Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

ÁREA DE ARQUITECTURA

Y

TECNOLOGÍA DE COMPUTADORES

HERRAMIENTA DE ANÁLISIS DE VULNERABILIDADES EN PÁGINAS WEB

DOCUMENTO Nº1

MEMORIA

AUTOR: D. CALVO RUBIO, RODRIGO

TUTOR: D. JOAQUÍN ENTRIALGO CASTAÑO

FECHA: Julio 2020

Índice

1. Introducción

El presente documento contiene la memoria del proyecto "*Multi-Web Vulnerability Scanner*", una aplicación cliente-servidor diseñada para realizar análisis de vulnerabilidades de páginas web.

Hoy en día, la seguridad constituye una parte fundamental en cualquier tipo de infraestructura tecnológica.

Los innumerables avances tecnológicos dan lugar a un gran número de incompatibilidades y errores lo que conlleva nuevas posibilidades de afectar a la eficiencia de las aplicaciones.

Al objeto de asegurar la funcionalidad de un sistema existen una serie de herramientas que se caracterizan por tratar de encontrar posibles fallos o inconsistencias en las diferentes componentes de una aplicación web.

La motivación de este proyecto es el diseño de una aplicación libre que permita al usuario realizar análisis personalizados de vulnerabilidades de páginas web utilizando varias de las herramientas de análisis de vulnerabilidades disponibles de manera gratuita y, de esta forma, proporcionar el mayor nivel de seguridad en su trabajo.

Como objetivo principal se puede decir que se basa en una aplicación web capaz de lanzar diferentes herramientas de análisis de vulnerabilidades, desplegadas en un servidor y obtener el resultado de éstas. Estas herramientas son las encargadas de realizar el escaneo en sí, por lo que habrá que tener en cuenta cómo funciona realmente cada herramienta y adaptar el resultado a la estructura de resultados de la aplicación.

Una de las principales características es la flexibilidad de la aplicación para poder tanto implementar nuevas herramientas como eliminar las menos oportunas o que se consideren obsoletas; de este modo, se pueden actualizar de forma dinámica los recursos de la aplicación, dotándola de una mayor longevidad en términos útiles y , por tanto haciéndola más sostenible.

En la actualidad, existen una gran cantidad de herramientas de escaneo de vulnerabilidades disponibles que pueden cumplir una función similar, pero como ya he citado, se busca aportar una mayor transparencia y flexibilidad al usuario en cuanto a la ejecución de cada herramienta y el consecuente resultado obtenido.

La aplicación está orientada para aquellos que buscan someter sus proyectos web a múltiples y diversos análisis de seguridad, con la posibilidad de perfeccionar y/o determinar el uso de las tecnologías más convenientes para garantizar la inmunidad de sus productos. No tiene como objetivo fomentar su uso de forma ilegítima.

2. Descripción del producto

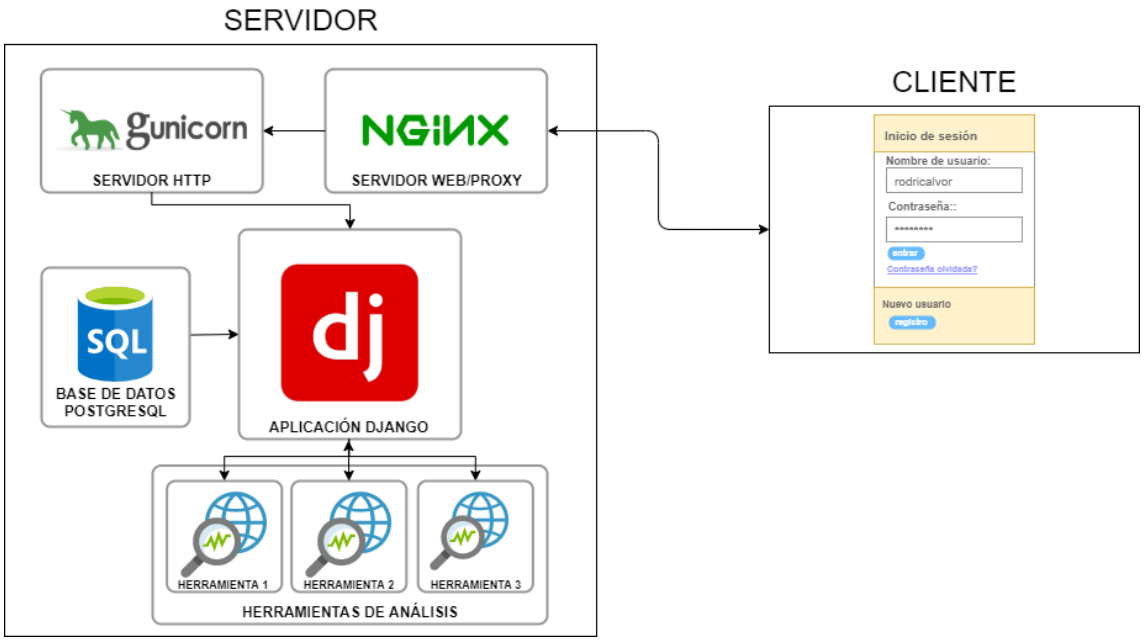

Figura 1: Diagrama general de la aplicación

El producto se presenta como una aplicación web que sólo requiere el registro como usuario y, que está compuesta por varios componentes:

- Una **aplicación web** desarrollada en **Django** (framework de código abierto escrito en **Python**), el cual se caracteriza, entre muchas cosas, por tres principios:
    1. **Velocidad**, lo que ayuda al desarrollador a la hora de obtener resultados de forma instantánea.
    2. **Seguridad**, aporta al desarrollador de una serie de recursos que permiten establecer una manera segura de gestión de desarrollo, además de evitar que se cometan fallos de seguridad, como SQL Injection o XSS.
    3. **Escalabilidad**, permite una gestión más eficiente de una gran cantidad de tráfico.

  Éstos están reflejados en ejemplos tales como Instagram o Mozilla (desarrollados con Django), los cuáles requieren de los principios citados por su gran importancia para su correcto funcionamiento.

En esta aplicación, se presentarán una serie de **herramientas de análisis de vulnerabilidades** disponibles que el usuario podrá utilizar en base a sus preferencias. Posteriormente, se presentará una serie de vulnerabilidades clasificadas por una serie de criterios, tales como, tipo y gravedad.

- Un **servidor de Ubuntu**, donde se alojarán tanto la aplicación web como las herramientas, por lo que deberá de mantener la aplicación activa y ejecutar las herramientas de acuerdo con la funcionalidad anteriormente comentada.

  A esto se puede añadir el uso de:

  - **Celery,** que permite mantener una cola de ejecución de tareas asíncronas basadas en el paso de las tareas distribuidas (también llamadas workers), centrada en operaciones en tiempo real.
  - **RabbitMQ** (message broker), encargado de recibir y enviar las tareas, así como de distribuirlas en las diferentes colas de ejecución asíncronas de Celery.
  - **Base de Datos PostgreSQL** (result backend) encargado de guardar los resultados obtenidos de la ejecución de las tareas.
- Una comunicación cliente-servidor con conexión **HTTPS**

3. Documentos del proyecto

3.1. Documento 1: Memoria

El presente documento se encuentra dividido en los siguientes apartados:

1. Introducción
2. Descripción del producto
3. Documentos del proyecto
4. Tecnologías empleadas
5. Problemas y soluciones
6. Posibles ampliaciones
7. Conclusiones

3.2. Documento 2: EVS, Estudio de Viabilidad del Sistema

Documento de evaluación de la viabilidad del sistema elaborado según los criterios de Métrica v3. Está distribuido en:

1. Establecimiento del alcance del sistema
2. Estudio de la situación actual
3. Definiciones de requisitos del sistema

4. Estudio de alternativas de solución
5. Selección de la solución

3.3. Documento 3: ASI, Análisis del siste4ma de Información

Documento de análisis del sistema a desarrollar, elaborado según los criterios de Métrica v3. Está distribuido en:

1. Alcance del sistema
2. Requisitos
3. Especificación de subsistemas
4. Análisis de subsistemas
5. Modelo de datos
6. Definición de los prototipos de la interfaz
7. Análisis de consistencia
8. Especificación del plan de pruebas

3.4. Documento 4: DSI, Diseño del Sistema de Información

Documento de Diseño del Sistema a desarrollar, elaborado según los criterios de Métrica v3. Está distribuido en:

1. Diseño de la arquitectura del sistema
2. Modelado estático del sistema
3. Modelado dinámico del sistema
4. Prototipos finales de la interfaz
5. Diseño del plan de pruebas

3.5. Documento 5: Manual de usuario

Este documento será de utilidad como manual de usuario, compuesto por:

1. Introducción
2. Inicio de sesión
3. Administradores
4. Usuarios
5. Recursos

3.6. Documento 6: Desarrollo del plan de pruebas

El desarrollo y ejecución de pruebas que fue planteado previamente en los documentos ASI y DSI. Contiene los siguientes apartados:

1. Detalles del plan de pruebas
2. Pruebas de funcionamiento

3.7. Documento 7: Planificación y presupuesto

La planificación temporal de la aplicación y el cálculo del presupuesto final para la realización del proyecto contiene los siguientes apartados:

1. Planificación
2. Presupuesto

4. Tecnologías empleadas

En este apartado se describen todas las tecnologías empleadas durante la realización de este proyecto.

4.1. Componentes

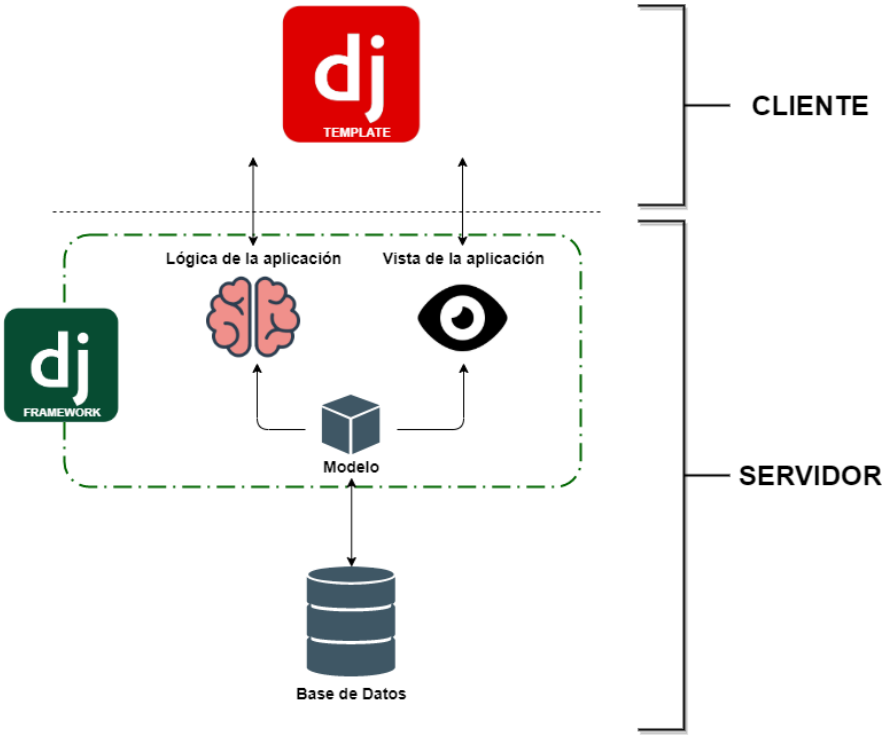

Ilustración 1: Django - Interacción entre componentes

4.1.1. Aplicación Django

Django es un framework de aplicaciones web basado en Python, el cual permite simplificar muchas tareas complejas de desarrollo relacionado con la administración y gestión aplicaciones web. Teniendo en cuenta que está escrito en Python (lenguaje multiparadigma orientado a legibilidad de código), hace que todo el desarrollo tenga una mayor disposición de recursos, propiamente presentes en Python y, en consecuencia, una mayor funcionalidad sin tener que depender de servicios externos.

Todo esto, siendo gratuito y *Open Source*, creado por Django Software Foundation.

El framework se encuentra fuertemente inspirado en el patrón de desarrollo **MVC** (Modelo-Vista-Controlador), por lo que la estructura presente en todos los proyectos desarrollados en éste presenta una estructura compuesta por:

- **Arquitectura**
    - **“Modelo”**: el modelo de datos viene formado por una serie de tablas en la Base de Datos de Django, presente de forma externa al código y con un acceso más simplificado al no ser necesario el uso de SQL.
    - **“Vista”**: la presentación de la aplicación se basa en HTML. Django hace uso de un motor de plantillas (template engine) y un cargador de plantillas (template loader), permitiendo hacer uso de páginas como “base y abstracción” para una posterior implementación.
    - **“Controlador”**: en Django se denomina “views”, algo un poco contradictorio. Permite atender la petición HTML en función de su mapeo.
- **Soporte de bases de datos**
    - Soporte para **PostgreSQL**, MySQL, SQLite 3 y Microsoft SQL Server
    - Una vez se han creado los modelos, Django nos proporciona una abstracción de la base de datos a través de su API, lo que facilita toda la gestión de objetos.

- **Soporte de servidores Web**
    - Incluye un servidor web destinado a la etapa de desarrollo
    - En producción se recomienda: Apache 2 y soporta WSGI, lo que pone a disposición el uso de otros servidores como **Lighttpd o nginx**.

- **Aclaración de uso del patrón MVC**

  Como se ha comentado anteriormente, Django se encuentra aparentemente inspirado en el patrón de diseño MVC, esto es debido a que existe una “vista”, lo que puede llevar a confusión, ya que en realidad, se trata de donde se describen los datos que serán representados, no como se verán.

  Realmente cuando se desarrolló sólo se trataba de conseguir un framework que cumpliese su función lo mejor posible, sin apegarse a nada en específico.

  Aun así, Django hoy se puede considerar un framework “MTV”, compuesto por un modelo, un template y una vista.

- **Aclaración sobre las Peticiones HTTP**

  En este apartado se describe a grandes rasgos el procedimiento de Django para procesar una petición. Cuando éste recibe una petición (request), lo primer que se hace es

generar un objeto HttpRequest que la representa y servirá como referencia para su posterior utilidad.

Para tratar la resolución de URLs utilizamos un módulo denominado *URL configuration*, con el que generamos lo que se denomina el esquema **URL dispatcher**, a través del cual fácilmente somos capaces de mapear las URLS a funciones presentes en la vista. Es decir, podremos ejecutar cualquier función con una URL determinada, siempre y cuando haya una configuración definida en el esquema.

- **Middleware**

  Django posee una especie de "plugin", que permite tratar la salida y entrada de datos de forma global, éste se denomina *Middleware*. Está compuesto por varias componentes implementadas en los ajustes de Django, las cuáles realizan funciones específicas.

  Existen varias componentes que vienen directamente implementadas por el framework, como, por ejemplo, *AuthenticationMiddleware*, que nos permite asociar a cada usuario diferentes peticiones.

  Estas componentes también son denominadas clases middleware; la ventaja es que se pueden ejecutar en más de un punto, como si se tratase de una implementación global:

  - **Middleware en peticiones**: permite modificar el objeto de la petición o devolver una respuesta concreta; se ejecuta justo después de la creación del objeto HttpRequest y antes de resolver la URL.
  - **Middleware en vistas**: permite ejecutar operaciones tanto antes como después de la ejecución de la vista; se ejecuta justo después de la resolución de la URL.
  - **Middleware en respuestas**: permite realizar modificaciones finales en la respuesta obtenida; se ejecuta justo después de la creación del objeto response.

Teniendo en cuenta lo anteriormente explicado, se puede comprender de una forma simple cómo funciona una aplicación desarrollada en Django. Además, en este caso, la aplicación desarrollada ha requerido del apoyo de software explícito destinado al framework para una gestión óptima de las tareas asíncronas generadas por los diferentes análisis que se puedan realizar en la aplicación.

De todas las herramientas disponibles se ha optado por implementar **Celery**, un administrador de colas de tareas de forma asíncrona.

Éste permite agendar o programar de forma temporal la ejecución de una serie de tareas. Todo esto centrado en que el procesamiento de la cola de tareas sea en tiempo real.

Junto a esta herramienta se debe de implementar un **message broker** que implemente el protocolo **AMQP**. En este caso, se ha optado por usar **RabbitMQ** debido a su estabilidad y fluidez con Celery, siendo posible el uso de Redis u otro agente disponible con la misma funcionalidad.

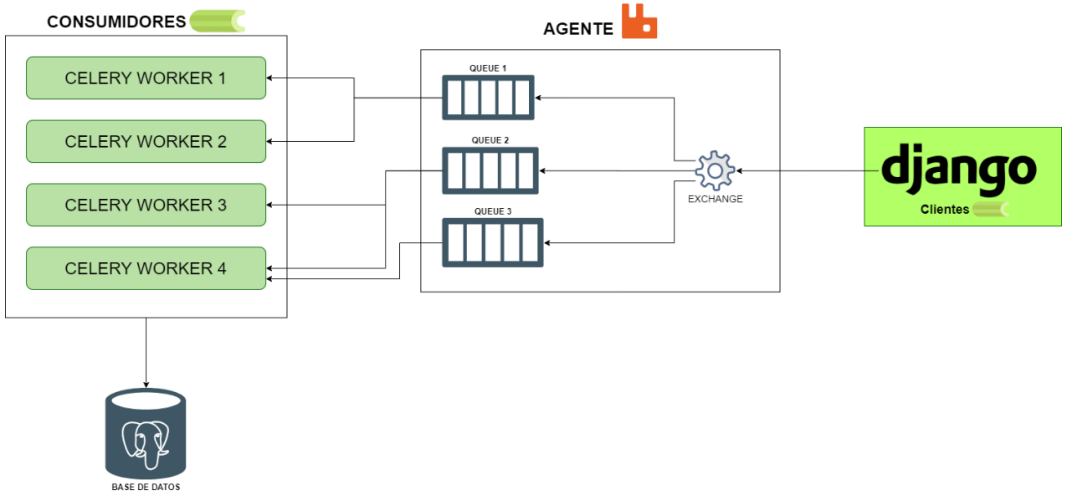

Ilustración 2: Lógica de Celery + RabbitMQ

Para simplificar la explicación, primero voy a detallar en qué consiste Celery, después se comentará la utilidad de RabbitMQ y por último explicaré la aplicación de estas dos herramientas en la aplicación.

1. **Celery**

   Como he comentado Celery nos permite simplificar el manejo y distribución de tareas asíncronas. Como principal concepto debemos de entender en qué consiste un productor y un consumidor en términos de la aplicación.

   Productor se concibe como el cliente o publicador de información; en este caso, se trata de cuando ejecutamos una acción dentro de la aplicación que vaya enlazada con una tarea propia de la herramienta, es decir, cuando ejecutamos el análisis existe una tarea de Celery que se procesa en ese momento y que lo primero que realiza es enviar una serie de mensajes.

   Consumidor se concibe como el receptor, administrador y procesador de la información enviada por el productor. En Celery disponemos de trabajadores (**Celery Workers**) que se encargan de ejecutar y gestionar la ejecución de las tareas o mensajes.

   Se debe tener claro que Celery tampoco genera por sí solo una cola de tareas, sino que, necesita de un medio de transporte o agente (broker) en el que apoyarse. Por lo que se puede ver a Celery como una capa o funda que envuelve al *message broker.*

2. **RabbitMQ**

   RabbitMQ es un agente de mensajes (Message Broker). Como ya hemos dicho, este agente se encarga de enrutar o clasificar los mensajes o tareas recibidas en una o varias colas.

   Como ya hemos comentado, los criterios que permiten asignar un trabajo en una cola u otra, obviamente, son propios de la atribución de una tarea a un usuario. Aquí existe una pieza clave llamada *Exchange* o **intercambiador,** el cual se encarga de asignar cada tarea a cada cola en base a estos criterios que son encontrados en la tarea o mensaje. En este caso se hace uso de un único Exchange que permite de forma predeterminada aportar prioridad a las tareas siguiendo la lógica de una cola FIFO.

   Nótese que el término tarea es bastante subjetivo ya que se trata de un mensaje recibido y procesado para ser ejecutado como una tarea, por lo que, a lo largo de esta explicación, puede hacerse referencia de las dos formas.

   Hasta este punto comprendemos como un productor de Celery (o cliente Celery) envía un mensaje con la tarea que es clasificado por el intercambiador de agente de mensajes RabbitMQ en una cola, para más adelante, este mensaje ser enviado a un consumidor concreto (o Celery Worker), que se encarga de procesar el mensaje y ejecutar la tarea correspondiente.

Cuando finalice la tarea se obtiene un resultado, por lo que se hace uso de la Base de Datos Postgres para guardar cada uno de los resultados obtenidos por cada tarea. De esta forma, conseguimos tener correctamente identificada cada tarea en base a su id y con un resultado de su ejecución.

4.1.2. Servidor de Ubuntu

Este servidor viene caracterizado por todas las propiedades de Ubuntu, es decir:

- Una amplia documentación y apoyo mantenido por toda su comunidad
- Software Libre, sin pago de licencia
- Gran soporte de versiones prolongado
- Alta integración con herramientas de uso común para la autenticación
- Compatibilidad con gran cantidad de protocolos y medios que facilitan su comunicación

En esta aplicación, el servidor cumple la función principal de administrar y salvaguardar toda la información necesaria para la correcta ejecución de la aplicación además de la información que pueda ser recogida y las herramientas que son ejecutadas en concordancia con esta.

Podemos decir, por tanto, que el servidor contendrá la aplicación desarrollada en **Django**, la base de datos **PostgreSQL**, un servidor HTTP ***Gunicorn*** y un servidor web ***Nginx***.

- Nginx se trata de un servidor web/proxy open-source. Viene caracterizado por presentar una arquitectura asíncrona y manejada por eventos, lo que lo convierte en una de las mejores opciones en términos de velocidad y escalabilidad.

  Como breve aclaración podemos decir que en Nginx es donde llegan las peticiones de internet, éste las captura rápidamente y filtra las que realmente no deben llegar a la aplicación.
- Green Unicorn (Gunicorn), se trata de un servidor HTTP capaz de soportar WSGI y Django de forma nativa. Éste administra los workers de forma automática además de no llegar a consumir muchos recursos.

  Como breve aclaración podemos decir que se encarga de traducir las peticiones que recibe de nginx al formato con que Django trabaja.

Todo esto anteriormente mencionado recoge la lógica del proyecto convirtiendo a este servidor en la base y pieza clave para el funcionamiento de la aplicación.

4.1.3. Base de Datos PostgreSQL

La base de datos se ha desarrollado en PostgreSQL, sistema de gestión de bases de datos relacional. También conocido como **Postgres**, creada con el objetivo de ser versátil y extensible, se caracteriza por ser capaz de soportar concurrencia.

Se autoproclama como el sistema de base de datos relacional más avanzado del mundo, a pesar de su poca extensión, éste se encuentra dirigido por una comunidad de desarrolladores, que a su vez trabajan en su propio desarrollo.

4.2. Entornos de desarrollo

4.2.1. Visual Studio Code

Visual Studio Code se trata de un editor de código fuente multiaplicación capaz de soportar múltiples lenguajes de programación. Se caracteriza por ser una herramienta fácilmente trasladable a diferentes entornos de desarrollo.

Desarrollado por Microsoft, principalmente se basa en una interfaz simple a la par que satisface las necesidades de los desarrolladores, permitiendo a cada usuario implementar una serie de extensiones disponibles y que facilitan el flujo de trabajo.

Teniendo en cuenta su auge y su potencial, he optado por utilizarlo como principal medio de desarrollo y acceso a todo el código del proyecto.

4.2.2. Microsoft Terminal

Microsoft Terminal se trata de un emulador de una terminal destinada para Windows 10, desde ella podemos tener acceso al símbolo de sistema, PowerShell, WSL y SSH.

Desarrollado en C++, lo he considerado como un medio fácil de interactuar con el servidor en vez de usar otro cliente como Putty o directamente Powershell, simplemente se trata de las preferencias de cada uno, ya que de lo que se trata realmente en este sentido es como te desenvuelvas con cada herramienta.

Su utilidad principal ha sido como medio de comunicación con el servidor para tareas de administración y configuración del propio servidor, además de gestión de las propias herramientas de trabajo.

4.2.3. Control de versiones: Git

Git ha brindado un buen soporte de cada una de las sesiones de desarrollo realizadas a lo largo de la evolución de la aplicación.

Tanto como medida de prevención como de eficiencia, Git aporta robustez a cualquier proyecto que necesite de un periodo de desarrollo, dando constancia de cada cambio y permitiendo comunicar los diferentes colaboradores de un mismo proyecto.

4.2.4. Herramientas de Desarrollador de Microsoft Edge

Como apoyo directo al desarrollo web se han utilizado las herramientas de desarrollador de Microsoft Edge (basado en Chrome), dando principal importancia a la monitorización del depurador y la consola con el objetivo de definir correctamente la comunicación de los componentes.

4.3. Otro software empleado

4.3.1. Sistema Operativo: Windows 10

Se ha primado el uso de herramientas integradas en el entorno de Microsoft, de tal forma que se asegura un funcionamiento sincronizado del sistema, además de una mayor fluidez y consistencia.

La importancia de la cohesión existente entre este software minimiza el riesgo de incompatibilidades y problemas que puedan ocasionarse en caso de tratarse de herramientas externas al entorno del propio sistema operativo.

4.3.2. Virtual Box

Como medida de desarrollo se ha generado una máquina virtual con el Servidor de Ubuntu instalado con el objetivo de facilitar el acceso y el manejo directo del éste y todas sus componentes.

4.3.3. Diseño de modelos: diagrams.net

Herramienta útil para el diseño de diagramas y grafos aportados como documentación del proyecto.

Gratuita e intuitiva, permite desarrollar varios diseños en base a tus requerimientos, además de poseer una gran facilidad a la hora de exportar en múltiples formatos todo tu trabajo.

4.3.4. Diseño de Diagrama de Gantt

Para el diseño de este diagrama he optado por mantener la utilización de la misma herramienta utilizada para realizar los diagramas: diagramas.net.

4.3.5. Desarrollo de la documentación: Office 365

Toda la documentación ha sido creada y editada con herramientas de esta aplicación.

5. Problemas y Soluciones

A lo largo del desarrollo del proyecto, se pueden destacar una serie de dificultades principales

5.1. Implementación de Celery

Como ya he comentado en la descripción de la aplicación, Celery constituye una herramienta añadida a la aplicación de Django que permite gestionar tareas asíncronas; de esta forma, el uso de las herramientas de escaneo se realiza de forma externa al procesamiento de la aplicación web.

La dificultad encontrada en el uso de Celery ha sido el hecho de que se pueden dar dos posibilidades en la ejecución de un análisis: que se ejecute con normalidad, finalice y devuelva un resultado o que se detenga un cierto tiempo después de iniciarse, a petición del cliente.

El hándicap ha sido encontrar un método con el que se pueda guardar el resultado de la tarea a pesar del hecho de que finalice de forma inesperada e inmediata.

Se ha requerido de generar un sistema que aplica un “polling”, es decir, una comprobación periódica del estado del proceso; de esta forma, podemos saber cuál es la situación del análisis de forma prácticamente inmediata y constante y mantener la mayor cantidad de información respecto a su ejecución guardada.

De esta forma, podemos asociar a cada tarea, de una forma más segura, la información que consideremos necesaria.

5.1.1. Estado de las tareas

A lo largo de la ejecución de una tarea, ésta va adquiriendo diferente valor en su estado. Un problema importante encontrado en el desarrollo ha sido el hecho de tener que guardar y comprobar que cada tarea haya finalizado y, en concreto, si ha sido a petición del usuario o porque finalizó por sí sola.

5.2. Herramientas de análisis

Cuando se implementa una nueva herramienta de análisis se debe de tener en cuenta que cada una de ellas no debe de compartir ninguna relación con otra, por lo que el trato que realicemos para la correcta ejecución de una no se puede abstraer para utilizarse en otra.

Por esto, se consideran varias dificultades en su implementación:

5.2.1. Exportación de resultados

Uno de los principales factores en la ejecución de las herramientas es la obtención de un resultado que se adhiera a nuestro modelo de datos. Por ello, se elaboran de forma concreta un sistema de traducción de funciones básicas, con el fin de abstraer las mismas funciones en las diferentes herramientas.

5.2.2. Variedad en su lógica

En cada una de estas herramientas se presenta una diferente estructura y posiblemente un diferente desarrollo en otro lenguaje y/o tecnología. Se debe apreciar que no todas las herramientas son capaces de detener de forma forzosa su ejecución y producir un resultado, por lo que se requiere de una comprobación añadida a la herramienta que permita determinar si esta posee tal propiedad.

6. Posibles Ampliaciones

Todo el proyecto ha seguido una dinámica aplicada con vistas al futuro.

Desde un principio, se ha tenido como objetivo generar una aplicación open source que permita de forma simple su utilización en cualquier tipo de entorno lícito, facilitando su implementación en diferentes ámbitos y pensada como algo llevable a cualquier proyecto web.

El hecho de que posea tal facilidad de implementación de nuevas herramientas permite que sea una gran medida de seguridad en desarrollo web.

Hoy en día las herramientas de seguridad se encuentran en auge debido a su demanda y la mayoría de las empresas grandes han generado sus propias metodologías frente a las posibles vulnerabilidades encontradas en sus aplicaciones. Con esta aplicación, enfocada de forma transparente al cliente, se podrá personalizar el análisis sin tener que depender de ninguna otra herramienta. Ya que de por sí las herramientas se actualizan y son las que de verdad ejercen el análisis, lo que conseguimos es centralizar los resultados obtenidos de éstas y mostrarlos de una forma más directa al cliente.

7. Conclusiones

Tras el desarrollo del proyecto se ha alcanzado el objetivo planteado a su inicio: desarrollar una herramienta de análisis de vulnerabilidades en páginas web.

Se pueden dividir las conclusiones en dos aspectos:

- Desde el aprendizaje: Ha sido una forma de plantear muchas aptitudes obtenidas en los diferentes cursos académicos de la carrera en un proyecto, ya que trata de cubrir muchos ámbitos de desarrollo y en varios entornos.
  Puedo decir que antes de iniciar el proyecto no había usado apenas Python, pero teniendo en cuenta las estadísticas presentes respecto a la demanda de este lenguaje y obviamente su simpleza y flexibilidad respecto a otros lenguajes me ha resultado muy atractivo.

  Además de que no había utilizado nunca ni el framework de desarrollo Django ni ninguna de las herramientas utilizadas en éste, puedo decir que me ha exigido tener que realizar un estudio planificado de cada una de las características y posibilidades presentes en el desarrollo de éste.

  También cabe mencionar que, personalmente, siempre me ha gustado aprender nuevas tecnologías, pero no quita el hecho de que no es algo de lo que haya hecho uso ni utilizado, lo que supone un diferente planteamiento de desarrollo respecto a una tecnología tratada a lo largo del grado.

- Desde el desarrollo: Lo principal a la hora de realizar una aplicación es saber si lo que tienes en mente a desarrollar es realmente posible y si lo es, entonces, tratar de mantenerlo los más planificado posible y sabiendo cuánto tiempo se requiere para cada parte de la aplicación.

  El hecho de tener que aprender a medida que te encuentras desarrollando ha hecho que tenga que destinar más tiempo del realmente necesario tanto buscando vías de desarrollo obsoletas, incipientes o directamente inviables como ejemplificando en pruebas más simples la utilidad de cada componente. Todo esto principalmente requiere de una motivación encontrada sobre todo en el ámbito de la seguridad y web. Por lo general poder utilizar un lenguaje que no posea una curva de aprendizaje elevada y que además posea una alta cantidad de recursos se considera un lujo en términos de desarrollo, por lo que, a la par que digo que me ha supuesto un reto tener que estudiar y desarrollar desde cero, he tenido a mi disposición muchas facilidades en comparación a otros entornos de desarrollo.

También cabe destacar que el desconocimiento de hasta qué punto el proyecto era viable en términos funcionales ha hecho que muchas de las ideas planteadas en un principio se vean modificadas en la fase de desarrollo.

6. Se ha conseguido el planteamiento inicial de conseguir una aplicación que ejecute análisis a aplicaciones web, pero teniendo en cuenta la evolución de estas tecnologías, es cuestión de tiempo que, si no se mantiene actualizada, pierda muchas de las propiedades.

8. Bibliografía

Durante el desarrollo de la aplicación se ha necesitado de varias fuentes para consultar procedimientos de elaboración y errores.

Destacaremos como principales páginas:

- DigitalOcean Community: se trata de una página web en la que la sección de comunidad presenta una serie de tutoriales de configuración de los servidores.
- StackOverflow: se trata de una página web comunitaria, dónde se encontrará principalmente apoyo entre usuarios y soluciones a problemas esporádicos.
- Djangoproject documentación: se trata de la página principal de Django dónde encontraremos una sección destinada al aprendizaje de su desarrollo.

- Diagrams.net: se trata de la página usada para elaborar todos los diagramas presentes en el proyecto.
- Celery documentación: se trata de la página utilizada para la instalación y configuración de Celery en nuestra aplicación.
- RabbitMQ Tutorials: se trata de la página utilizada para la instalación y configuración de el servidor rabbit en nuestra aplicación.