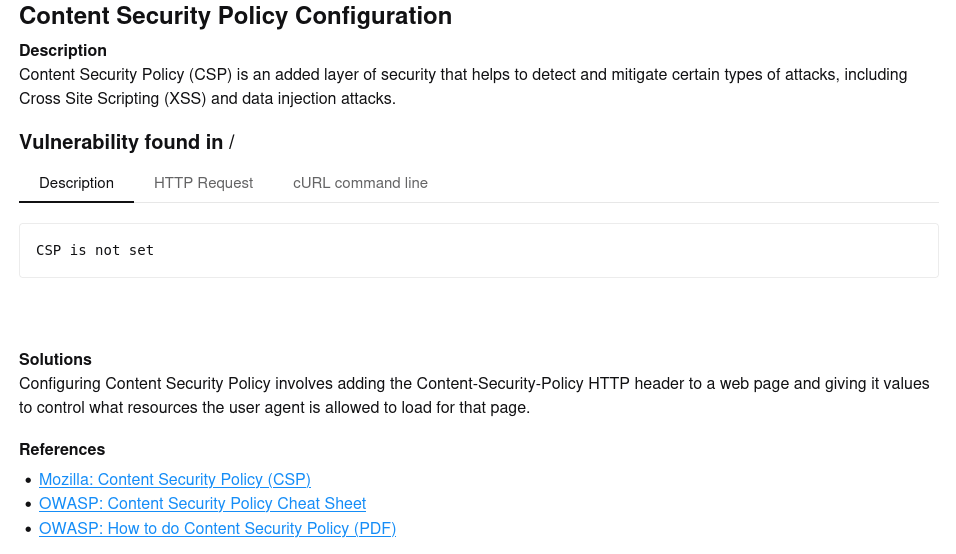
Para explotar esta vulnerabilidad hemos realizado varios ataques:

1. Obtener información privilegiada: al tener acceso al código fuente y a los mensajes de error detallados, podemos obtener información privilegiada sobre la arquitectura y la lógica de la aplicación. Esto nos da acceso a información sobre bases de datos, servicios externos, claves de API, entre otros datos sensibles.
2. Explotar errores de seguridad específicos: al ver los mensajes de error detallados, podemos explotar otras vulnerabilidades a partir de esta, como inyecciones SQL, inyecciones de comandos, vulnerabilidades de desbordamiento de búfer...

* Mitigar la vulnerabilidad

La solución es tan sencilla como cambiar TEMPLATE\_DEBUG a False.

# Content Security Policy no está configurado

Ejecutando el comando wapiti -u <http://192.168.56.101:9898>, wapiti genera un informe de las vulnerabilidades que encuentra, una de ellas es:

La vulnerabilidad que menciona el informe de wapiti, "CSP is not set", se refiere a la falta de una Política de Seguridad de Contenidos. La CSP es un mecanismo de seguridad que ayuda a mitigar los riesgos de ataques.

* Explotación de la vulnerabilidad
* XSS (Cross-Site Scripting): inyectamos scripts maliciosos en formularios, campos de entrada o en URLs, aprovechando así la falta de restricciones impuestas por una CSP.
* Clickjacking: superponemos contenido malicioso sobre el sitio web objetivo, engañando a los usuarios para que hagan clic en elementos que realizan acciones no deseadas.
* Inyección de contenido no deseado: inyectamos contenido no deseado en el sitio web objetivo, como anuncios, redirecciones o contenido ofensivo.

Para explotar la vulnerabilidad hemos creado una página web maliciosa que incluye un iframe que cargue la página 192.168.56.101:9898 como su contenido principal:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Ataque de Clickjacking</title>

<style>

iframe {

position: absolute;

width: 100%;

height: 100%;

border: none;

opacity: 0.5; /\* Opacidad para que el usuario no note el iframe \*/

z-index: 1; /\* Colocamos el iframe por encima del contenido superpuesto \*/

}

.overlay {

position: absolute;

width: 100%;

height: 100%;

z-index: 2; /\* Colocamos el contenido superpuesto por encima del iframe \*/

}

.overlay-content {

position: absolute;

transform: translate(-50%, -50%);

text-align: center;

font-size: 24px;

color: red;

}

</style>

</head>

<body>

<iframe src="http://192.168.56.101:9898" frameborder="0"></iframe>

<div class="overlay">

<div class="overlay-content">Haz clic aquí para reclamar tu premio</div>

</div>

</body>

</html>

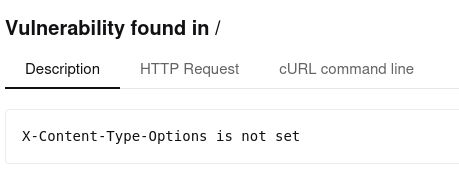
Esta página la podemos enviar a un usuario que verá la página web objetivo (192.168.56.101:9898) superpuesta con un mensaje o contenido persuasivo. Cuando haga click para interactuar con el contenido superpuesto, en realidad estará haciendo click en nuestra página web.

* Mitigar la vulnerabilidad

La solución es tan fácil como configurar la política de seguridad de contenido agregando el encabezado HTTP Content-Security-Policy a una página web y darle valores para controlar qué recursos el agente de usuario puede cargar para esa página.

# X-Content-Type-Options no está configurado

Wapiti nos alerta de la siguiente vulnerabilidad en su informe:



Como podemos ver, nos avisa de que "X-Content-Type-Options" no está configurado, esto permite que los navegadores puedan intentar adivinar el tipo de contenido de una respuesta HTTP si no se especifica explícitamente. Lo que permite que se puedan realizar ataques de tipo de contenido (Content-Type), donde el navegador interpreta incorrectamente el tipo de contenido del recurso, permitiendo que un atacante ejecute código malicioso en el contexto del sitio.

* Explotación de la vulnerabilidad

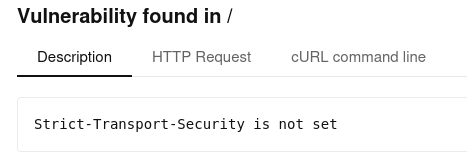
Para explotarla enviamos archivos maliciosos (por ejemplo, archivos HTML o JavaScript) con encabezados HTTP manipulados para que los navegadores los interpreten de manera incorrecta, lo que nos permite ejecutar código malicioso en el navegador de la víctima.

* Mitigar la vulnerabilidad

Para mitigar esta vulnerabilidad, debemos de configurar adecuadamente el encabezado HTTP "X-Content-Type-Options" en la aplicación web. Para ello debemos de agregar el encabezado HTTP "X-Content-Type-Options" en la configuración del servidor web. El valor recomendado es "nosniff", que indica al navegador que no debe realizar el MIME sniffing y debe respetar el tipo de contenido declarado en el encabezado "Content-Type".

# Strict-Transport-Security no está configurado

Wapiti nos alerta de la siguiente vulnerabilidad en su informe:



La vulnerabilidad "Strict-Transport-Security (STS) not set" se refiere a la falta de configuración de la política de seguridad de transporte estricta en un sitio web. Esta política es una medida de seguridad completamente necesaria que protege a los usuarios contra ataques de tipo Man-in-the-Middle (MITM) y asegura que la comunicación entre el navegador y el servidor se realice a través de una conexión segura utilizando HTTPS. En la página de Tannen dado que STS no está configurado, los navegadores no reciben instrucciones específicas para solicitar una conexión segura a través de HTTPS en futuras visitas al sitio. Lo que permite que los usuarios sean vulnerables a ataques donde un atacante puede interceptar la comunicación entre el navegador y el servidor, comprometiendo la seguridad de los datos transmitidos.

* Explotación de la vulnerabilidad

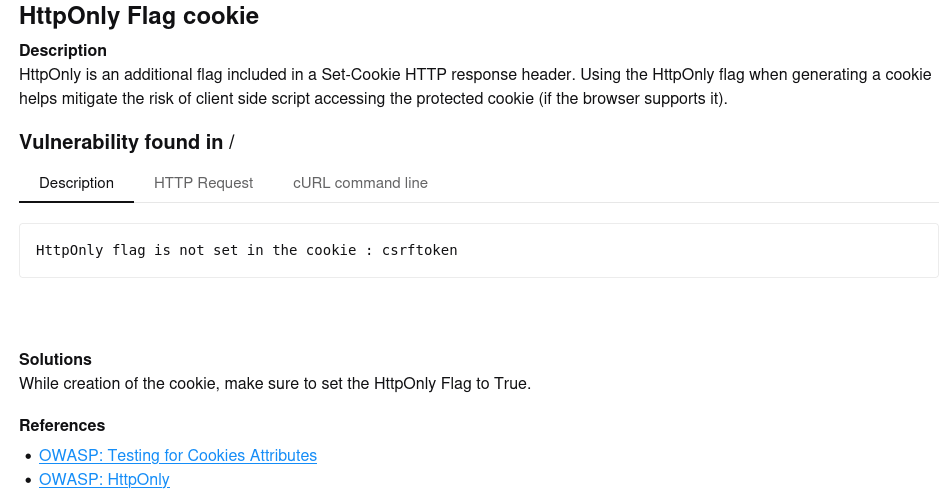
Para explotar esta vulnerabilidad tenemos varias ideas:

* Ataques de intercepción de datos: podemos aprovechar la falta de STS para interceptar las solicitudes HTTP del usuario y redirigirlas a un sitio web malicioso o comprometido.
* Ataques de sesión: podemos lanzar ataques de secuestro de sesión, donde interceptamos las cookies de sesión del usuario y las utilizamos para suplantar la identidad del usuario.
* Phishing y ataques de redireccionamiento: podemos utilizar la falta de STS para crear páginas de phishing que imitan al sitio legítimo y engañar a los usuarios para que ingresen información confidencial.
* Mitigar la vulnerabilidad

La solución es configurar la política de seguridad de transporte estricta (STS) en el servidor web. Esto se puede hacer agregando el encabezado HTTP Strict-Transport-Security a las respuestas del servidor. Este encabezado especifica un período de tiempo durante el cual el navegador debe acceder al sitio solo a través de HTTPS.

# El indicador HttpOnly no está configurado en la cookie

Wapiti nos alerta de la siguiente vulnerabilidad en su informe:



La vulnerabilidad "HttpOnly flag not set in the cookie" se refiere a la falta de configuración de la bandera HttpOnly en una cookie específica, como el token CSRF (Cross-Site Request Forgery). La bandera HttpOnly es una medida de seguridad que se puede establecer en una cookie para indicar al navegador que la cookie no debe ser accesible a través de scripts del lado del cliente. De esta manera prevenimos ataques de secuestro de cookies y protegemos la información sensible almacenada en ellas.

* Explotación de la vulnerabilidad

Para explotarla podemos ejecutar scripts en el lado del cliente, para acceder y robar la cookie que no tiene configurada la bandera HttpOnly, lo cual nos permite comprometer la sesión del usuario y realizar acciones en nombre del usuario o acceder a información sensible almacenada en la cookie.

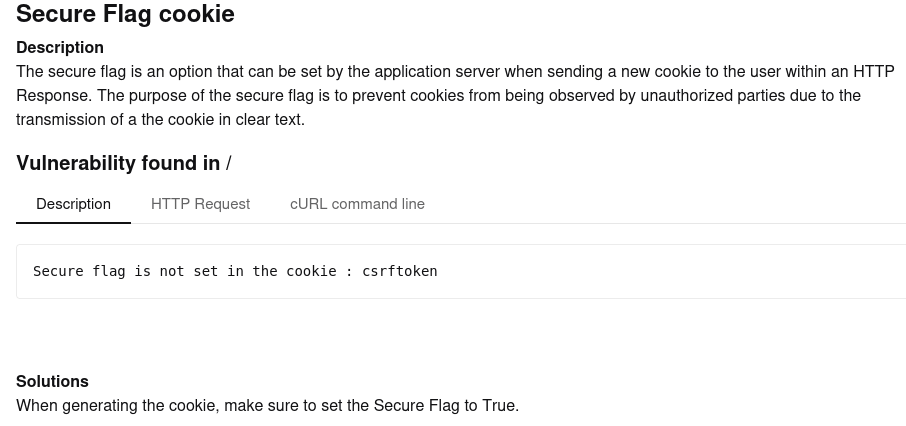
* Mitigar la vulnerabilidad

La solución es configurar la bandera HttpOnly como true en la cookie. Se puede hacer al establecer la cookie en el servidor con la opción HttpOnly.

Ejemplo de django: response.set\_cookie('csrftoken', value, httponly=True)

# La bandera segura no está configurada en la cookie: csrftoken

Wapiti nos alerta de la siguiente vulnerabilidad en su informe:



La vulnerabilidad "Secure flag not set in the cookie: csrftoken" se refiere a la falta de configuración de la bandera "Secure" en una cookie específica, como el token CSRF (Cross-Site Request Forgery). La bandera "Secure" es una medida de seguridad que se puede establecer en una cookie para indicar al navegador que solo debe enviar la cookie a través de una conexión segura HTTPS. Ayuda a prevenir la exposición de la cookie a ataques de interceptación de tráfico en redes no seguras.

* Explotación de la vulnerabilidad

Para explotarla podemos interceptar el tráfico de red en una conexión no segura y capturar la cookie que no tiene configurada la bandera "Secure", lo que nos permite acceder al token CSRF y realizar acciones en nombre del usuario.

* Mitigar la vulnerabilidad

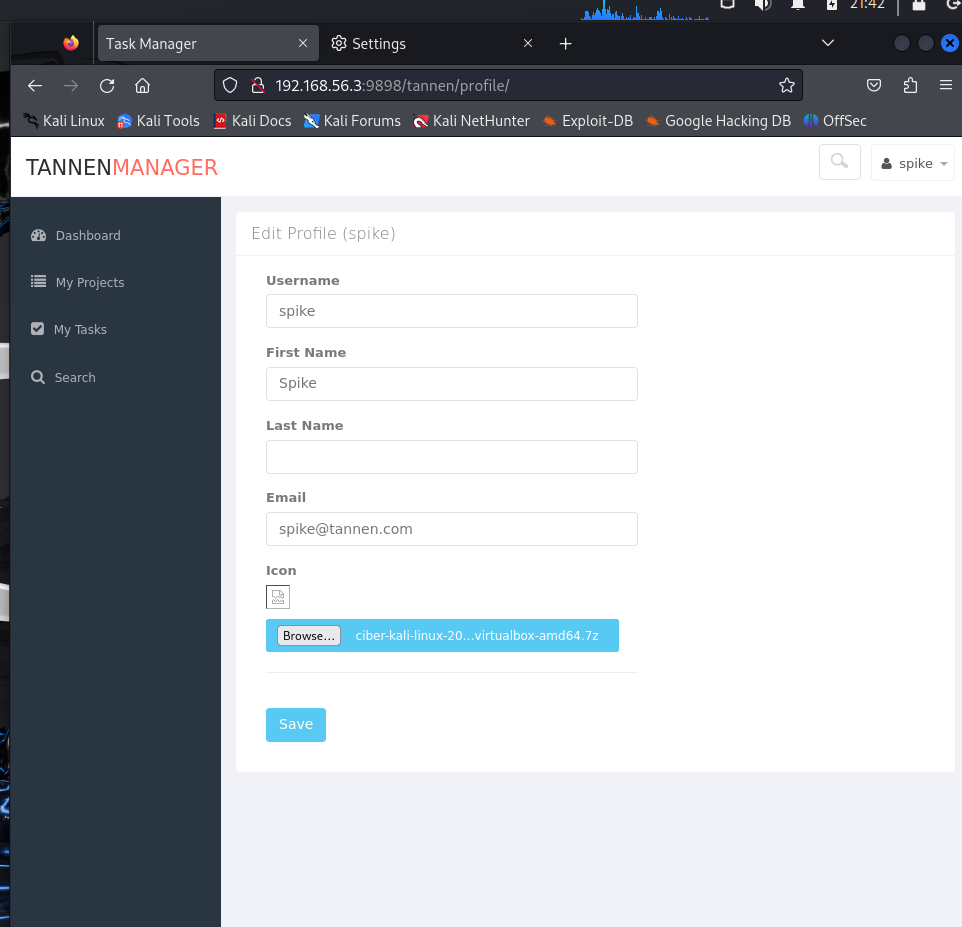
La solución es configurar la bandera "Secure" en la cookie. Se puede hacer al establecer la cookie en el servidor con la opción "secure=True". Ejemplo django: “response.set\_cookie('csrftoken', value, secure=True)”

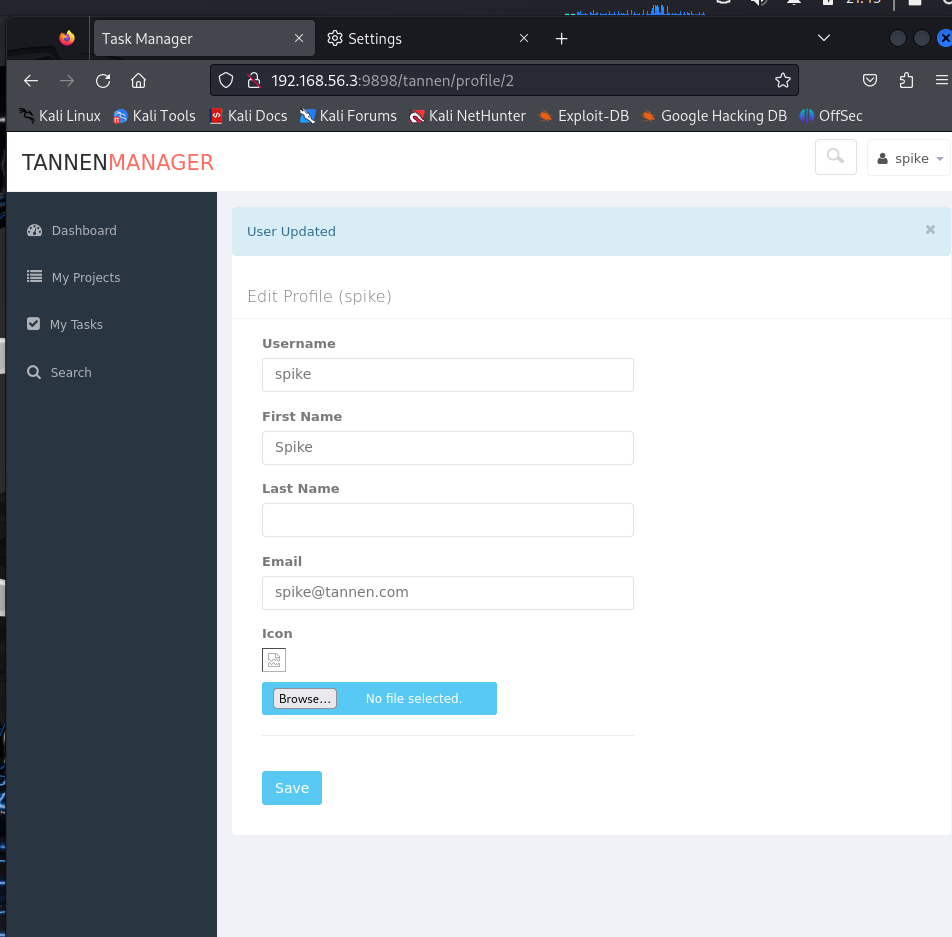
# Límite de tamaño de archivos no configurado

No existe control respecto al tamaño del archivo que se puede subir como foto de perfil. Por lo general, los sitios web deben imponer límites de tamaño de archivo para evitar que los usuarios suban archivos muy grandes que podrían consumir muchos recursos del servidor o incluso provocar ataques de denegación de servicio (DoS).

* Explotación de la vulnerabilidad

Para explotar esta vulnerabilidad podemos subir un archivo de un tamaño enorme al servidor para sobrecargarlo.



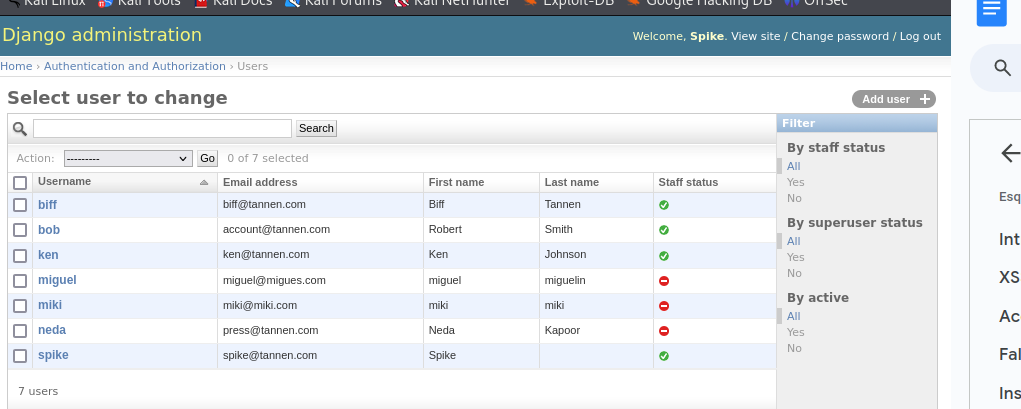


* Mitigar de la vulnerabilidad

La solución para evitar la explotación de esta vulnerabilidad es implementar controles adecuados para limitar el tamaño de los archivos que se pueden subir como fotos de perfil, como establecer un límite de tamaño máximo y validar los archivos cargados para garantizar que sean de tipo seguro y no contengan contenido malicioso.

# Privilegios de administrador no configurados.

El usuario spike, el cual no debería de tener privilegios de administrador, en realidad si los tiene, pudiendo hacer cambios en la base de datos y viendo información confidencial. Esta vulnerabilidad implica una incorrecta gestión de los privilegios de usuario en un sistema.



* Explotación de la vulnerabilidad

Para explotar esta vulnerabilidad, desde el usuario "spike" podemos realizar varias acciones:

.

* Realizar cambios en la base de datos: podemos modificar datos sensibles, como información de usuarios o registros financieros.
* Acceder a información confidencial: podemos acceder a información confidencial que normalmente estaría restringida para los usuarios no administradores.
* Mitigar de la vulnerabilidad

Para solucionar esta vulnerabilidad, podemos implementar siguientes medidas:

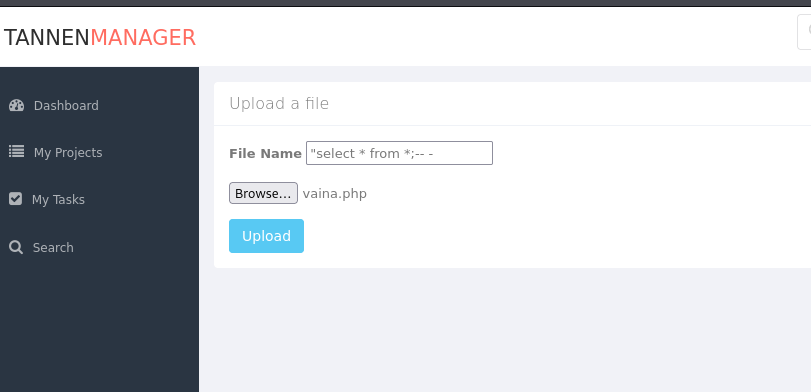
* Revisar y corregir los permisos de usuario: debemos revisar la configuración de permisos para asegurarse de que los usuarios solo tengan los privilegios necesarios para realizar sus funciones. Es decir debemos de eliminar cualquier acceso de usuario que no sea necesario para evitar que usuarios no autorizados obtengan privilegios de administrador.
* Implementar controles de acceso adecuados: debemos implementar controles de acceso adecuados para restringir el acceso a información confidencial y funcionalidades de administrador a los usuarios que no lo necesitan.

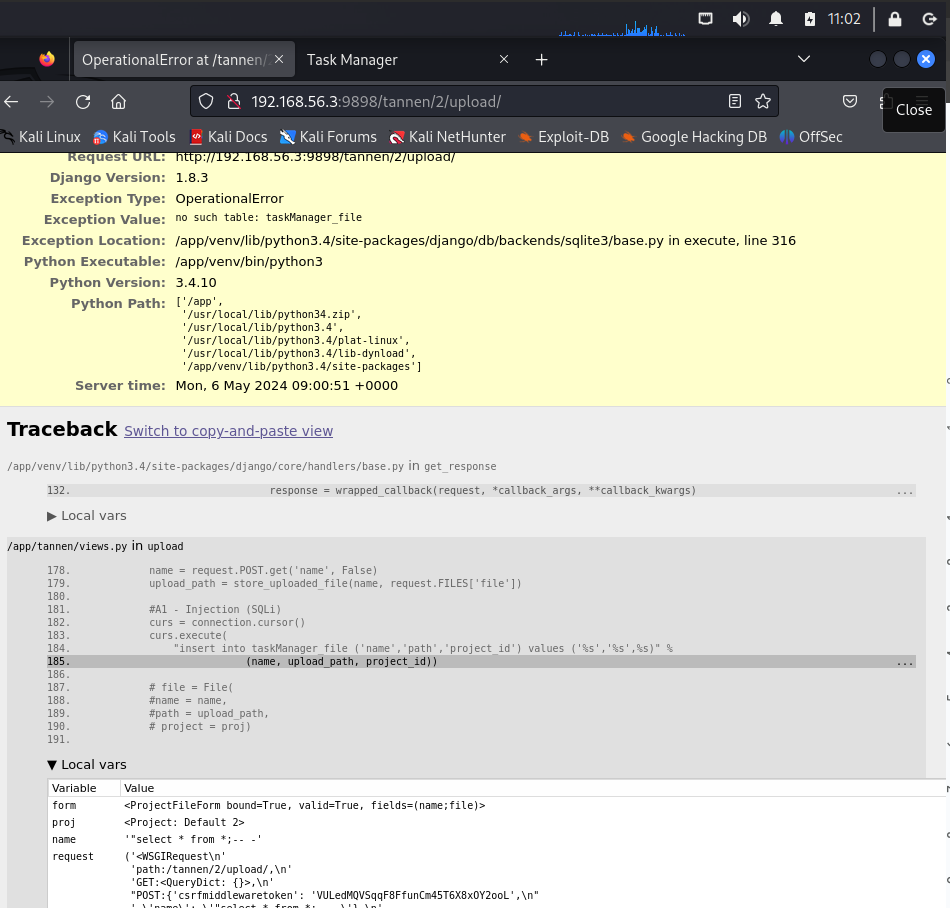
# SQL INJECTION:

La inyección SQL es un tipo de ataque en el que se inyecta una consulta SQL malintencionada a través de datos proporcionados por el cliente hacia una aplicación. La base de datos puede ejecutar la consulta SQL inyectada, lo que permite al atacante evitar las restricciones de seguridad, leer, modificar o borrar datos y realizar operaciones administrativas en la base de datos. Como resultado, esto puede llevar a obtener acceso no autorizado a información confidencial, pérdida de datos y compromiso de la integridad de los mismos.

* Explotación de la vulnerabilidad

La explotación ha consistido en la siguiente inyección de código sql, Podemos ver en el debugger de django como esto consta de una inyección SQL.





* Mitigar la vulnerabilidad

Algunas de las soluciones para mitigar de la inyección SQL son:

* Preparar consultas y parámetros: asegura que los valores ingresados sean tratados únicamente como datos y no como parte del código SQL.
* Validar y limpiar las entradas: con esto nos aseguramos de que cumplan con los requisitos esperados (por ejemplo, tipos de datos, formato), limpiar las entradas sirve para eliminar elementos maliciosos potenciales.
* Uso de listas de control de acceso (ACL): limitar los derechos de usuario en la base de datos solo a lo necesario para realizar sus funciones.
* Gestión de errores y mensajes: configurar la aplicación para que maneje los errores sin revelar información detallada sobre la base de datos.