Treball Fi de Carrera

ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMES

Facultat de Matemàtiques
Universitat de Barcelona

SEGURIDAD EN APLICATIVOS WEB

Adrián Hermoso Metaute

Director: Eloi Puertas Prats
Realitzado en: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 20 de septiembre de 2013

Índice

1.- Introducción y objetivos

El acceso a internet y en concreto a las tecnologías web de forma ubicua es toda una realidad hoy en día. Cada vez es más común encontrarse con aplicaciones web, en lugar de aplicaciones nativas, dando estas primeras soporte a una mayor cantidad de dispositivos y plataformas.

Una de las mayores implicaciones que tiene este hecho es que se pasa del modelo de datos descentralizado que suelen tener las aplicaciones nativas a un modelo centralizado donde todos los datos de los usuarios y de la aplicación estarán en un mismo servidor o grupo de servidores.

Mientras que este “nuevo” planteamiento presenta claramente un conjunto sustancial de ventajas respecto a las aplicaciones nativas también pone de manifiesto una serie de inconvenientes. Cualquier tipo de incidencia en materia de seguridad podría afectar la confidencialidad, integridad o disponibilidad de todos los datos asociados a la aplicación o a los usuarios de ésta.

Es por ello que los objetivos que persigue este estudio son:

- Analizar la importancia de la seguridad en general y en concreto sobre aplicaciones web y de como una herramienta como es la **auditoría de seguridad** puede ayudar a identificar fallos en estos aplicativos que pudieran vulnerar la seguridad de la información.

- Entender la naturaleza de las **vulnerabilidades** qué son, cómo surgen, qué implicaciones tienen y sobretodo dar a conocer una serie de recursos que puedan ayudar tanto a desarrolladores como a auditores a identificarlas y mitigarlas.

- Proveer de un sencillo **aplicativo** con interfaz gráfica que tiene como objeto alcanzar las dependencias mínimas en software de terceros siendo a su vez fácilmente extensible programáticamente. Dicho aplicativo deberá facilitar tanto al **auditor** como al desarrollador a localizar vulnerabilidades en aplicaciones web.

Por lo tanto además del análisis de la seguridad en aplicativos web con especial atención en las vulnerabilidades se propone el desarrollo de un ligero proxy de peticiones HTTP que es considerada una de las herramientas más útiles para la evaluación de vulnerabilidades por ser permitir trabajar de una forma muy gráfica a un nivel muy bajo.

1.1.- Estructura de la memoria

La memoria está organizada en los siguientes capítulos:

- **Capítulo 1.- Introducción:** capitulo actual dónde se introduce el tema propuesto así como los objetivos que persigue este proyecto. También se resume de forma breve que contiene cada uno de los capítulos de los que consta este trabajo.

- **Capítulo 2.- Estado del arte**: se detalla brevemente el actual estado de algunas de las herramientas que pueden ser utilizadas para evaluar la seguridad de sitios web. En concreto se analizan brevemente dos de las más conocidas.

- **Capítulo 3.- Análisis de la seguridad en sistemas web:** en este capítulo se introducen varios conceptos y temas que servirán para concienciar al lector sobre la importancia de la seguridad en los aplicativos web así como para entender las ventajas y limitaciones de las auditorías de seguridad y en concreto de los análisis de vulnerabilidades que forman parte de estas. Además se analizará cómo los análisis de vulnerabilidades pueden ayudar a identificar los distintos tipos de vulnerabilidades que pudieran existir.

- **Capítulo 4.- Propuesta de aplicativo para la evaluación de seguridad en webs:** este capítulo detalla las tareas de análisis y definición de requisitos, el diseño de la aplicación y las herramientas y tecnologías utilizadas, así como parte del proceso de implementación y las pruebas realizadas para evaluar el software.

- **Capítulo 5.- Planificación y costes:** en este capítulo se muestra la planificación inicial del proyecto cuyos resultados se comparan con los experimentales recogidos durante el proceso, además se presenta el coste final del proyecto

- **Capítulo 6.- Conclusiones:** en este capítulo se rememorará los objetivos que buscaba este objetivo y se compararán con los resultados obtenidos. Además se expondrán los problemas experimentados y se presentarán propuestas de mejora para el proyecto presentado

- **Capítulo 7.- Bibliografía:** aquí se incluyen las referencias bibliográficas mencionadas a lo largo del estudio

- **Anexos:** éste recoge las referencias no bibliográficas así como un manual de instalación, configuración y uso del proyecto presentado.

2.- Estado del arte

Todas las tecnologías web, al igual que la mayor parte de las asociadas con internet, evolucionan y se desarrollan a un ritmo vertiginoso. Día a día se presentan innovaciones y se aprueban borradores de las mismas que son adaptados rápidamente por navegadores y servidores.

La carrera por ser el primero en adaptar estas tecnologías, junto con la falta de experiencia, dejadez o malas prácticas por parte de los desarrolladores, hace que en ocasiones se incurran en errores que pueden dar lugar a vulnerabilidades de seguridad de distinta gravedad.

Se puede observar que, en tanto que existe una multitud de desarrolladores diseñando y adaptando éstas novedades, no hay tantos que se dediquen al desarrollo de herramientas que permitan evaluar la seguridad de las aplicaciones web. Esta circunstancia constituye claramente una traba para los auditores de seguridad que tratan de detectar los posibles fallos de seguridad en los aplicativos.

Si además, se tiene en consideración que muchas de las herramientas disponibles para la evaluación de seguridad web presentan una serie de inconvenientes como que algunas están obsoletas, bien sea porque son antiguas y no mantenidas o porque están pensadas de forma monolítica, otras bien son de código cerrado, o comerciales además caras, o demasiado complejas…, se presenta un escenario en el que la falta de soluciones es la tónica predominante que impide la adaptación al ritmo con el que suceden los cambios

Véanse por ejemplo estas dos conocidas herramientas que se podrían utilizar para realizar auditorías de seguridad y que presentan características similares a la solución propuesta como puede ser el proxy pasivo de mensajes HTTP o el modo de interceptado y edición de estos mismos mensajes:

La primera de ellas **Burp** de la compañía PortSwigger es una herramienta comercial bastante potente y no excesivamente cara si es comparada con otras herramientas similares disponibles en el mercado. Sin embargo debido a su naturaleza de código cerrado y su licencia privativa no es posible para la comunidad de software libre crear trabajos derivados bien sea extendiéndola con nuevas funcionalidades o aprovechando código de ésta para otras herramientas.

La segunda Zed Attack Proxy o ZAP por el contrario es una herramienta libre y de código abierto creada por OWASP(Open Web Application Security Project). Desarrollada en Java íntegramente dispone de una API para python que facilita el desarrollo de extensiones. Se debe observar sin embargo que una vez instalada ocupa un considerable espacio en disco duro, y que si además se tiene en cuenta que requiere la máquina virtual de java para ejecutarse, y el interprete de python en caso de se quiera realizar ninguna extensión, se convierte en una aplicación poco portable.

Dadas estas observaciones se puede concluir que, en un campo en continua evolución, donde nuevas tecnologías aparecen y son adaptadas rápidamente, **existe un vacío en cuanto a herramientas de código abierto lo suficientemente genéricas, modulares, portables y fácilmente extensibles**.

3.- Análisis de la seguridad en sistemas web

Para el desarrollo del análisis es necesario tener en cuenta las siguientes consideraciones:

- Cuando se hace mención a “auditorías” o “auditorías de seguridad” se hace referencia concretamente a las auditorías de seguridad de aplicativos web. De igual modo se utilizará el término “auditoría” para referenciar a los análisis de vulnerabilidades que forman parte de estas.

- La mayor parte de las veces que se hace alusión a las vulnerabilidades, salvo que se especifique lo contrario, se estará haciendo referencia a vulnerabilidades que puedan presentar un riesgo de seguridad.

- Se utilizarán indistintamente los términos anglosajones request y response para referirse a petición y respuesta respectivamente prefiriéndose los últimos sobre los primeros.

3.1.- Auditorías de seguridad

Previo a de definir que es una auditoría de seguridad primero se va a tratar de definir qué se entiende por **auditoría** y por **seguridad**:

- Una **auditoría** es la evaluación o revisión de las cualidades de un sistema, persona, objeto, proceso, producto, etc, para saber cómo se posiciona éste de acuerdo a un marco de referencia o una serie de criterios.

- Según la Real Academia Española(RAE) **seguridad** es la cualidad de seguro. Y seguro es todo aquello libre y exento de todo peligro, daño o riesgo.

Así pues se podría entender por auditoría de seguridad aquel **proceso** que una vez llevado a cabo permite **evaluar** e **identificar** de forma sistemática el estado de la **seguridad**(en este caso de un aplicativo web) en relación a una serie de **criterios** o **normas**.

Mediante este proceso lo que se tratará de identificar aquellos riesgos que pudieran afectar a la confidencialidad, integridad y/o disponibilidad(**CIA**, de las siglas anglosajonas Confidentiality, Integrity and Availability) de un aplicativo y de los sistemas asociados con éste, identificando **vulnerabilidades.**

3.1.1.- ¿Por qué son necesarias las auditorías?

Para entender el motivo por el cual son necesarias las auditorías de seguridad, primero se debe entender las **motivaciones** que existen para el **desarrollo** de una **aplicación** y qué **valores** tienen éstas para que requieran ser **protegidas**. De esta forma, dejando a un lado los sistemas y plataformas sobre los que funcionan las aplicaciones web y centrándose en éstas últimas, se identifican diferentes motivos que pueden llevar a un desarrollador o un equipo de éstos a programar una.

Muy probablemente y especialmente si se trata de un equipo administrado por una empresa, éste tenga un **interés económico** de forma directa o indirecta. Bien ya sea monetizando directamente la aplicación y cobrando por tanto una **tarifa por uso** o indirectamente rentabilizándola sometiéndola a un proceso de marketing con tal de crear marca, llegar antes que los productos de la competencia al usuario final y facilitando la rápida adopción de ésta e incrementar la presencia en el mercado.

Aún incluso **no** teniendo ningún **interés económico**, puede haber otras causas que lleven a un desarrollador o equipo a trabajar en una herramienta específica. Por ejemplo, con el objetivo de cubrir una necesidad concreta y ofrecer la solución de forma gratuita y **solidaria**, darse a conocer y promocionarse incrementando su valor personal, o demostrar las capacidades del equipo para desarrollar un trabajo o simplemente por puro altruismo.

Independientemente del ánimo que hay tras el desarrollo de la aplicación se debe observar que ésta **trabajará constantemente** con **datos e información**. Es importante entonces entender el **valor** de la información que se trata y qué representa.

Si se observa la definición asociada a las Tecnologías de la Información (TI) **se** entiende que la **información** es un conjunto de datos organizados que pueden ser interpretados como un mensaje, o lo que es lo mismo, es el conocimiento comunicado o recibido (mensaje) con respecto a un hecho en particular o circunstancia.

Puede darse el caso de que la mucha de la información con la que la aplicación pueda tratar tenga carácter confidencial. Dicha información podría variar y ser desde **datos introducidos** directamente por los usuarios y clientes, por ejemplo datos de una transferencia en una aplicación bancaria o pagos realizados en una web de subastas, hasta **datos inferidos** o calculados gracias a los hábitos y costumbres del usuario al utilizar dicha aplicación como por ejemplo que búsquedas realiza para saber que le interesa o que gustos tiene.

Aunque no toda la información con la que tratan las aplicaciones es de carácter confidencial, ni la filtración de ésta tiene porque atentar contra la privacidad de nadie, sí que se podría considerar quizá igual de valiosa. Ésta, por ejemplo, podría representar parte de la **inteligencia** del negocio, especialmente si es algo que se ha generado específicamente para este aplicativo.

Una propiedad de la información es que ésta adquiere diferentes valores según el interés de diferentes personas o entidades. Lo que para unos puede parecer insignificante para otros es realmente valioso. Así que la información con la que nuestra aplicación trabaje tendrá más o menos valor a los ojos de un potencial atacante

Tras este breve análisis sobre el valor de la información se puede deducir que para una aplicación ésta es crítica y que por lo tanto tener las medidas correctas de seguridad para protegerla en todo momento es muy importante. Lo que interesará garantizar es que la aplicación tiene en consideración lo que se conoce como los tres pilares de la seguridad de la información o **CIA** y que quedan presentadas de forma visual en la figura 3.1:

- **Confidentiality (Confidencialidad)**: que no se invada la privacidad de los usuarios comprometiendo la confidencialidad de sus datos y/o acciones, es decir, que sólo tengan **acceso** a dicha información las personas debidamente autorizadas.

- **Integrity (Integridad)**: de forma similar a la confidencialidad, interesa conservar la integridad de los datos garantizando que la información es sólo modificada por aquellos que cuentan con los apropiados permisos. apropiados.

- **Availability (Disponibilidad)**: asegurar que la información esté disponible cuando se requiere.

***Figura 3.1**: Los tres pilares de la seguridad de la información*

Teniendo claro que estos datos van a ser utilizados constantemente, que serán los que aportan la mayor parte del valor de la aplicación y que la criticidad que estos puedan adquirir puede tener un gran impacto en caso de ser filtrados o destruidos es evidente que estas tres áreas requieren un esfuerzo con tal de protegerlos adecuadamente. Lo que se querrá evitar será la filtración, destrucción o alteración por personas no autorizadas.

Tanto si se piensa desde el punto de vista de los desarrolladores como del de los usuarios uno de los hechos que dará más valor o menos a el aplicativo es que ésta sea segura y que la información esté protegida en las tres áreas

Lo que se consigue con una auditoría de seguridad es garantizar a los usuarios que sus datos y sistemas están siendo protegidos con las medidas de seguridad apropiadas generando confianza en la compañía, entidad o propietario de la aplicación.

Si la auditoría está realizada además por un tercero independiente ésta aportará un valor añadido al producto, ampliando el abanico de potenciales clientes y consolidando/asegurando los que ya se tienen

Entendidas las motivaciones que hay para el desarrollo de una aplicación, el valor de la información con la que ésta trabaja, el porqué ésta debe ser protegida y sobretodo, en qué tres áreas se debe trabajar para proteger ésta, se puede entender porqué hacer una auditoría de seguridad es recomendable y necesaria. Aún con más motivo, si detrás del aplicativo hay una motivación económica o la vulneración de ésta pudiera implicar una mala propaganda y afectar a la marca o imagen empresarial hasta el punto de suponer su declive y desaparición.

3.1.2.- Diferentes enfoques para realizar un análisis de vulnerabilidades

Una de las diferentes fases de la que consta una auditoría es el análisis o evaluación de vulnerabilidades que se podrá enfocar utilizando diversas estrategias. Los resultados variarán considerablemente dependiendo de estos enfoques:

- **Black box (Caja negra):** Para este tipo de análisis los auditores se ponen en la piel de un atacante el cual no tienen conocimiento alguno de la aplicación o de los sistemas asociados a ésta.

- **White box (Caja blanca):** En las análisis de caja blanca se presenta a los auditores con todo el conocimiento del aplicativo así como con una copia del código fuente para que la revisen a conciencia.

- **Grey box (Caja gris):** Este enfoque consiste en una mezcla de los dos anteriores. En esta ocasión se facilitara a los técnicos una parte de información acerca del funcionamiento de la aplicación y de los sistemas con los que interactua y con ella procederán a realizar el análisis de vulnerabilidades

Si se invierte el mismo tiempo en hacer la misma auditoría con cada uno de los tres enfoques se observarán una serie de diferencias.

El enfoque más realista para realizar la evaluación de vulnerabilidades será el de caja negra, puesto que los auditores tendrán el mismo conocimiento que el que pudiera tener un posible atacante. Se deberá por lo tanto entender que éste podría generar algún que otro falso positivo o no encontrar ninguna vulnerabilidad aún existiendo éstas.

La revisión más exhaustiva y por lo consiguiente fiable sería la de caja blanca, pero sin embargo, ésta también será la más laboriosa y costosa económicamente. Las auditorías de caja blanca suelen incluir una revisión de código con la que los auditores tendrán total conocimiento sobre la aplicación y podrán identificar de forma sencilla los puntos más

críticos de esta o si por ejemplo se repiten ciertos patrones de fallos que pudieran derivar en vulnerabilidades. Con este enfoque y debido a la complejidad que entraña es posible que no se pueda finalizar a tiempo la evaluación si la ventana acordada para la evaluación no es lo suficientemente grande para el alcance definido.

Por último, si se enfoca el análisis de vulnerabilidades con una metodología de caja gris, se conseguiría que los auditores estuvieran en una posición aproximada a la de un atacante pero con la ventaja de que pueden identificar con más facilidad aquellas partes más críticas o que requieran más atención.

Se debe por lo tanto entender que el hecho de elegir entre un enfoque u otro variará dependiendo de los intereses y presupuesto del que se disponga.

3.1.3.- Limitaciones

El haber concluido una auditoria, no es garantía de que la aplicación sea 100% segura. Se debe comprender que no existe un producto totalmente seguro y que siempre hay riesgos. De igual modo se debe observar que en una auditoría de seguridad es posible que no se identifiquen todos los fallos que hay en la aplicación web. Hay una serie de factores que podrían afectar la precisión de los resultados:

- **Presupuesto:** la realización de una auditoría o de una auditoría los suficientemente rigurosa se podría ver afectada por el impacto económico que tendría ésta sobre el coste total del proyecto.

- **Tiempo**: éste es un factor que generalmente, aunque no necesariamente, está relacionado con el punto anterior. No será posible identificar todas las vulnerabilidades existentes en el sistema o aplicativo debido a una limitación de tiempo, que puede venir impuesta por una limitaciones en el presupuesto

- **Alcance**: por diversos motivos puede haber parte de la aplicación o sistemas asociados con ésta que queden fuera del alcance de la auditoría dejando, por lo tanto, parte de ella sin auditar.

- **Factores externos:** aún siendo un software de desarrollo propio no es raro en encontrarse con el uso de librerías de terceros sobre las que no se tiene control. En caso de ser estas vulnerables, nuestra aplicación también podría verse afectada.

Es evidente entonces realizar un análisis de vulnerabilidades no garantiza que la aplicación evaluada sea segura. Debe quedar claro que **no** hay aplicación 100% segura y que una auditoría de este sólo sirve para garantizar unos mínimos o que se ha expresado un interés en proteger el aplicativo.

3.1.4.- Resultados

Así pues, la auditoría de seguridad formará parte del último paso de en la implementación de medidas defensivas, servirá para evaluar la efectividad de éstas y como resultado generará un informe en el que se detallarán las vulnerabilidades encontradas y posiblemente una serie de recomendaciones para solventarlas.

Si el informe de resultados no es favorable habrá dos opciones para cada una de las vulnerabilidades encontradas: la primera sería aceptar el riesgo y la segunda solventar el fallo. Una vez finalizado este proceso se deberá volver a evaluar otra vez si se han corregido apropiadamente

3.2.- Vulnerabilidades

Como se ha mostrado en el apartado anterior, el informe de resultados de una auditoría, detallará la serie de vulnerabilidades encontradas. Por vulnerabilidad se entenderá cualquier debilidad en la aplicación que pudiera resultar de manera directa o indirecta en un riesgo para la organización, que ha desarrollado ésta, o para los usuarios que la utilizan.

Se debe observar que un atacante aprovechándose de estas vulnerabilidades tendrá un objetivo que bien puede ser el usuario de la aplicación o bien los activos sobre los que corre la aplicación.

3.2.1.- ¿Por qué surgen?

Vale la pena recalcar, como se mencionó en el apartado “3.1.3.- Limitaciones de las auditorias” , que no es posible tener una aplicación libre de vulnerabilidades. Por ello es importante entender la naturaleza de éstas y por qué surgen, cuestiones que se entenderán con los siguientes ejemplos.

Por un lado, si se habla de **sistemas** se observará que en muchas ocasiones surgen por utilizar ficheros de configuración por defecto o por copiar éstos de un sistema a otro o incluso de internet cuando no van a tener los mismos requisitos ni características.

Otro de los fallos que se pueden encontrar y que hace que los sistemas puedan ser vulnerables es el configurar mal los permisos de los ficheros. Un fallo al configurar estos vulneraría lo que se ha definido anteriormente como CIA, los tres pilares de la seguridad de la información.

También es bastante común encontrarse en sistemas servicios disponibles los cuales no son necesarios y que por ignorancia o desconocimiento nunca son deshabilitados. Además se debe tener en cuenta que estos vendrán configurados con los parámetros por

defecto, que en la mayoría de ocasiones suelen ser bastante permisivos. Otro fallo habitual es encontrarse con el uso de contraseñas por defecto con las que el fabricante distribuye sus productos que en caso de ser cambiadas suele ser por contraseñas débiles que no siguen ningún tipo de política de contraseñas que exijan unos mínimos de complejidad.

Por el otro, en el **software** los fallos más comunes aparecen por otros motivos. No es nada extraño encontrarse con proyectos mal planificados dónde las exigentes fechas de entrega hace que los desarrolladores se centren más en el desarrollo de las funcionalidades de la aplicación que en implementar controles de seguridad.

Tampoco es sorprendente encontrarse con programadores que desconocen cómo desarrollar de forma segura, así que aún teniendo un proyecto bien planificado se observará que por ejemplo, los datos con los que trabaja la aplicación, o se validan mal o no se validan.

Relacionado con el punto anterior son las validaciones en el lado del cliente. En las aplicaciones donde existe una comunicación constante entre cliente y servidor(por ejemplo entre un navegador y una aplicación web) es una práctica errónea y común validar los datos en el lado del cliente y no volverlos a validar en el servidor. Este tipo de validaciones exclusivas en el cliente suelen ser fáciles de evadir.

Comunes a las dos son por ejemplo la falta de actualizaciones bien sea por ejemplo de los controladores de los que dependen diversos sistemas o bien de librerías de terceros depende el software hace que, si por ejemplo éstos son vulnerables, nuestro sistema o aplicación también pueda serlo.

En la figura 3.2 se presentan algunos de los fallos más comunes que acaban resultando en vulnerabilidades del sistema.

Sistemas	Software
Configuración generica	Mala planificación
Servicios innecesarios habilitados	Desconocimiento sobre desarrollo seguro
Falta de actualizaciones	Falta de actualizaciones
Débil política de contraseñas	Falta de validación o validación insuficiente
	Validación en el lado del cliente

***Figura 3.2**: Algunos ejemplos de los fallos más comunes que resultan en vulnerabilidades*

3.2.2.- Evaluando el riesgo: métricas

Siempre que se hable de vulnerabilidades se hablará de **riesgo**. Para entender completamente el significado de riesgo se deberá también definir conceptos como **complejidad** de explotación e **impacto**.

La **complejidad** especificará la dificultad a la que se enfrentará un atacante para tratar de aprovecharse o explotar una determinada vulnerabilidad.

Por otro lado el **impacto** indicará entonces el alcance o daño que resultará sobre el sistema, aplicativo, usuarios y/o información así como los daños colaterales que podrían surgir si se aprovechase de forma efectiva esta vulnerabilidad, es decir, de que manera se vulneran la CIA.

Con estos dos conceptos claros se puede definir el **riesgo** como aquel valor que identificará la gravedad de una vulnerabilidad siempre teniendo en cuenta la complejidad de explotación y el impacto de estas

Al riesgo y por lo consiguiente al impacto y a la complejidad se le podrán otorgar dos tipos de valoraciones una **objetiva** otra **subjetiva**.

Tanto para hacer una valoración objetiva como subjetiva se plantearán para cada vulnerabilidad una serie de cuestiones que ayudarán a determinar la complejidad de explotación y el impacto que suponen.

La única diferencia es que cuando se valore el riesgo, que representa una vulnerabilidad, de forma **subjetiva**, se tendrá en cuenta la naturaleza de la aplicación y de los datos que utiliza para luego realizar una serie de preguntas en relación a ésta.

Imagínese por ejemplo el siguiente escenario en el que existen dos aplicaciones; una es una web de aficionados a coches de miniaturas donde los usuarios pueden comprar y vender modelos además de listar los que tienen y la otra es una aplicación bancaria que permite realizar a los clientes transferencias, pagos a entidades y consultar sus movimientos bancarios.

Ambas tienen la misma vulnerabilidad que además es relativamente sencilla de explotar y permite a un atacante ver todos los datos de otros usuarios Con una valoración objetiva de esta vulnerabilidad se obtendría la misma puntuación para ambas aplicaciones, sin embargo, con una valoración subjetiva se observaría que el riesgo presentado es diferente para las dos.

Esto se deberá a que aunque a pesar de que el impacto es el objetivamente el mismo y en las dos se comprometen datos, los datos bancarios son considerados más sensibles o críticos. La complejidad de explotación en las dos en este caso será la misma

Para **valorar** el **impacto** de una vulnerabilidad por ejemplo, se debería responder a las siguientes preguntas: ¿Dada la naturaleza de los datos filtrados tendrá ésta un impacto económico directo? ¿Qué importancia tiene el activo vulnerado, es un sistema secundario o quizá es básico para el funcionamiento correcto de la aplicación? ¿Qué tipo de datos se perderían o filtrarían, son estos el núcleo de nuestra aplicación? ¿De que manera se vulnerará la privacidad de los usuarios, se filtrarán sólo direcciones de correo o será sin embargo todos los datos personales que se guardan? ¿Es el activo accesible desde internet o sólo desde la red local?

Por otro lado para **valorar** la **complejidad** se podrían plantear que plantear preguntas como las siguientes: ¿Es complicado reproducir la vulnerabilidad? ¿Tiene la aplicación habilitados los mensajes de depuración o presenta al usuario detalles del error que le pudiesen ayudar a explotar la vulnerabilidad? ¿Se deberá convencer al usuario para que realice alguna tarea como: visitar un link especifico o visitar otro sitio web para explotarla?

En la figura 3.3 se puede ver una serie de preguntas que ayudarán a determinar el impacto y la complejidad de una vulnerabilidad.

Preguntas impacto	Preguntas complejidad
¿Dada la naturaleza de los datos filtrados tendrá ésta un impacto económico directo	¿Es complicado reproducir la vulnerabilidad?
Qué importancia tiene el activo vulnerado, es un sistema secundario o quizá es básico para el funcionamiento correcto de la aplicación?	¿Tiene la aplicación habilitados los mensajes de depuración o presenta al usuario detalles del error que le pudiesen ayudar a explotar la vulnerabilidad?
¿Qué tipo de datos se perderían o filtrarían, son estos el núcleo de nuestra aplicación?	¿Se deberá convencer al usuario para que realice alguna tarea como: visitar un link especifico o visitar otro sitio web para explotarla?
¿De que manera se vulnerará la privacidad de los usuarios, se filtrarán sólo direcciones de correo o será sin embargo todos los datos personales que se guardan?	
¿Es el activo accesible desde internet o sólo desde la red local?	

***Figura 3.3**: Ejemplos de preguntas para determinar el impacto y la complejidad de una vulnerabilidad*

Dependiendo de las respuestas a estas preguntas se les asignará unas puntuaciones que tras ser introducidas en una fórmula indicarán el riesgo que presenta una determinada vulnerabilidad. Se podría por ejemplo crear una fórmula como la de la figura 3.4 que indicaría el riesgo.

Riesgo = 0.5 * Impacto + 0.5 * (10 – Complejidad)

***Figura 3.4**: Fórmula inventada que podría determinar el riesgo de una vulnerabilidad*

Si en al fórmula anterior de la figura 3.4 en base a las respuestas de las preguntas anteriores, se les asigna valores del 0 al 10 tanto a impacto como a probabilidad se obtendrá un valor de riesgo del 0 al 10 indicando el riesgo que representará dicha vulnerabilidad

3.2.3.- Common Vulnerability Scoring System (CVSS)

Como se ha especificado anteriormente, la fórmula de la Figura 3.4 es un ejemplo y puede no reflejar de forma precisa el riesgo que representa una vulnerabilidad. Con el fin de encontrar un sistema de evaluación de vulnerabilidades preciso y estandarizarlo en la industria de la seguridad informática, el National Infrastructure Advisory Council(NIAC), oficina de Estados Unidos encargada de la seguridad de infraestructuras críticas tanto físicas como computacionales, comenzó en 2003 el desarrollo de el **Common Vulnerability Scoring System(CVSS)**.

El CVSS, que se encuentra ahora mismo en su segunda versión, trata de medir la gravedad de una vulnerabilidad en tres áreas mirando de proveer así una valoración bastante completa sobre la vulnerabilidad. Como resultado, generará cuatro puntuaciones, una para cada área y una cuarta global, además de un vector de texto que indicará las valoraciones individuales para cada una de las áreas. Las tres áreas evaluadas son:

- **Métricas base:** es el área que se centra en medir la complejidad y el impacto de una vulnerabilidad de forma objetiva. Esta a su vez se divide en los valores que se observan en la figura 3.5.

***Figura 3.5**: Valoración base del vector CVSS donde se puntúan de forma objetiva la complejidad y el impacto*

- **Métricas temporales:** área que me mide aquellas características que evolucionan con la vulnerabilidad a lo largo del tiempo y que están basadas en las métricas base. Con estas se consigue aportar una valoración subjetiva y se puntúan en tres áreas como las que se ven en la figura 3.6.

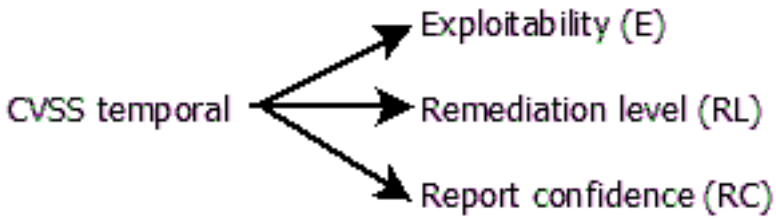

***Figura 3.6**: Diferentes valores que se evalúan en el vector temporal de CVSS*

- **Métricas ambientales:** esta área se basa en las dos anteriores y contempla las características que dependen de la implementación o de los sistemas asociados con la aplicación vulnerable. En la figura 3.7 se puede observar los valores que se evalúan.

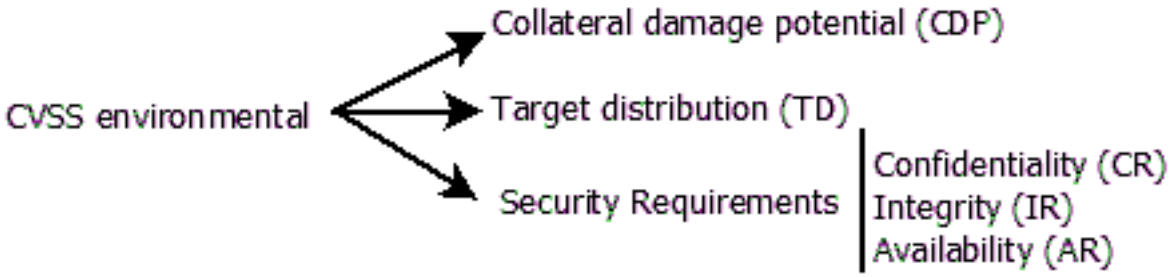

***Figura 3.7**: Diferentes valores que se evalúan en el vector temporal de CVSS*

Normalmente para evaluar una vulnerabilidad utilizando el CVSS lo que se acostumbra a utilizar es una calculadora[1] que ayudará a puntuar los distintos valores. Cada una de estas áreas se divide en diversos campos o sub-áreas que se puntuarán de forma individual y no numérica. Esto significa que la calculadora presentará una serie de opciones predefinidas que aportarán a la puntuación global un valor diferente dependiendo de la opción escogida.

Aunque no es necesario proveer de métricas temporales o ambientales para obtener una valoración utilizando una calculadora CVSS, si que es recomendable puesto que de otro modo la valoración sería exclusivamente objetiva.

El uso de CVSS como sistema para evaluar el riesgo que representa una vulnerabilidad está bastante extendido hoy en día, a parte de las empresas del sector privado y en específico aquellas que hacen auditorías de seguridad, hay muchas organizaciones que han adaptado éste sistema para evaluar la gravedad de las vulnerabilidades:

- MITRE con su Common Vulnerability Database (**CVE**)[2]. CVE es una base de datos de vulnerabilidades que pretende proveer de nombres comunes para los fallos conocidos públicamente facilitando así el intercambio de datos e información entre las diferentes herramientas, bases de datos y servicios.

- NIST con el National Vulnerability Database (**NVD**)[3] que es un sistema parecido al CVE pero mantenido y promovido por el gobierno de los estados unidos.

- Open Source Vulnerability Database (**OSVDB**)[4] que al igual que las dos anteriores es una base de datos de vulnerabilidades creada y mantenida por la comunidad open source que pretende ser imparcial a cualquier entidad, organismo o empresa.

En la figura 3.8 se puede ver un ejemplo de vulnerabilidad listada en NVD. En rojo quedan resaltados el identificador utilizado por CVE y el vector de CSSVv2 así como la puntuación base.

Vulnerability Summary for **CVE-2013-4113**

Original release date:07/13/2013

Last revised:08/30/2013

Source: US-CERT/NIST

Overview

ext/xml/xml.c in PHP before 5.3.27 does not properly consider parsing depth, which allows remote attackers to cause a denial of service (heap memory corruption) or possibly have unspecified other impact via a crafted document that is processed by the xml_parse_into_struct function.

Impact

CVSS Severity (version 2.0):

CVSS v2 Base Score: **6.8** (MEDIUM) (**AV:N/AC:M/Au:N/C:P/I:P/A:P**)

Impact Subscore: 6.4

Exploitability Subscore: 8.6

CVSS Version 2 Metrics:

Access Vector: Network exploitable; Victim must voluntarily interact with attack mechanism

Access Complexity: Medium

Authentication: Not required to exploit

Impact Type:Allows unauthorized disclosure of information; Allows unauthorized modification; Allows disruption of service

Figura 3.8: vulnerabilidad listada en NVD. En rojo quedan resaltados el identificador utilizado por CVE y el vector de CSSVv2 así como la puntuación base

3.3.- Open Web Application Security Project (OWASP)

Con el fin de agrupar y clasificar todo el conocimiento en relación al desarrollo seguro y seguridad en aplicaciones web, así como de proveer tanto documentación, como herramientas y metodologías libres y de código abierto nace en 2004 un organismo sin ánimo de lucro bajo el nombre de Open Web Application Security Project(**OWASP**)[5].

Dicho organismo está formado tanto por empresas como instituciones académicas e individuales donde los miembros contribuyen tanto económicamente con una subscripción anual como con colaboraciones en los distintos proyectos que promueve. Algunos de los proyectos más conocidos se pueden encontrar en la siguiente lista:

- **OWASP Application Security Verification Standard (ASVS)**: proyecto que trata de establecer una serie de verificaciones o comprobaciones a alto nivel con tal de clasificar la aplicación bajo uno de los cuatro niveles de seguridad que reconoce.

- **OWASP Enterprise Security API(ESAPI)**: librería de programación disponible para varios lenguajes de programación que facilita a los desarrolladores a programar aplicaciones más seguras.

- **OWASP Development guide**: guía con consejos y buenas prácticas para garantizar el desarrollo seguro de aplicaciones. Está disponible para varios lenguajes de programación.

- **OWASP Top 10[6]**: lista consensuada con la comunidad, actualizada aproximadamente cada tres años y que contiene las diez vulnerabilidades más críticas.

- **OWASP Testing guide[7]**: esta guía además de recopilar un conjunto de técnicas para buscar las vulnerabilidades más comunes dentro de aplicaciones web detalla una serie de procedimientos que se deberían seguir en el procesos iterativo de desarrollo del software para garantizar unos buenos estándares de seguridad.

- **OWASP Zed Attack Proxy(ZAP)**: aplicación potente pero poco portable que aporta bastantes funcionalidades con tal de tratar de ayudar en la identificación de vulnerabilidades de aplicativos web. Algunas de estas funcionalidades, como ya se ha mencionado anteriormente, son la inspección pasiva de tráfico HTTP y HTTPS, o el escaneo automático de vulnerabilidades.

3.3.1.- OWASP Top 10

Como se ha mencionado en el apartado anterior el OWASP Top 10 es una lista consensuada de las vulnerabilidades más críticas que se pueden encontrar en aplicativos web. Se debe distinguir y notar que son las diez más críticas y no las diez más comunes. Esta lista se actualiza y libera cada tres años donde la última versión liberada es del año 2013 e identifica las siguientes vulnerabilidades:

- **A1 Injection**: se consideran vulnerabilidades dentro de esta categoría aquellas en los que el atacante es capaz de explotarlas mandando texto sin formato con tal de aprovecharse del interprete de datos. Estos ataques pueden venir tanto de fuentes externas de datos, como podrían ser formularios en la aplicación web, como de fuentes internas de datos como bases de datos.

- **A2 Broken authentication and session management**: en este caso los atacantes se aprovecharán de debilidades en el sistema de autenticación o de control de sesiones(enumeración de cuentas, contraseñas, identificadores de sesión, etc) para suplantar usuarios.

- **A3 Cross-site scripting (XSS)**: son aquellas en las que el atacante se aprovecha de los fallos de validación de datos que la aplicación realiza permitiéndole ejecutar código malicioso en el navegador del usuario.

- **A4 Insecure direct object references**: este tipo de vulnerabilidades permite a los atacantes aprovecharse de la aplicación alterando parámetros que referencian directamente a elementos del sistema o usuarios sobre los cuales no tiene autorización.

- **A5 Security misconfiguration**: vulnerabilidades de esta categorías son aquellas que permiten al atacante acceso a áreas restringidas debido a malas prácticas o fallos durante la configuración del la aplicación o de los sistemas asociados. Ejemplos de estos serían: usuarios y contraseñas por defecto, páginas en desuso o de administración accesibles, vulnerabilidades conocidas sin parchear, ficheros y directorios sin proteger, etc

- **A6 Sensitive data exposure**: se clasifican en esta categoría aquellas vulnerabilidades en la que los atacantes se pueden aprovechar de datos críticos que han sido expuestos por una mala gestión de estos. Por ejemplo si se quisiese atacar a los algoritmos criptográficos no se haría directamente, sino que se trataría de robar las claves privadas o realizar ataques “man-in-the-middle” para robar los datos mientras estos se encuentran en tránsito entre el servidor remoto y el navegador del usuario

- **A7 Missing function level access control**: todas aquellas vulnerabilidades donde el atacante mediante la alteración del valor de una URL o de un parámetro acceda a una función sobre la cual no tiene privilegios pertenecerán a esta categoría

- **A8 Cross-site request forgery (CSRF)**: los atacantes que traten de aprovecharse de estas vulnerabilidades mirarán de engañar al usuario legitimo de la aplicación en enviar una petición concreta mediante un link, una imagen, una vulnerabilidad

de XSS o cualquier otra técnica, con tal de aprovecharse de los privilegios que dicho usuario posee en la aplicación, siempre y cuando este esté autenticado

- **A9 Using components with known vulnerabilities**: si un componente utilizado en la aplicación tiene vulnerabilidades conocidas y estas podrían ser detectadas mediante un análisis automático se clasificaran en esta categoría.

- **A10 Unvalidated redirects and forwards**: los ataques que se aprovechan de las vulnerabilidades de esta categoría lo hacen mediante un link legitimo de la aplicación que una vez cargado les redirigirá hacia otro sitio sobre el cual el atacante puede tener control

OWASP Top 10 2013 (Current version)
A1 Injection
A2 Broken Authentication and Session Management
A3 Cross-Site Scripting (XSS)
A4 Insecure Direct Object References
A5 Security Misconfiguration
A6 Sensitive Data Exposure
A7 Missing Function Level Access Control
A8 Cross-Site Request Forgery (CSRF)
A9 Using Components with Known Vulnerabilities
A10 Unvalidated Redirects and Forwards

***Figura 3.9**: Tabla resumen del OWASP Top 10 2013*

Se puede observar en la figura 3.9 la última versión liberada del OWASP Top 10(2013) y en la figura 3.10 las dos versiones anteriores en las que se puede ver la evolución de esta clasificación a lo largo de los años. Es interesante ver como más de una de estas vulnerabilidades se repiten como por ejemplo los ataques de cross-site scripting o los de inyección.

OWASP Top 10 2007	OWASP Top 10 2010
A1 Cross Site Scripting (XSS)	A1 Injection
A2 Injection Flaws	A2 Cross Site Scripting (XSS)
A3 Malicious File Execution	A3 Broken Authentication and Session Management
A4 Insecure Direct Object Reference	A4 Insecure Direct Object References
A5 Cross Site Request Forgery (CSRF)	A5 Cross Site Request Forgery (CSRF)
A6 Information Leakage and Improper Error Handling	A6 Security Misconfiguration
A7 Broken Authentication and Session Management	A7 Insecure Cryptographic Storage
A8 Insecure Cryptographic Storage	A8 Failure to Restrict URL Access
A9 Insecure Communications	A9 Insufficient Transport Layer Protection
A10 Failure to Restrict URL Access	A10 Unvalidated Redirects and Forwards

***Figura 3.10**: Anteriores versiones del OWASP Top 10 dónde se puede observar la evolución de éste*

3.3.2- OWASP Testing guide

Además de la serie de consideraciones a tener en cuenta durante el desarrollo del software con tal de alcanzar unos buenos estándares de seguridad en las aplicaciones web, la guía de testing de la OWASP contiene un conjunto de técnicas que trata de detallar como buscar e identificar las diferentes vulnerabilidades que pueden existir en estos aplicativos.

La última versión de la guía(v3) identifica nueve áreas sobre las que se deberá trabajar con tal de identificar potenciales fallos de seguridad que pudieran afectar a la aplicación. Estas áreas se presentan de la siguiente manera:

- **Configuration Management Testing**: las pruebas descritas en esta categoría están orientadas a identificar fallos en las políticas de gestión de configuración.

Muchas veces los escaneos infraestructurales o perimetrales que puedan ser llevados a cabo revelarán información como puede ser, métodos HTTP permitidos, funciones administrativas y configuraciones infraestructurales.

- **Authentication Testing**: en este área se evaluarán todas las secciones de la web que estén relacionadas con los procesos de autenticación como puede ser el formulario de identificación, si es posible enumerar usuarios o si por ejemplo los mecanismos de captcha funcionan adecuadamente

- **Session Management Testing**: las técnicas descritas en esta sección se focalizarán en evaluar los controles de seguridad para las sesiones que se establecen para controlar las interacciones de un usuario concreto, se mirará por ejemplo si substituyendo la sesión de un usuario por la de otro se podrá suplantar su identidad o si por ejemplo estas se destruyen de forma correcta

- **Authorization Testing**: las pruebas englobadas en esta categoría van dirigidas a evaluar si los controles de autorización funcionan correctamente, por ello se mirará por ejemplo si los usuarios tienen acceso única y exclusivamente a los datos sobre los que están autorizados o si estos pudieran de alguna forma escalar privilegios

- **Business Logic Testing**: los fallos de lógica de negocio presentados en esta categoría son quizá los más difíciles de identificar, pues requieren un profundo conocimiento de la aplicación y si existen son exclusivos y únicos para cada aplicación. Se tratará de buscar aquellas funcionalidades en las cuales alterando el flujo normal del aplicativo puedan ser aprovechadas para beneficio del atacante. Se puede pensar en estos como aquellas funcionalidades que no cumplen correctamente las especificaciones del diseño, que no siguen los casos de uso.

- **Data Validation Testing**: en esta categoría se recogen las pruebas que se recomiendan hacer para evaluar si las funcionalidades que reciben datos, ya sean originarios del usuario, de otros servicios o de la base de datos, validan éstos de

forma correcta y si los datos son limpiados de tal manera que no puedan afectar a los intérpretes que los vayan a utilizar.

- **Testing for Denial of Service**: esta sección engloba las pruebas dirigidas a evaluar si la aplicación es resistente a ataques de denegación de servicio. Ejemplos de pruebas asociadas serían por ejemplo: si la base de datos acepta caracteres comodín como '%' que pudiera ralentizar a e incluso inhabilitar o si es posible por ejemplo bloquear las cuentas de todos los usuarios del sistema.

- **Web Services Testing**: las pruebas de esta categoría se centran en evaluar lo que se conoce como 'web services' y que se pueden definir como servicios más o menos específicos a los cuales otras aplicaciones hacen peticiones. Una prueba que se podría realizar es tratar de enumerar todas las llamadas disponibles o ver si estas permiten acceso sin validar si el usuario está autorizado a realizar dicha llamada.

- **AJAX Testing**: en esta última sección de la guía se detalla como se debe evaluar la seguridad de las llamadas asíncronas hechas con JavaScript. Recoge una serie de pruebas basadas en las secciones anteriores pero enfocadas a esta tecnología

3.4.- Identificación de vulnerabilidades

Entendidos los conceptos de que es una vulnerabilidad, que representa y como se valora la gravedad o riesgo de ésta se procederá a explicar el proceso de identificación. Para ello se analizará los diferentes aspectos que se deberán tener en cuenta, antes, durante y después de este proceso, así como que herramientas pueden ser útiles para realizar esta búsqueda

3.4.1.- Requisitos

El proceso de buscar debilidades en el software es un proceso lento y laborioso que por lo general se puede enfocar de dos maneras distintas: manualmente o automáticamente. Se debe tener en cuenta que en ocasiones no es posible automatizar el proceso de búsqueda y que la automatización de éste puede dar lugar a falsos positivos que requerirán una posterior validación manual.

Sea cual sea el enfoque utilizado para buscar vulnerabilidades se debe saber primero que se está buscando y de que manera se debe hacer. La búsqueda requiere además de conocimientos de desarrollo, un profundo conocimiento de las tecnologías sobre las que se pretende buscar las vulnerabilidades, en este caso se necesitará tener un amplio conocimiento de las tecnologías web.

Por lo tanto además de conocimientos de desarrollo web(PHP, Python, Java, JavaScript, etc) y de administración de servidores web(Apache, nginx, Microsoft IIS, etc), el auditor deberá saber, por ejemplo, cómo funciona el protocolo HTTP, cómo se encapsula éste sobre SSL/TLS para dar lugar a HTTPS, diferentes sistemas de bases de datos y sus lenguajes de consulta, nociones programación asíncrona para evaluar tecnologías como AJAX, el formato de los mensajes MIME, etc además de otros conocimientos sobre redes y protocolos como DNS y FTP, etc.

El hecho de que cada aplicación sea diferente y la extensa lista de conocimientos requeridos hace que buscar vulnerabilidades sea una ardua y difícil tarea.

3.4.2.- Las diferentes etapas de la búsqueda

La búsqueda de vulnerabilidades se deberá plantear como un proceso que constará de tres partes como se observa en la figura 3.11 y se detalla más abajo:

***Figura 3.11**: Las tres partes de la búsqueda de vulnerabilidades: reconocimiento, identificación y validación*

Reconocimiento: el primer paso del proceso será definir el objetivo y los sistemas asociados al aplicativo para así obtener una visión general del entorno.

En este paso se tratará de identificar por ejemplo el servidor web sobre el que corre la aplicación, el lenguaje de programación utilizado para su desarrollo, la estructura de directorios utilizada por la aplicación, determinar si la web utiliza SSL y si utiliza que protocolos de cifrado permite, etc.

Identificación: una vez se ha reconocido el entorno y se ha ganado una visión global del aplicativo así como de los sistemas y tecnologías asociados a éste se procederá a tratar de identificar las vulnerabilidades o los diferentes vectores de ataque que pudieran existir.

Para ello la búsqueda se podrá realizar siguiendo las indicaciones de la guía de testing de la OWASP y centrándose en las nueve áreas de trabajo detalladas en el apartado 3.3.2, se mirará de garantizar que se cubre la mayor parte de la aplicación en el tiempo definido al inicio de la auditoría.

Validación: en este último paso se validará si la vulnerabilidad es realmente explotable y en caso de ser así se evaluará, posteriormente, el riesgo que ésta representa.

Con tal de realizar esta verificación se requerirá utilizar un navegador web actualizado, especialmente si se trata de una vulnerabilidad cuyo vector de ataque se ejecutará en el navegador del usuario, por ejemplo un fallo de XSS o de CSRF. Esto es así puesto que interesará comprobar si es posible aprovecharse del fallo en los navegadores que utilizan los usuarios de la aplicación.

Aunque es una vulnerabilidad en la aplicación web la que hace posible ataques como XSS, se debe tener en cuenta que los distintos motores encargados de convertir los lenguajes de "markup" como HTML o XML utilizados por los diferentes navegadores incorporan medidas de seguridad con tal de proteger a los usuarios de estos. Por lo tanto es posible que aunque una aplicación sea vulnerable, su riesgo sea más bajo de lo inicialmente estimado debido a esto.

3.4.3.- Herramientas

Conocido el proceso para buscar, dónde y cómo encontrar información acerca de los distintos tipos de vulnerabilidades web además de entender cuales son las más críticas sólo queda por entender que herramientas serán más útiles o cuáles pueden facilitar la tarea de búsqueda.

Aunque existen herramientas genéricas para buscar de forma automática casi cada tipo de vulnerabilidad, éstas por lo general no suelen ser fiables. Los cambiantes entornos y las diferentes naturalezas de los sitios web hace que éstas acaben requiriendo verificación manual.

Así que en cualquier caso se acabará requiriendo un análisis manual de vulnerabilidades. Si así fuese se podría utilizar el navegador, como se ejemplifica en la figura 3.12, pero pronto se observará que éste queda pequeño ya que no proporciona control total sobre las peticiones y respuestas que envía y recibe de y hacia el servidor remoto. De igual modo utilizar una navegador para la evaluación manual puede resultar más complicado puesto que no está pensado para esta tarea.

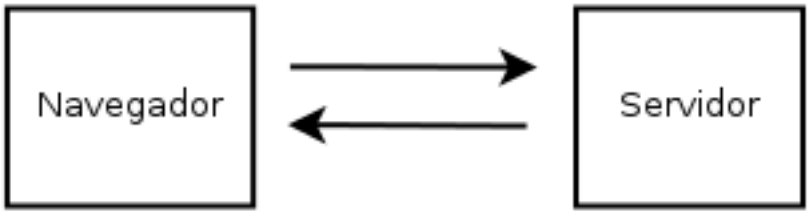

***Figura 3.12**: Navegador conectado directamente al servidor remoto*

Así que con tal de realizar un análisis de vulnerabilidades manual se necesitará otro tipo de aplicación que ofrezca más control sobre las peticiones y respuestas Es por ello que las herramientas que se presentan más útiles son las que se conocen como proxys de peticiones web o **proxys HTTP**.

***Figura 3.13**: Navegador conectado al servidor remoto mediante un proxy*

La mayoría de proxys de peticiones http permiten ver de forma sencilla e intuitiva el tráfico entre el navegador y el servidor puesto que se sitúan en mitad de la conexión y todo el tráfico es pasado a través de éstos como ha sido representado esquemáticamente en la figura 3.13.

4.- Propuesta de aplicativo para la evaluación de seguridad en webs

Se propone como parte del proyecto de investigación el desarrollo de un sencillo pero a la vez versátil y fácilmente extensible proxy de peticiones web que formará el núcleo de la aplicación sobre el cuál se proveerá con una básica interfaz gráfica de utilidad para los análisis de vulnerabilidades.

Para ello se han definido una serie de objetivos que se han tenido en cuenta durante el diseño e implementación de la herramienta:

- Tiene que ser un aplicativo sencillo e intuitivo, tanto para el uso de la interfaz gráfica como para el desarrollo de trabajos derivados.

- A ser posible debe ser independiente de librerías de terceros.

Dicha aplicación se conocerá bajo el nombre de **Web Security Framework** o por su acrónimo **WSF**.

Este capítulo está organizado en las siguientes secciones:

- **Casos de uso:** sección en la que se detallan los casos de uso que indican como deberá interactuar la aplicación con el usuario.

- **Diseño:** esta sección recoge todos los aspectos evaluados durante el diseño de WSF como son los requisitos tanto funcionales como no funcionales, la arquitectura del sistema y los diagramas de clase.

- **Herramientas y tecnologías:** las tecnologías y herramientas que han sido utilizadas a lo largo del proyecto se recogen en esta sección.

- **Implementación:** en esta sección se habla sobre alguna de las particularidades de la implementación.

- **Resultados:** en el último apartado de esta sección se presentan alguna de las pruebas más significativas realizadas para evaluar la correcta funcionalidad del aplicativo

4.1.- Casos de uso

Teniendo presentes los objetivos presentados en la sección anterior se han definido una serie de casos de uso que indican como deberá interactuar la aplicación con el usuario. Éstos ayudarán a definir e identificar potenciales requisitos que fuesen necesarios para el desarrollo y uso de la aplicación

4.1.1.- Inicio aplicación

Nombre: Inicio aplicación
Tipo: Primaria
Actores: Usuario, WSF
Descripción: El usuario ejecuta la aplicación
Curso normal de los acontecimientos:

Usuario	WSF
Inicia el aplicativo llamando al interprete de python y al fichero principal desde la línea de comandos	
	Inicializa el proxy en un hilo de ejecución
	Inicializa la interfaz

4.1.2.- Inicio modo interceptado

Nombre: Inicio modo interceptado

Tipo: Primario

Actores: Usuario, WSF

Descripción: Permite al usuario comenzar el modo de interceptado que permitirá interceptar tanto peticiones como respuestas entre el navegador y el servidor remoto.

Curso normal de los acontecimientos:

Usuario	WSF
Activa el modo de interceptado con el botón correspondiente de la pestaña "Intercept"	
	La interfaz gráfica pone el proxy en modo interceptado

Curso alternativo de los acontecimientos:

- **Caso A**: El modo de interceptado ya esta activado; el mismo botón presenta otra etiqueta("Stop") que ahora lo desactivará.

4.1.3.- Petición o respuesta recibida

Nombre: Petición o respuesta recibida

Tipo: Primario

Actores: WSF

Descripción: La aplicación recibe una petición por parte del navegador del usuario o una respuesta por parte del servidor y la presenta por pantalla en la pestaña del histórico

Requisitos: El servidor remoto ha enviado una respuesta a una petición anterior del usuario o el navegador del usuario ha enviado una petición

Curso normal de los acontecimientos:

WSF
Recibe una petición o una respuesta y la muestra en la pestaña del histórico
Reenvía la petición o respuesta recibida al destinatario correspondiente si el modo de interceptado no esta activo

Curso alternativo de los acontecimientos:

- **Caso A**: El modo de interceptado esta activado con lo que pasa al caso de uso 4.1.4

4.1.4.- Petición o respuesta interceptada

Nombre: Petición o respuesta interceptada

Tipo: Secundario

Actor: WSF

Descripción: La aplicación recibe una petición por parte del navegador del usuario o una respuesta por parte del servidor y la presenta por pantalla para que el usuario decida si quiere alterarla o no, todo esto lo hará antes de reenviarla al destinatario correspondiente

Requisitos: El modo de interceptado está activo y se llega a este caso desde el caso de uso 4.1.3

Curso normal de los acontecimientos:

WSF
Muestra la respuesta o petición en modo texto en el editor de la pestaña de interceptado y espera acción por parte del usuario

4.1.5.- Edición de petición o respuesta interceptada

Nombre: Edición de petición o respuesta interceptada

Tipo: Secundario

Actores: Usuario, WSF

Descripción: El usuario edita la petición o la respuesta interceptada y apreta el botón de reenviar al destinatario correspondiente

Requisitos: Se llega a este estado des de el caso de uso 4.1.4

Curso normal de los acontecimientos:

Usuario	WSF
Edita una petición o una respuesta interceptada	
Aprieta el botón de enviar	
	Reconstruye la petición o la respuesta y la reenvía al destinatario correspondiente

4.1.6.- Fin modo interceptado

Nombre: Fin modo interceptado

Tipo: Primario

Actores: Usuario, WSF

Descripción: Permite al usuario parar el modo de interceptado

Curso normal de los acontecimientos:

Usuario	WSF
Presiona el botón de “Stop” de la ventana Intercept”	
	La interfaz desactiva el modo de interceptado del proxy

Curso alternativo de los acontecimientos:

- **Caso A**: El modo de interceptado ya esta activado; el mismo botón presenta otra etiqueta(“Stop”) que ahora lo desactivará.
- **Caso B**: Si hay una petición o respuesta interceptada esta se reenviará al destinatario correspondiente y se parara el modo de interceptado

4.1.7.- Carga sesión

Nombre: Carga sesión

Tipo: Primario

Actores: Usuario, WSF

Descripción: El usuario decide cargar en WSF una sesión de trabajo guardad previamente

Curso normal de los acontecimientos:

Usuario	WSF
Elige el fichero que desea cargar desde el menú ‘File’ submenú ‘Load session’	
	La nueva sesión es cargada y la interfaz actualizada con los datos de ésta

Curso alternativo de los acontecimientos:

- **Caso A**: Se han realizado peticiones en la sesión actual. WSF presenta al usuario la posibilidad de guardar la sesión actual antes de cargar la otra

4.1.8.- Guarda sesión

Nombre: Guarda sesión

Tipo: Primario

Actores: Usuario, WSF

Descripción: El usuario decide salvar la sesión de trabajo en una ruta específica

Curso normal de los acontecimientos:

Usuario	WSF
Elige desde el menú 'File; la opción 'Save session' e introduce el nombre del fichero y la ruta donde quiere guardar dicha sesión de trabajo	'
	La sesión es guardad en la ruta especificada

4.1.9.- Cierra aplicación

Nombre: Cierra aplicación

Tipo: Primario

Actores: WSF, Usuario

Descripción: El usuario cierra la aplicación

Curso normal de los acontecimientos:

Usuario	WSF
Sale de la aplicación presionando la cruz de la barra de controles	
	Presenta al usuario la opción de guardar la sesión en caso de haberse realizado alguna petición
	Destruye el fichero de sesión temporal
	Cierra el interfaz

4.2.- Diseño aplicativo

Este apartado recoge todos los aspectos evaluados durante el diseño de WSF como son los requisitos tanto funcionales como no funcionales, la arquitectura del sistema y los diagramas de clase.

4.2.1.- Requisitos

En esta sección se evalúan tanto los requisitos funcionales como los no funcionales que se han establecido para el desarrollo de Web Security Framework.

4.2.1.1- Requisitos funcionales

WSF ha sido desarrollada y probada en un sistema operativo GNU/Linux por lo tanto se espera quede requisitos funcionales que se pueden clasificar en dos áreas: núcleo y entorno gráfico:

- **Núcleo**: Como su propio nombre indica responde a características independientes de la interfaz gráfica y básicas para el funcionamiento de la aplicación

Requisito	Descripción
Modular	El núcleo de WSF deberá ser modular hecho que implica que sus componentes deberán ser en la medida de lo posible independientes y re-aprovechables.
Portable	WSF deberá ser en la medida de lo posible auto-contenido e independiente de librerías de terceros.

- **Entorno gráfico:** éste deberá ser no bloqueante e incluir las siguientes funcionalidades:

Requisito	Descripción
Histórico de peticiones y respuestas	Mostrará por pantalla la lista de peticiones que se han ido realizando al servidor remoto. Cuando se seleccione una también mostrará la respuesta recibida por parte del servidor
Editor de mensajes interceptados	Permitirá poner el proxy en modo bloqueante, significando que parará todos los mensajes que reciba, y permitirá editar estos para posteriormente mandarlos alterados
Cargar y salvar sesiones	Durante el uso de la aplicación las peticiones y respuestas que se vayan recibiendo y enviando durante la sesión de trabajo se podrán salvar en ficheros que posteriormente podrán ser cargados

4.2.1.2- Requisitos no funcionales

Durante el desarrollo de la aplicación se ha tenido en cuenta los siguientes estrictos requisitos no funcionales:

Requisito	Descripción
Conexión internet	La aplicación sólo requerirá conexión a internet para realizar auditorías a sitios externos. Si el sitio está alojado en la red local ésta no dependerá de internet.
Lenguaje desarrollo	WSF se desarrollará en la versión 2.x de python en lugar de la 3.x debido a un mayor número de librerías disponibles para dicha versión.

Idioma	Todos los nombres de variables, clases y comentarios del código fuente de la aplicación estarán en inglés así como lo estará interfaz gráfica. La idea detrás de esta decisión es llegar a más gente y facilitar los trabajos derivados.

4.2.2.- Arquitectura del sistema

Como se puede apreciar en la figura 4.1 WSF se deberá situar en medio de la conexión entre el navegador del usuario y del servidor con tal de que éste pueda recibir el tráfico HTTP que se origina en el navegador.

***Figura 4.1**: Navegador conectado al servidor remoto utilizando WSF como proxy*

A nivel arquitectural se puede imaginar esta caja de WSF como dos grandes bloques que interaccionan entre sí tal y como se ve en la figura 4.2. El primero de ellos será el proxy que contendrá toda la lógica de la recepción y envío de peticiones y respuestas así como parte de los controles de los mecanismos de sincronizado que requiera la interfaz gráfica(GUI).

El segundo bloque será el encargado de la lógica de la interfaz gráfica con la que interaccionará el usuario así como de la gran parte del control de los mecanismos de sincronizado que se utilizarán para que esta sea no bloqueante.

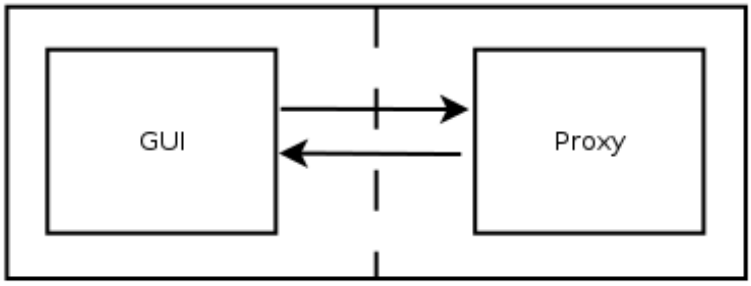

***Figura 4.2**: Arquitectura de WSF. Los dos grandes bloques que se identificarán*

4.2.3.- Diagramas de clases

El diagrama general de clases de la aplicación resulta bastante sencillo como se puede observar en la figura 4.3

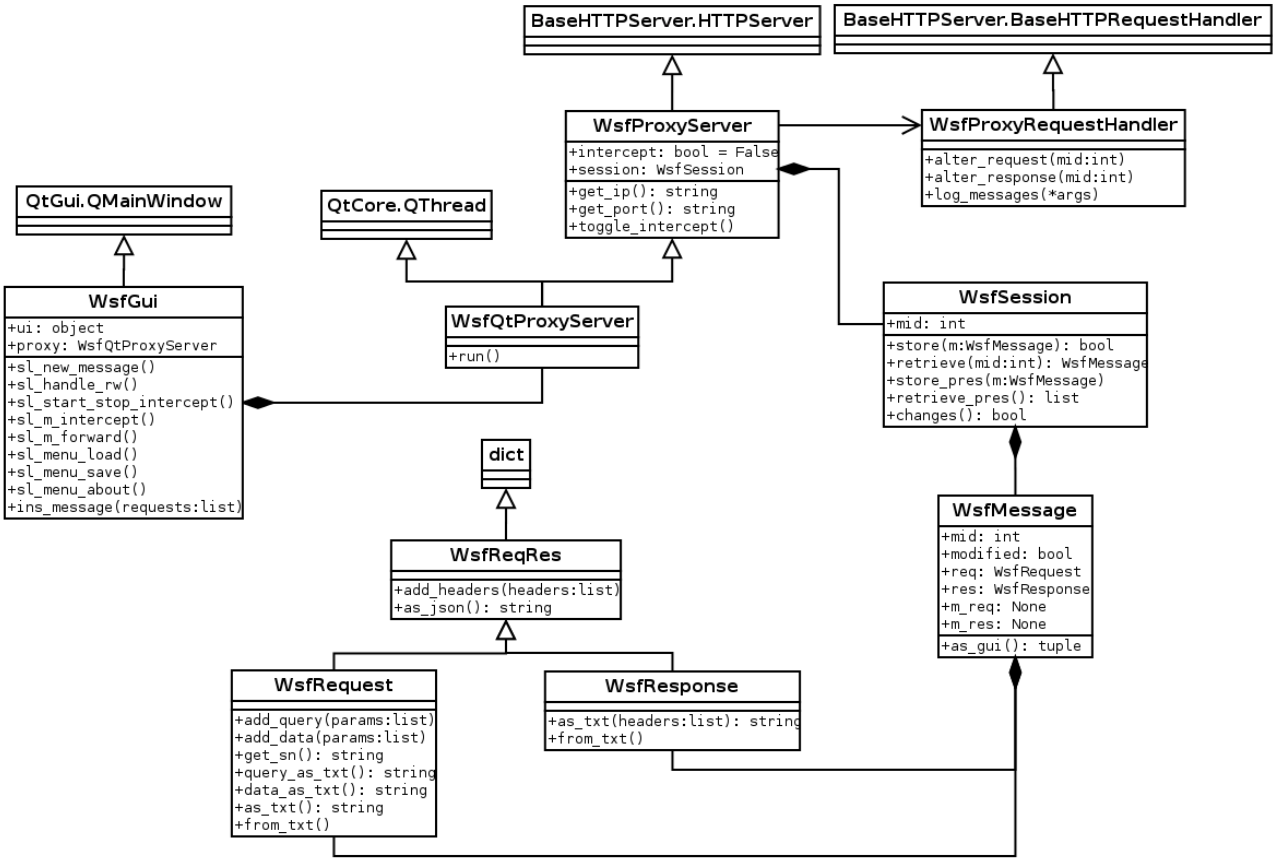

***Figura 4.3**: Aspecto que tiene el diagrama de clases de la aplicación*

4.2.3.1.- WsfReqRes

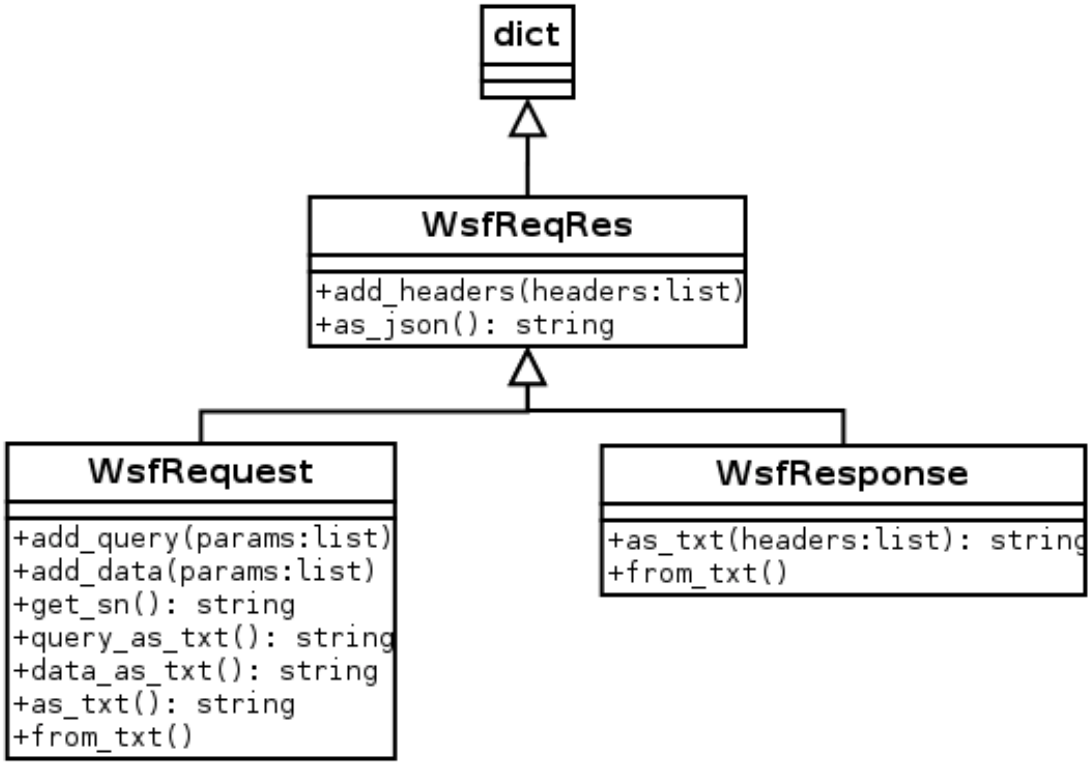

***Figura 4.4**: Aspecto que tiene el diagrama de clases de la aplicación*

WsfReqRes, como se ve en la figura 4.4, será una clase que heredará y extenderá el objeto de python "dict" que representa un diccionario. Sobre ésta se implementarán funcionalidades comunes que heredarán las clases WsfRequest y WsfResponse. WsfRequest y WsfResponse contendrán los objetos lógicos de peticiones y respuestas respectivamente.

Estas dos subclases además contendrán los métodos específicos para el uso de cada una. En especial es interesante destacar los métodos as_txt() y from_txt() dónde el primero permitirá devolver la representación en modo texto de dicha petición o respuesta y el segundo que dada una petición o una respuesta en modo texto creará el objeto correspondiente.

4.2.3.2.- WsfSession y WsfMessage

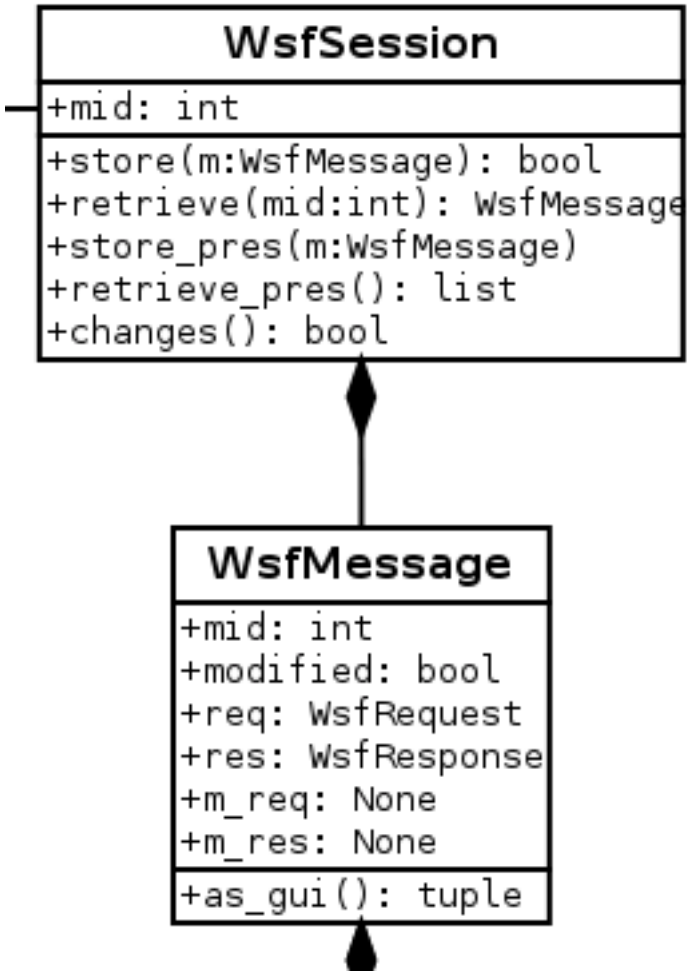

***Figura 4.5**: Diagramas de clases de WsfMessage y WsfSession*

WsfMessage será la clase que una vez instanciada hará de contenedora de los mensajes intercambiados entre el servidor remoto y el navegador del usuario, esto es, cada par de petición y respuesta se representará con una instancia de WsfMessage como se observa en la figura 4.5.

WsfSession por el contrario será la clase que contendrá todos los mensajes de una sesión de trabajo y que proveerá de los métodos necesarios para guardarlos o recuperarlos.

4.2.3.3.- WsfProxyServer y WsfProxyRequestHandler

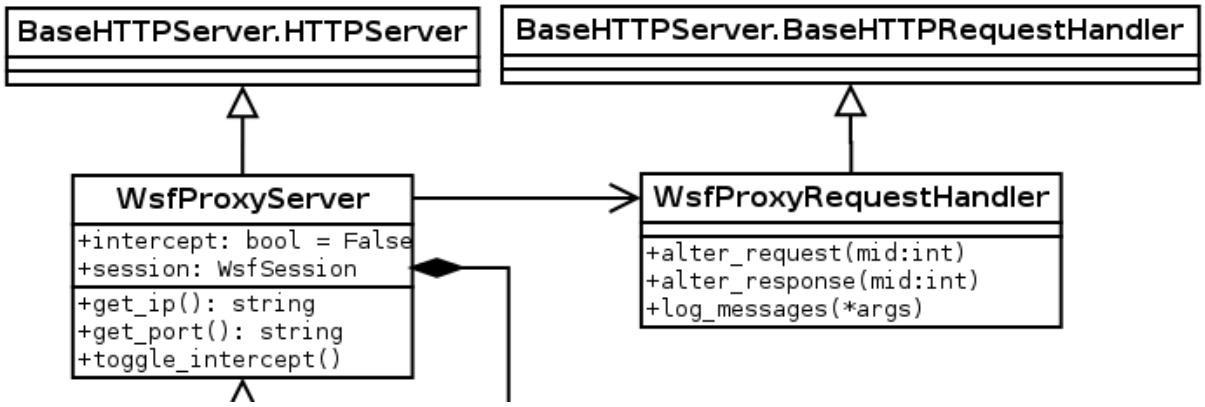

***Figura 4.6**: Diagramas de clase de WsfProxyServer y WsfProxyRequestHandler*

Estrechamente relacionadas como se ve en la figura 4.6 estas dos clases manejarán toda la lógica de conexiones y decidirán que hacer con las peticiones recibidas. Concretamente WsfProxyServer será un servidor de peticiones HTTP el cual instanciará la clase del tipo WsfProxyRequestHandler que se encargará de procesar cada una de las peticiones recibidas en caso de ser necesario o indicado por el usuario.

4.2.3.4.- WsfQtProxyServer

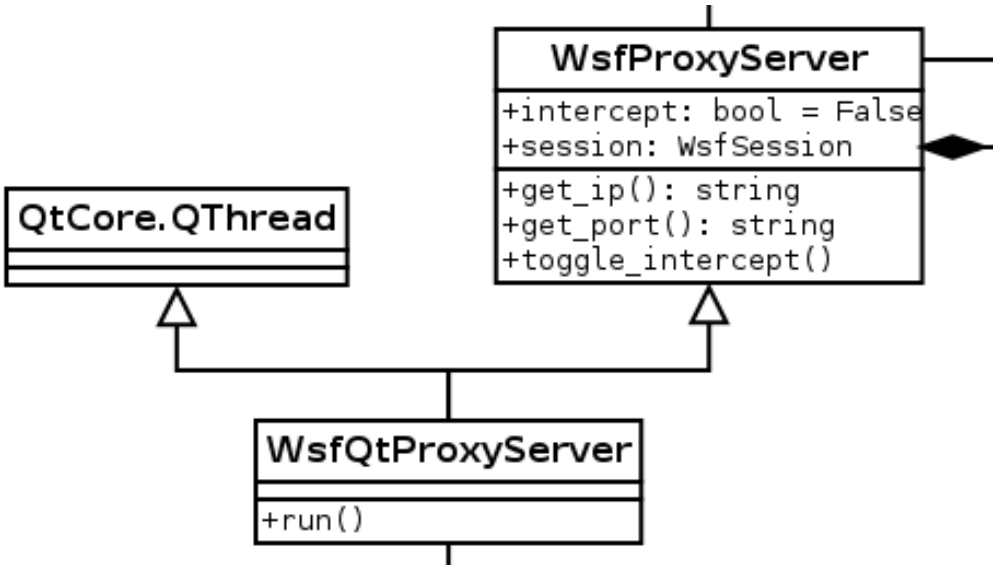

***Figura 4.7**: Diagrama de clase de WsfQtProxyServer*

WsfQtProxyServer hereda de WsfProxyServer y QThread y actuará como envoltorio de éstas dos con el añadido de que se encargará gestionar los mecanismos necesarios para el sincronizado entre el proxy la interfaz gráfica. El detalle de esta clase se puede ver en la figura 4.7.

4.2.4.5.- WsfGui

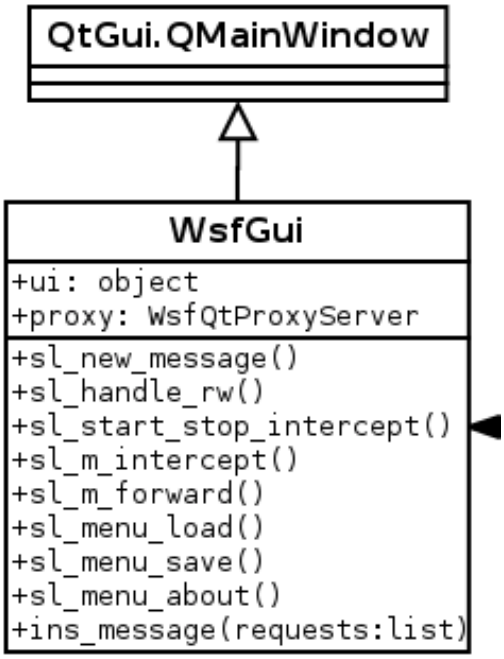

***Figura 4.8**: Diagramas de clase de WsfProxyServer y WsfProxyRequestHandler*

Una vez instanciada WsfGui, cuyo diagrama de clase se puede ver en la figura 4.8, construirá la ventana principal del programa y contendrá toda la lógica de ésta, así como los diferentes controles y eventos que otras clases podrán utilizar para notificar de cambios a la interfaz.

4.3.- Herramientas y tecnologías

A lo largo del proyecto se han utilizado una serie de tecnologías y herramientas que vale la pena estudiar brevemente.

4.3.1- Tecnologías

Con los objetivos anteriormente definidos se han seleccionado las tecnologías que mejor parecían adaptarse a las necesidades:

- **Python**: potente lenguaje de programación de alto nivel, libre y de código abierto. Altamente versátil, pues es posible desarrollar en python siguiendo diversos paradigmas de programación como podría ser programación imperativa, funcional u orientada a objetos, en este caso caso se ha escogido el último. Además dispone de un extenso conjunto de módulos que permiten realizar todo tipo de tareas.

- **Qt**: es un potente framework(conjunto de librerías y herramientas relacionados entre sí y que han sido diseñados para resolver un problema específico) orientado al desarrollo de interfaces gráficas para aplicaciones que rivaliza directamente con GTK. La preferencia de utilizar Qt sobre GTK es debido al potente módulo que facilita la integración con python y también a una muy mejor documentación.

- **SQLite**: es una base de datos relacional auto-contenida en un único fichero que no requiere de servidor o configuración alguna para el funcionamiento de esta. Esto la convierte en la base de datos ideal para pequeños proyectos que necesiten todo el potencial de una base de datos relacional además de en la candidata perfecta para almacenar datos asociados a una aplicación de forma persistente.

4.3.2- Herramientas

Para la realización de este proyecto se han utilizado las siguientes herramientas:

- **Vim**: versátil editor de texto gratuito y de código abierto basado en el famoso y extendido edior 'vi'. Vim se ha utilizado como la principal y única herramienta para el desarrollo de código aprovechando está disponible para diversas plataformas así como las características que hacen que desarrollar en éste sea realmente cómodo, como por ejemplo el resaltado de sintaxis, la búsqueda de patrones en el texto utilizando expresiones regulares o las macros de edición.

- **QtDesigner**: herramienta perteneciente al framework de Qt. QtDesigner permite de forma sencilla y muy visual diseñar interfaces de aplicaciones. Éstas serán guardada en un fichero de definiciones que posteriormente pueden ser cargado y modificado dinámicamente por el código de la aplicación.

4.4.- Implementación

Toda la aplicación gira en torno a un módulo de python llamado **wsf**. Éste módulo a su vez se divide en tres sub-módulos, como se observa en la figura 4.9, dónde cada uno de ellos contiene las siguientes funcionalidades:

- **wsf.http**: módulo encargado de la gestión del proxy. Dentro de este se encuentra un único fichero python llamado **proxy.py**. Dicho fichero contiene toda la lógica de las conexiones que se realizarán entre el navegador y el proxy, y entre el proxy y el servidor remoto. El planteamiento a nivel lógico es sencillo, está compuesto por un servidor de peticiones y un agente responsable de procesar las peticiones, por cada petición que reciba el servidor se ejecutará dicho agente que actuará según lo establecido.

- **wsf.session**: por un lado gestiona los mensajes, o lo que es lo mismo los pares de petición y respuesta, y por el otro gestiona el control de sesiones, que son todos aquellos mensajes pertenecientes a una sesión de trabajo. Estas funcionalidades están divididas en dos ficheros: **messages.py** y **session.py** respectivamente.

- **wsf.ui**: este último módulo contiene las funcionalidades asociadas a la interfaz gráfica. Dentro de él se puede encontrar dos ficheros: **wsf.ui** y **wsfgui.py**. El primero de ellos, wsf.ui, es un fichero XML que define todos los elementos de la interfaz gráfica: su tamaño, posición, propiedades, etc., el segundo, wsfgui.py, contiene la lógica de la interfaz gráfica así como un proxy heredado de wsf.http.proxy y con las modificaciones necesarias para que se pueda ejecutar en un hilo independiente.

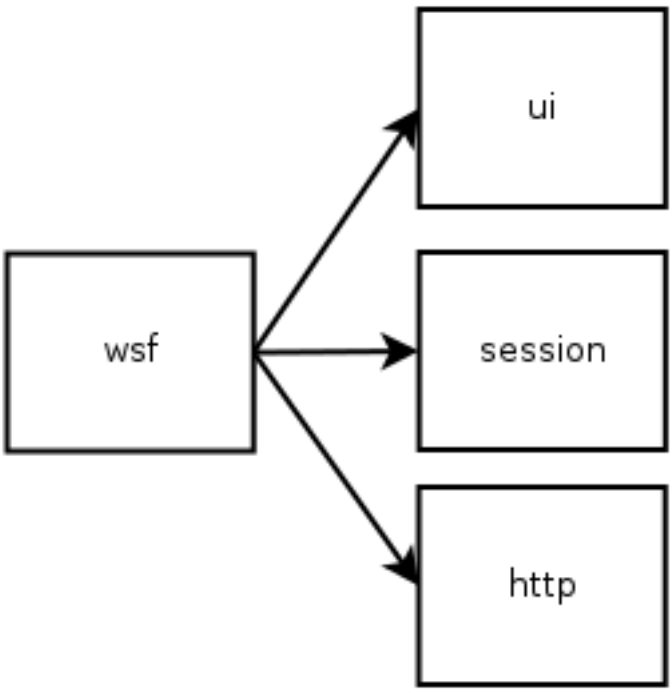

***Figura 4.9**: Arquitectura de WSF. Los dos grandes bloques que se identificarán*

Esta sencilla estructuración posibilita la invocación de la mayoría de las distintas funcionalidades de forma independiente al resto. Por ejemplo si se quisiese instanciar un objeto de petición sin necesidad de cargar todo el módulo wsf se tendría que hacer como

se demuestra en la figura 4.10.

```
from wsf.session.messages import WsfRequest as wReq

w = wReq(method=”GET”, path=”/”, protocol=”1.1”, scheme=”http://”,
netloc=”ub.edu”, url=”http://ub.edu/”)
```

Figura 4.10: Ejemplo de uso del módulo wsf en el que se instancia un objeto de petición sin necesidad de cargar todo el módulo

El ejemplo de la figura 4.10 además muestra como sería posible incluir este módulo en otros proyectos cumpliendo así los requisitos de modularidad y portabilidad.

4.5.- Resultados

Mostrada y entendida la arquitectura de la aplicación y cómo se ha implementado WSF es momento para mostrar algunos ejemplos de ésta en funcionamiento.

4.5.1.- Inspección pasiva de tráfico HTTP

Una de las tareas que permite la aplicación es el inspeccionado de los mensajes HTTP mandados entre el navegador del usuario y el servidor remoto. Esta funcionalidad es útil para hacerse una idea de cómo funciona el aplicativo web, con que servicios interactúa y de que manera lo hace.

Este es el comportamiento por defecto de WSF cuando esta se ha configurado como servidor de proxy del navegador. Todas las peticiones y respuestas HTTP se verán reflejadas en la pestaña “History” de WSF.

En la siguitente figura 4.11 se detalla como interaccionan todas las partes involucradas en el proceso. Se puede observar como WSF ha sido separado en los dos bloques principales descritos en el apartado anterior “4.4.- Implementación”. Debajo se puede

encontrar una detallada explicación de este proceso:

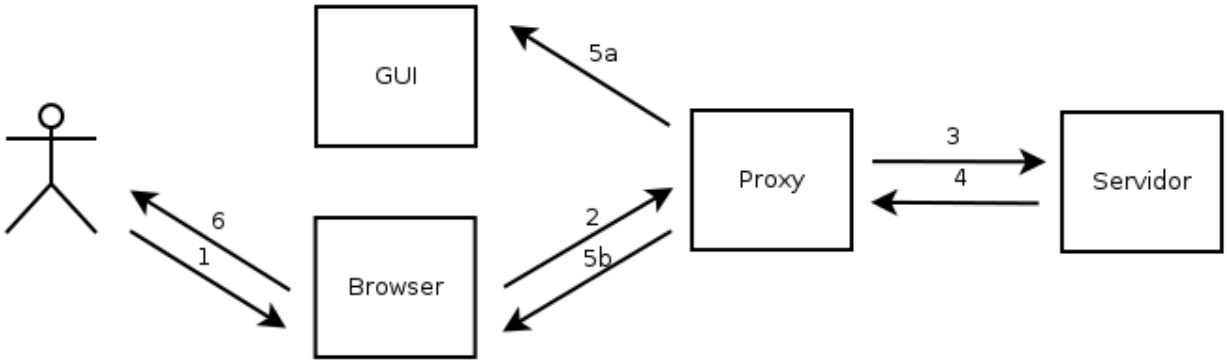

***Figura 4.11**: Inspección de tráfico HTTP con WSF*

1. El usuario interactua con el navegador y visita un sitio web.
2. El navegador, que ha sido configurado previamente para utilizar WSF como proxy, envía esta petición al proxy.
3. El proxy reenvía la petición original del navegador al servidor remoto.
4. El servidor remoto responde a la petición y la devuelve al proxy que es el que ha iniciado dicha conexión.
5. Aquí se ejecutan dos acciones de forma simultanea.
    a. Se actualiza la interfaz con la petición y la respuesta recibida Figura 4.12.
    b. Se redirecciona la respuesta al navegador.
6. El usuario ve la respuesta de los datos inalterados procesados en el navegador.

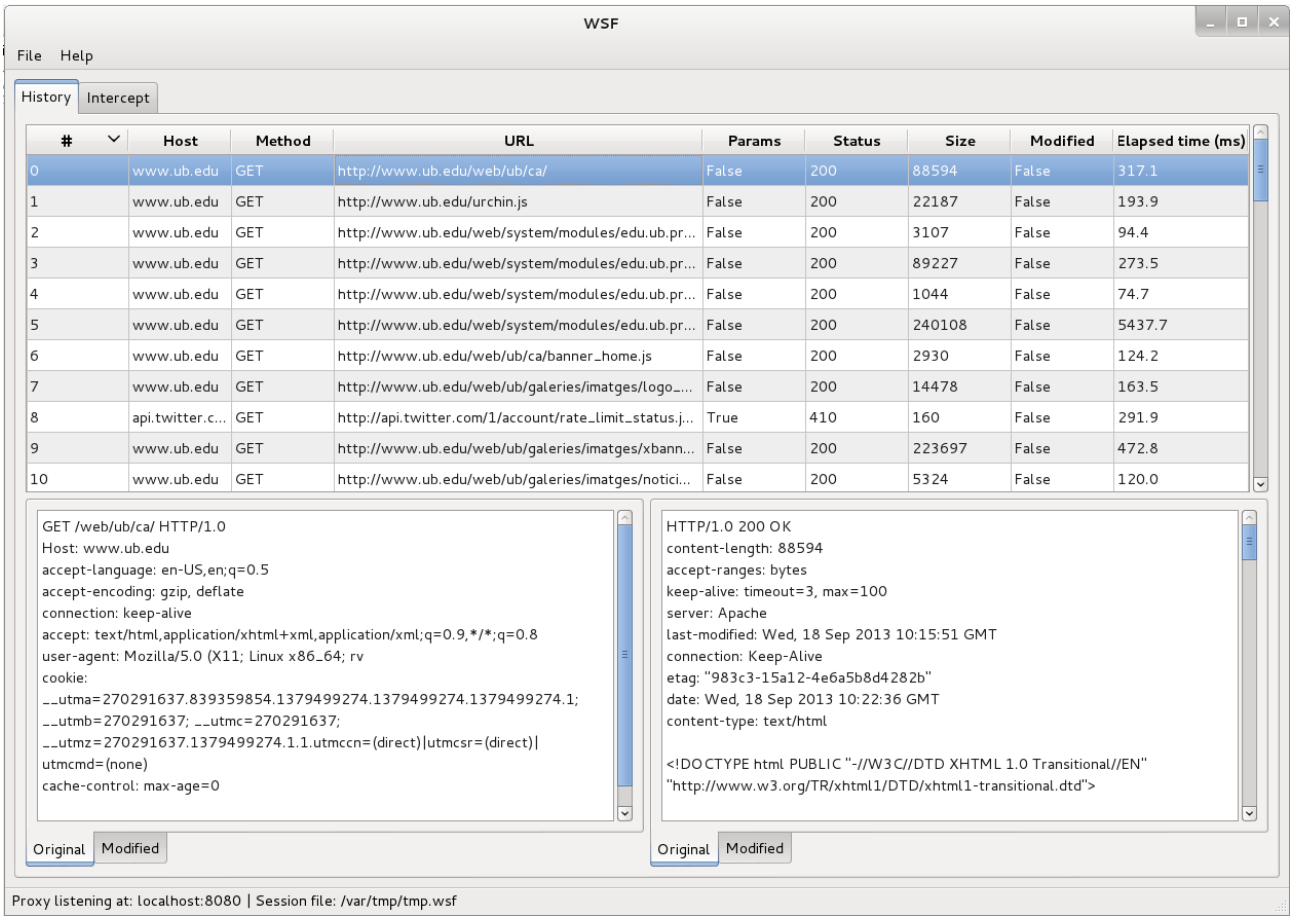

***Figura 4.12**: Interfaz actualizada con las peticiones y respuestas recibidas*

Como se observa en la figura 4.12 WSF es capaz de analizar de forma pasiva el tráfico de mensajes HTTP que hay entre el navegador y el servidor remoto.

4.5.2.- Interceptado de tráfico HTTP

El interceptado de tráfico es activado bajo la pestaña “Intercept”(figura 4.14) y es relativamente similar a la inspección pasiva de tráfico HTTP. La única diferencia es que cada vez que llega tráfico al proxy y antes de enviarlo al servidor o navegador según corresponda este se presentará en la interfaz permitiendo al usuario alterarlo.

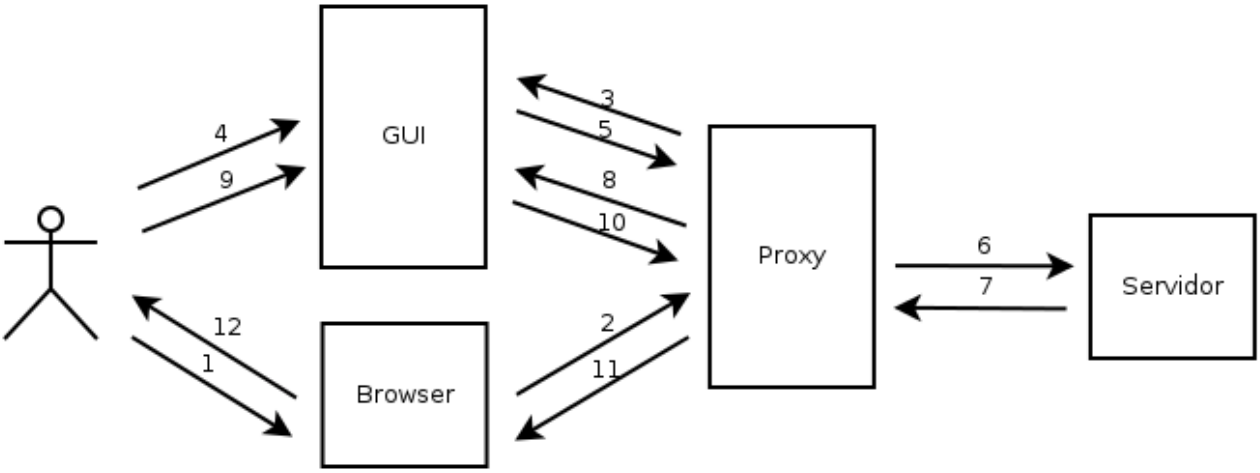

***Figura 4.13**: Todas las partes involucradas en el interceptado y modificación de tráfico HTTP con WSF*

Se puede observar en figura 4.13 como interaccionan todas las partes involucradas en el interceptado de tráfico:

1. El usuario interactua con el navegador y visita un sitio web.
2. El navegador, que ha sido configurado previamente para utilizar WSF como proxy, envía esta petición al proxy.
3. El proxy notifica a la interfaz gráfica y ésta presenta por pantalla al usuario lo que el navegador ha mandado hacia el servidor remoto.
4. El usario decide si modificar la petición.
5. Posteriormente presiona el botón “Forward” que notificará al proxy que puede redirigir esta al servidor remoto.
6. El proxy reenvía la petición original del navegador al servidor remoto.
7. El servidor remoto responde a la petición y la devuelve al proxy que es el que ha iniciado dicha conexión.
8. El proxy notifica a la interfaz gráfica y ésta presenta por pantalla al usuario la respuesta del servidor(figura 4.14).
9. El usuario decide si modificar la respuesta del servidor(figura 4.15).
10. Posteriormente presiona el botón “Forward” que notificará al proxy que puede redirigir esta al navegador.
11. Se redirecciona la respuesta al navegador.
12. El usuario ve la respuesta procesada por el navegador(figura 4.16).

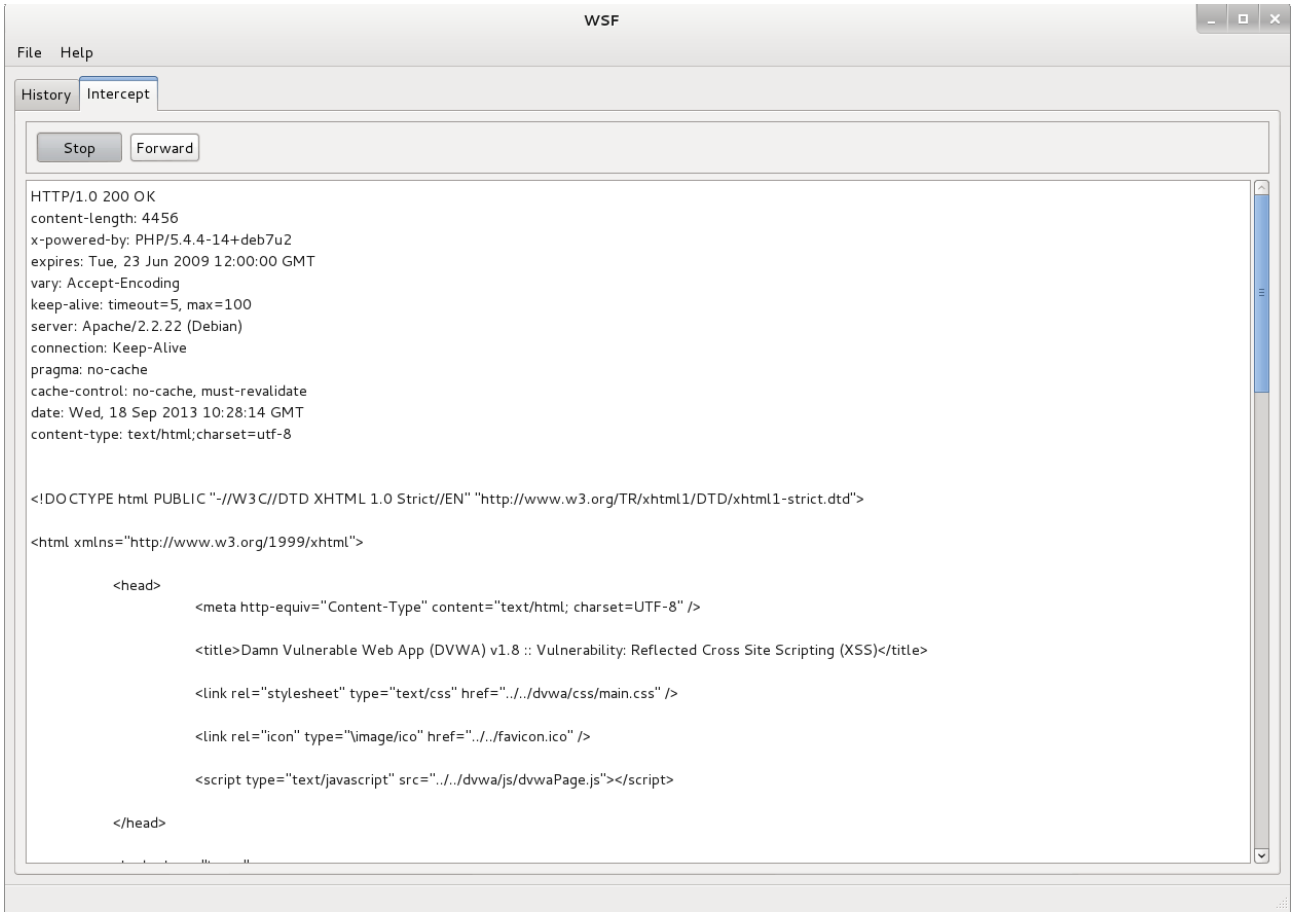

***Figura 4.14**: La funcionalidad de interceptado muestra al usuario la respuesta del servidor*

En las figuras 4.15 y 4.16 se evidencia como la modificación y el posterior renvio de la respuesta al navegador del cliente permiten la modificación en caliente de estos mensajes HTTP.

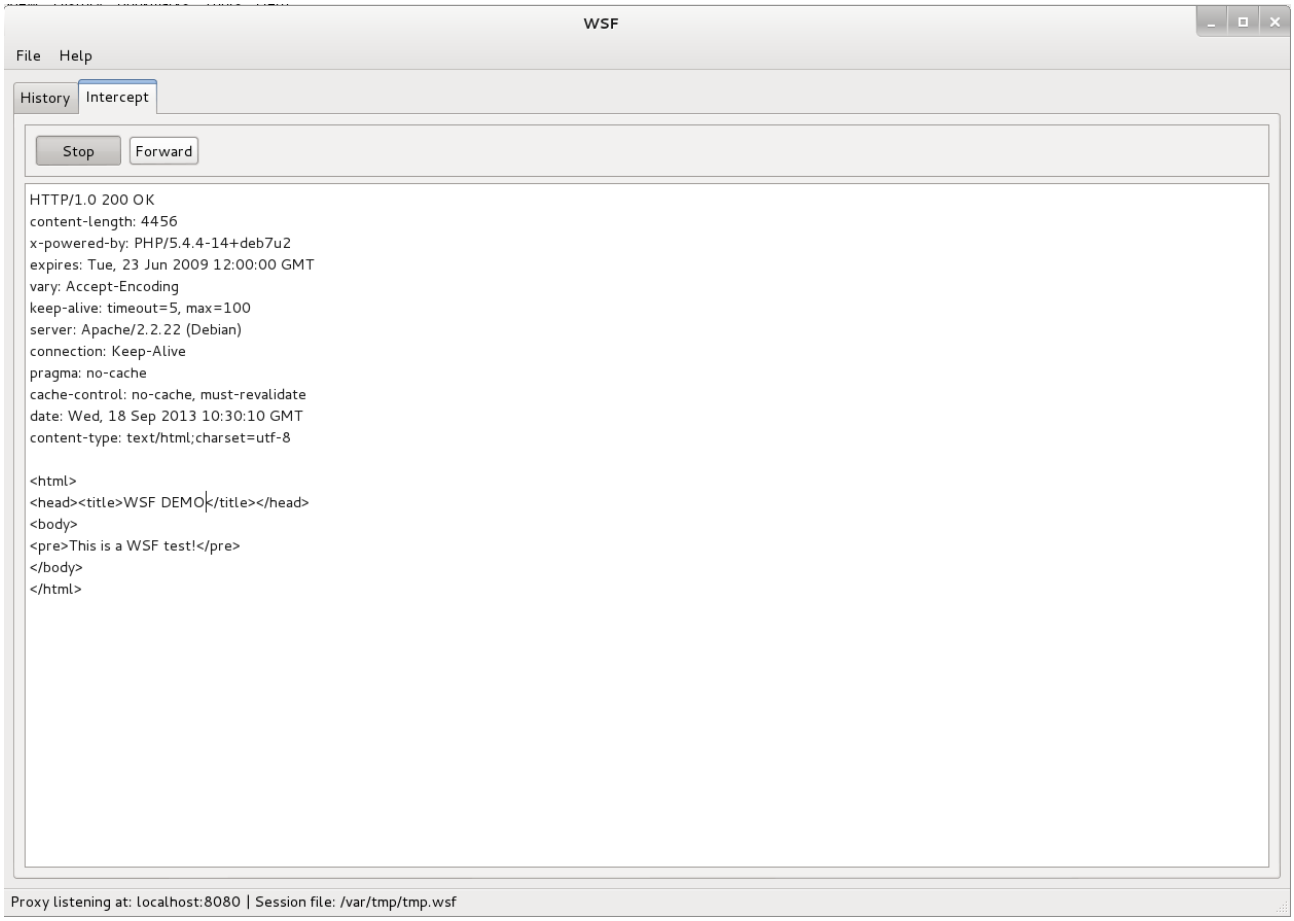

***Figura 4.15**: El usuario modifica esta manualmente*

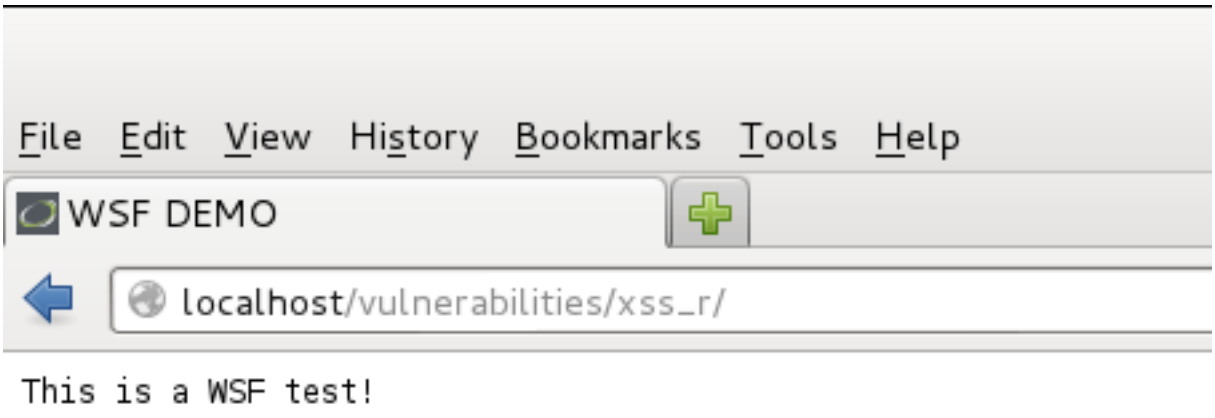

***Figura 4.16**: La respuesta modificada se devuelve al navegador*

5.- Planificación y costes

Durante la planificación inicial se tuvieron en cuenta las siguientes consideraciones:

- Las semanas laborales constan de 5 días
- Cada jornada es de 8 horas útiles
- Los posibles días festivos que ocurriesen durante el desarrollo del proyecto serían contabilizados como laborables

Con estas consideraciones en mente se definió la lista de tareas de la figura 5.1 que se darían a lo largo del proyecto y se estimó el tiempo que costaría finalizar cada una de ellas.

	Estimado (Días)
Análisis del problema	2.5
Definición casos de uso	1.5
Evaluación tecnologías	5.0
Diseño	7.5
Implementación núcleo	22.0
Implementación interfaz	10.0
Pruebas validación	6.5
Documentación	9.0

Total días	64.0
Total horas	512

***Figura 5.1**: Lista de tareas definida y estimación del coste de cada una*

Dada esta lista de tareas y la estimación de días para cada una de ellas se puede observar que la planificación inicial estimaba que el proyecto estaría finalizado en **64 días** laborables o en un total de **512 horas** que se pretendían distribuir tal y como se indica en el diagrama de Gantt de la figura 5.2.

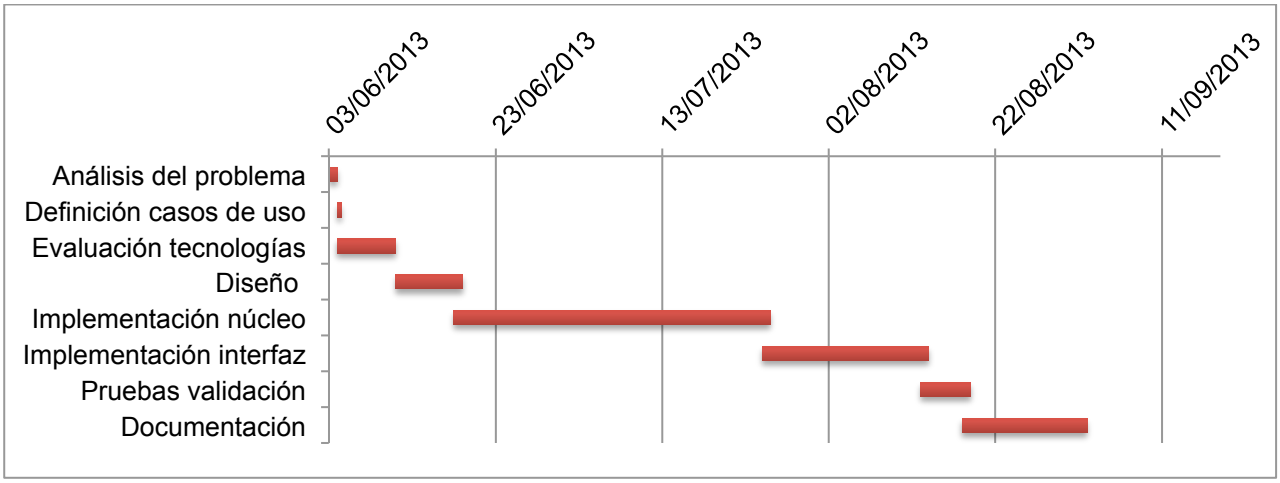

***Figura 5.2**: Diagrama de Gantt de los datos estimados*

Durante el desarrollo se ha experimentado una serie de eventualidades que han tenido un impacto directo sobre la planificación y obviamente sobre el coste total del proyecto. En la figura 5.3 se presenta el diagrama de Gantt de tareas realizadas a lo largo del desarrollo del proyecto actualizado con los datos reales:

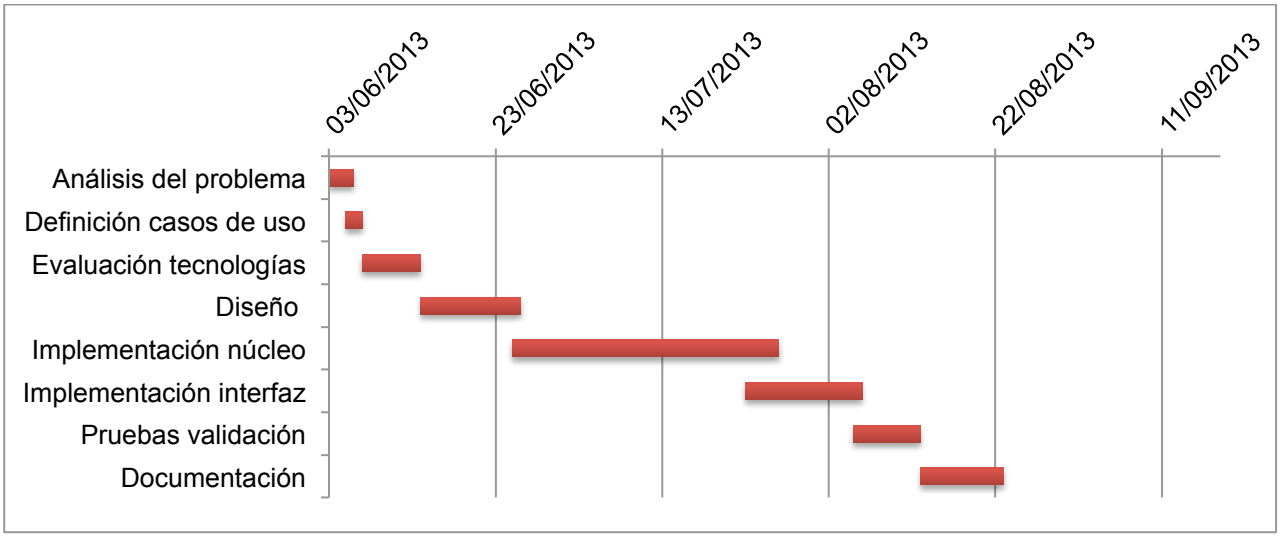

***Figura 5.3**: Diagrama de Gantt con los datos reales*

En la tabla de la figura 5.4 se puede apreciar una comparativa entre los datos estimados y los reales donde en rojo quedan resaltados los resultados finales.

	Estimado (Días)	Real (días)	Real (horas)
Análisis del problema	2.5	1	8
Definición casos de uso	1.5	0.5	4
Evaluación tecnologías	5.0	5	40
Diseño	7.5	6	48
Implementación núcleo	22.0	27.5	220
Implementación interfaz	10.0	15	120
Pruebas validación	6.5	4	32
Documentación	9.0	11	88

Total días	64.0	**70.0**
Total horas	512	**560**

***Figura 5.4**: Comparativa entre los datos estimados y los reales donde en rojo quedan resaltados los resultados finales*

Se puede observar entonces que los 64 días laborables estimados al inicio del proyecto se han convertido en 70 lo que supone al coste del proyecto una carga adicional de 48 horas.

560 horas * 25€/hora = 14.000€

Si se tiene en cuenta que el programador que desarrolla el proyecto cobra aproximadamente 25€ por hora se observará que el coste total del proyecto es de 14.000€.

6.- Conclusiones

Al inicio del estudio se definían tres objetivos que éste había de cumplir y que se han alcanzado en mayor o menor grado. Debido a la naturaleza de éstos, los dos primeros han tenido un enfoque más teórico frente al tercero que ha sido completamente práctico.

El primero de los objetivos pretendía concienciar sobre la importancia de la seguridad de la información y en concreto sobre cómo una herramienta como es la auditoría de seguridad beneficiará tanto a los desarrolladores de una aplicación como a sus usuarios.

La segunda meta era explicar la naturaleza de las vulnerabilidades, qué son, cómo surgen y que implicaciones tienen y sobretodo dar a conocer una serie de recursos que puedan ayudar tanto a desarrolladores como a auditores a identificarlas y mitigarlas.

Se considera que se han alcanzado ambos objetivos y que el lector de este documento estará ahora concienciado sobre la importancia de la seguridad en los aplicativos web. Además será capaz de utilizar las referencias presentadas para indagar más sobre el tema.

Por último se ha provisto de un sencillo pero portable proxy de peticiones HTTP que facilitará tanto a un auditor como a un desarrollador a localizar vulnerabilidades en aplicaciones web.

Se debe observar que las soluciones que hay en el mercado ahora mismo son bastante más maduras que el prototipo propuesto, sin embargo, esto no implica que trabajando más en esta herramienta y siempre teniendo en cuenta las diferentes debilidades identificadas, se pueda conseguir un producto competente y de calidad.

Por lo tanto se puede concluir que sería interesante seguir trabajando en el desarrollo del proyecto, siempre teniendo en cuenta claros los requisitos de modularidad y portabilidad, puesto que estos eran las elementos diferenciadores de esta herramienta.

6.1.- Problemas y propuestas de mejora

En esta sección se pueden observar alguno de los problemas experimentados durante la implementación del proyecto así como numerosas propuestas de mejora que harían la aplicación mucho más versátil.

Se debe recordar que la aplicación presentada es un prototipo y que se comenzó el diseño sin conocimiento alguno de los módulos y tecnologías e implicadas con lo cual se esperaba encontrar problemas de diversa índole que pudieran afectar al funcionamiento final de la aplicación.

6.1.1.- Problemas

A lo largo de la implementación se han experimentado problemas que no habían sido advertidos o habían sido menospreciados en las fases de diseño y dónde algunos han llegado incluso a limitar el aplicativo. La mayoría de ellos han podido ser resueltos, sin embargo hay un grupo que se ha presentado especialmente difícil y que ha requerido más atención de la que inicialmente se ha había previsto en la planificación del proyecto.

Este grupo surge debido a la poca homegenización que hay en los módulos de python en el momento de tratar con datos. Se puede observar por ejemplo que mientras que unos módulos trabajan con cadenas y datos codificados en ascii, otros lo harán con datos en unicode en diversos formatos UTF-8, UTF-16, etc, hecho que requiere la constante conversión de datos y que presenta problemas de compatibilidades si uno de los módulos o funcionalidades no soporta uno de estos tipos.

Si además se tiene en cuenta la complejidad que presenta el protocolo HTTP debido a los diferentes métodos de transferencia y codificaciones soportadas, se observa que el desarrollo se convierte en una tediosa tarea dónde se tiene que poner especial atención y dónde un pequeño error en el manejo y procesado de estos datos puede resultar en la aplicación inutilizada.

6.1.2.- Propuestas de mejora

Gracias a los problemas identificados durante la implementación del prototipo y debido a un mayor conocimiento de las tecnologías y protocolos implicados se han identificado una serie de areas en las que se podría y se debería trabajar con tal mejorar la aplicación y hacer que además esta se adapte aún más a los objetivos y requisitos definidos inicialmente:

- En un futuro desarrollo se optaría por el uso de sockets frente a las librerías utilizadas. Esto persigue dos objetivos, el primero de ellos tener más control sobre el protocolo y evitar o mitigar los problemas experimentados con las diferentes codificaciones y el segundo hacer el aplicativo aún más portable.

- El uso de sockets también permitiría convertir el proxy en un proxy de mensajes como HTTP como HTTPS añadiendo soporte de forma sencilla a SSL y TLS.

- También sería necesario escribir una librería para el manejo de cookies y de otros mecanismos de autenticación.

- Ahora mismo la aplicación no ofrece ningún tipo de inteligencia en cuanto a las peticiones ya realizadas, esto significa por ejemplo que estará continuamente descargando contenidos estáticos como imagenes y hojas de estilo que ya había descargado anteriormente. Se propone que esta en un futuro ofrezca algún mecanismo que guarde este tipo de contenidos reduciendo así considerablemente el tráfico y por lo consiguiente aumentando el rendimiento.

- Sería recomendable añadir una funcionalidad que permitiese escribir requests directamente en modo texto y que la aplicación se encargase de reconstruirlas y enviarlas al servidor remoto así como de recibir las respuestas.

- Añadir soporte para la representación de HTML e imagenes en otra pestaña reduciendo así la interacción con el navegador

7.- Bibliografía

[1] NIST CVSS Calculator

http://nvd.nist.gov/cvss.cfm?calculator&version=2

[2] MITRE CVE

http://cve.mitre.org/

[3] NIST NVD

http://nvd.nist.gov/

[4] OSVDB

http://www.osvdb.org/

[5] OWASP

https://www.owasp.org/

[6] OWASP Top 10

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[7] OWASP Testing guide

https://www.owasp.org/index.php/OWASP_Testing_Project

A.- Anexos

A.1.- Recurso no bibliográficos

OWASP ZAP - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Portswigger Burp - http://portswigger.net/

Python - http://python.org/

Qt - http://qt-project.org/

SQLite - http://sqlite.org/

A.2.- WSF manual

Web Security Framework or WSF is a versatile HTTP proxy with graphical interface developed by Adrián Hermoso that intends to aid both developers and security researchers in identifying vulnerabilities within web applications.

1.- Requirements and installation

Prior to using the application and in order to be able to successfully run the application the following requirements must be met:

- WSF was developed on a GNU/Linux operating system and while it might run in other operating systems it won't be supported for those.
- Python 2.x branch must be installed on the system and specifically the version of it should be 2.7.3 or greater.
- The following modules that can be installed with the command-line tool 'pip' are required:
    - PyQt4
    - requests
    - simplejson

2.- Configuration

WSF requires little to no configuration, however, it is possible to perform minimal tweaks on it that will allow the user to start the proxy on a different interface and port. It is also possible to define another location for the temporal working session files.

These can be modified by opening the 'wsfapp.py' with your favourite text editor and editing the following variables: WSF_IP, WSF_PORT and WSF_DBFILE which default to:

WSF_IP="localhost"
WSF_PORT=8080
WSF_DBFILE="/var/tmp/tmp.wsf"

Once ‘wsfapp.py’ is configured it can be run with the following statement:

```
$ python wsfapp.py
```

This will launch the application main window, which looks as it is presented in the figure 2.1, where we will be able to see in the status bar at the bottom left-hand corner the current connection details as seen in detail in the figure 2.2.

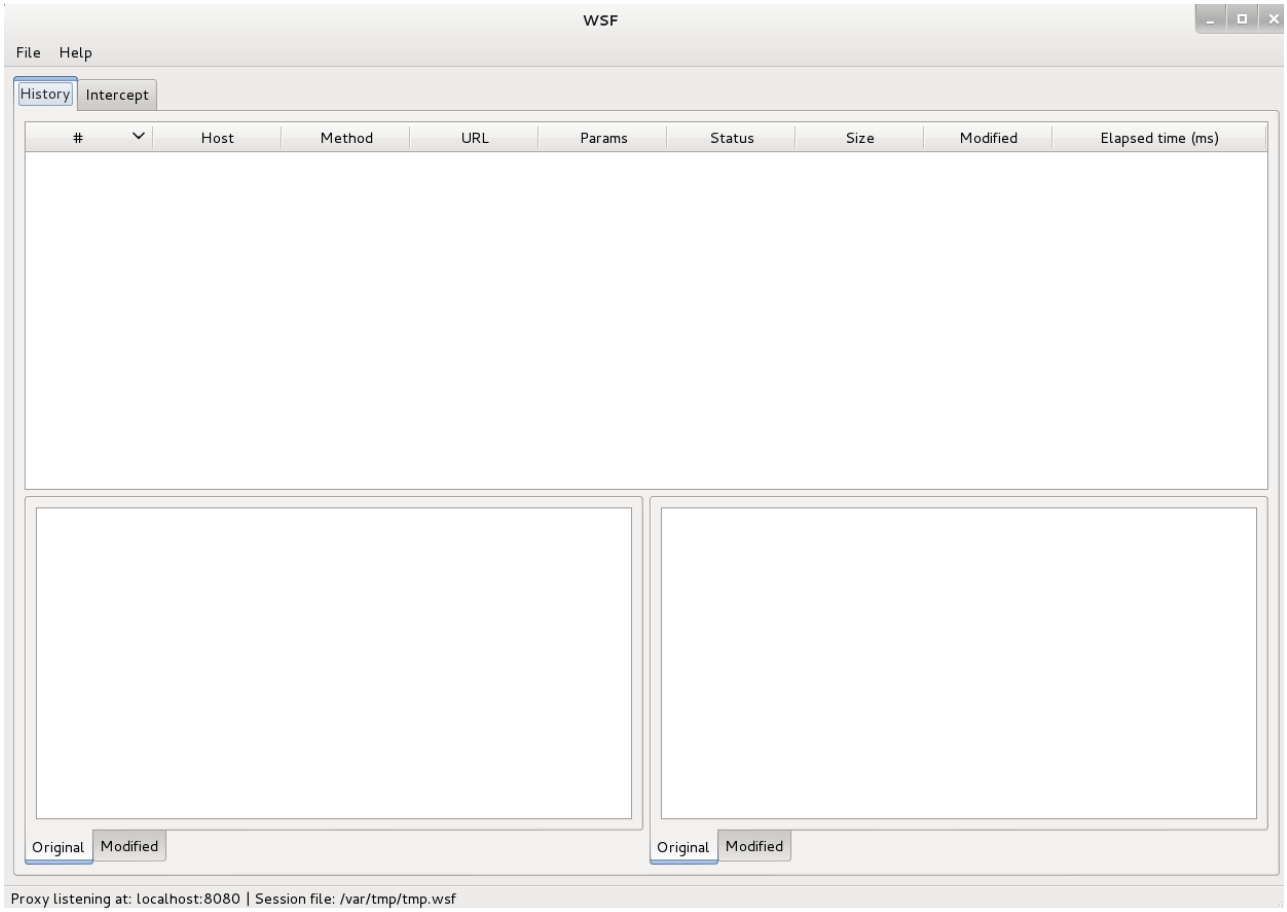

***Figure 2.1**: Main window of the application*

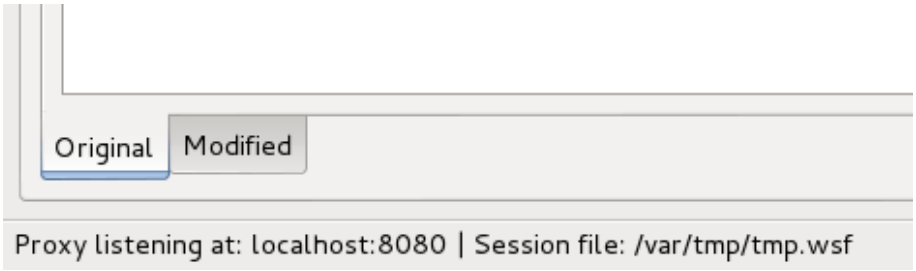

***Figure 2.2**: Detail of the status bar showing the proxy details*

The browser has to be configured with these connection settings in order to route all traffic to WSF. For example in Firefox it can be done through the 'Options – Preferences' menu in the 'Advanced' submenu and then in the 'Network' tab by clicking in the connection settings.

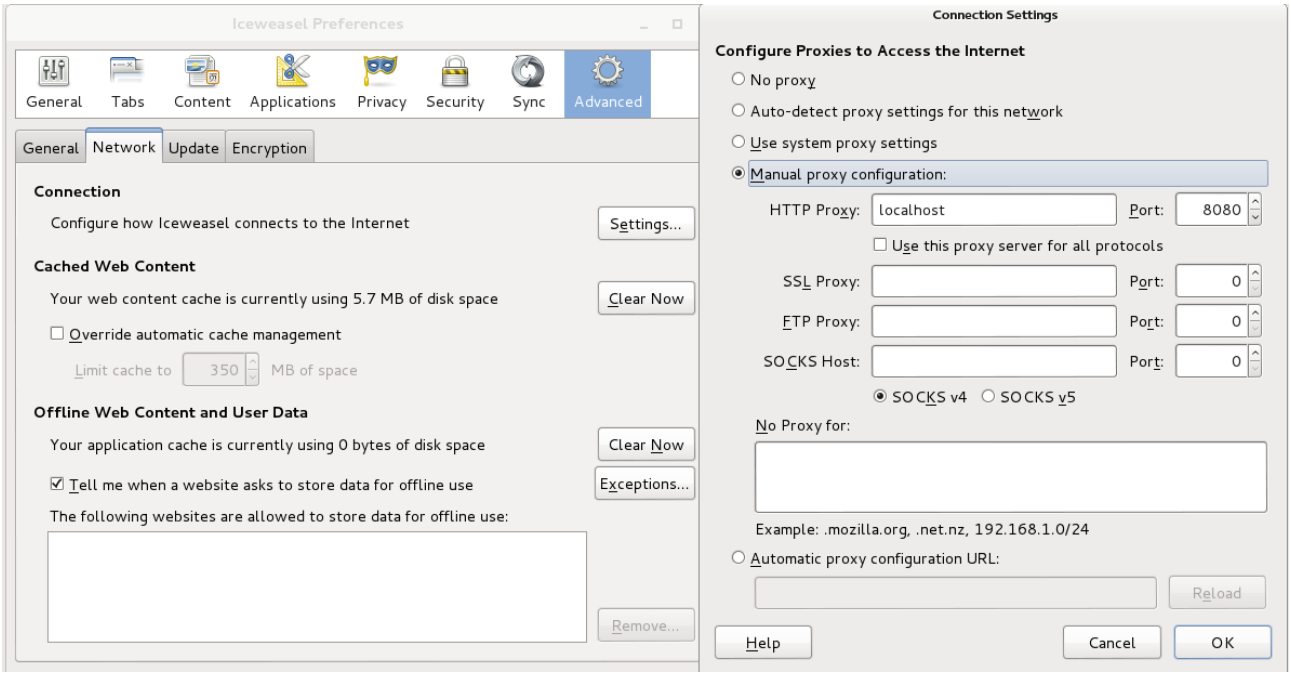

***Figure 2.3**: Configuring WSF in Firefox*

3.- Usage

Once launched WSF provides different functionalities that we will analyse in the following sections.

3.1.- Passive sniffing

By default and once the browser is configured WSF will act as a passive sniffer of the on-going HTTP traffic between the remove server and the user's browser. All the HTTP traffic, while representable by text strings, will be shown in the user interface as seen in the figure 3.1.

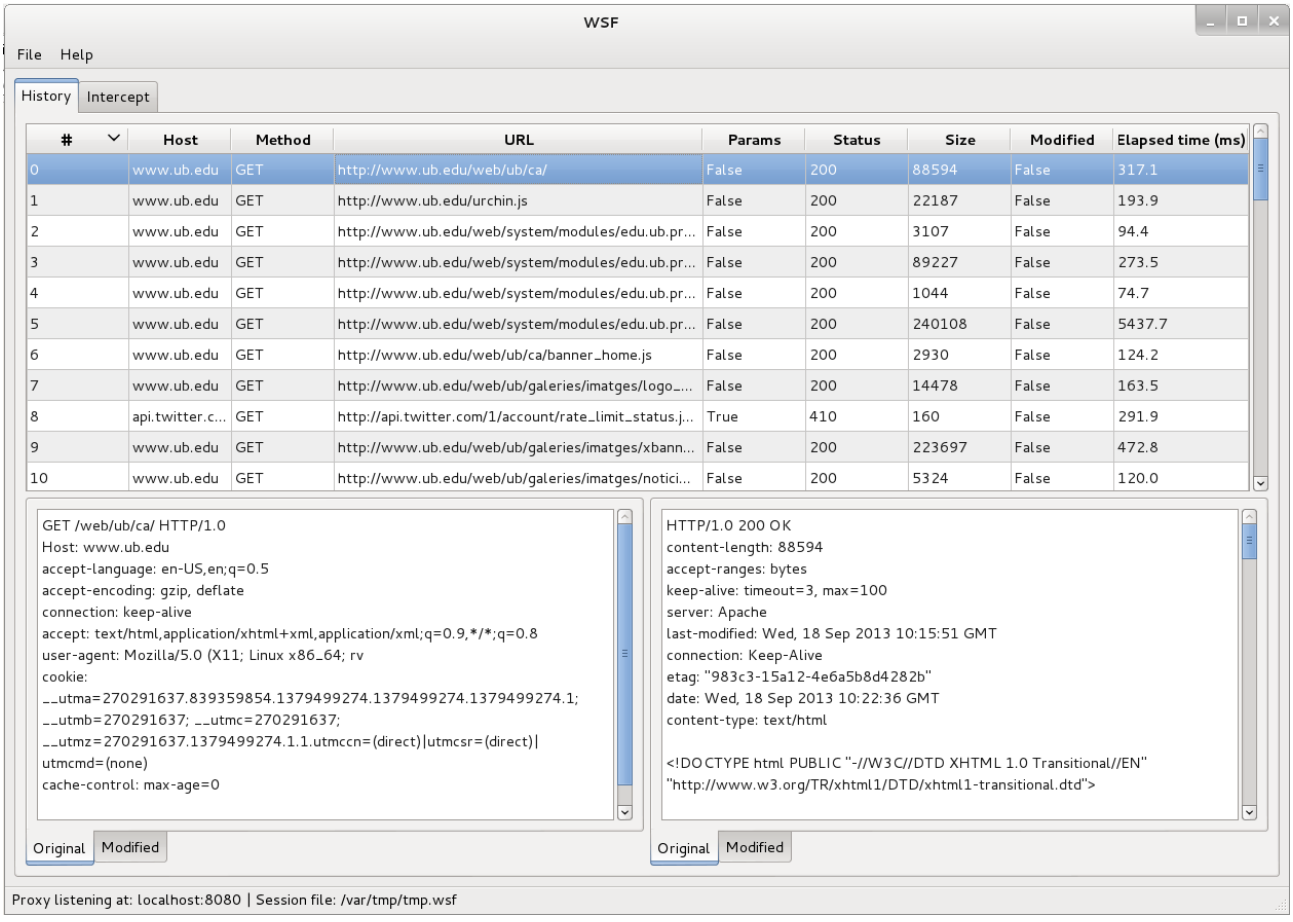

***Figure 3.1**: Configuring WSF in Firefox*

Just by browsing websites the interface will be updated with a list of the messages sent and received, as shown in the middle centre of the figure 3.1, and the user will be able to inspect in more detail both the request and the response in the bottom left and bottom right, of the same figure, respectively.

3.2.- Tampering

The interactive tampering of text requests and responses can be achieved in the Intercept tab presented in the main window. To activate it the ‘Start’ button must be pressed.

As soon as any web is browsed it will present to the user the request sent as it can be seen in the figure 3.2. Once presented it will wait for the user input on whether to modify it or forward it without tampering..

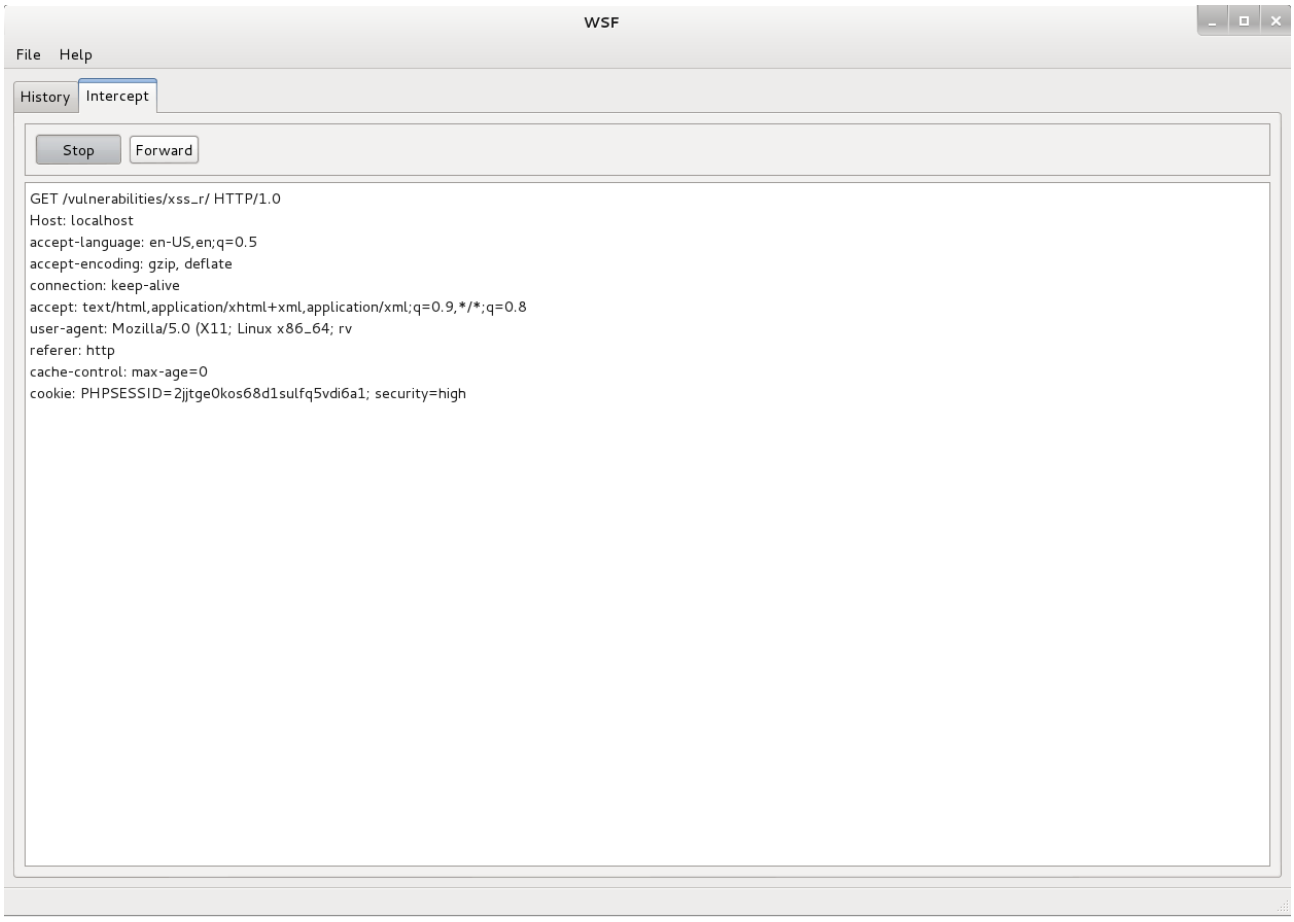

***Figure 3.2**: Intercepted request waiting for user action*

In the figure 3.3 we can appreciate how the response is also presented to the user so it can be modified.

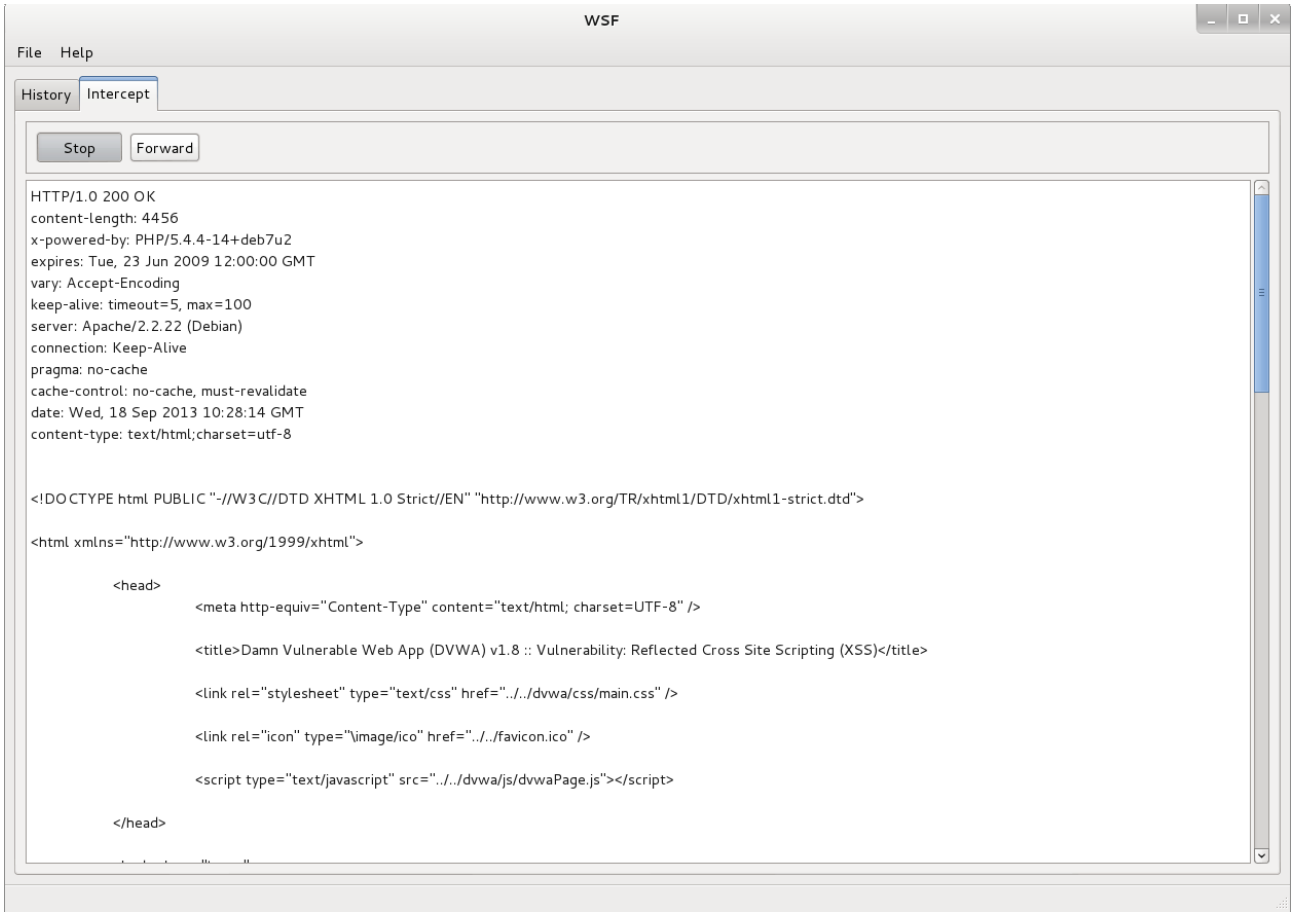

***Figure 3.3**: Intercepted response waiting for user action*

Imagine then that the previous user tampered with the response seen in the figure 3.3 and had typed into it something similar to the contents of the figure 3.4. If at this stage the user presses the 'Forward' button, the response will be sent to the user's browser and the contents of if will be rendered as the ones shown in the figure 3.5.

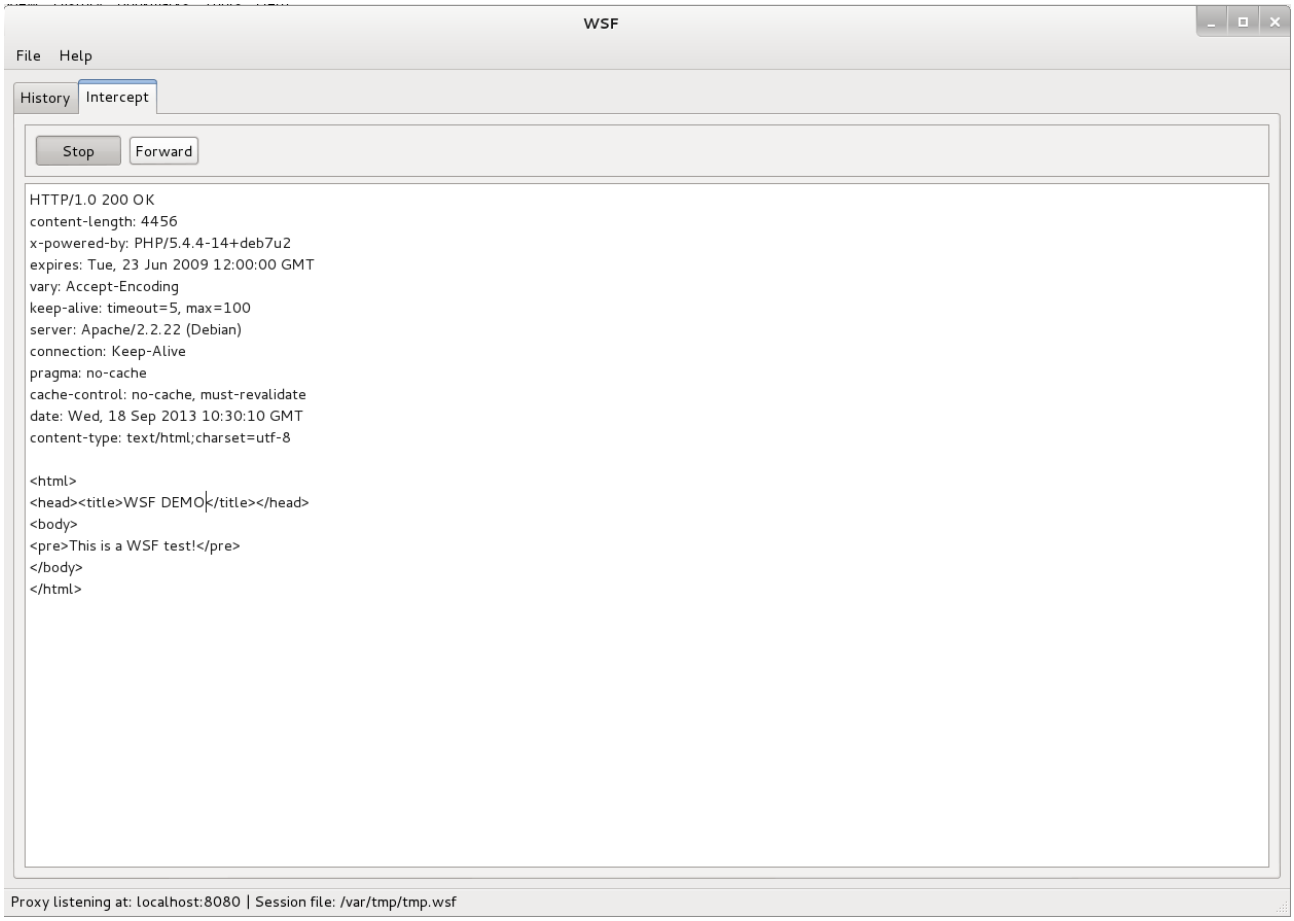

***Figure 3.4**: User tampered with the response*

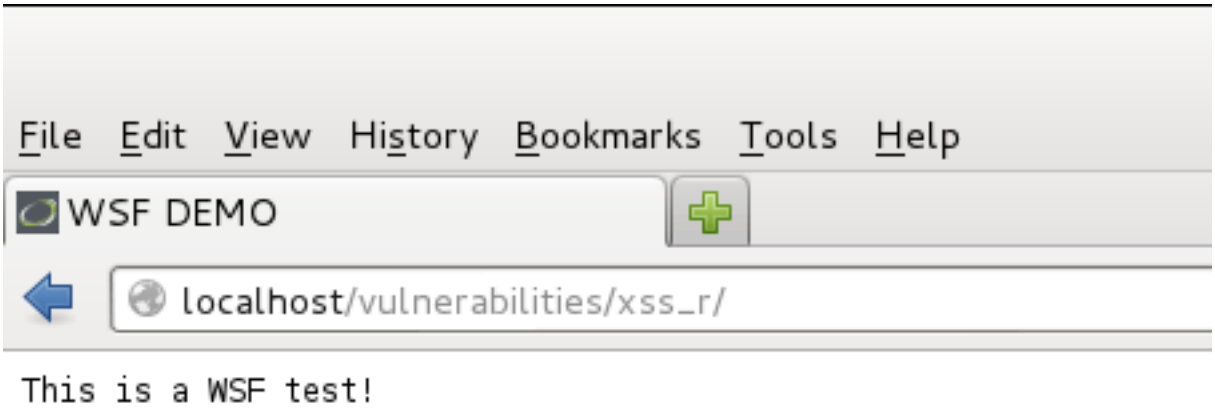

***Figure 3.5**: The tampered response gets rendered in the user;'s browser*

If the user takes a look again to the main 'History' tab he or she will realise that the 'Modified' tag is now set to 'True' and that both the contents original and the tampered request and response are accessible from the tabs within the bottom as is it shown in the figure 3.6.

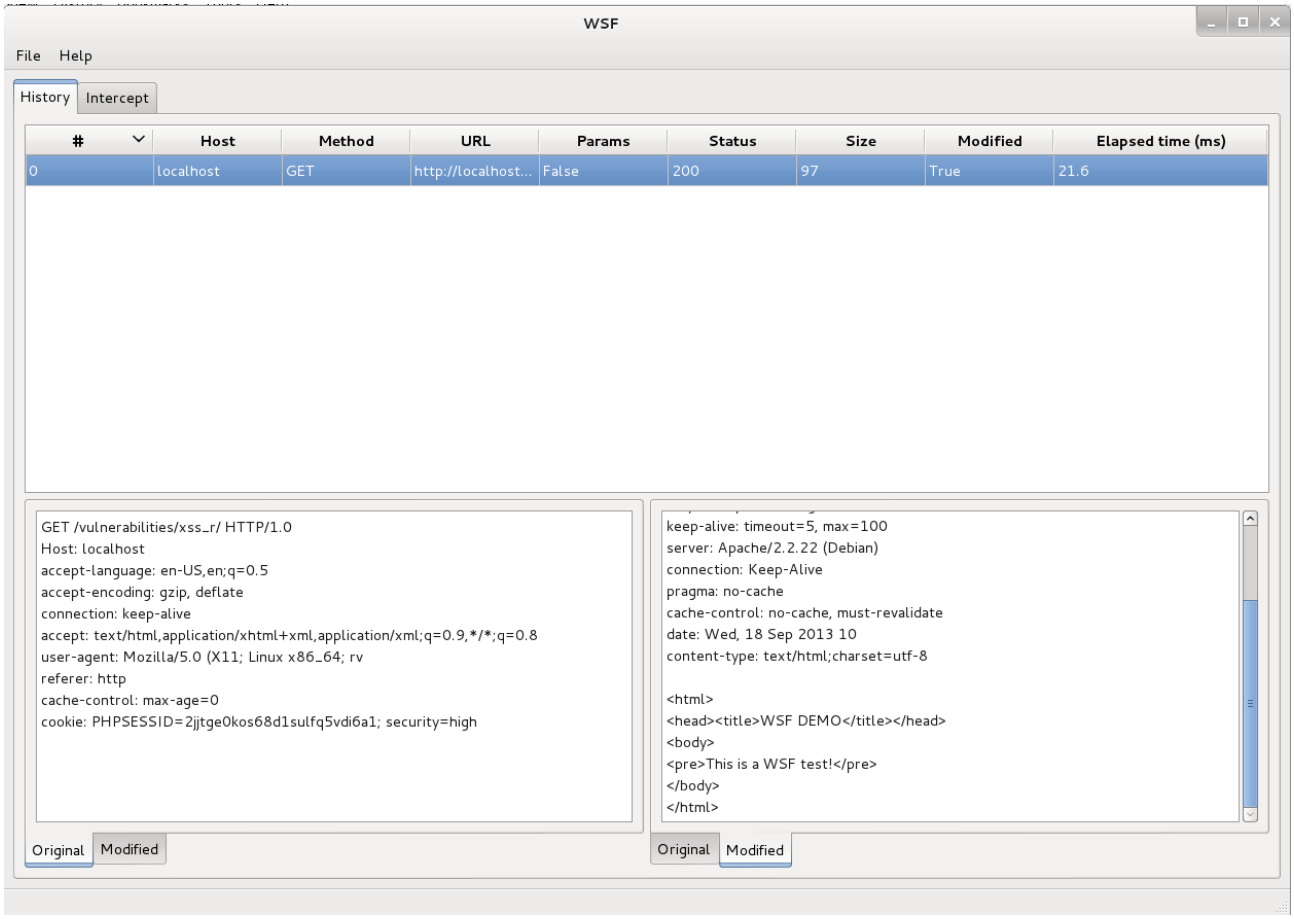

***Figure 3.6**: both the contents original and the tampered request and response are accessible from the tabs within the bottom*

3.3.- Saving and loading sessions

It is also possible for the user to save and load working sessions from the 'File' menu following an intuitive process. The application by default will store a temporary session file in the file path specified in the WSF_DBFILE variable if the program is closed and this file is not saved all the progress will be lost.

The save option prompts the user for a file name and a path in which the session file is to be saved as it can be seen in the figure 3.7.

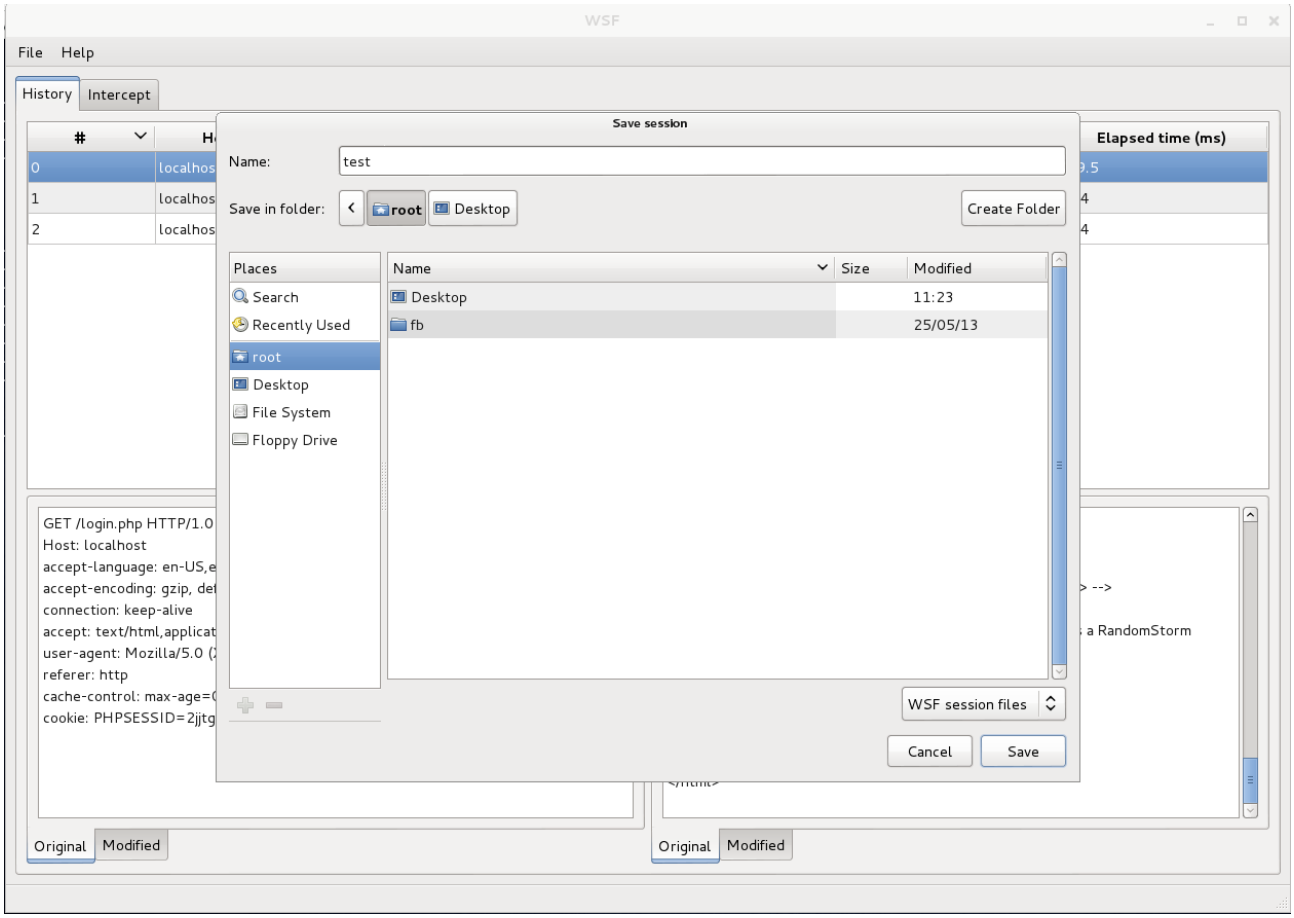

***Figure 3.7**: The save option prompts the user for a file name and a path in which the session file is to be saved*

Once saved the session can be loaded, just by browsing to the path where it was saved and selecting it(figure 3.8) the application will reload all the data from it and the work can be resumed as it is shown in the figure 3.9

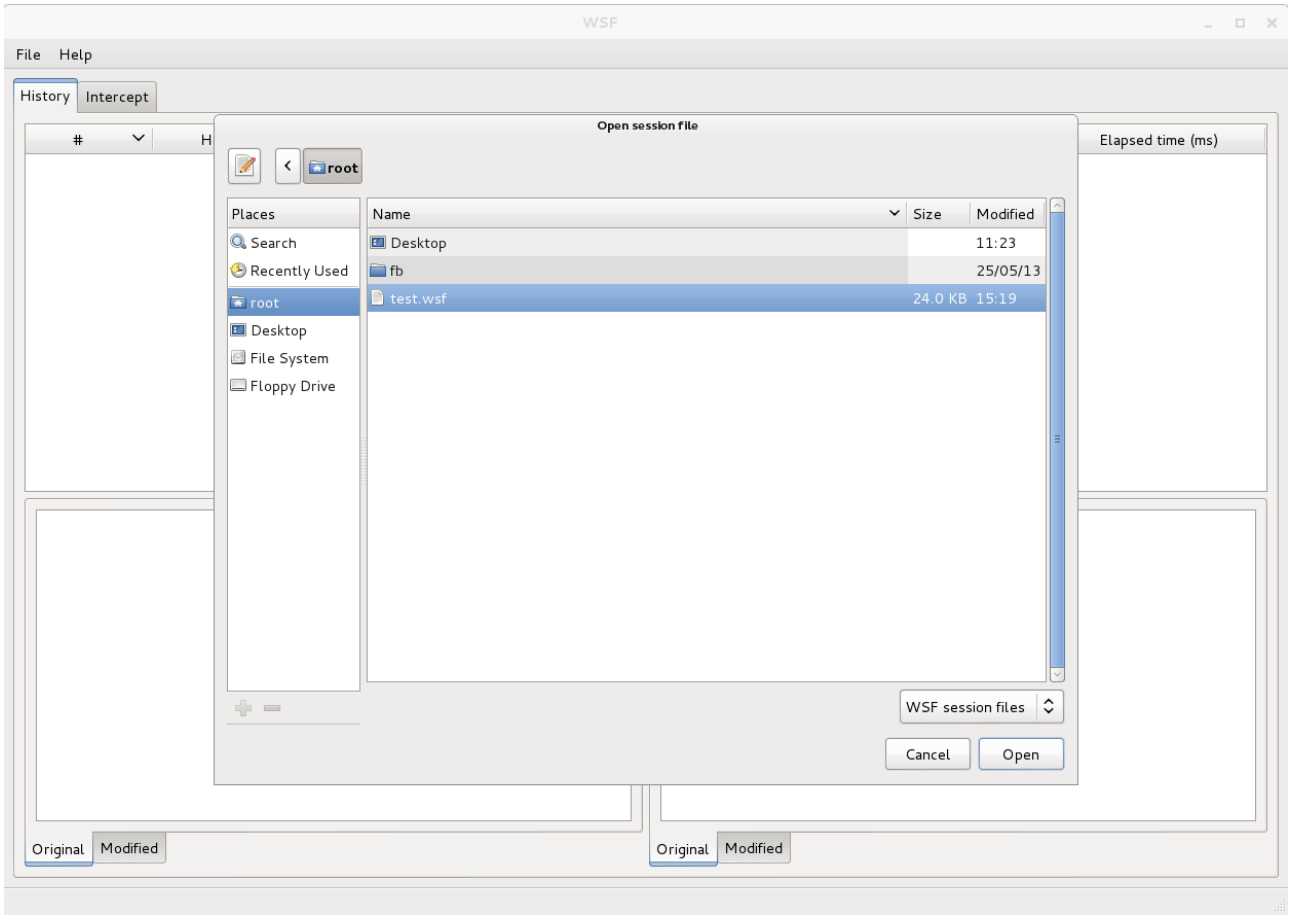

***Figure 3.8**: Browsing to the path where a session was saved and selecting it*

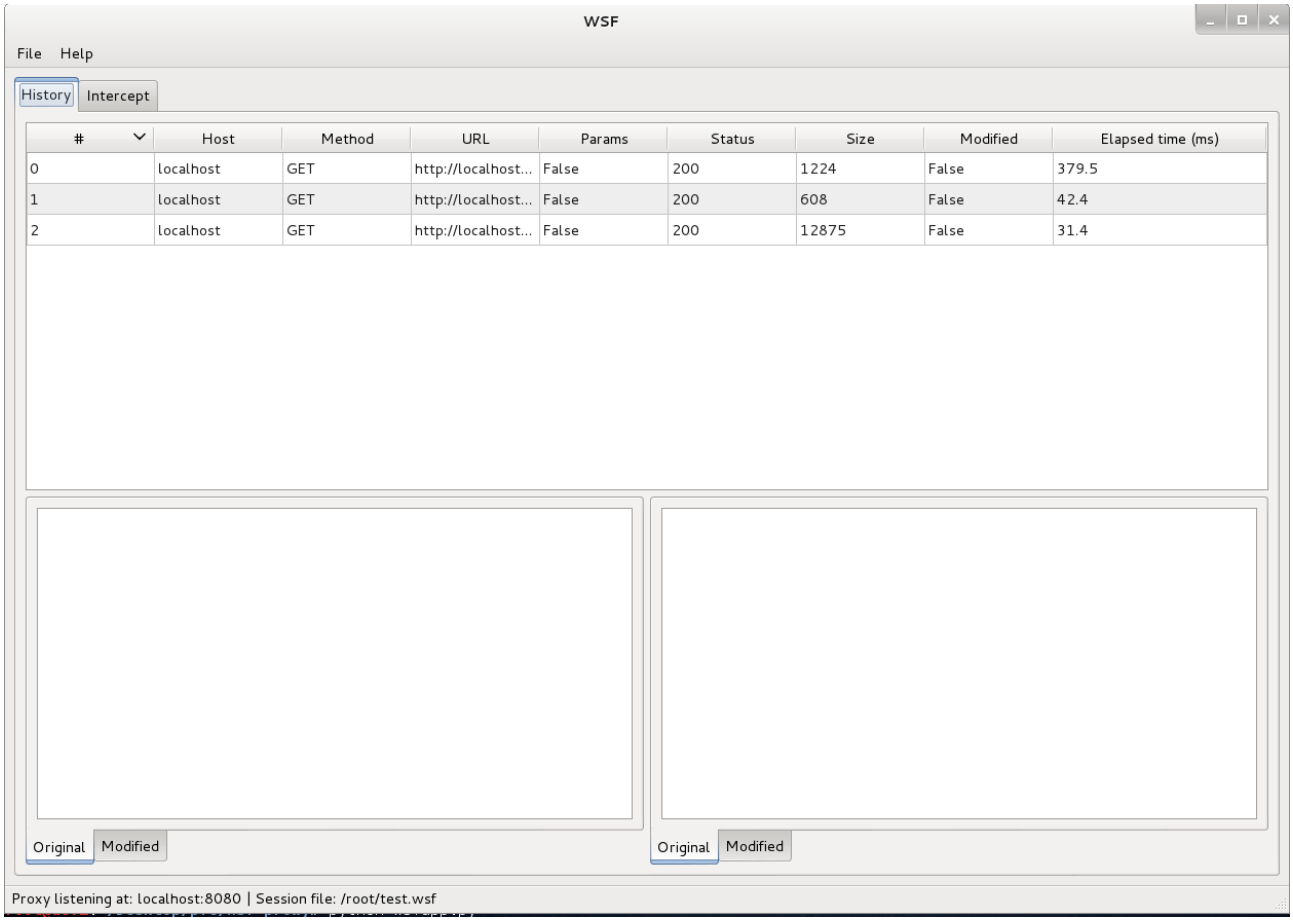

***Figure 3.9**: The application reloads all the data from that session and the work can be resumed*