UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Adaptación de la API FHScanHTTPCore para su integración con múltiples lenguajes de programación.

Autor:
Óscar Mallo Raposo
Directores:
Purificación Cariñena Amigo
Andrés Tarascó Acuña

Grado en Ingeniería Informática
Septiembre 2013

Trabajo de Fin de Grado presentado en la Escola Técnica Superior de Enxeñaría de la Universidad de Santiago de Compostela para la obtención del Grado en Ingeniería Informática.

D. Andrés Tarascó Acuña, Director en Tarlogic Security, S.L. y **Dña. Purificación Cariñena Amigo**, profesora del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela.

INFORMAN:

Que la presente memoria titulada *"Adaptación de la API FHScanHTTPCore para su integración con múltiples lenguajes de programación"*, presentada por **D. Óscar Mallo Raposo** para superar los créditos correspondientes al trabajo de fin de grado de la titulación Grado en Ingeniería Informática, se realizó bajo nuestra dirección en la empresa Tarlogic Security S.L.

Y para que conste que así a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a 9 de Septiembre de 2013.

Los directores,

Andrés Tarascó Acuña Purificación Cariñena Amigo

El alumno,

Óscar Mallo Raposo

Tabla de contenido

1. Introducción 9
   1.1 Contextualización del proyecto. 9
   1.2 Objetivos del proyecto. 10
   1.3 API HTTPCore 10
   1.4 Herramientas para la realización de auditorías de seguridad en entornos web. 11
   1.5 Glosario y aclaración de términos empleados. 13
   1.6 Organización del documento. 17
   1.7 Aclaraciones del proyecto 18
2. Análisis de requisitos 19
   2.1 Casos de uso 19
      2.1.1 Descripción de actores 19
      2.1.2 Descripción de casos de uso 20
   3.2 Análisis de requisitos del software. 44
      3.2.1 Restricciones de diseño 45
      3.2.1 Requisitos funcionales. 47
      3.2.2 Requisitos no funcionales 58
      3.2.3 Requisitos de interfaz. 58
      3.2.4 Requisitos de calidad. 59
      3.2.5 Requisitos de evolución. 59
      3.2.6 Requisitos del proyecto. 60
      3.2.7 Requisitos de seguridad. 60
   3.3 Matriz de trazabilidad. 62
3. Gestión del proyecto. 63
   3.1 Metodología de desarrollo. 63
      3.1.1 Análisis del alcance de cada incremento. 64
   3.2 Gestión de riesgos 66
      3.2.1 Riesgos de gestión del proyecto 67
      3.2.2 Riesgos de desarrollo 68
   3.3 Estructura de descomposición de trabajo. 70
   3.4 Planificación temporal. 72
   3.5 Seguimiento 74
   3.6 Gestión de la configuración 76
   3.7 Gestión de costes 76

4. Arquitectura, tecnologías y herramientas. 77
4.1 Problemática inicial. 77
4.2 Arquitectura del sistema. 80
4.2.1 Esquema funcional de la aplicación. 80
5.3.2 Esquema de interacción entre componentes 83
5.3.3 Tecnologías empleadas. 85
5. Diseño 89
5.1 Modelo estructural del sistema. 89
5.2 Modelo Vista Presentador 89
5.3 Iteración 1. 92
5.4 Iteración 2 95
5.5 Iteración 3 97
5.6 Iteración 4 99
5.7 Iteración 5 102
5.8 Patrones de diseño 109
5.9 Diagramas de secuencia 111
6. Implementación. 115
6.1 Consideraciones Iniciales 115
6.2 Detalles de implementación del Wrapper 116
6.3 Detalles de implementación de las clases Proxy en C# 117
6.4 Modelo de acceso a la API HTTPCore 119
6.5 Decoración de nombres 119
6.6 Decoración de nombres y bibliotecas de vínculos dinámicos 120
6.7 Fugas de memoria (Memory Leaks) 121
6.8 Convenciones de llamada (Calling Conventions) 122
6.9 Representación y formato de tipos de datos 123
6.10 Prácticas de desarrollo seguro 125
7. Pruebas y validación. 127
7.1 Pruebas funcionales 127
7.2 Prueba de seguridad 147
7.3 Evaluación heurística. 149
8. Conclusiones y trabajo futuro. 151
8.1 Conclusiones 151
8.2 Trabajo futuro 152
ANEXO 153

A. Manual técnico. 153
A.1 Especificación de funciones de la API exportadas en la biblioteca de vínculo dinámico a través de los wrappers. 153
A.2 Creación de scripts 190
B. Manual de usuario. 195
B.1 Secciones de la interfaz 195
Bibliografía 217

1. Introducción

1.1 Contextualización del proyecto.

La problemática del software inseguro se ha convertido en uno de los principales retos técnicos de la actualidad. El despliegue de aplicaciones expuestas a Internet sin la implementación adecuada de controles de seguridad, siguiendo una metodología de desarrollo seguro supone un riesgo muy elevado para las empresas, especialmente para aquellas cuyo modelo de negocio depende de la red de redes.

Con la revolución de las tecnologías de la información, las plataformas web se han convertido en una de las herramientas fundamentales de las compañías para la expansión de sus actividades económicas, desarrollando aplicaciones web o aplicativos móviles que se comunican a través del protocolo de comunicaciones HTTP. El despliegue de este tipo de aplicaciones supone por lo tanto, que una gran cantidad de activos clave para la organización se puedan ver afectados, ante una vulneración de la seguridad. Consecuentemente el impacto asociado a la exposición de los activos afecta a la actividad económica de una organización.
Existen distintos enfoques que permiten la evaluación de la seguridad de los sistemas de información, sin embargo las auditorías técnicas de seguridad se han convertido en una de las opciones preferidas por parte de las empresas para reducir el riesgo de las mismas ante ataques informáticos.

Para la realización de una auditoría de seguridad sobre entornos web es necesario disponer de un conjunto de herramientas que nos proporcionen la capacidad de llevar a cabo las distintas pruebas de seguridad, siguiendo una metodología concreta como puede ser la “Guía de pruebas de OWASP” (1). A pesar de que existe una gran cantidad de herramientas enfocadas a la realización de auditorías y ejecución de pruebas de seguridad sobre el software, en ocasiones se presentan situaciones específicas en las que es necesario automatizar ciertos procedimientos de pruebas mediante la implementación de scripts que hacen uso de librerías que permiten la gestión de las comunicaciones HTTP/HTTPs. Sin embargo existen ciertos escenarios en los que el rendimiento es crítico para la realización de las pruebas de seguridad, por lo tanto es necesario disponer una API implementada en un lenguaje de programación de alto nivel que disponga de un alto rendimiento de ejecución especialmente en la realización de procedimientos iterativos.

1.2 Objetivos del proyecto.

A continuación se exponen los objetivos del proyecto:

- Investigar las distintas alternativas para permitir que la API HTTPCore (2) pueda ser utilizada desde cualquier lenguaje de programación con el fin de servir de apoyo en la creación de pruebas de seguridad específicas en entornos web.

  Para la consecución de este objetivo será necesario:

  - Identificar aquellas funcionalidades de la API que deben ser expuestas, abstrayendo a los usuarios de la API de los detalles de implementación de la misma.
  - Ofrecer soporte para el desarrollo de scripts implementados con cualquier lenguaje de programación bajo entornos Windows.

- Desarrollo de una aplicación grafica que permita la ejecución de auditorías de seguridad. La herramienta deberá disponer de las siguientes funcionalidades:

  - Permitir de interceptación de peticiones HTTP/s enviadas desde cualquier aplicación externa.
  - Permitir la redirección o el bloqueo de las peticiones interceptadas.
  - Permitir el envío de peticiones HTTP construidas manualmente.
  - Permitir de ejecución de procedimientos iterativos de prueba.
  - Disponer de un IDE de desarrollo que permita la creación, compilación y ejecución de scripts que interactúen con las funcionalidades de la API HTTPCore.

1.3 API HTTPCore

La API HTTPCore es una API (Application Programming Interface) multiplataforma implementada en C++ que ofrece soporte de múltiples funcionalidades, a través de las cuales es posible gestionar comunicaciones HTTP y HTTPs. Las características más destacables son:

- Capacidad de establecer conexiones HTTP y HTTPs.
- Envío de peticiones HTTP/s y acceso a las respuestas asociadas.
- Soporte de autenticación a través de los métodos Basic, Digest y NTLM.
- Soporte de redirecciones.
- Soporte para la creación de proxies HTTP y HTTP/s.
- Compresión y descompresión automática de datos utilizando los algoritmos gzip y deflate.
- Soporte multihilo.
- Gestión automática de cookies.

- Sistema de callbacks que permiten modificar el comportamiento de la API mediante la manipulación flexible de los datos de, tanto en las peticiones HTTP como de las respuestas.

1.4 Herramientas para la realización de auditorías de seguridad en entornos web.

La evaluación de la seguridad de un aplicativo web requiere la utilización de un conjunto de herramientas para llevar a cabo la ejecución de las pruebas de seguridad oportunas. Después de los navegadores, la herramienta más útil a la hora de realizar una auditoría en entornos web son los proxies de interceptación. Si bien los primeros proxies solo disponían de funcionalidades de interceptación y modificación de peticiones, a día de hoy este tipo de herramientas han evolucionado hasta convertirse en entornos de desarrollo de pruebas, que integran múltiples componentes diseñados específicamente para facilitar la realización de tareas comunes durante la ejecución de los procedimientos de prueba.

El funcionamiento básico de un proxy de interceptación se basa principalmente en disponer de un servidor que recibe todas las peticiones HTTP enviadas por parte de una aplicación cliente (como puede ser un navegador web), de forma que es posible visualizar los datos enviados, y permitir la manipulación de los mismos, con el objetivo de comprobar el comportamiento de una aplicación ante distintos valores de entrada, permitiendo verificar la existencia de vulnerabilidades cuando no se realiza una validación correcta de los parámetros proporcionados.

Es necesario destacar también, la existencia de herramientas automáticas que permiten la identificación de vulnerabilidades, sin embargo los resultados de éstas siempre son limitados ya que existen ciertos tipos de debilidades que no son pueden ser comprobadas. Un ejemplo claro son los fallos de implementación de lógica de negocio. Ante esta situación siempre es necesario emplear una aproximación mixta durante la ejecución de las pruebas, utilizando tanto herramientas automáticas/semiautomáticas como manuales.

A continuación se comentan algunas de las herramientas semiautomáticas más conocidas para la realización de auditorías de seguridad sobre entornos web.

Burp Suite

Burp Suite es una plataforma integrada para la ejecución de pruebas de seguridad en entornos web. Proporciona gran cantidad de módulos que ofrecen soporte a todas las fases del proceso de pruebas facilitando la realización de las mismas.

Sus funcionalidades más destacables son:

- Proxy de interceptación, que permite inspeccionar el tráfico entre el navegador y un sistema destino.
- Escáner de vulnerabilidades que permite la detección automática de fallos de seguridad.
- Capacidad de inspección y descubrimiento de recursos de forma automática.
- Módulo de fuzzing que permite la ejecución de procedimientos de prueba iterativos.
- Soporte multiplataforma.

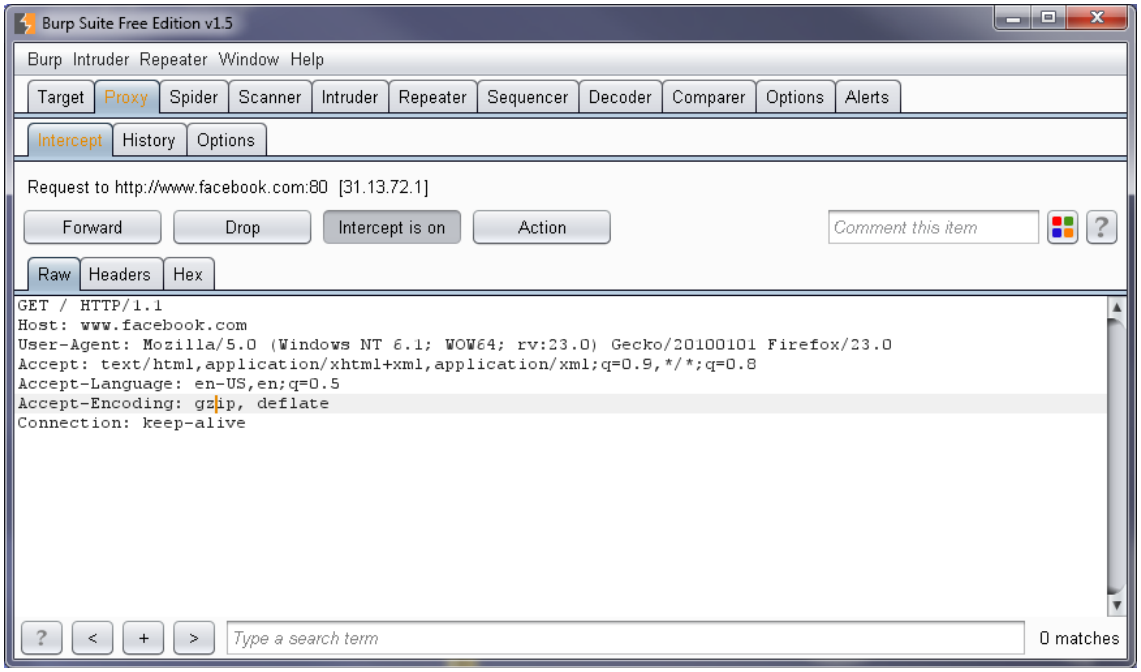


Figura 1: Módulo de interceptación de Burp Suite.

Burp Suite se distribuye en dos versiones: una comercial que incluye todas las funcionalidades y otra versión gratuita más limitada.

OWASP Zed Attack Proxy

OWASP ZAP es un proyecto de código abierto patrocinado por la organización Open Web Application Security Project, encargada de concienciar y promover la seguridad de entornos web. ZAP es una herramienta que, al igual que Burp Suite permite la identificación y ejecución de pruebas de seguridad en entornos web disponiendo en mayor o menor medida de las mismas funcionalidades, siguiendo una aproximación semiautomática para la realización de los procedimientos de prueba.

La diferencia fundamental con respecto a Burp Suite es que ZAP se distribuye de forma completamente gratuita.

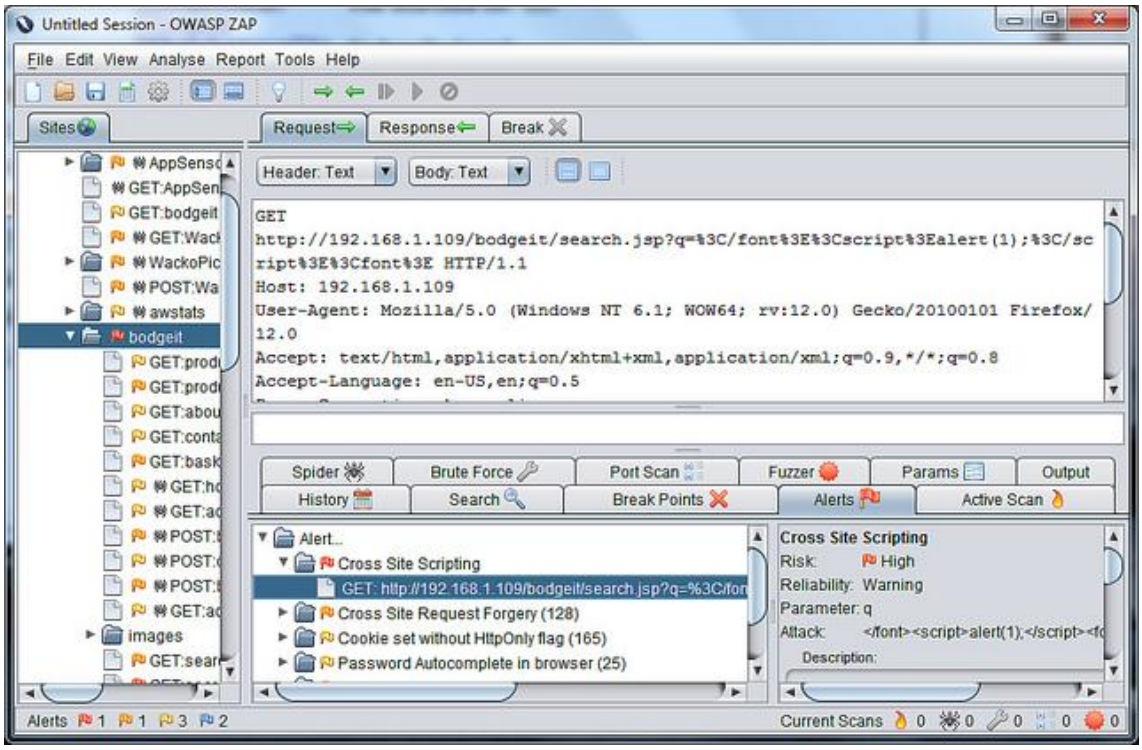


Figura 2: OWASP Zed Attack Proxy.

A pesar de que la herramienta que se pretende desarrollar comparte funcionalidades con ZAP y con Burp Suite el elemento diferenciador, es la capacidad de crear y ejecutar scripts desde la propia herramienta, utilizando las funcionalidades exportadas de la API HTTPCore.

1.5 Glosario y aclaración de términos empleados.

Con el objetivo de aportar mayor claridad, a continuación se definen un conjunto de términos empleados a lo largo de todo el documento.

Motor de comunicaciones HTTP/s: El motor de comunicaciones es el núcleo de la API HTTPCore. Se refiere principalmente a aquellas funcionalidades de la API que permiten establecer y gestionar un canal de comunicación empleando el protocolo HTTP.

Servidor Proxy HTTP/s / Proxy de interceptación: Servidor que permite la interceptación de peticiones HTTP y HTTPs enviadas por parte de un cliente a un servidor destino, haciendo de intermediario en el envío y recepción de las mismas. La utilización de este término lo largo del documento hace referencia a las funcionalidades del servidor proxy proporcionado por la API HTTPCore.

Callbacks: La definición de callback que figura en la Wikipedia (3) es la siguiente: "Una devolución de llamada o retrollamada (en inglés: callback) es una función "A" que se

usa como argumento de otra función "B". De modo que, cuando se invoca la función "B", ésta ejecuta "A".

El concepto de callback se representa a través del siguiente diagrama.

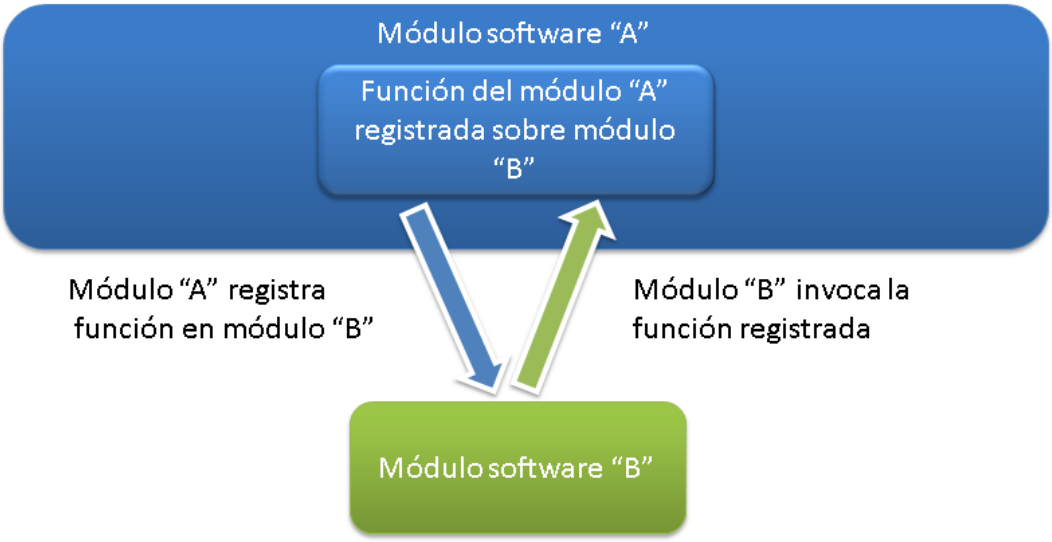


Figura 3: Diagrama ilustrativo del concepto de callback.

La API permite definir callbacks a través de los cuales es posible llevar a cabo la manipulación flexible de los datos de peticiones y respuestas HTTP con el objetivo de modificar el comportamiento del motor de comunicaciones HTTP o del servidor proxy. La API define cuatro puntos en los es posible definir un callback:

- Antes del envío de una petición por parte del motor de comunicaciones HTTP.
- Antes del envío de una petición por parte del servidor proxy HTTP.
- Tras la recepción de una petición por parte del motor de comunicaciones HTTP.
- Tras la recepción de una petición por parte del proxy HTTP.

Fuzzing (4)**:** Procedimiento de prueba a través del cual es posible descubrir fallos en el software al proporcionar valores de entrada erróneos o imprevistos. A lo largo del documento se utiliza este término para hacer referenciar a la funcionalidad que permite ejecutar procedimientos iterativos de prueba, ejecutados mediante el envío de múltiples peticiones HTTP en las que se reemplaza una parte de la misma por un conjunto de valores definidos.

Payload: En el contexto del proyecto el término "Payload" hace referencia al valor/valores utilizados para reemplazar una parte definida de una petición HTTP, durante la ejecución de pruebas de fuzzing.

Repeater: El término "Repeater" hace referencia a la funcionalidad de la aplicación que permite la construcción de peticiones HTTP para su envío en la aplicación que se pretende desarrollar.

Wrapper: En el contexto del proyecto un wrapper es una interfaz que permite proporcionar acceso a las funcionalidades de la API HTTPCore.

Código administrado (5)**:** El término código administrado (Managed code en inglés) es un término acuñado por Microsoft para identificar el código fuente de un programa que es ejecutado bajo el control del entorno de ejecución CLR (Common Language Runtime) en la plataforma Microsoft .NET.

Código no administrado (6)**:** El término código no administrado hace referencia al código de un programa que es ejecutado fuera del contexto del entorno de ejecución CLR.

Petición HTTP: Mensaje del protocolo HTTP enviado por parte de un cliente con el objetivo de solicitar una acción sobre un recurso concreto en un servidor.

Respuesta HTTP: Mensaje del protocolo HTTP que es enviado como respuesta por parte de un servidor, tras la recepción de una petición HTTP.

Clase proxy: En programación orientada a objetos una clase es una representación abstracta de una entidad, a través de la cual es posible especificar tanto atributos (propiedades) como métodos (comportamientos). El término "clase proxy" se utiliza a lo largo del proyecto para identificar a aquellas clases que implementan el patrón de diseño "Proxy Pattern" (7).

Acrónimos

API (Application programming Interface): Conjunto de rutinas, protocolos o herramientas para la construcción de aplicaciones software.

UML (Unified Modeling Language) (8): Lenguaje gráfico empleado para especificar, construir y documentar un sistema.

CLR (Common Language Runtime) (9): Entorno de ejecución de la plataforma .NET de Microsoft, encargado de la ejecución del código y de proporcionar servicios que facilitan el proceso de desarrollo.

CIL (Common Intermediate Language) (10): CIL es un lenguaje de programación de bajo nivel empleado para la construcción de código intermedio independiente de la plataforma o arquitectura del procesador. Durante la compilación de código desarrollado en algún lenguaje de programación compatible con la plataforma. NET el

código fuente es convertido a código CIL, para a continuación ser compilado a código nativo que puede ser ejecutado por el procesador.

HTTP (HyperText Transfer Protocol). Protocolo de aplicación empleado para la transmisión de documentos en la "World Wide Web".

HTTPs (HyperText Transfer Protocol Secure): Protocolo de aplicación basado en el protocolo HTTP que permite llevar a cabo la comunicación de los datos a través de un canal de comunicaciones cifrado.

1.6 Organización del documento.

El presente documento está compuesto por un total de 8 capítulos y dos anexos. A lo largo del documento se exponen las distintas fases y procedimientos que se han llevado a cabo para la consecución de los objetivos propuestos.

Capítulo 2 – Análisis: En este capítulo se realiza un análisis de los objetivos del proyecto para concluir con la especificación de requisitos, definida en base a los casos de uso identificados.

Capítulo 3 – Gestión del proyecto: Este capítulo aborda todos los aspectos relativos a la gestión del proyecto: elección de un ciclo de vida de desarrollo, análisis de riesgos, definición de una planificación temporal, gestión de la configuración y análisis de costes.

Capítulo 4 – Arquitectura, tecnologías y herramientas: A lo largo del capítulo 4 se define la arquitectura del sistema, y se comentan las tecnologías empleadas para el desarrollo del proyecto.

Capítulo 5 – Diseño: Aborda los aspectos de diseño de la aplicación, se analizan por separado cada uno de los subsistemas para finalmente mostrar el diseño global de la aplicación.

Capítulo 6 – Implementación: En este capítulo se analizan y detallan las decisiones que se han llevado a cabo a lo largo del proyecto desde el punto de vista de su implementación.

Capítulo 7 – Pruebas: En el capítulo siete se detallan todas las pruebas realizadas sobre el software con el objetivo de validar que los requisitos se hayan cumplido correctamente.

Capítulo 8 – Conclusiones y trabajo futuro: Se detallan las conclusiones del proyecto y se analizan distintas medidas de ampliación y mejora del proyecto.

Manual Técnico: Anexo en el que se detalla las distintas llamadas que permiten la invocación de las funcionalidades de la API.

Manual de usuario: Proporciona aquellas instrucciones necesarias para que un usuario pueda hacer uso de la herramienta.

1.7 Aclaraciones del proyecto

Este TFG ha sido desarrollado para la empresa **Tarlogic Security S.L** (11), de forma que se han tenido que respetar ciertas condiciones a la hora de llevar a cabo la redacción del documento, por lo que es posible que se eche en falta un mayor detalle de la implementación de las soluciones.

El proyecto sienta la base para ejecución de un proyecto mayor en el que se integrarán las funcionalidades implementadas. Como consecuencia de esta situación no se publicará como parte del trabajo de fin de grado, ni el código fuente que permite ofrecer la interoperabilidad de la API, ni la aplicación desarrollada.

2. Análisis de requisitos

A lo largo de esta sección se describen los procedimientos de análisis empleados para la identificación de las funcionalidades de la aplicación a través de los datos recogidos en las distintas reuniones mantenidas con el cliente.

2.1 Casos de uso

El modelado de casos de uso es una técnica de captura de información empleada durante de la fase de planificación de un proyecto, para asistir en la identificación de los requisitos funcionales de un sistema.

En el contexto de la ingeniería del software los casos de uso son un elemento fundamental en el proceso de desarrollo ya que permiten representar la interacción de los usuarios y un sistema de cómputo a través sus funcionalidades, sin la necesidad de especificar los detalles de implementación. Los casos de uso se definen textualmente como un flujo de eventos e incluyen generalmente un escenario principal y uno o varios escenarios alternativos.

El lenguaje de modelado unificado (UML) contempla los diagramas de casos como una representación visual que permite describir la relación que existe entre los agentes externos de un sistema (actores) y el propio sistema.

Los diagramas de casos de uso están compuestos por los siguientes elementos:

- **Actores**: Se emplea el término actor para identificar a los agentes externos que interactúan con el sistema, de esta forma un actor podría describir a tanto a un usuario como a otro sistema que interactúa con la plataforma.

- **Casos de uso**: Describen acciones concretas que permiten lograr un objetivo discreto para el usuario.

- **Relaciones**: Muestran la relación existente entre distintos casos de uso.

2.1.1 Descripción de actores

Para la interacción con el sistema se han identificado los siguientes actores:

- **Auditor de seguridad**: Actor por defecto que interactúa con las funcionalidades de la aplicación.

- **Aplicación**: Actor que representa cualquier aplicación que permita el envío de peticiones HTTP/s a través de un proxy.

2.1.2 Descripción de casos de uso

A continuación se detalla una lista de casos de uso que describen las funcionalidades de la aplicación que se va a desarrollar.

Para la obtención y especificación de los requisitos del proyecto se seguirá el formato detallado a continuación:

Código del caso de uso, siguiendo el formato: CU-NÚMERO

Nombre	Nombre identificativo del caso de uso.
Descripción	Descripción breve del propósito del caso de uso.
Actores	Nombres de los actores que participan en el caso de uso.
Importancia	Criticidad del caso uso. Los posibles valores son los siguientes: • **Alto**: Su implementación es estrictamente necesaria para el funcionamiento del sistema. • **Medio**: La ausencia de esta funcionalidad limita en gran medida el funcionamiento del sistema. • **Bajo**: Su implementación es prescindible para el funcionamiento del sistema.
Precondición	Condiciones necesarias para la realización del caso de uso.
Postcondición	Descripción del estado del sistema una vez que se ha llevado a cabo el caso de uso.
Escenario principal	Secuencia de acciones que tienen lugar cuando el caso de uso se lleva a cabo correctamente.
Escenario alternativo	Define un curso alternativo de eventos cuando la secuencia de acciones no se realizó correctamente.

(*) Aclaración: No se han documentado casos de uso correspondientes a la API de programación ya que no representan acciones visibles para el usuario, sin embargo sí que se lleva a cabo una especificación de requisitos y documentación formal de la misma.

Los casos de uso de la aplicación se pueden agrupar en cuatro subsistemas principales:

- Subsistema de proxy de interceptación.
- Subsistema de módulo de repetición de peticiones HTTP.
- Subsistema de fuzzing HTTP.
- Subsistema IDE de scripting.

Casos de uso del subsistema *"proxy de interceptación"*

CU-01

Nombre	Iniciar proxy de interceptación.
Descripción	La aplicación permite iniciar un proxy de interceptación de comunicaciones HTTP/HTTPs en una dirección IP y un puerto concreto.
Actores	Auditor de seguridad
Importancia	**ALTA**
Precondición	No aplica.
Postcondición	El sistema inicia el servidor proxy en una dirección IP y puerto especificado por el usuario.
Escenario principal	1. El sistema muestra un listado de direcciones IP asignadas a cada una de las interfaces de red del equipo. 2. El usuario selecciona la dirección IP en la que desea establecer el proxy. 3. El usuario especifica el puerto en el que desea establecer el proxy. 4. El usuario hace clic sobre el botón "Start Proxy". 5. La aplicación inicia el proxy en el puerto y dirección IP especificada.
Escenario alternativo	1A. El sistema detecta que no existen interfaces de red con direcciones asignadas. El sistema no permite el inicio del proxy, ya que no se ha especificado una dirección IP. 1B. El sistema detecta que el puerto en el que se desea iniciar el proxy está siendo utilizado por otro proceso. El sistema no permite iniciar el proxy y muestra un mensaje de error. 1C. El puerto especificado no tiene el formato adecuado. El sistema muestra un mensaje de error.

CU-02

Nombre	Detener proxy de interceptación.
Descripción	La aplicación permite detener el servidor proxy iniciado previamente.
Actores	Auditor de seguridad.
Importancia	**MODERADA**
Precondición	Es necesario que el proxy de interceptación se haya iniciado previamente.
Postcondición	El sistema detiene el proxy de interceptación, y libera los recursos que estaba utilizando.
Escenario principal	1. El usuario hace clic sobre el botón "Stop proxy". 2. El sistema detiene el proxy de interceptación
Escenario alternativo	1. El sistema detecta un error a la hora de detener el servidor proxy. La aplicación muestra un mensaje de error por pantalla.

CU-03

Nombre	Configurar proxy
Descripción	La aplicación permite establecer los parámetros de configuración del servidor proxy.
Actores	Auditor de seguridad.
Importancia	**MODERADA**
Precondición	N/A
Postcondición	El sistema establece los parámetros de configuración del servidor proxy.
Escenario principal	1. El usuario accede a la sección de opciones de proxy. 2. El usuario establece los parámetros de configuración.
Escenario alternativo	El usuario establece un parámetro de configuración incorrecto. La aplicación informa al usuario de la situación mediante un mensaje de error.

CU-04

Nombre	Activar interceptación de peticiones HTTP/s
Descripción	El sistema permite la interceptación de peticiones HTTP enviadas desde un sistema cliente (navegador web u otra aplicación que se comunica a través del protocolo HTTP). La petición es mostrada por pantalla, y se permite al usuario redirigir la petición hacia el sistema destino, o por el contrario bloquearla, de modo que ésta no se llegaría a enviar.
Actores	Auditor de seguridad
Importancia	**ALTA**
Precondición	El proxy de interceptación de peticiones HTTP/s debe haber sido iniciado previamente.
Postcondición	El sistema permite la interceptación de peticiones HTTP/s, permitiendo que éstas sean mostradas por pantalla a medida que son recibidas.
Escenario principal	1. El usuario hace clic sobre el botón "Intercept On". 2. El sistema desbloquea en la interfaz gráfica los botones que permiten la redirección y el bloqueo de las peticiones interceptadas.
Escenario alternativo	N/A

CU-05

Nombre	Desactivar interceptación de peticiones HTTP/s
Descripción	Al desactivar de la funcionalidad de interceptación de peticiones HTTP/s el sistema registra todas las peticiones recibidas y las redirige de forma automática hacia sistema destino. En este modo de funcionamiento las funcionalidades que permiten la redirección o el bloqueo de peticiones de forma manual se encuentran desactivadas
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	El proxy de interceptación de peticiones HTTP/s debe haber sido

	iniciado previamente.
Postcondición	El sistema bloquea los componentes de la interfaz gráfica que permiten la redirección o el bloqueo de una petición interceptada.
Escenario principal	1. El usuario selecciona el botón "Intercept Off". 2. El sistema bloquea los componentes gráficos que permiten la redirección o el bloqueo de las peticiones HTTP/s interceptadas
Escenario alternativo	N/A

CU-06

Nombre	Enviar petición interceptada al módulo de repetición.
Descripción	El sistema permite configurar el módulo de repetición en base a los datos de una petición HTTP interceptada.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	Debe existir una petición HTTP/s interceptada.
Postcondición	El sistema establece los parámetros de configuración del módulo de repetición, con los datos de una petición interceptada.
Escenario principal	1. Disponiendo de una petición interceptada, el usuario selecciona el módulo al que desea reenviar la petición HTTP. 2. El usuario selecciona el módulo de repetición ("repeater"). 3. El usuario hace clic en el botón "Send". 4. La aplicación configura los parámetros del módulo de repetición en base a los datos de la petición interceptada.
Escenario alternativo	1. Sin existir una petición interceptada el usuario hace clic en el botón "Send". 2. La aplicación muestra un mensaje de error.

CU-07

Nombre	Enviar petición HTTP al módulo de fuzzing.
Descripción	El sistema permite configurar el módulo de fuzzing en base a los datos de una petición HTTP/s interceptada.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	Debe existir una petición HTTP/s interceptada.
Postcondición	El sistema establece los parámetros de configuración del módulo de fuzzing, con los datos de la petición HTTP/s interceptada.
Escenario principal	1. Disponiendo de una petición interceptada el usuario selecciona el módulo al que desea reenviar la petición HTTP. 2. El usuario selecciona el módulo de fuzzing (“fuzzer”). 3. El usuario hace clic en el botón “Send”. La aplicación configura los parámetros del módulo de fuzzing en base a los datos de la petición interceptada.
Escenario alternativo	1. Sin existir una petición interceptada el usuario hace clic en el botón “Send”. 2. La aplicación muestra un mensaje de error.

CU-08

Nombre	Redirigir petición interceptada
Descripción	La aplicación permite que una petición enviada a través del proxy sea redirigida hacia el sistema destino.
Actores	Auditor de seguridad, aplicación.
Importancia	**ALTA**
Precondición	Debe existir una petición HTTP/s interceptada por el proxy.
Postcondición	▪ La aplicación permite la redirección de la petición interceptada a través del proxy hacia el sistema destino. ▪ La respuesta asociada es redirigida a la aplicación que hace uso del proxy. ▪ La aplicación añade la petición enviada, así como la respuesta asociada al historial de navegación.

Escenario principal	1. El proxy recibe una petición de un agente externo (navegador o aplicación que se comunica a través de HTTP). 2. Los datos de la petición son mostrados al usuario. 3. El usuario hace clic sobre el botón "Forward". 4. La petición es enviada al sistema destino y la respuesta correspondiente enviada a la aplicación origen. 5. Tanto los datos de la petición redirigida como los datos de la respuesta son añadidos al historial de navegación del proxy.
Escenario alternativo	N/A

CU-09

Nombre	Bloquear petición HTTP/s interceptada.
Descripción	La aplicación permite al usuario bloquear una petición interceptada, de forma que ésta no se llega a enviar al sistema de destino.
Actores	Auditor de seguridad, aplicación.
Importancia	**ALTA**
Precondición	Debe existir una petición HTTP/s interceptada por el proxy.
Postcondición	▪ La aplicación bloquea la redirección de la petición hacia el sistema destino. ▪ La aplicación que navega a través del proxy recibe un mensaje de error, en el que se informa que la petición fue bloqueada.
Escenario principal	1. El proxy recibe una petición de un agente externo (navegador o aplicación que se comunica a través de HTTP). 2. Los datos de la petición son mostrados al usuario. 3. El usuario hace clic sobre el botón "Block". 4. La petición se descarta. 5. Se envía un mensaje a la aplicación que navega a través del proxy informando que la petición fue bloqueada.
Escenario alternativo	N/A

CU-10

Nombre	Visualizar peticiones realizadas.
Descripción	La aplicación permite visualizar tanto una petición HTTP redirigida a través del proxy, como su respuesta correspondiente.
Actores	Auditor de seguridad
Importancia	**MODERADA**
Precondición	Es necesario que existan peticiones HTTP que hayan sido procesadas por el proxy de interceptación, y que éstas hayan sido añadidas al historial de navegación.
Postcondición	La aplicación muestra los datos de la petición HTTP enviada a través del proxy, así como los datos de la respuesta correspondiente.
Escenario principal	1. El usuario accede a la sección de visualización de peticiones realizadas. 2. El usuario selecciona una petición realizada. 3. La aplicación muestra una ventana en la que se permite visualizar los datos de una petición HTTP, así como los datos de respuesta.
Escenario alternativo	N/A

CU-11

Nombre	Modificar petición interceptada.
Descripción	La aplicación permite la modificación de una petición interceptada por el proxy.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	Debe existir una petición HTTP/s interceptada por el proxy.
Postcondición	▪ La petición interceptada es modificada en base a los cambios realizados por el usuario.

Escenario principal	1. El usuario visualiza la petición HTTP interceptada. 2. El usuario modifica los valores de las cabeceras o el cuerpo de la petición HTTP. 3. La petición es modificada.
Escenario alternativo	1. El usuario borra el contenido de la petición interceptada. 2. La petición se borra.

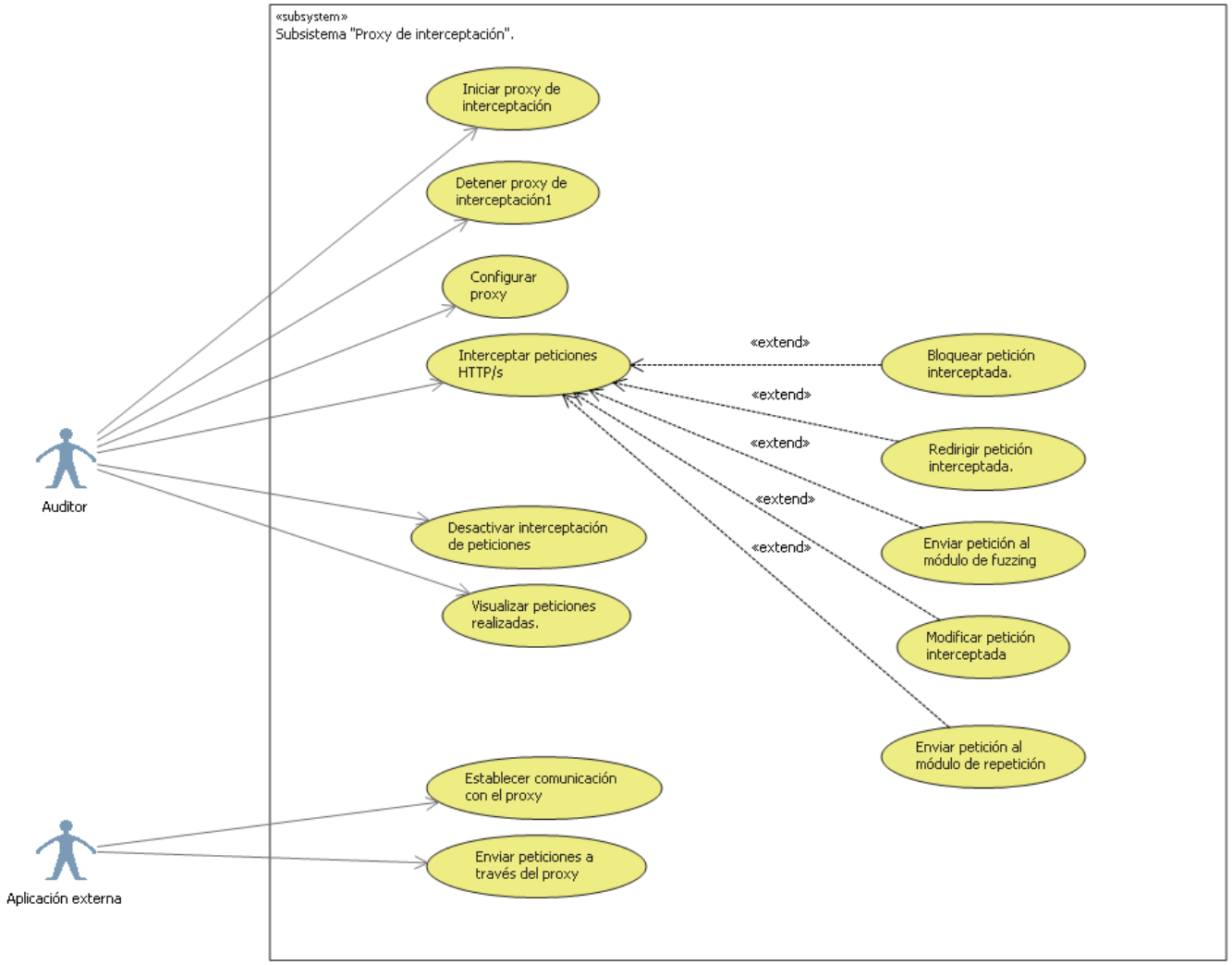


Figura 4: Casos del uso del módulo "Proxy de interceptación"

Casos de uso del subsistema *"módulo de repetición"*

CU-12

Nombre	Enviar petición HTTP/s.
Descripción	El sistema permite la construcción de peticiones HTTP y el envío de las mismas a un sistema destino. Tras el envío, los datos de respuesta se muestran por pantalla.
Actores	Auditor de seguridad
Importancia	**ALTA**
Precondición	Todos los parámetros de módulo de repetición deben estar debidamente configurados.
Postcondición	La aplicación envía la petición HTTP y muestra la respuesta al usuario.
Escenario principal	A. 1. El usuario envía al módulo de repetición una petición HTTP interceptada. 2. El usuario visualiza los datos de la petición HTTP. 3. El usuario selecciona el botón "Send". 4. La petición es enviada al sistema destino. 5. Los datos de respuesta son mostrados por pantalla. B. 1. El usuario accede al módulo de repetición. 2. El usuario establece los parámetros de configuración de la petición HTTP que va a ser enviada. 3. El usuario selecciona el botón "Send". 4. La petición es enviada al sistema destino. 5. Los datos de respuesta son mostrados por pantalla.
Escenario alternativo	1. El usuario establece las opciones de configuración proporcionando valores incorrectos. La aplicación informa de la situación mostrando un mensaje de error.

CU-13

Nombre	Establecer sistema objetivo del módulo de repetición.
Descripción	El sistema permite establecer los parámetros que definen el sistema objetivo (aquel al que será enviada la petición). Las opciones de configuración son: • El nombre del host, • Puerto de destino, • Establecimiento de la conexión a través de un canal de comunicaciones cifrado (HTTPs).
Actores	Auditor de seguridad
Importancia	**ALTA**
Precondición	N/A
Postcondición	Las directivas de configuración se aplican al módulo de repetición.
Escenario principal	1. El usuario establece las opciones de configuración del módulo de repetición. 2. Las directivas de configuración son aplicadas.
Escenario alternativo	1. Las directivas de configuración establecidas son incorrectas. La aplicación muestra un mensaje de error.

A continuación se detalla el diagrama de casos de uso para el subsistema “Módulo de repetición”.

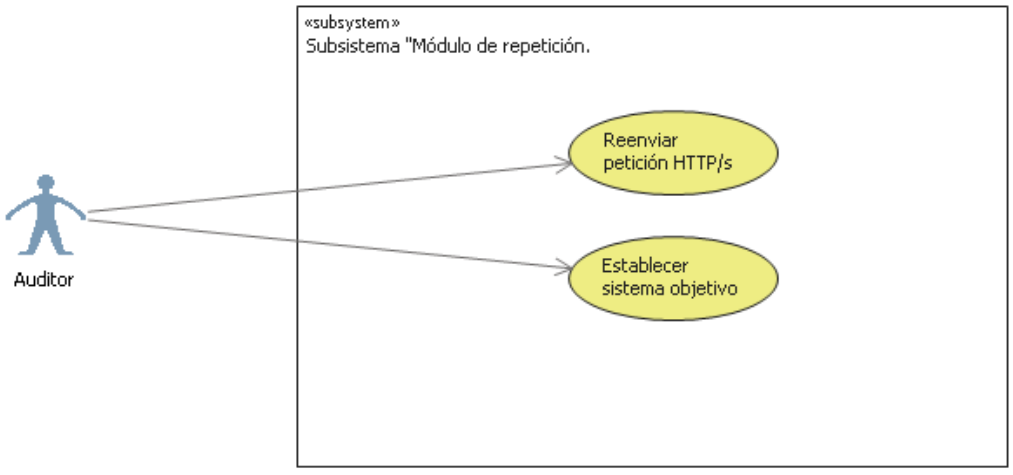


Figura 5: Diagrama de casos de uso para el subsistema "Módulo de repetición".

Casos de uso del subsistema *"módulo de fuzzing"*

CU-14

Nombre	Establecer sistema objetivo del módulo de fuzzing.
Descripción	El sistema permite establecer los parámetros que definen el sistema objetivo (aquel al que serán enviadas las peticiones). Se permite configurar tanto el nombre del host como el puerto del sistema destino, y si la petición debe ser enviada a través de un canal de comunicaciones cifrado HTTPs.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	N/A
Postcondición	Las directivas de configuración se aplican al módulo de fuzzing.
Escenario principal	1. El usuario accede al módulo de fuzzing. 2. El usuario establece los parámetros de configuración especificando el nombre del host destino, el puerto del host destino, y si la petición debe ser enviada a través de un canal de comunicaciones cifrado (HTTPs).
Escenario alternativo	1. El usuario establece una directiva de configuración errónea. La aplicación informa al usuario de la situación a través de un mensaje de error.

CU-15

Nombre	Añadir parámetro de fuzzing.
Descripción	El sistema debe permitir establecer los parámetros sobre los que se llevará a cabo el proceso de fuzzing.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	Debe existir una petición HTTP configurada en el módulo de fuzzing.
Postcondición	Los parámetros de fuzzing son añadidos en el sistema.

Escenario principal	1. El usuario selecciona una sección de la petición HTTP para establecerla como parámetro de fuzzing. 2. La aplicación añade la sección seleccionada a la lista de parámetros. 3. La aplicación muestra los parámetros seleccionados en la sección de selección de ataques y payloads.
Escenario alternativo	N/A

CU-16

Nombre	Eliminar parámetros de fuzzing
Descripción	El sistema debe permitir eliminar los parámetros sobre los cuales se va a realizar el proceso de fuzzing.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	N/A
Postcondición	El sistema elimina los parámetros previamente seleccionados.
Escenario principal	1. El usuario accede al módulo de fuzzing. 2. El usuario selecciona el botón “Clear parameters”. 3. El sistema elimina todos los parámetros establecidos previamente.
Escenario alternativo	N/A

CU-17

Nombre	Configurar payloads.
Descripción	El sistema debe permitir definir los valores que serán empleados para reemplazar a los parámetros seleccionados durante la ejecución de pruebas de fuzzing.
Actores	Auditor de seguridad.
Importancia	**ALTA**

Precondición	Es necesario que se hayan definido previamente los parámetros sobre los cuales se realizará el proceso de fuzzing.
Postcondición	El sistema genera internamente una lista de payloads para el parámetro seleccionado.
Escenario principal	1. E usuario accede a la sección de configuración de payloads del módulo de fuzzing. 2. En función del tipo de ataque seleccionado el usuario puede configurar un parámetro definido previamente. 3. El usuario establece las opciones de configuración del tipo de payload seleccionado. 4. El sistema genera internamente una lista de payloads.
Escenario alternativo	1. El usuario introduce un valor erróneo para el tipo de payload seleccionado. El sistema informa al usuario acerca de la situación mostrando un mensaje de error.

CU-18

Nombre	Seleccionar tipo de ataque.
Descripción	El sistema debe permitir seleccionar el tipo de prueba de fuzzing que se desea ejecutar.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	N/A
Postcondición	El sistema muestra, para el tipo de prueba seleccionada los parámetros y payloads que se pueden definir.
Escenario principal	1. El usuario accede a la sección de payloads del módulo de fuzzing. 2. El usuario selecciona el tipo de ataque que desea realizar. 3. El sistema muestra las opciones correspondientes para el tipo de ataque seleccionado.
Escenario alternativo	N/A

CU-19

Nombre	Iniciar ataque de fuzzing
Descripción	El sistema debe permitir ejecutar el procedimiento de fuzzing sobre los parámetros y payloads configurados.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	▪ El usuario debe haber configurado tanto el host, como el puerto destino. ▪ El usuario debe haber seleccionado un tipo de procedimiento de fuzzing concreto. ▪ El usuario debe haber seleccionado los parámetros sobre los que se realizará el proceso de fuzzing. ▪ El usuario debe haber configurado los payloads asociados a los parámetros seleccionados.
Postcondición	El sistema muestra una lista en la que se muestran todas las peticiones enviadas, así como los datos de respuesta.
Escenario principal	1. El usuario accede al módulo de fuzzing. 2. El usuario selecciona el host y puerto concreto. 3. El usuario selecciona el tipo de procedimiento de fuzzing. 4. El usuario selecciona los parámetros sobre la petición HTTP. 5. El usuario asigna los payloads a los parámetros. 6. El usuario hace clic sobre el botón “Start Attack”.
Escenario alternativo	El usuario establece un valor incorrecto en alguno de los pasos descritos en el escenario principal. La aplicación informa de la situación mediante un mensaje de error.

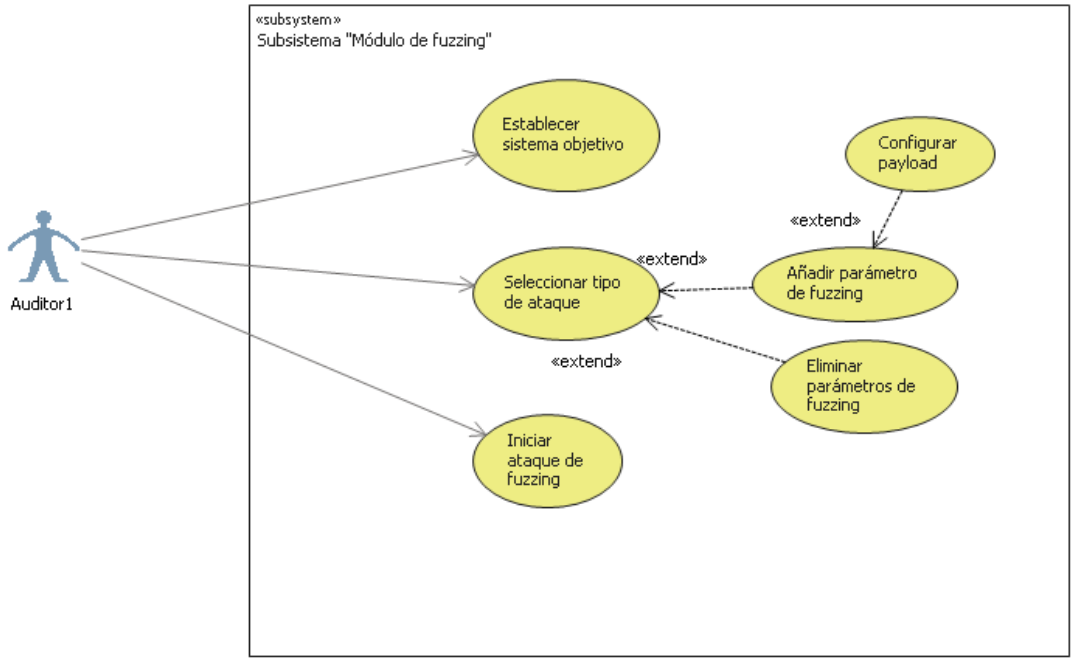


Figura 6: Diagrama de casos de uso para el subsistema "Módulo de fuzzing".

Casos de uso del subsistema *"módulo de scripting"*

CU-20

Nombre	Crear script
Descripción	El sistema permite la creación de scripts a través de la aplicación.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	N/A
Postcondición	El sistema crea el script, manteniendo una referencia interna del mismo.
Escenario principal	1. El usuario hace clic sobre el botón "New Script". 2. El usuario selecciona el tipo de scripts que desea crear. 3. El sistema crea una nueva pestaña a través de la cual será posible desarrollar el código del script.
Escenario alternativo	N/A

CU-21

Nombre	Abrir script
Descripción	El sistema permite abrir un fichero de texto que contiene el código fuente de un script.
Actores	Auditor de seguridad.
Importancia	**MODERADA**
Precondición	N/A
Postcondición	El sistema abre el fichero seleccionado manteniendo una referencia del mismo.
Escenario principal	1. El usuario hace clic sobre el botón "Open script". 2. La aplicación muestra por pantalla una ventana a través de la cual es posible seleccionar el fichero que se desea abrir. 3. El sistema crea una nueva pestaña mostrando el contenido del fichero seleccionado.
Escenario alternativo	N/A

CU-22

Nombre	Guardar script.
Descripción	El sistema debe permitir guardar un fichero de script creado a través de la aplicación.
Actores	Auditor de seguridad.
Importancia	**MODERADA**
Precondición	Es necesario que se haya creado o se haya abierto un script previamente.
Postcondición	El sistema guarda el script en el equipo del usuario.
Escenario principal	1. El usuario hace clic sobre el botón "Guardar". 2. Se muestra por pantalla una ventana a través de la cual es posible seleccionar el directorio en el que se desea guardar el script.

	3. El sistema guarda el script en el directorio seleccionado.
Escenario alternativo	1. Si la cuenta de usuario con la que se ejecuta la aplicación no dispone de privilegios suficientes para guardar un fichero en el directorio seleccionado, se muestra un mensaje de error por pantalla.

CU-23

Nombre	Cerrar script
Descripción	El sistema debe permitir cerrar los scripts abiertos previamente.
Actores	Auditor de seguridad.
Importancia	**MODERADA**
Precondición	Es necesario que se haya creado o se haya abierto un script previamente.
Postcondición	El sistema cierra el componente gráfico en el que se muestran los datos del script.
Escenario principal	1. El usuario selecciona el botón que permite cerrar la pestaña que contiene el script. 2. Se cierra la pestaña que contiene el script.
Escenario alternativo	N/A

CU-24

Nombre	Compilar script.
Descripción	El sistema debe permitir la compilación de los scripts editados a través de la aplicación.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	Es necesario que se haya abierto un script previamente.
Postcondición	El sistema compila el script y almacena los resultados de la compilación

Escenario principal	1. El usuario dispone de un script abierto previamente. 2. El usuario selecciona el botón "Compile". 3. Los resultados de compilación son mostrados por pantalla.
Escenario alternativo	1. Si la compilación del script falla se muestran por pantalla los errores de compilación.

CU-25

Nombre	Ejecutar script.
Descripción	El sistema debe permitir la ejecución de los scripts creados a través de la aplicación.
Actores	Auditor de seguridad.
Importancia	**ALTA**
Precondición	Es necesario que previamente se haya compilado el script que se desea ejecutar.
Postcondición	El sistema ejecuta el script y muestra por pantalla los resultados de la ejecución.
Escenario principal	1. El usuario dispone de un script abierto previamente. 2. El usuario selecciona el botón "Run". 3. El sistema ejecuta el script y muestra los resultados por pantalla.
Escenario alternativo	1. Si el script falla se informa al usuario a través de un mensaje de error.

CU-26

Nombre	Especificar parámetros de entrada del script.
Descripción	El sistema debe permitir especificar los posibles parámetros de entrada necesarios para la ejecución de un script mediante la definición de un documento XML.
Actores	Auditor de seguridad.
Importancia	**MODERADO**

Precondición	Es necesario que se haya creado o se haya abierto un script previamente.
Postcondición	N/A
Escenario principal	1. El usuario dispone de un script abierto. 2. El usuario accede a la pestaña que permite especificar los parámetros de entrada de un script concreto. 3. El usuario establece los parámetros de entrada del script.
Escenario alternativo	N/A

CU-27

Nombre	Añadir script al repositorio de scripts.
Descripción	La aplicación mantiene dos repositorios de scripts: un repositorio de scripts de auditoría a través de los cuales es posible ejecutar pruebas sobre un sistema concreto y otro repositorio que contiene scripts de callback, que permiten modificar el comportamiento del motor de comunicaciones HTTP o del servidor proxy. La aplicación permite guardar en el repositorio seleccionado un script concreto.
Actores	Auditor de seguridad.
Importancia	**MODERADA**
Precondición	Es necesario que se haya creado o se haya abierto un script previamente.
Postcondición	La aplicación añade el script al repositorio seleccionado y crea un fichero con el contenido del script en una carpeta localizada dentro del directorio de trabajo de la aplicación.
Escenario principal	1. El usuario dispone de un script abierto. 2. El usuario selecciona el botón “Add script to library”. 3. La aplicación muestra una ventana a través de la cual es posible seleccionar el repositorio en el que se desea guardar el script. 4. El usuario hace clic sobre el botón “Add”. 5. El script se añade al repositorio seleccionado.

Escenario alternativo	N/A

CU-28

Nombre	Activar script de callback
Descripción	La aplicación permite activar scripts de callback a través de los cuales es posible modificar el comportamiento del servidor proxy o del motor de comunicaciones HTTP ante el envío de una petición o la recepción de una respuesta.
Actores	Auditor de seguridad.
Importancia	**MODERADA**
Precondición	Es necesario que se haya compilado el script de callback previamente.
Postcondición	El sistema activa el script de callback y lo añade a la lista de callbacks del motor de comunicaciones HTTP o del servidor proxy.
Escenario principal	1. El usuario selecciona un script de callback a través del repositorio de scripts de callback. 2. El usuario hace clic en botón "Activate callback". 3. El callback es añadido al motor de comunicaciones HTTP o al servidor proxy.
Escenario alternativo	1. El código del script tiene errores. La aplicación muestra un mensaje de error. 2. El script ya se encuentra activo. La aplicación muestra un mensaje de error.

CU-29

Nombre	Desactivar script de callback.
Descripción	La aplicación permite desactivar los scripts de callbacks que fueron activados previamente.
Actores	Auditor de seguridad.
Importancia	**MODERADA**
Precondición	Es necesario que se haya activado un script de callback previamente.
Postcondición	El sistema elimina el callback de la lista de callbacks del motor de comunicaciones HTTP o del servidor proxy.
Escenario principal	1. El usuario selección un script de *callback* activo. 2. El usuario hace clic sobre el botón "deactivate *callback*." 3. La aplicación desactiva el script de callback seleccionado.
Escenario alternativo	1. Si el script de *callback* seleccionado no estaba activo la aplicación muestra un mensaje de error.

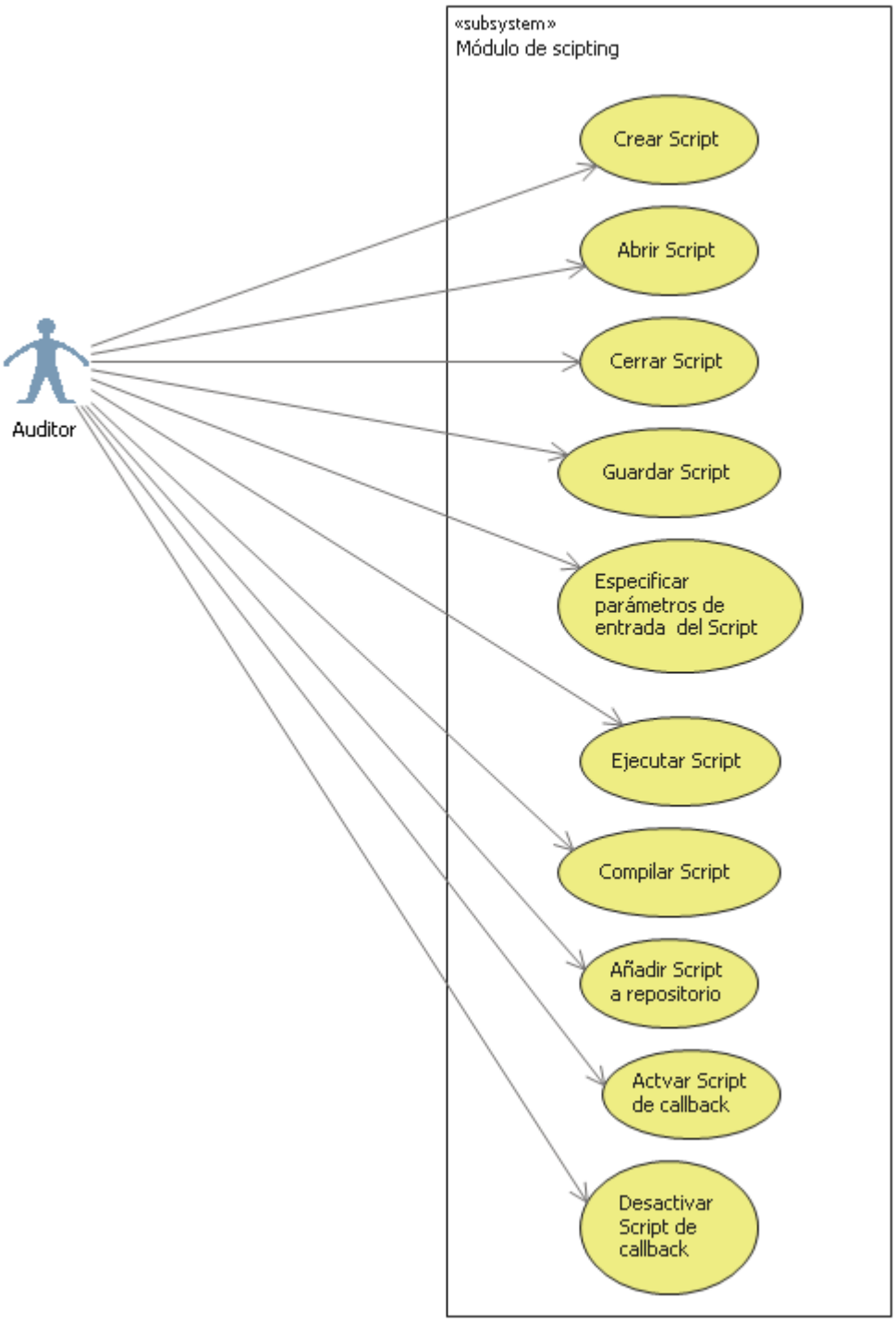


Figura 7: Diagrama de casos de uso para el subsistema "Módulo de scripting".

3.2 Análisis de requisitos del software.

La especificación de requisitos de software describe desde una perspectiva formal el comportamiento del sistema a desarrollar, representando de forma completa e inequívoca las características del sistema.

Para llevar a cabo la documentación de requisitos se seguirá el modelo planteado por el estándar IEEE 830-1998 (12). Los requisitos planteados se definen en base a las siguientes categorías:

- **Restricciones de diseño:** Imponen una serie de restricciones que limitan las opciones de implementación a la hora de crear un producto. Se especifican como RDn, donde n es el número del requisito.

- **Requisitos funcionales**: Reflejan las funcionalidades que debe disponer el producto desarrollado. Se especifican como RFn, donde n es el número del requisito.

- **Requisitos de interfaz:** Requisitos que establecen las necesidades de interacción del software con otros sistemas o usuarios. Se especifican como RIn, donde n es el número del requisito.

- **Requisitos de calidad:** Define una serie exigencias de calidad que debe cumplir el producto. Se especifican como RCn, donde n es el número del requisito.

- **Requisitos del proyecto:** Requisitos que condicionan el desarrollo del proyecto. Se especifican como RPRn, donde n es el número del requisito.

- **Requisitos de evolución:** Definen aspectos de diseño con el objetivo de flexibilizar la adaptación del producto a condiciones o exigencias que podrían surgir a futuro. Se especifican como REVn, donde n es el número del requisito.

- **Requisitos de seguridad:** Definen aquellos aspectos de seguridad que deben ser cumplidos por parte del sistema desarrollado. Se especifican como RSn, donde n es el número del requisito.

En base a los casos de uso definidos se han identificado los siguientes requisitos:

3.2.1 Restricciones de diseño

REQUISITO RD1

Título: La parte grafica de la aplicación debe ser desarrollada utilizando la tecnología Windows Forms.

Importancia: Esencial

Validez:

- *Medible*: Verificar que todos los componentes gráficos de la aplicación están desarrollados utilizando la tecnología *Windows Forms*.
- *Alcanzable*: Todos los componentes gráficos utilizados por la aplicación deben estar desarrollados utilizando la API *Windows Forms*.
- *Relevante*: La implementación de la parte gráfica de la aplicación utilizando la tecnología *Windows Forms,* permitirá a futuro la ejecución de la herramienta en entornos Linux a través del framework Mono Develop (13). Actualmente Mono solo soporta interfaces gráficas desarrolladas en esta tecnología. Sin embargo medida que el soporte de funcionalidades de .NET avance será posible desarrollar la parte gráfica de la aplicación utilizando otra tecnología.

REQUISITO RD2

Título: La aplicación debe disponer de una arquitectura de diseño que permita reemplazar la tecnología empleada en el desarrollo de la parte gráfica de la aplicación.

Importancia: Esencial

Validez:

- *Medible*: Verificar que la arquitectura de diseño de la aplicación permite el reemplazo de la tecnología empleada en la implementación de la parte gráfica.
- *Alcanzable*: Plantear la arquitectura de la aplicación utilizando el patrón de diseño Modelo Vista Presentador, de forma que sea posible separar la lógica de presentación, de la lógica de negocio.
- *Relevante*: A futuro será posible utilizar otra tecnología diferente para el desarrollo de la parte gráfica de la aplicación.

REQUISITO RD3

Título: La aplicación debe ser desarrollada utilizando la versión 4 del framework .NET

Importancia: Esencial

Validez:

- *Medible*: Verificar que no se utiliza ninguna funcionalidad de las versiones 4.5 o 5.0 del framework .NET
- *Alcanzable*: Configurar el entorno de programación para hacer uso de la versión 4.0 del framework .NET.
- *Relevante*: Ya que a futuro se pretende ofrecer soporte multiplataforma de herramienta., la aplicación deberá ser desarrollada utilizando la versión 4.0 del framework .NET ya que es la versión soportada por Mono Develop. (13)

REQUISITO RD4

Título: Se debe permitir la interoperabilidad de la API HTTPCore con cualquier lenguaje de programación.

Importancia: Esencial

Validez:

- *Medible*: Se debe verificar que las funcionalidades fundamentales de la aplicación son expuestas en una biblioteca de vínculo dinámico (DLL).
- *Alcanzable*: Se debe desarrollar una interfaz (wrapper) que permitan exportar las funcionalidades públicas de la API, a través de una biblioteca de vínculo dinámico. Desde el lenguaje de programación C# será necesario definir clases proxy que permitan abstraer los detalles de acceso a la DLL.
- *Relevante*: La API HTTPCore dispone de todas las funcionalidades que permiten la gestión de comunicaciones HTTP/s, así como la creación y gestión de proxies HTTP.

REQUISITO RD5

Título: El sistema debe hacer uso exclusivo de la API HTTPCore para todas las comunicaciones HTTP/s

Importancia: Esencial

Validez:

- *Medible*: Para el cumplimiento de este requisito se debe verificar que cualquier comunicación HTTP/s se realiza utilizando las funcionalidades exportadas de la API *HTTPCore*.
- *Alcanzable*: Cualquier comunicación HTTP/s se debe llevar cabo utilizando la las funcionalidades de la API HTTPCore exportadas en una DLL.
- *Relevante*: La API dispone de las funcionalidades necesarias para la implementación de la aplicación que se va a desarrollar.

3.2.1 Requisitos funcionales.

Requisitos funcionales del *wrapper* que permite exportar las funcionalidades de la API HTTPCore

Los siguientes requisitos representan las funcionalidades de la API *HTTPCore* que deben ser exportadas a través del wrapper.

REQUISITO RF1

Título: El wrapper debe exportar la funcionalidad de la API que permite el inicio conexiones HTTP/s

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento del requisito es necesario verificar que: utilizando las funcionalidades de la API exportadas a través de los wrappers es posible establecer una conexión HTTP/s con un sistema concreto.
- *Alcanzable:* Es necesario exportar la funcionalidad a través de una biblioteca de vínculo dinámico (DLL)
- *Relevante:* Se trata de una característica fundamental de la API, que será utilizada por la aplicación que se pretende desarrollar.

REQUISITO RF2

Título: El *wrapper* debe exportar la funcionalidad que permite terminar conexiones HTTP/s establecidas previamente.

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento del requisito es necesario verificar que utilizando las funcionalidades de la API exportadas a través de los wrappers es posible llevar a cabo la terminación de una conexión HTTP/s, establecida previamente.
- *Alcanzable:* Es necesario exportar la funcionalidad a través de una biblioteca de vínculo dinámico (DLL)
- *Relevante:* Se trata de una característica fundamental de la API, que será utilizada por la aplicación que se pretende desarrollar.

REQUISITO RF3

Título: El *wrapper* debe exportar la funcionalidad que permite interactuar con los distintos campos de una petición HTTP.

Importancia: Esencial

Validez:

- *Medible:* Verificar que los cambios realizados sobre las cabeceras o el cuerpo de una petición HTTP se llevan a cabo correctamente.
- *Alcanzable:* Es necesario exportar la funcionalidad a través de una biblioteca de vínculo dinámico (DLL)
- *Relevante:* Se trata de una característica fundamental de la API, que será utilizada por la aplicación que se pretende desarrollar

REQUISITO RF4

Título El *wrapper* debe exportar la funcionalidad que permite obtener las respuestas de las peticiones HTTP utilizando los métodos de la API.

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento del requisito se debe verificar que una vez enviada una petición HTTP es posible obtener la respuesta correspondiente utilizando las funcionalidades de la API expuestas por el wrapper.
- *Alcanzable:* Es necesario exportar la funcionalidad a través de una biblioteca de vínculo dinámico (DLL)
- *Relevante:* Se trata de una característica fundamental de la API, que será utilizada por la aplicación que se pretende desarrollar

REQUISITO RF5

Título: El *wrapper* debe exportar la funcionalidad que permite iniciar un proxy HTTP.

Importancia: Esencial

Validez:

- *Medible:* El requisito se cumplirá si es posible iniciar un proxy HTTP a través del cual es posible enviar peticiones HTTP desde una aplicación externa.
- *Alcanzable:* Es necesario exportar la funcionalidad a través de una biblioteca de vínculo dinámico (DLL)
- *Relevante:* Se trata de una característica fundamental de la API, que será utilizada por la aplicación que se pretende desarrollar

REQUISITO RF6

Título: El *wrapper* debe exportar la funcionalidad que permite detener un proxy HTTP establecido previamente.

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento del requisito es necesario verificar que una vez que se ha invocado la terminación del proxy HTTP, el canal de comunicaciones es liberado, y no es posible continuar enviando peticiones a través del mismo.
- *Alcanzable:* Es necesario exportar la funcionalidad a través de una biblioteca de vínculo dinámico (DLL)
- *Relevante:* Se trata de una característica fundamental de la API, que será utilizada por la aplicación que se pretende desarrollar

REQUISITO RF7

Título: El *wrapper* debe exportar la funcionalidad que permite establecer las opciones de configuración del proxy HTTP.

Importancia: Esencial

Validez:

- *Medible:* El requisito se cumplirá si, cualquier opción del proxy HTTP configurada a través de la funcionalidad exportada a través del wrapper se establece correctamente.
- *Alcanzable:* Es necesario exportar la funcionalidad a través de una biblioteca de vínculo dinámico (DLL).
- *Relevante:* Se trata de una característica fundamental de la API, que será utilizada por la aplicación que se pretende desarrollar

REQUISITO RF8

Título: El *wrapper* debe exportar la funcionalidad que permite establecer las opciones de configuración del motor de comunicaciones

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento del requisito será necesario verificar que cualquier opción del motor de comunicaciones HTTP se establece correctamente tras invocar esta funcionalidad.

- *Alcanzable:* Es necesario exportar la funcionalidad a través de una librería dinámica.
- *Relevante:* Se trata de una característica fundamental de la API, que será utilizada por la aplicación que se pretende desarrollar

REQUISITO RF9

Título: El *wrapper* debe exportar las funciones que permiten interactuar con las funcionalidades de *callback* definidas en la API.

Importancia: Esencial

Validez:

- *Medible:* El wrapper debe permitir la creación, activación y gestión de callbacks a través de los métodos definidos en la API.
- *Alcanzable:* Es necesario exportar la funcionalidad a través de una librería dinámica.
- *Relevante:* Se trata de una característica fundamental de la API, que será utilizada por la aplicación que se pretende desarrollar

Requisitos funcionales de la aplicación

A continuación se muestran los requisitos funcionales propios de la aplicación que se va a desarrollar.

REQUISITO RF10

Título: La aplicación debe permitir iniciar un proxy de interceptación en una dirección IP y puerto especificado por el usuario.

Importancia: Esencial

Validez:

- *Medible:* Se debe verificar que el proxy se inicia en la dirección IP y puerto especificado por el usuario.
- *Alcanzable:* Para la implementación de esta funcionalidad se hará uso de la funcionalidad de inicio de proxy de la API, exportada a través de los *wrappers*.
- *Relevante:* La creación de un proxy HTTP permitirá la interceptación de las comunicaciones HTTP/s establecidas a través del mismo.

REQUISITO RF11

Título: La aplicación debe permitir detener un proxy HTTP iniciado previamente.

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento del requisito se debe verificar que una vez que se ha invocado la terminación del proxy, el canal de comunicaciones se libera correctamente.
- *Alcanzable:* Es necesario utilizar de la funcionalidad de inicio de proxy de la API, exportada a través de los *wrappers*.
- *Relevante:* La modificación de ciertos parámetros de configuración, requiere el reinicio del proxy, por lo tanto es necesario disponer de una funcionalidad que permite llevar a cabo la detención del mismo.

REQUISITO RF12

Título: La aplicación debe permitir el registro de las peticiones HTTP que se envían a través del proxy.

Importancia: Esencial

Validez:

- *Medible:* Se debe verificar que las peticiones HTTP enviadas desde un agente externo son registradas por el proxy HTTP.
- *Alcanzable:* Es necesario establecer una función de callback que permita obtener tanto los datos de las peticiones enviadas, como de las respuestas.
- *Relevante:* Se trata de una funcionalidad imprescindible de cualquier herramienta de auditoría en entornos web, ya que es posible visualizar que datos se están enviando a un servidor destino, en cada petición.

REQUISITO RF13

Título: La aplicación debe permitir la activación o desactivación del módulo de interceptación de peticiones HTTP/s.

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento del requisito se debe verificar que las peticiones HTTP enviadas desde un agente externo son interceptadas y mostradas por pantalla, permitiendo decidir al usuario, si la petición debe ser redirigida al

sistema destino o si debe ser bloqueada. La desactivación del módulo de interceptación supondrá la redirección automática de todas las peticiones enviadas a través del proxy.

- *Alcanzable:* Para la implementación de esta funcionalidad se establecerá una función de callback que permita obtener los datos de todas las peticiones enviadas a través del proxy.
- *Relevante:* Se trata de una funcionalidad imprescindible de cualquier herramienta de auditoría en entornos web, ya que es posible visualizar que datos se están enviando a un servidor web en cada petición.

REQUISITO RF14

Título: La aplicación debe permitir bloquear/rechazar las peticiones HTTP que han sido interceptadas por el proxy.

Importancia: Esencial

Validez:

- *Medible:* Se debe verificar que, ante el bloqueo de una petición HTTP, el agente externo que envío la petición recibe un mensaje de error informando la situación de bloqueo.
- *Alcanzable:* Para la implementación de esta funcionalidad se establecerá una función de callback que permitirá la interceptación de peticiones. El bloqueo de una petición HTTP se llevara a cabo tras recibir un evento de bloqueo, tras hacer clic sobre el botón “Block”.
- *Relevante:* El bloqueo de peticiones HTTP que pasan a través del proxy permiten cancelar acciones no deseadas durante la realización de una auditoria.

REQUISITO RF15

Título: La aplicación debe permitir la redirección de peticiones HTTP enviadas a través del proxy hacia un sistema de destino.

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento del requisito se debe verificar que, cuando se permite la redirección de una petición HTTP a través del proxy, el sistema destino recibe la petición y la respuesta correspondiente es redirigida a través del proxy hacia la aplicación que inicio la comunicación.

- *Alcanzable:* Para la implementación de esta funcionalidad se establecerá una función de callback que permitirá la interceptación de peticiones. La redirección de una petición HTTP se llevara a cabo tras recibir un evento de redirección.
- *Relevante:* Se trata de una funcionalidad imprescindible de cualquier proxy de interceptación web.

REQUISITO RF16

Título: La aplicación debe permitir la modificación de las peticiones HTTP interceptadas.

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento de este requisito se verificará que la modificación de una petición HTTP interceptada se lleva a cabo correctamente.
- *Alcanzable:* Para la implementación de esta funcionalidad se establecerá una función de callback que permitirá la interceptación de peticiones. La modificación de los datos de la petición interceptada se reflejaran en la estructura interna que gestiona una petición HTTP concreta.
- *Relevante:* Se trata de una funcionalidad imprescindible para cualquier proxy de interceptación web, ya que permite la manipulación de los datos de entrada enviados al sistema objetivo que se pretende auditar.

REQUISITO RF17

Título: La aplicación debe disponer de un módulo que permita el reenvío de peticiones HTTP a un host y puerto concreto.

Importancia: Esencial

Validez:

- *Medible:* Se debe verificar que la petición es enviada al host y puerto de destino especificado por el usuario.
- *Alcanzable:* La aplicación debe hacer uso de las funcionalidades exportadas de la API que permiten el envío de peticiones HTTP/s.
- *Relevante:* El envío de peticiones HTTP/s es una característica necesaria de cualquier herramienta de auditoría de seguridad sobre entornos web

.

REQUISITO RF18

Título: La aplicación debe permitir realizar procedimientos de *fuzzing* sobre peticiones HTTP.

Importancia: Esencial

Validez:

- *Medible:* La aplicación debe disponer de un módulo que permita llevar a cabo procesos iterativos de *fuzzing* de peticiones HTTP.
- *Alcanzable:* El proceso debe permitir la selección de los parámetros sobre los cuales se realizaran las pruebas. Se deben definir además distintas estrategias de *fuzzing* sobre los parámetros configurables, permitiendo:
    - El reemplazo de todos los parámetros seleccionados al mismo tiempo utilizando un único payload en cada iteración.
    - El reemplazo de cada uno de los parámetros seleccionados de forma separada utilizando un único payload en cada iteración.
    - Asignación de una lista concreta de valores para cada parámetro seleccionado. Durante la realización del procedimientos iterativo cada parámetro es reemplazado por un valor de la lista de forma independiente.
- *Relevante:* Las funcionalidades son necesarias para la herramienta ya que permiten la identificación de vulnerabilidades en aplicativos web mediante la utilización de procedimientos de prueba basados en técnicas de fuzzing.

REQUISITO RF19

Título: La aplicación permitirá la creación y edición de scripts desarrollados en C#, que hacen uso de la API *HTTPCore* a través de un entorno de desarrollo integrado en la propia herramienta.

Importancia: Esencial

Validez:

- *Alcanzable:* La aplicación dispondrá una de entorno de desarrollo a través del cual será posible crear o abrir scripts.
- *Relevante:* De cara a aportar un elemento distintivo a la herramienta de auditoria se presenta la funcionalidad de creación de scripts. El desarrollo de scripts permitirá implementar procedimientos automáticos de prueba, haciendo uso de la API HTTPCore.

REQUISITO RF20

Título: La aplicación debe permitir la compilación de scripts.

Importancia: Esencial

Validez:

- *Medible:* El requisito se cumplirá si el sistema permite la compilación de scripts y el almacenamiento de los objetos de compilación resultantes, necesarios para su posterior ejecución.
- *Alcanzable:* La aplicación hará uso de las funcionalidades del framework .NET para llevar a cabo la compilación de los scripts desarrollados en C# que hacen uso de las funcionalidades exportadas de la API *HTTPCore.* Para permitir la compilación de los scripts será necesario que estos implementen una interfaz de programación concreta. Se contemplan dos tipos de interfaces:
  - *Interfaz de scripts de auditoria*: La interfaz de auditoria proporciona un conjunto de métodos, que deberán ser implementados por un script concreto para permitir su ejecución. Los scripts de auditoria harán uso de las funcionalidades exportadas de la API.
  - *Interfaz de scripts de callback:* La interfaz de callback proporcionará un conjunto de métodos que deberán ser implementados por un script concreto con el objetivo de permitir su ejecución. El objetivo de los scripts de callback es modificar el comportamiento del motor de comunicaciones HTTO o del proxy interceptación.
- *Relevante:* Los scripts serán desarrollados en C# y harán uso la API a través de un conjunto de clases proxy que abstraen los detalles de acceso a la DLL, por lo tanto es necesario definir una funcionalidad que permita la compilación.

REQUISITO RF21

Título: La aplicación debe permitir la ejecución de los scripts compilados a través de la aplicación.

Importancia: Esencial

Validez:

- *Medible:* El requisito se cumplirá si el sistema permite la ejecución de scripts compilados a través de la aplicación. La aplicación deberá permitir visualizar los resultados de ejecución por pantalla.
- *Alcanzable:* La aplicación hará uso de las funcionalidades del framework .NET para llevar a cabo la ejecución de los scripts compilados a través de la

aplicación. Se contempla la ejecución de dos tipos de scripts en función de la implementación de las interfaces definidas en el requisito RF-20.

- *Scripts de auditoría:* Los scripts de auditoría harán uso de la API de comunicación HTTP. Tras su ejecución será posible visualizar los resultados de ejecución por pantalla. Este tipo de script puede aceptar parámetros de entrada especificados como un documento XML que podrá ser definido a través de la aplicación. La validación del formato y parámetros de entrada deberá ser implementada a través de los métodos de validación definidos en la interfaz de script de auditoría
- *Scripts de callback:* La ejecución de los scripts de callback permitirá la modificación del comportamiento del proxy o del motor de comunicaciones HTTP. Para su ejecución será necesario llevar a cabo la activación de los mismos (*RF-API-22*)

- *Relevante:* Dado que los scripts deben ser desarrollados en C#, es necesario proveer un método que permita la ejecución de los mismos.

REQUISITO RF22

Título: La aplicación debe permitir la activación y desactivación de la ejecución de los scripts de callback.

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento del requisito se verificará que la activación de un script callback concreto permite la modificación del motor de comunicaciones HTTP o del proxy. Se deberá verificar que la desactivación del script de callback restaura el comportamiento por defecto del motor de comunicaciones o del proxy.
- *Alcanzable* La activación de un script de callback supondrá su ejecución, en alguno de los puntos del proceso de recepción y envío de peticiones HTTP. Un script de callback activo podrá, a través de los métodos definidos en la API modificar las cabeceras o el cuerpo tanto de una petición HTTP como de una respuesta en alguno de los siguientes puntos del proceso de envío/recepción de peticiones.

  - *Antes del envío de una petición a través del motor de comunicaciones HTTP.*
  - *Tras la recepción de una petición a través del motor de comunicaciones HTTP*
  - *Antes de la redirección de una petición HTTP enviada a través del proxy.*

- *Tras la recepción de una petición HTTP a través del proxy.*

- *Relevante:* La activación y desactivación de los scripts de callback permite definir procedimientos de modificación de peticiones y respuestas, que pueden ser de utilidad ante escenarios en los que es necesario realizar un filtrado de los datos.

REQUISITO RF23

Título: La aplicación debe disponer de un repositorio de scripts que permita el almacenamiento de los mismos.

Importancia: Esencial

Validez:

- *Medible:* Para el cumplimiento de este requisito se verificará que la aplicación implementa un módulo que permite visualizar una lista de scripts integrados en la herramienta.
- *Alcanzable:* El repositorio de scripts permitirá el almacenamiento y categorización de scripts en función de sus funcionalidades. Los scripts de auditoria podrán categorizarse según las fases de un *test de intrusión.*

  - *Reconocimiento/Enumeración*
  - *Escaneo de vulnerabilidades.*
  - *Explotación de vulnerabilidades.*
  - *Postexplotación.*

  Los scripts de callback podrán categorizarse en función del punto de ejecución del mismo, como se describió en el requisito RF-22.
- *Relevante:* Esta funcionalidad permite almacenar un conjunto de scripts que pueden ser utilizados durante la realización de una auditoría. La disponibilidad de esta funcionalidad aporta mayor valor a la herramienta final.

3.2.2 Requisitos no funcionales

REQUISITO RNF1

Título: El idioma de la aplicación debe ser el inglés.

Importancia: Opcional

Validez:

- *Alcanzable*: Cualquier texto presentado de la aplicación debe utilizar como idioma el inglés.
- *Relevante*: De cara ofrecer una uso internacional de la herramienta se plantea el inglés como el idioma por defecto en la aplicación.

REQUISITO RNF2

Título: La aplicación debe estar desarrollada en C#.

Importancia: Esencial

Validez:

- *Medible*: Verificar que el proyecto de desarrollo está desarrollado utilizando lenguaje de programación C#
- *Alcanzable*: Todo el código de la aplicación debe estar desarrollada en C#. El acceso a la API *HTTPCore* debe realizarse a través de clases proxy, que abstraen los detalles de acceso a la DLL y que permiten reconstruir el comportamiento de la API desde una perspectiva orientada a objetos. (Detalle de implementación)
- *Relevante*: El conjunto de funcionalidades ofrecidas por .NET y la escasez de vulnerabilidades identificadas sobre este framework han condicionado la elección del mismo.

3.2.3 Requisitos de interfaz.

REQUISITO RI1

Título: La interfaz gráfica de la aplicación debe ser adaptable al espacio del escritorio. Igualmente los componentes de la interfaz deben disponer de capacidades de redimensionamiento.

Importancia: Esencial

Validez:

- *Medible*: Verificar que es posible redimensionar tanto la ventana principal de la aplicación, así como los distintos componentes de la misma.
- *Alcanzable*: Emplear contenedores gráficos que permitan el redimensionamiento de los elementos de la interfaz.

- *Relevante*: La visualización de los datos en un espacio correcto facilita la realización de la auditoria. El incumplimiento de este requisito limitaría la usabilidad de la aplicación.

3.2.4 Requisitos de calidad.

REQUISITO RC1

Título: No deben existir fugas de memoria asociados a la creación, y no liberación de los objetos instanciados a través del código no administrado (6) de la API.

Importancia: Esencial

Validez:

- *Medible*: Verificar mediante una herramienta de profiling de código que no se producen fugas de memoria en la API.
- *Alcanzable*: Implementación de una estrategia de liberación automática de objetos en memoria a través de la implementación de clases proxy que heredan de la interfaz IDisposable (14) en C#.
- *Relevante*: La existencia de fugas de memoria degradaría en gran medida el funcionamiento de la aplicación.

3.2.5 Requisitos de evolución.

REQUISITO REV1

Título: La aplicación debe disponer de una arquitectura modular, que permita de forma flexible la integración de nuevas funcionalidades.

Importancia: Esencial

Validez

- *Medible*: La aplicación debe soportar la integración de nuevos módulos.
- *Alcanzable*: Llevar a cabo la separación de la lógica de presentación con respecto a la lógica de negocio y el modelo de datos.
- *Relevante*: El cumplimiento de este requisito es necesario para la integración de nuevas funcionalidades a futuro.

3.2.6 Requisitos del proyecto.

REQUISITO RPR1

Título: Tanto la aplicación como la documentación deben estar finalizadas y entregadas antes del día 9 de Septiembre de 2013.

Importancia: Esencial

Validez

- *Medible*: El requisito se cumplirá cuando el proyecto haya concluido y la documentación haya sido entregada.

3.2.7 Requisitos de seguridad.

REQUISITO RS1

Título: El wrapper debe validar correctamente los datos de entrada para cada funcionalidad concreta.

Importancia: Esencial

Validez

- *Medible*: Se deberá verificar que las rutinas de validación implementadas en los métodos exportados por la API se ejecutan correctamente.
- *Alcanzable*: Los métodos exportados deben implementar rutinas de validación de datos para verificar que los datos de entrada son correctos.
- *Relevante*: Dado que la aplicación hace uso de código no administrado (6) a través de la librería HTTPCore la existencia de un fallo de corrupción de memoria supondría el cierre inesperado de la aplicación.

REQUISITO RS2

Título: El wrapper debe comprobar la validez de los punteros que referencian los objetos creados por la API.

Importancia: Esencial

Validez

- *Medible*: Se deberá verificar que los métodos invocados devuelven un código de error cuando se detecta un puntero nulo.
- *Alcanzable*: Los métodos exportados implementarán rutinas de validación de punteros, fallando de forma segura ante la detección de un puntero nulo.

- *Relevante*: Dado que la aplicación hace uso de código no administrado a través de la librería HTTPCore la existencia de un fallo de corrupción de memoria supondría el cierre inesperado de la aplicación.

3.3 Matriz de trazabilidad.

CASOS DE USO	REQUISITOS FUNCIONALES																						
	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8	RF9	RF10	RF11	RF12	RF13	RF14	RF15	RF16	RF17	RF18	RF19	RF20	RF21	RF22	RF23
CU-1					☑					☑													
CU-2						☑					☑												
CU-3							☑																
CU-4				☑	☑				☑	☑			☑										
CU-5					☑				☑	☑			☑										
CU-6																							
CU-7																							
CU-8					☑					☑					☑								
CU-9														☑									
CU-10												☑											
CU-11			☑	☑												☑							
CU-12	☑	☑	☑	☑													☑						
CU-13																	☑						
CU-14																		☑					
CU-15																		☑					
CU-16																		☑					
CU-17																		☑					
CU-18																		☑					
CU-19	☑	☑	☑	☑														☑					
CU-20																			☑				
CU-21																			☑				
CU-22																			☑				
CU-23																			☑				
CU-24																				☑			
CU-25	☑	☑	☑	☑				☑													☑		
CU-26																							
CU-27																							☑
CU-28									☑													☑	
CU-29									☑													☑	

Figura 8: Matriz de trazabilidad

3. Gestión del proyecto.

La aplicación de una metodología de trabajo a la hora de ejecutar un proyecto es fundamental para evitar el fracaso del mismo, así como para garantizar que, dentro del alcance definido se cumplen los requisitos, teniendo en cuenta las restricciones temporales y de costes. La gestión de un proyecto de software permitirá asegurar tanto la calidad de los procesos como del producto que se va a desarrollar.

Los aspectos más relevantes en la gestión de un proyecto son: la selección de un metodología de desarrollo que permita adaptarse a las necesidades de la organización y del equipo de desarrollo con el fin de alcanzar los objetivos definidos, la gestión de riesgos del proyecto, la definición de una planificación temporal que permita organizar los recursos humanos involucrados en las distintas fases del ciclo de vida del proyecto y del producto para cumplir con las estimaciones de plazos y de costes, y la gestión de la configuración, que permitirá asegurar la integridad del producto y la gestión de cambios.

En este capítulo se abordan todos los temas asociados a la gestión de un proyecto.

3.1 Metodología de desarrollo.

Los procesos asociados al desarrollo del software requieren la definición de un entorno de trabajo disciplinado que permita estructurar, planificar y, en definitiva controlar el proceso de desarrollo del producto.

Mediante la adopción de un ciclo de vida de desarrollo del software se establece un enfoque sistemático de trabajo a través de una serie fases por las que se debe pasar durante la construcción de un sistema. La elección de una metodología de desarrollo adaptada al equipo de trabajo y las necesidades del proyecto es una de las decisiones más importantes ya que puede marcar la diferencia entre el éxito y el fracaso del mismo.

El ciclo de vida empleado para la ejecución de este proyecto ha sido el modelo incremental, en el cual el producto se desarrolla a través de un conjunto de incrementos que permiten obtener como resultado de cada uno de ellos un módulo funcional del software. La versión final del producto es la integración de los resultados sucesivos.

Una de las ventajas principales del ciclo de vida incremental es que es posible obtener una retroalimentación continua por parte del cliente tras el desarrollo de cada iteración. La elección de este ciclo de vida parte de la necesidad de disponer de versiones funcionales a medida que se va a desarrollando el proyecto, así como la flexibilidad que aporta al equipo de desarrollo, permitiendo afrontar una parte del problema en cada iteración.

El modelo de desarrollo incremental planteando en el proyecto sigue el esquema representando en el siguiente diagrama, en el que cada incremento representa el desarrollo de un subsistema del software.

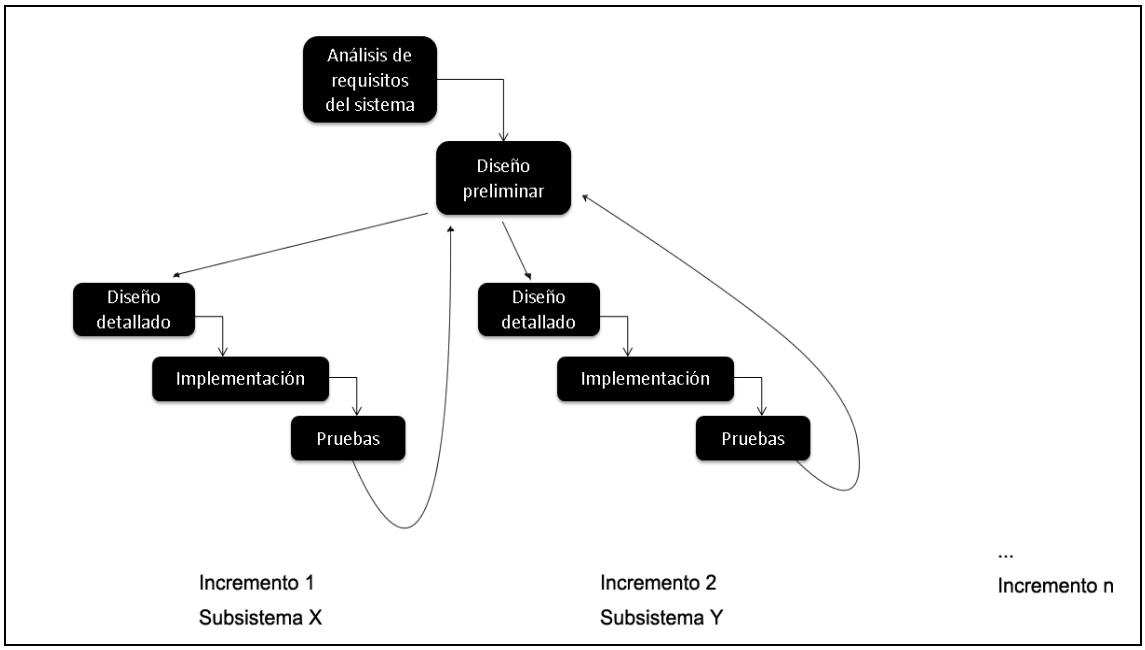


Figura 9: Ciclo de vida de desarrollo incremental

El desarrollo del proyecto se plantea como 5 iteraciones o incrementos, cada uno de ellos con un conjunto de requisitos asociados, cuya implementación tendrá como resultado una versión funcional del software.

3.1.1 Análisis del alcance de cada incremento.

La finalización de la iteración 1 permitirá disponer de un módulo funcional que permite la interacción con la API HTTPCore desde cualquier lenguaje de programación, permitiendo establecer comunicaciones a través del protocolo HTTP/s.

Iteración 1	
RF-1	El wrapper debe exportar la funcionalidad de la API que permite el inicio conexiones HTTP/s
RF-2	El wrapper debe exportar la funcionalidad que permite terminar conexiones HTTP/s establecidas previamente
RF-3	El wrapper debe exportar la funcionalidad que permite interactuar con los distintos campos de una petición HTTP.
RF-4	El wrapper debe exportar la funcionalidad que permite obtener las respuestas de las peticiones HTTP utilizando los métodos de la API.
RF-5	El wrapper debe exportar la funcionalidad que permite iniciar un proxy HTTP.

RF-6	El wrapper debe exportar la funcionalidad que permite detener un proxy HTTP establecido previamente.
RF-7	El wrapper debe exportar la funcionalidad que permite establecer las opciones de configuración del proxy HTTP.
RF-8	El wrapper debe exportar la funcionalidad que permite establecer las opciones de configuración del motor de comunicaciones.
RF-9	El wrapper debe exportar las funciones que permiten interactuar con las funcionalidades de callback definidas en la API.

Tabla 1: Alcance de la iteración 1.

La tabla 2 muestra el conjunto de requisitos que se llevarán a cabo en la iteración 2. La finalización de esta iteración permitirá disponer de una aplicación gráfica que implementa un proxy con capacidades de interceptación de peticiones HTTP.

Iteración 2	
RF-10	La aplicación debe permitir iniciar un proxy de interceptación en una dirección IP y puerto especificado por el usuario.
RF-11	La aplicación debe permitir detener un proxy HTTP iniciado previamente.
RF-12	La aplicación debe permitir el registro de las peticiones HTTP que se envían a través del proxy.
RF-13	La aplicación debe permitir la activación o desactivación del módulo de interceptación de peticiones HTTP/s.
RF-14	La aplicación debe permitir bloquear/rechazar las peticiones HTTP que han sido interceptadas por el proxy
RF-15	La aplicación debe permitir la redirección de peticiones HTTP enviadas a través del proxy hacia un sistema de destino.
RF-16	La aplicación debe permitir la modificación de las peticiones HTTP interceptadas.

Tabla 2: Alcance de la iteración 2.

Tras la finalización de la iteración 3 se dispondrá de un módulo funcional que permite la construcción y reenvío de peticiones HTTP definidas por el usuario.

Iteración 3	
RF-17	La aplicación debe disponer de un módulo que permita el reenvío de peticiones HTTP a un host objetivo y puerto concreto.

Tabla 3: Alcance de la iteración 3.

Tras el desarrollo de la iteración número 4 será posible realizar procedimientos iterativos de fuzzing sobre el protocolo HTTP.

Iteración 4	
RF-18	La aplicación debe permitir realizar procedimientos de fuzzing

	sobre peticiones HTTP.

Tabla 4: Alcance de la iteración 4.

La finalización de la iteración 5 permitirá la creación, edición, compilación y ejecución de scripts. Tras este último incremento se obtendrá una versión completa del sistema.

Iteración 5	
RF-19	La aplicación permitirá la creación y edición de scripts desarrollados en C#, que hacen uso de la API HTTPCore a través de un entorno de desarrollo integrado en la propia herramienta.
RF-20	La aplicación debe permitir la compilación de scripts
RF-21	La aplicación debe permitir la ejecución de los scripts compilados a través de la aplicación.
RF-22	La aplicación debe permitir la activación y desactivación de la ejecución de los scripts de callback.
RF-23	La aplicación debe disponer de un repositorio de scripts que permita el almacenamiento de los mismos.

Tabla 5: Alcance de la iteración 5.

3.2 Gestión de riesgos

Una tarea fundamental en la gestión de proyectos es, la identificación y análisis de riesgos que pueden afectar tanto al proyecto como al producto que se va a desarrollar, de forma que sea posible anticipar las posibles amenazas y emprender acciones correctivas que permitan mitigar su exposición.

La fase de gestión de riesgos comprende varias etapas:

1. Identificación de riesgos: Permite descubrir los riesgos que pueden afectar tanto al proyecto como al producto.
2. Análisis de riesgos: Esta etapa comprende la valoración de probabilidades y consecuencias de una posible materialización de los riesgos.
3. Planificación de riesgos: En esta fase se definen los planes de contingencia que permiten abordar la materialización de los riesgos.
4. Supervisión de los riesgos: Permite establecer un seguimiento de los riesgos y llevar a cabo una revisión de los planes de contingencia para la mitigación de los mismos.

La identificación de riesgos en las fases tempranas del ciclo de vida de desarrollo es crítica, ya que permitirán minimizar los problemas que estos puedan ocasionar en etapas más avanzadas del proyecto. En este apartado se analizan los riesgos y se propone un plan de acción para su minimización, así como un plan de contingencia que define una serie de contramedidas para aquellos riesgos que lleguen a materializarse

Los riesgos identificados se detallarán siguiendo la siguiente estructura.

Identificador del riesgo. RK-Número.

Nombre: Nombre del riesgo
Descripción: Descripción breve del riesgo.
Probabilidad: Probabilidad de aparición del riesgo:

Muy Bajo < 10%
Bajo (10-25%)
Moderado (25-50%)
Alto (50-75%)
Muy Alto (> 75%)

Efecto Permite categorizar la criticidad que puede suponer la materialización de un riesgo.

Se categorizan como: catastrófico, serio, tolerable e insignificante.

Medidas preventivas: Descripción de las acciones a llevar a cabo para prevenir la aparición del riesgo.
Medidas correctivas: Descripción del plan de contingencia ante la materialización del riesgo.

3.2.1 Riesgos de gestión del proyecto

RK-01

Nombre: Estimaciones temporales irrealistas.

Descripción: La falta de experiencia en la gestión de proyectos software, y la envergadura del proyecto, dificultan llevar a cabo una estimación temporal precisa.

Probabilidad: Alta

Efecto Catastrófico, el incumplimiento de fechas de entrega puede suponer el fracaso del proyecto.

Medidas preventivas: Realizar un sobreestimación de las tareas del proyecto.

Medidas correctivas: Incrementar la carga de trabajo con el objetivo de evitar posibles retrasos como consecuencia de la posible materialización de un riesgo o de una mala planificación.

RK-02

Nombre: Identificación de requisitos incorrecta.

Descripción: Una identificación de requisitos incorrecta supone volver a realizar un análisis de los mismos, lo que provoca retrasos en la planificación.

Probabilidad: Media

Efecto Serio

Medidas preventivas: Mayor dedicación en la fase de análisis de requisitos.

Medidas correctivas: Analizar el estado del sistema y cómo afecta al estado actual del desarrollo del software. Llevar a cabo una re planificación del proyecto en base a los resultados obtenidos.

RK-03

Nombre: Alcance del proyecto inadecuado.

Descripción: El proyecto dispone de demasiadas funcionalidades con respecto al tiempo planteado para la ejecución del mismo.

Probabilidad: Media

Efecto Serio

Medidas preventivas: Es necesario delimitar el alcance del proyecto y llevar a cabo la identificación de aquellas funcionalidades opcionales.

Medidas correctivas: Rechazar aquellas funcionalidades que no sean críticas para el funcionamiento del software.

3.2.2 Riesgos de desarrollo

RK-04

Nombre: Pérdida de datos.

Descripción: El fallo del dispositivo de almacenamiento en el que se encuentra el repositorio de código y de la documentación supone tener que volver a realizar el trabajo.

Probabilidad: Baja

Efecto Catastrófico.

Medidas preventivas: Implementar un procedimiento para la realización de copias de seguridad.
Se debe almacenar el código y la documentación en un servidor independiente del equipo de desarrollo. Adicionalmente se sincronizará el repositorio con un servicio de almacenamiento en la nube como puede ser Dropbox.

Medidas correctivas: En el caso de materializarse el riesgo se deberán recuperar los datos del servidor de respaldo o del servicio de almacenamiento en la nube.

RK-05

Nombre: Identificación de fallos en la API HTTPCore.

Descripción: El descubrimiento de fallos en las funcionalidades de la API HTTPCore supone un retraso en la planificación, ya que es necesario dedicar tiempo a la corrección de los fallos y a la ejecución de tareas de depuración.

Probabilidad: Alta

Efecto Serio

Medidas preventivas: N/A

Medidas correctivas: Se debe asumir el riesgo. Se deberña realizar una re planificación del proyecto.

RK-06

Nombre: Fallos de diseño.

Descripción: Un mal planteamiento del diseño de la aplicación supone el replanteamiento del mismo y consecuentemente un retraso en la planificación.

Probabilidad: Media

Efecto Serio

Medidas preventivas: Dedicar un mayor esfuerzo a la fase de diseño del proyecto.

Medidas correctivas: Analizar cómo afecta al proyecto y realizar una re planificación adecuada.

3.3 Estructura de descomposición de trabajo.

Una estructura de descomposición de trabajo o WBS (Work Breakdown Structure) (14) es una descomposición jerárquica de los entregables del proyecto en componentes más pequeños y por lo tanto más fáciles de manejar. La EDT permite definir y organizar el alcance del proyecto basándose en los entregables de trabajo que deben ser ejecutados por parte del equipo de desarrollo con el fin de alcanzar los objetivos del proyecto.

El modelo jerárquico de la EDT permite a través de una estrategia top-down (de arriba abajo) llevar a cabo una descomposición de los entregables del proyecto siguiendo una estructura en varios niveles, donde cada nivel descendiente representa una definición cada vez más detallada del trabajo a realizar hasta que los entregables quedan definidos a nivel de paquetes de trabajo, donde la duración de las actividades y el coste de las mismas pueden estimarse de forma más fiable.

1. *Fase de análisis preliminar*

En esta primera fase se lleva a cabo la definición de objetivos del proyecto, la definición de los casos de uso y la especificación de requisitos, así como la identificación de los principales riesgos que pueden afectar al proyecto.

2. *Planificación*

La fase de planificación contempla todas aquellas tareas asociadas a la gestión y planificación del proyecto.

3. *Formación*

La fase de formación contempla todas aquellas tareas que requieren el estudio de las tecnologías, metodologías, así como el análisis de alternativas para la implementación del producto software que se va a desarrollar.

4. *Diseño*

Esta fase se lleva a cabo de un diseño de alto nivel que permite identificar la arquitectura del sistema y el modelo de interacción entre los componentes, con el objetivo de determinar cómo será la comunicación entre los distintos módulos software contemplados en el proyecto.

5. Ejecución del proyecto

La fase de ejecución concentra todas aquellas tareas asociadas a la construcción del software, siguiendo el modelo de ciclo de vida incremental planteado. En esta fase se llevan a cabo las tareas de análisis, diseño, implementación y pruebas para cada uno de los incrementos contemplados. Esta fase del proyecto es la que más dedicación requiere.

6. Documentación

La fase de documentación se lleva a cabo durante la realización de todo el proyecto y dispone de tareas de elaboración de la memoria y corrección de la misma.

A continuación se muestra un diagrama que representa la EDT del proyecto:

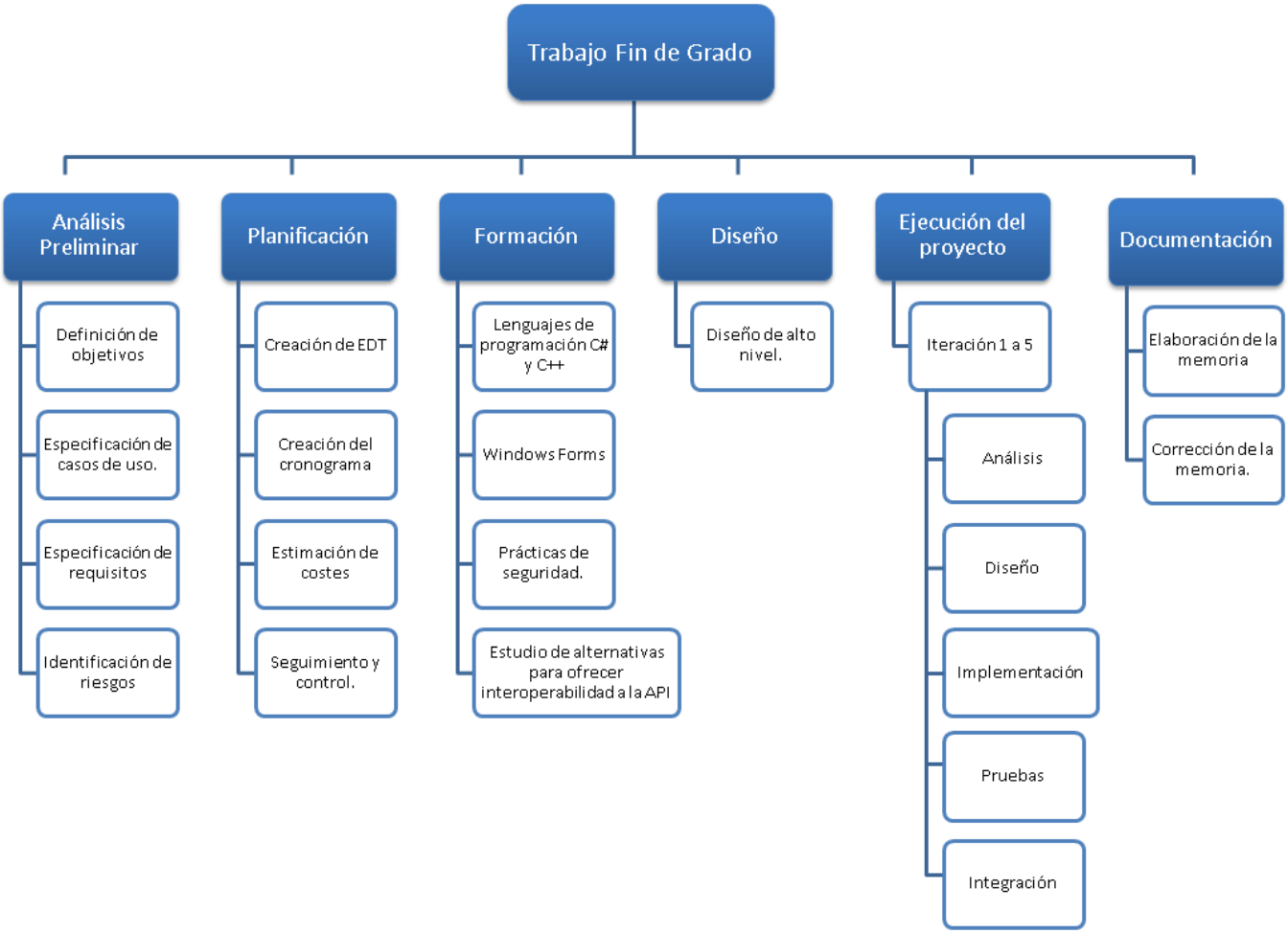


Figura 10: Estructura de descomposición de trabajo del proyecto.

3.4 Planificación temporal.

En esta sección se detalla la estimación temporal planteada inicialmente para las distintas tareas que conforman el proyecto. El calendario laboral propuesto está definido en base a jornadas de 5.5 horas de lunes a viernes, lo que supone un total de 30 horas semanales y una estimación total del proyecto de 460 horas. La fecha de inicio del proyecto es el día 08/05/2013 y la fecha de fin estimada el día 02/09/2013.

Los recursos del proyecto están representados por los roles: analista/programador y jefe de proyecto. El analista/programador, participa en todas aquellas tareas que requieren una perspectiva técnica. El rol jefe de proyecto participa en las tareas de gestión del proyecto. Como aclaración el tiempo dedicado a llevar a cabo reuniones con el cliente están incluidas dentro de la estimación temporal de aquellas tareas que así lo requieren.

A continuación se presentan los detalles de la estimación temporal a través de un Diagrama de Gantt que permite mostrar el tiempo de dedicación previsto para cada una de las tareas del proyecto.

El diagrama de Gantt se divide en un total de 6 fases diferenciadas como se documentó en la estructura de descomposición de trabajo.

La siguiente Figura representa la estimación temporal de las primeras 3 fases del proyecto: Análisis preliminar, planificación y formación. La tarea de seguimiento y control tiene una duración de 77 jornadas, ya que tiene como duración el tiempo de ejecución del proyecto.

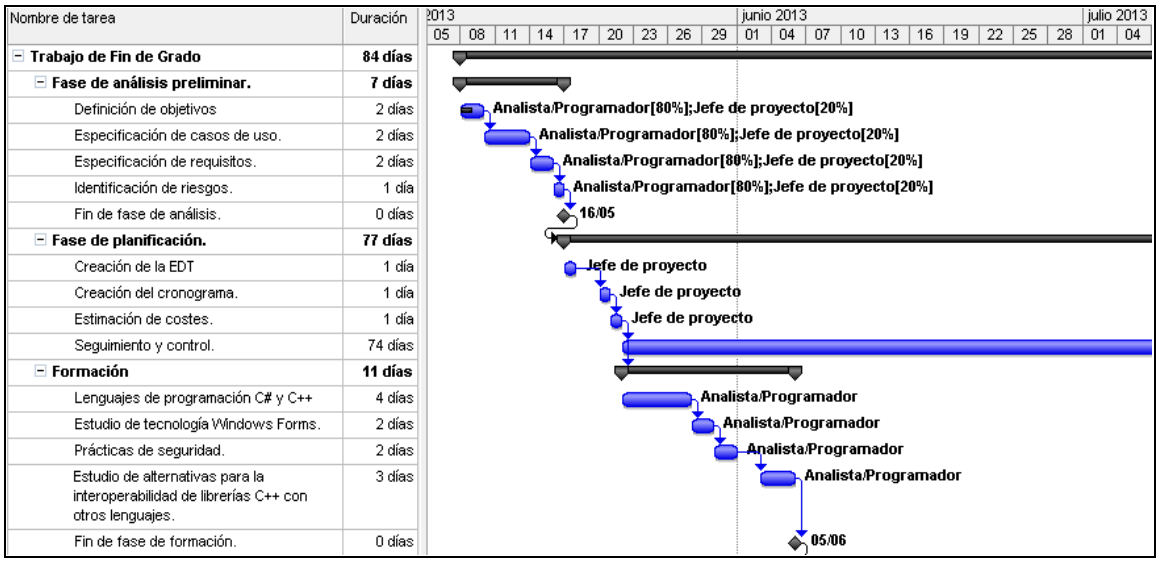


Figura 11: Diagrama de Gantt que representa la estimación temporal del proyecto (parte 1).

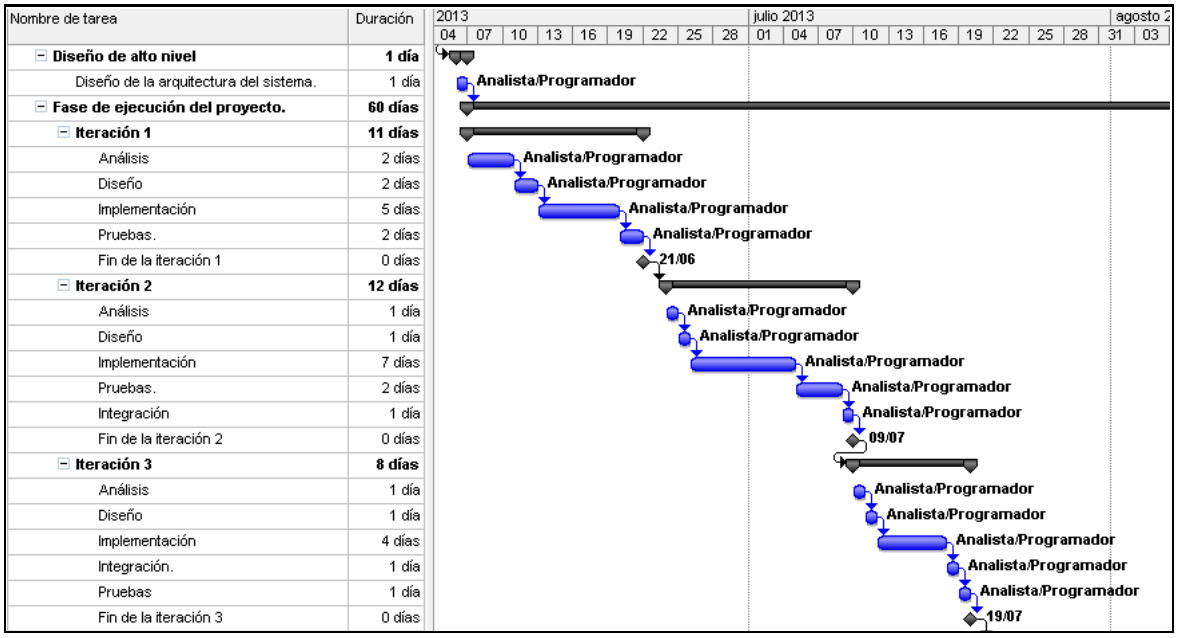


Figura 12: Diagrama de Gantt que representa la estimación temporal del proyecto (parte 2).

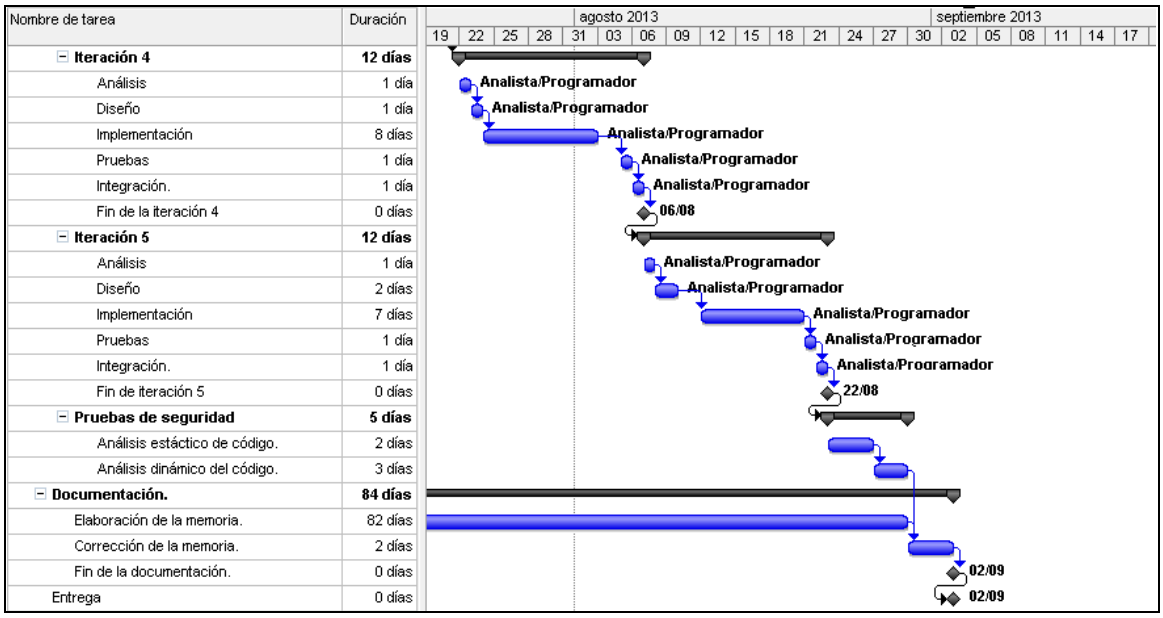


Figura 13: Diagrama de Gantt que representa la estimación temporal del proyecto (parte 3).

3.5 Seguimiento

En la ejecución del proyecto el riesgo RK-05 (Identificación de fallos en la API HTTPCore.) llegó a materializarse durante la fase de pruebas de la iteración 1, suponiendo un retraso de 3 jornadas sobre la planificación inicial, ya que fue necesario analizar los fallos del software y llevar a cabo tareas de depuración. La medida correctiva propuesta para dicho riesgo contemplaba la re planificación del proyecto y supuso la reducción del tiempo dedicado a la realización de las pruebas de seguridad de análisis estático y análisis dinámico.

Tras realizar las modificaciones que permiten adaptar la planificación a los cambios, la fecha de estimación final se sigue manteniendo. El diagrama de Gantt resultante tras la realización de las modificaciones sobre la planificación inicial se muestra a continuación.

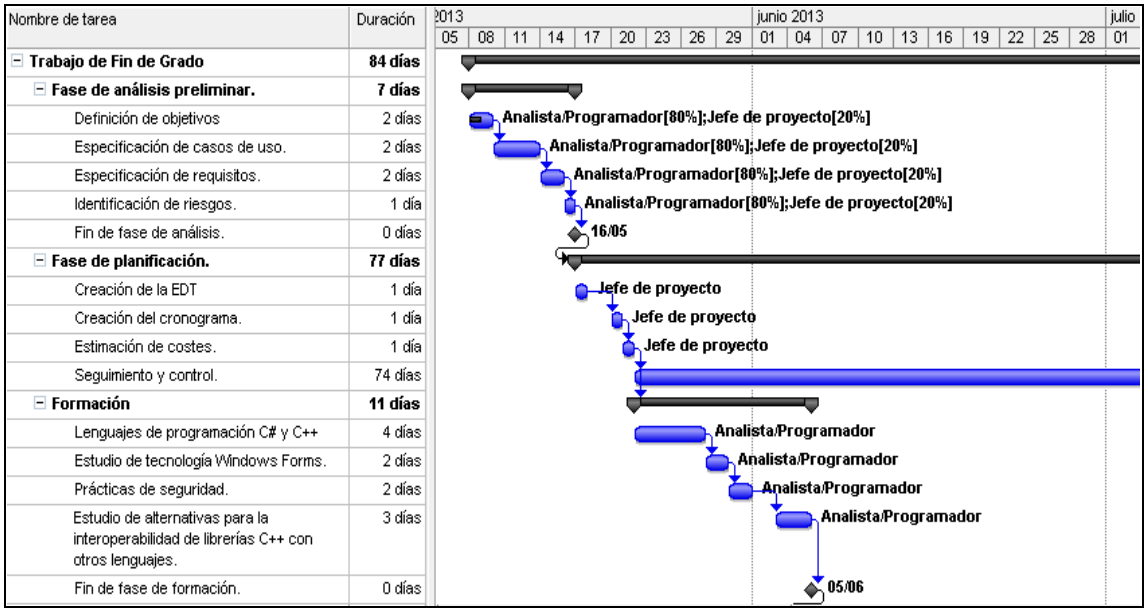


Figura 14: Diagrama de Gantt que representa la estimación temporal tras la re planificación (Parte 1).

Como se puede observar en las siguientes Figuras el riesgo RK-05 tuvo lugar en la tarea de pruebas de la iteración 1. Como medida de re planificación se limitó el tiempo de duración de las tareas de análisis estático y análisis dinámico del código, con el objetivo de mantener la fecha de terminación del proyecto según la planificación inicial.

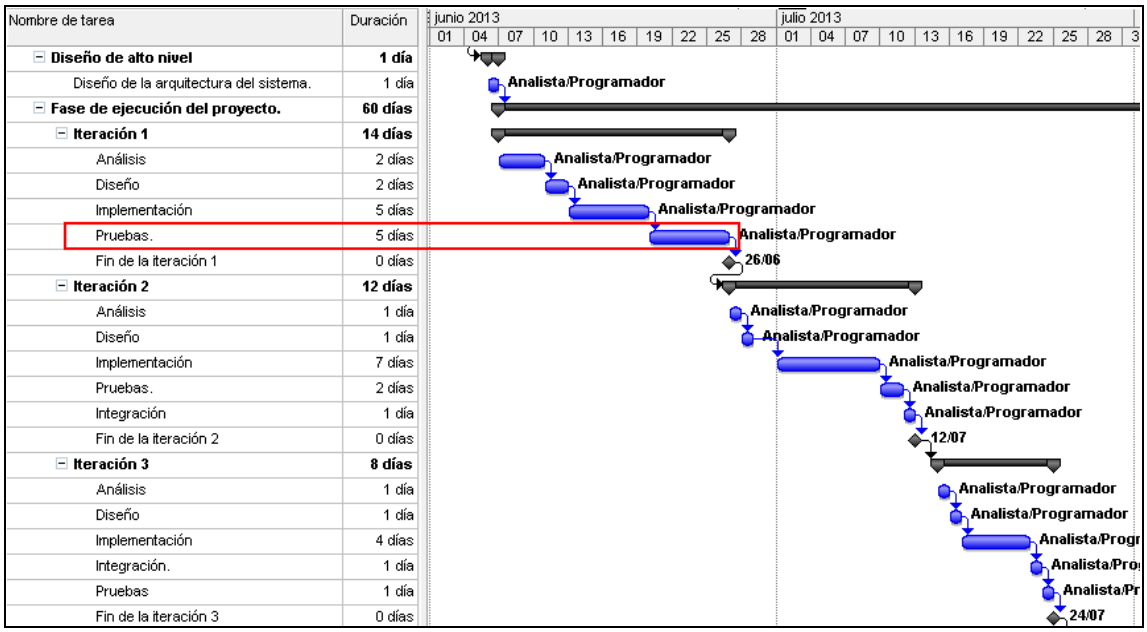


Figura 15: Diagrama de Gantt que presenta la estimación temporal tras la re planificación (Parte 2).

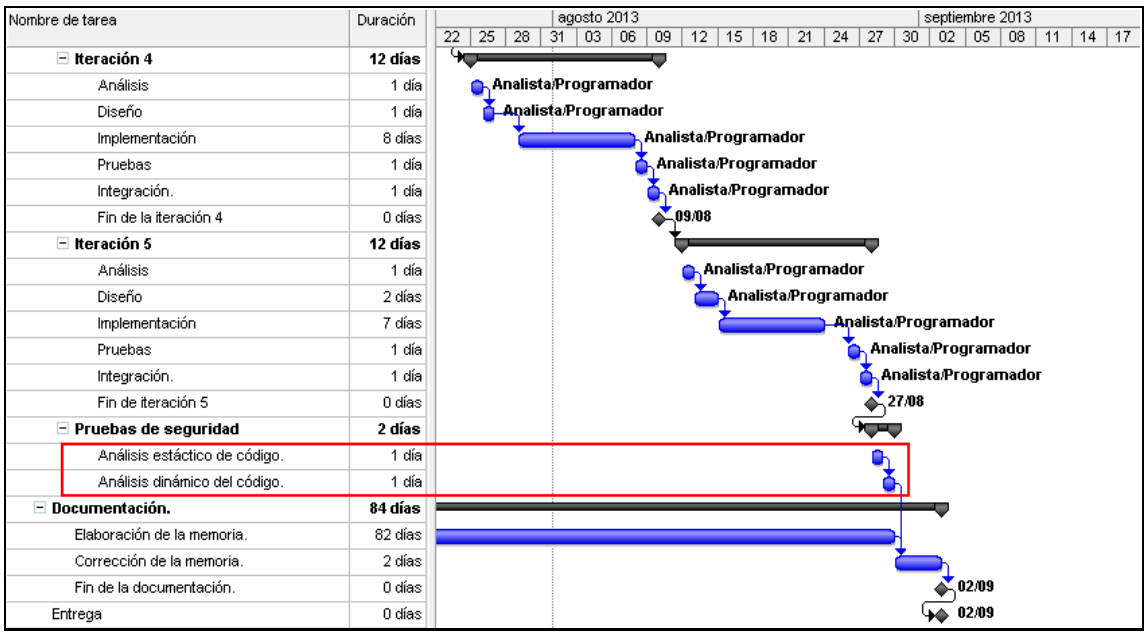


Figura 16: Diagrama de Gantt que representa la estimación temporal tras la re planificación (Parte 3).

3.6 Gestión de la configuración

La gestión de la configuración es un proceso cuyo propósito es el de establecer y mantener la integridad de cualquier elemento de trabajo asociado a cualquier etapa del ciclo de vida del proyecto. El objetivo principal de este proceso es del asegurar la integridad de los productos y servicios desarrollados mediante la definición de un procedimiento de gestión de cambios.

Para gestionar el control de versiones tanto del código fuente de los distintos componentes software, así como de toda la documentación generada se hará uso de un servidor de control de versiones basado en Git. Con el objetivo de evitar cualquier pérdida de algún elemento de configuración el repositorio estará sincronizado con la plataforma de almacenamiento en la nube Dropbox.

3.7 Gestión de costes.

En esta sección se analiza los costes del proyecto con el objetivo de determinar el coste total que supone la ejecución del proyecto. Los costes del proyecto analizados contemplan tanto los costes derivados de los recursos humanos que participan en el proyecto, así como de aquellos elementos hardware y software necesarios para la ejecución del mismo

Costes derivados de recursos humanos

Elemento	Cantidad	Precio unitario	Coste Total
Coste de personal	460 horas	11,50 €	5.290 €

Coste derivado de elementos hardware/software

Elemento	Cantidad	Precio unitario	Coste Total
Portátil Dell Vostro 3360	1	639 €	639 €
Licencia Visual Studio 2010 Professional	1	615 €	615 €

3.7.1 Coste total

Se ha contemplado un 20% de gastos adicionales asociados a gastos indirectos generados durante la ejecución del proyecto,

Concepto	Coste Total
Costes derivados de los recursos humanos	5.290 €
Costes derivados de elementos hardware/software	1.254 €
Subtotal	**6.544 €**
Costes indirectos	+20% (1.308,8 €)
Total	**7.852,8 €**

4. Arquitectura, tecnologías y herramientas.

En este capítulo se detalla la arquitectura global de la aplicación desde distintas perspectivas, así como las tecnologías empleadas para el desarrollo de los distintos componentes que integran la API *HTTPCore* y la aplicación final desarrollada en base a esta.

En primer lugar se analizan la problemática inicial y se comentan las distintas alternativas que permitirán ofrecer la interoperabilidad de la API con cualquier lenguaje de programación.

4.1 Problemática inicial.

Como ya se comentó en la introducción, la API HTTP permite la gestión de comunicaciones HTTP/s, ofreciendo la capacidad de enviar peticiones y recibir las respuestas asociadas a éstas, así como la posibilidad de crear de un servidor proxy HTTP a través del cual es posible llevar a cabo la interceptación de las comunicaciones. La API está desarrollada en C++ desde una perspectiva orientada a objetos y puede ser compilada tanto en entornos Windows como en entornos Linux.

Dado que, para llevar a cabo la invocación de la API, ésta debe ser compilada como un ejecutable las alternativas identificadas para permitir ofrecer su interoperabilidad son las siguientes

ALTERNATIVA 1

Arquitectura multiproceso cliente-servidor, en la cual, tanto la librería como la aplicación que haga uso de la misma se implemente como dos procesos independientes. La comunicación entre ellos se establece través de un mecanismo IPC (Inter-process communication) (15), o a través de tecnologías como CORBA (Common object Request Broker Architecture) (16)o RPC (Remote Procedure Call).

Figura 17: Comunicación entre la aplicación y la librería a través de un arquitectura cliente servidor

Ventajas:

- **Tratamiento de errores**: Al mantener espacios de memoria separados, un fallo en la gestión de memoria de la API HTTPCore no afectaría al espacio de memoria de la aplicación, de forma que ésta podría generar una excepción ante un evento inesperado sin suponer la pérdida de datos.

Inconvenientes:

- **Complejidad de la solución:** Para la implementación de esta solución es necesario definir una arquitectura cliente servidor y emplear un mecanismo de comunicación IPC. Dado que los métodos de comunicación IPC son dependiente del sistema operativo, sería necesario re-implementar la solución si se desea ofrecer soporte multiplataforma. La implementación de esta solución requiere además definir una interfaz de definición de lenguaje (IDL) en la cual se especifique el tipo y formato de los datos utilizados en la invocación de las funcionalidades, así como en la recuperación de los valores de retorno.

  El esfuerzo de desarrollo del esquema de comunicación de otros lenguajes de programación es mayor con respecto a la *ALTERNATIVA 2*.

- **Dificultad de depuración:** Al emplear un esquema multiproceso la depuración de errores es más complicada y tediosa, ya que es necesario realizar la depuración de forma separada para cada proceso.

- **Rendimiento:** El rendimiento de la solución es dependiente del mecanismo IPC empleado.

ALTERNATIVA 2

Exportar la API HTTPCore como una biblioteca compartida a través de la cual es posible invocar los métodos expuestos a través de esta.

*Ventajas***:**

- **Sencillez de la solución:** La implementación de la solución requiere únicamente exportar los métodos públicos a través de los cuales se interactuará con la API, así como definir una especificación ABI (Application Binary Interface) (17) que defina las funciones exportadas, el tamaño y formato de los tipos de datos que

son pasados como argumentos a cada función, así como la convención de llamadas (Calling Convention) definida durante la compilación de la DLL.

- **Gestión de cambios:** La gestión de los cambios es sencilla, cualquier modificación en las funcionalidades internas de la biblioteca dinámica, solo requeriría el reemplazo de la misma.

- **Rendimiento:** La aplicación que invoca los métodos de la biblioteca compartida no necesita implementar ningún tipo de mecanismo de gestión de la comunicación a través de un mecanismo IPC, por lo tanto el rendimiento de esta solución es superior a la planteada en la ALTERNATIVA 1. La implementación de esta alternativa permite además, reducir el tamaño de los ficheros ejecutables y ahorrar espacio en memoria ya que las funciones de la DLL se cargan bajo demanda y solo se mapean una única vez en memoria.

Inconvenientes:

- **Tratamiento de errores:** El tratamiento de errores en el caso de fallos de corrupción de memoria en la biblioteca compartida puede suponer la pérdida de datos de la aplicación si no se gestionan correctamente las excepciones.

Conclusión:

Se ha optado por la *ALTERNATIVA 2* para implementar la interoperabilidad de la API HTTPCore, los factores de sencillez de implementación y rendimiento han sido decisivos en su elección.

Los detalles de implementación de esta solución se describen en el **Capítulo 6**.

4.2 Arquitectura del sistema.

Una vez analizadas las alternativas de implementación de la solución de adaptación de la API para su integración con múltiples lenguajes de programación y definidos los casos de uso y requisitos del proyecto se procede a definir la arquitectura global del sistema.

Con el objetivo de aportar una mayor claridad en la descripción de la arquitectura del sistema, se presentan distintas perspectivas de la misma. En primer lugar se representará la arquitectura funcional de la aplicación integrando la biblioteca de vínculo dinámico.

4.2.1 Esquema funcional de la aplicación.

Desde una perspectiva funcional se diferencian dos secciones fundamentales: Las funcionalidades expuestas por la API HTTPCore (representadas en azul) y las funcionalidades implementadas por la aplicación a través de distintos módulos (verde).

El siguiente esquema representa la arquitectura de la aplicación según los requisitos identificados durante la fase de análisis.

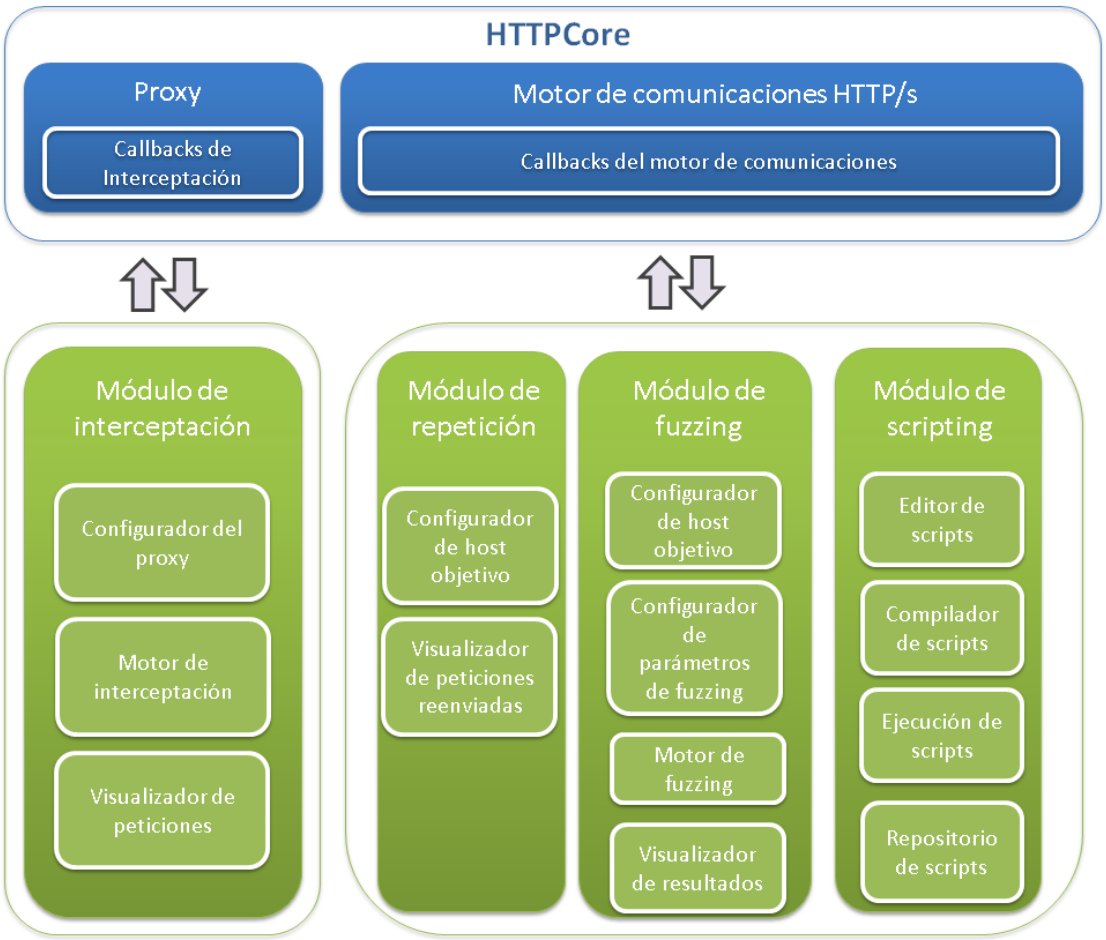


Figura 18: Esquema funcional del sistema.

Las funcionalidades de la API, empleadas por los distintos subsistemas de la aplicación se categorizan en dos módulos. El *proxy* y el *motor de comunicaciones HTTP/s*.

- El proxy permitirá la interceptación de peticiones y respuestas HTTP/s enviadas por parte de una aplicación externa como puede ser una navegador web conectado al proxy.

- El motor de comunicaciones representa aquellas funcionalidades de la API que permiten gestionar las comunicaciones HTTP (Envío y recepción de peticiones) con independencia del proxy.

Los distintos módulos representados en verde definen aquellos subsistemas de la aplicación.

- *Módulo de interceptación*: El objetivo de este módulo es el capturar las peticiones HTTP que pasan a través del proxy. La recepción de una petición HTTP en el proxy generará un evento que será capturado por la aplicación, permitiendo visualizar los datos de la petición interceptada, así como llevar a cabo la modificación de la misma y decidir si la petición debe ser redirigida al sistema destino o bloqueada.

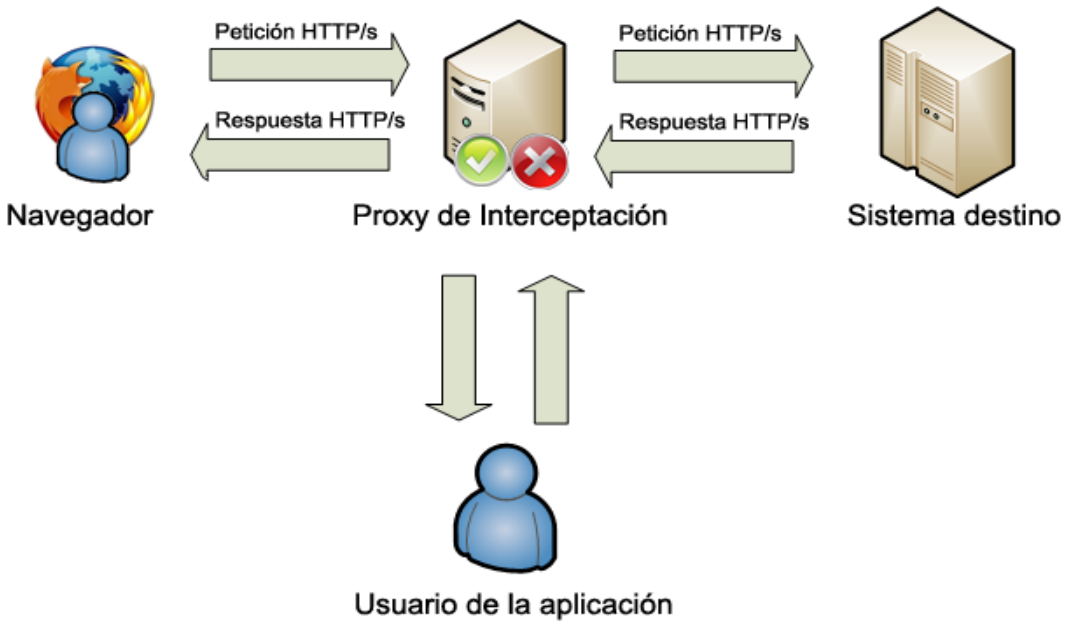


Figura 19: Arquitectura de comunicación del módulo de interceptación.

1. Un navegador establece una conexión con el proxy HTTP.
2. El navegador envía una petición HTTP a través del proxy.

3. Ante la recepción y procesamiento de una petición HTTP en el proxy (a través de los callbacks) se genera un evento que notificará a la aplicación la recepción de una petición.
4. La aplicación recoge los datos de la petición HTTP interceptada, de forma que estos puedan ser mostrados al usuario de la aplicación.
5. Si la interceptación se encuentra activada el usuario de la aplicación puede decidir si la petición debe ser bloqueada o redirigida al sistema destino.
   Si la interceptación está desactivada al petición se redirige de forma automática al sistema destino.
6. Si la petición es bloqueada se descarta y no se produce el envío de la misma. Si se permite la redirección se produce el envío de la petición.
7. La respuesta correspondiente es devuelta al proxy, y es procesada (a través de los callbacks de recepción, los cuales informan su recepción.

- *Módulo de repetición*: El objetivo de este módulo es permitir la construcción y el envío y recepción de peticiones HTTP de forma independiente.

- *Módulo de fuzzing*: Este módulo permite realizar procedimientos iterativos de fuzzing contra un sistema objetivo, reemplazando una parte seleccionada de una petición por un *payload* configurado por el usuario. El payload define el valor que reemplazará a la parte original seleccionada.

- *Módulo de scripting*: El objetivo de este módulo es proveer a la aplicación de un entorno de desarrollo de scripts. Permitiendo a través de éste compilar y ejecutar scripts.

5.3.2 Esquema de interacción entre componentes.

El siguiente diagrama representa el esquema de interacción de la aplicación desde una perspectiva de componentes permitiendo visualizar como es la interacción entre los distintos módulos, siguiendo el planteamiento definido en la *ALTERNATIVA2*

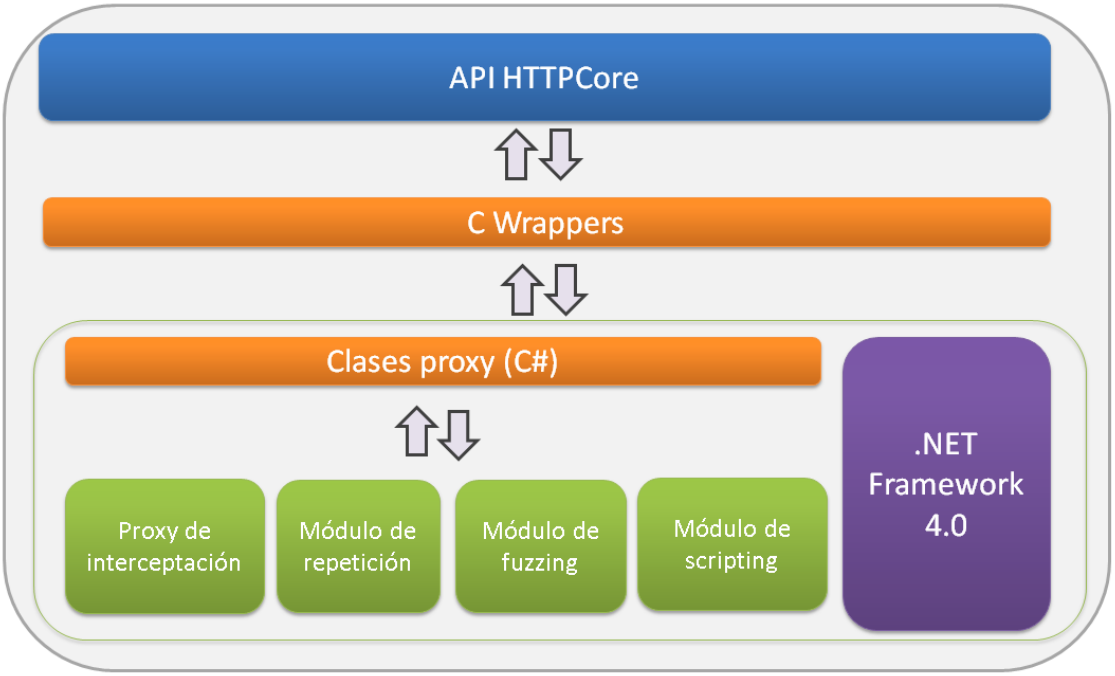


Figura 20: **Esquema de interacción entre componentes.**

El diagrama representa el modelo de interacción entre los elementos de la aplicación (representados en verde). Para permitir la interacción con la biblioteca dinámica se presenta un esquema multicapa:

El elemento "C Wrappers" exporta los métodos de la librería HTTPCore como funciones básicas de C.

El elemento "Clases proxy (C#)" permite reconstruir en el lenguaje C# la perspectiva orientada a objetos original de la API (ya que las funcionalidades se exportan como funciones básicas de C. Su objetivo principal proveer una capa de abstracción que permita ocultar los detalles de acceso a la biblioteca compartida.

Tanto los subsistemas de la aplicación como las clases proxy hacen uso de las funcionalidades del framework .NET para implementación de sus responsabilidades.

El siguiente diagrama representa como es la interacción entre elementos ejecutables.

Se distinguen tres elementos fundamentales:

- La biblioteca compartida *HTTPCore.dll* contiene la lógica de la librería HTTPCore, y expone los métodos que se desean exportar.
- La biblioteca *HTTPCoreCSharp.dll* que contiene las clases proxy desarrolladas en C#, que permiten reconstruir la perspectiva orientada a objetos de la API, así como abstraer los detalles de acceso a la misma
- La aplicación *FHScan.exe* que implementa las funcionalidades definidas en los requisitos funcionales y que hace uso de las funcionalidades de la API a través de las clases proxy.

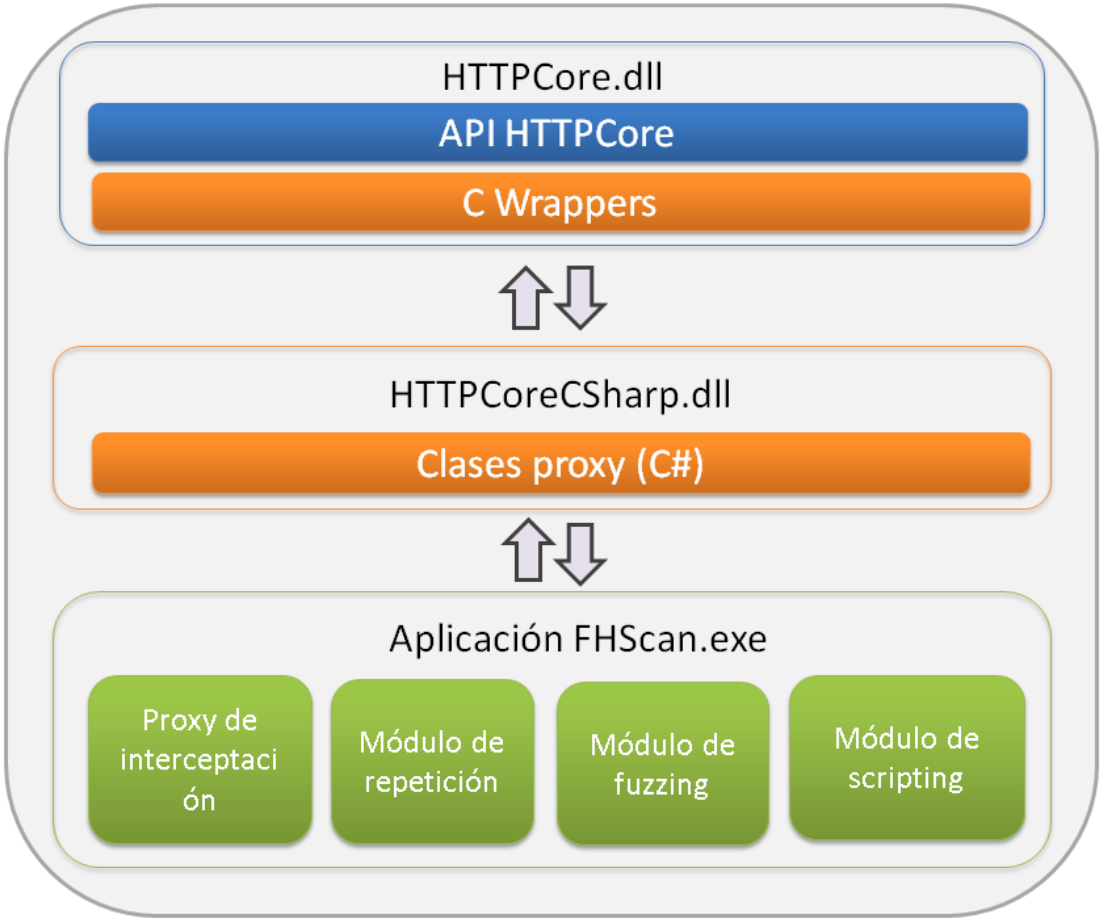


Figura 21: Representación de la interacción entre los distintos componentes ejecutables de la aplicación.

5.3.3 Tecnologías empleadas

5.3.3.1 Lenguajes de programación.

Lenguaje C

C (18) es un lenguaje de programación creado por *Dennis M. Ritchie* en los Laboratorios Bell. Es un lenguaje orientado originalmente al desarrollo de sistemas operativos, ya que a pesar de que es un lenguaje de alto nivel permite realizar operaciones a bajo nivel en el sistema.

En la realización del proyecto se ha utilizado el lenguaje C para la implementación de los wrappers que permiten exportar las funcionalidades de la API HTTPCore a través de una librería dinámica.

Lenguaje C#

C# es un lenguaje orientado a objetos de propósito general desarrollado por parte de Microsoft como parte del framework .NET. La ejecución de las aplicaciones desarrolladas en C# requiere de la utilización de un entorno de ejecución que dispone de un compilador JIT (Just in time) para la compilación del código intermedio independiente de la plataforma (*generado tras la compilación del código fuente*) a código nativo. A pesar de que originalmente la ejecución de programas desarrollados en C# estaba limitada para entornos Windows el framework Mono Develop permite la creación y ejecución de aplicaciones escritas en C# sobre distintas plataformas como puede ser Windows, GNU/Linux, Android, IOs y Mac OS X.

En la realización del proyecto se ha utilizado el lenguaje C# para:

- El desarrollo de las clases proxy que reconstruyen las funcionalidades de la API desde una perspectiva orientada a objetos.
- El desarrollo de la aplicación de auditoria FHScan.

Framework .NET

.NET es una plataforma de desarrollo creada por Microsoft para la creación de aplicaciones en entornos Windows. Proporciona soporte para una gran cantidad de lenguajes de programación como por ejemplo: C#, Visual Basic, Powershell, F#...

Los principales componentes del framework son:

- El entorno de ejecución CLR (*Common Language Runtime*) para lenguajes que cumplen la especificación CIL (*Common Intermediate Language*).
- Bibliotecas de clases base BCL, que pueden ser empleadas en la creación de aplicaciones.

Una de las características destacables del framework y que ha sido utilizada en el proyecto es el soporte de interoperabilidad de lenguajes a través de la tecnología P/Invoke (*Platform Invocation Services*) (19) que permite al código administrado (aquel ejecutado por la máquina virtual CLR) realizar llamadas a código nativo mediante la invocación de métodos definidos en librerías de enlace dinámico.

A continuación se muestra un esquema de la arquitectura del framework .NET en la versión 4.0 empleada en el desarrollo del proyecto.

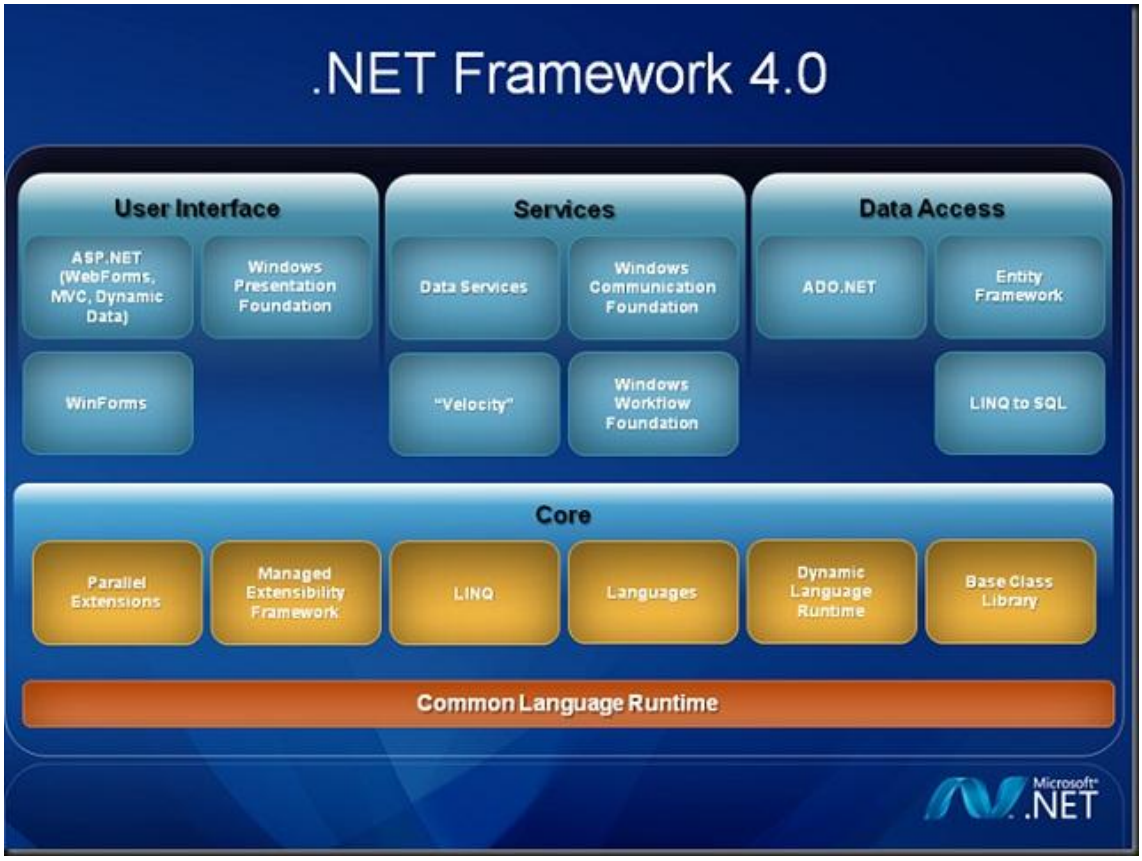


Figura 22: Diagrama de arquitectura del framework .NET 4.0

5.3.3.2 Herramientas de desarrollo

Microsoft Visual Studio 2010

Visual Studio 2010 es un entorno de desarrollo para sistema Windows con soporte para distintos lenguajes de programación como por ejemplo C#,C++,F# y Visual Basic.

Visual Studio dispone de una gran cantidad de funcionalidades que permite el desarrollo de proyectos de software desde las fases más tempranas del ciclo de vida de desarrollo del software, al permitir la creación de diagramas de casos de uso, diagramas clases, diagramas de actividad y diagramas de secuencia.

Se ha utilizado este entorno de desarrollo para la implementación de todo el código fuente del proyecto, así como para la creación de los diagramas de casos de uso, de clases y de secuencia.

5. Diseño

En este capítulo se documenta la fase de diseño de la aplicación. Se comenzará describiendo el diseño desde un nivel de abstracción alto y se irá avanzando hasta un nivel más bajo en el que se detallan los distintos elementos del sistema. A lo largo del capítulo se analizará por separado el diseño para cada una de las iteraciones planteadas en el ciclo de vida de desarrollo del software.

5.1 Modelo estructural del sistema.

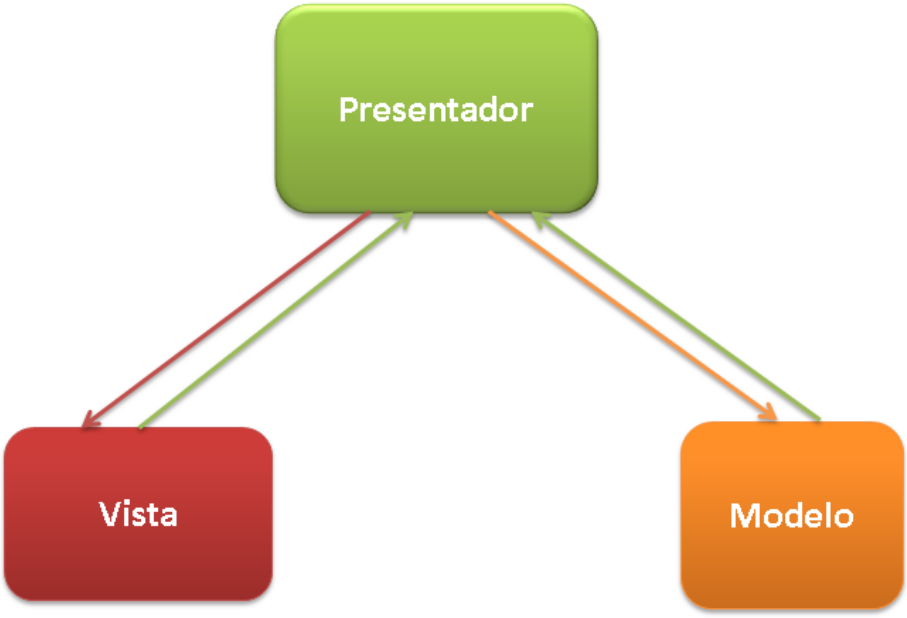
En este apartado se describe la arquitectura base para el desarrollo de la aplicación.

La arquitectura está organizada en distintas capas siguiendo el patrón de comportamiento/arquitectura MVP (Modelo Vista Presentador) (20), el cual permite llevar a cabo la separación de responsabilidades entre lógica de negocio y lógica de presentación. El Modelo Vista Presentador deriva del patrón MVC (Modelo Vista Controlador) (21) planteado originalmente por el arquitecto de software Martin Fowler.

5.2 Modelo Vista Presentador

El principio del patrón de diseño MVP consiste separar la aplicación en tres componentes fundamentales:

- **Modelo**: Componente que encapsula la lógica de negocio y el modelo de datos de la aplicación.
- **Vista**: Este componente representa la parte gráfica de la aplicación, implementada con cualquier tecnología que permita el desarrollo de interfaces gráficas.
- **Presentador:** Componente responsable de orquestar los distintos casos de uso de la aplicación asumiendo la responsabilidad de implementar la comunicación entre el modelo y la vista. La lógica de presentación se incluye en este componente.



Uno de los aspectos a destacar del patrón MVP es el hecho de que el presentador contiene la lógica de presentación. Los detalles de la vista se abstraen a través de una interfaz de programación, de forma que el presentador se comunica a través de una capa de abstracción de la vista, lo que convierte el presentador en un componente reutilizable, sobre el cual es posible llevar a cabo la verificación de las pruebas funcionales de la aplicación con independencia de la parte gráfica. Una ventaja de este esquema es la capacidad de reemplazar la tecnología gráfica empleada para la implementación de la interfaz gráfica, ya que no existe dependencia con el resto de componentes, el único requisito sería la implementación de la interfaz de programación que define la Vista a través de la cual interactúa el presentador.

Resumen de ventajas de implementación del patrón MVP:

- Separación de responsabilidades, aumentando la cohesión entre componentes y disminuyendo el acoplamiento de los mismos, al aislar la interfaz gráfica de la capa de negocio. Consecuentemente se mejora el mantenimiento de la aplicación por parte del equipo de desarrollo.
- Flexibiliza la realización de pruebas sobre la aplicación al separar la lógica de la aplicación de las vistas.
- Permite el intercambio de vistas.
- Mejora la productividad y eficiencia del proceso de desarrollo al disponer de elementos divisibles e independientes.

A continuación se muestra el modelo estructural de la aplicación empleando el patrón de diseño Modelo Vista Presentador. Se ha optado por llevar a cabo la separación de lógica de negocio de la aplicación (*HelperModel*) del modelo de datos (*DataModel*), de forma que ambos elementos sean reutilizables.

El siguiente diagrama muestra la interacción entre los distintos componentes. El esquema parte de la generación de un evento desde la interfaz gráfica.

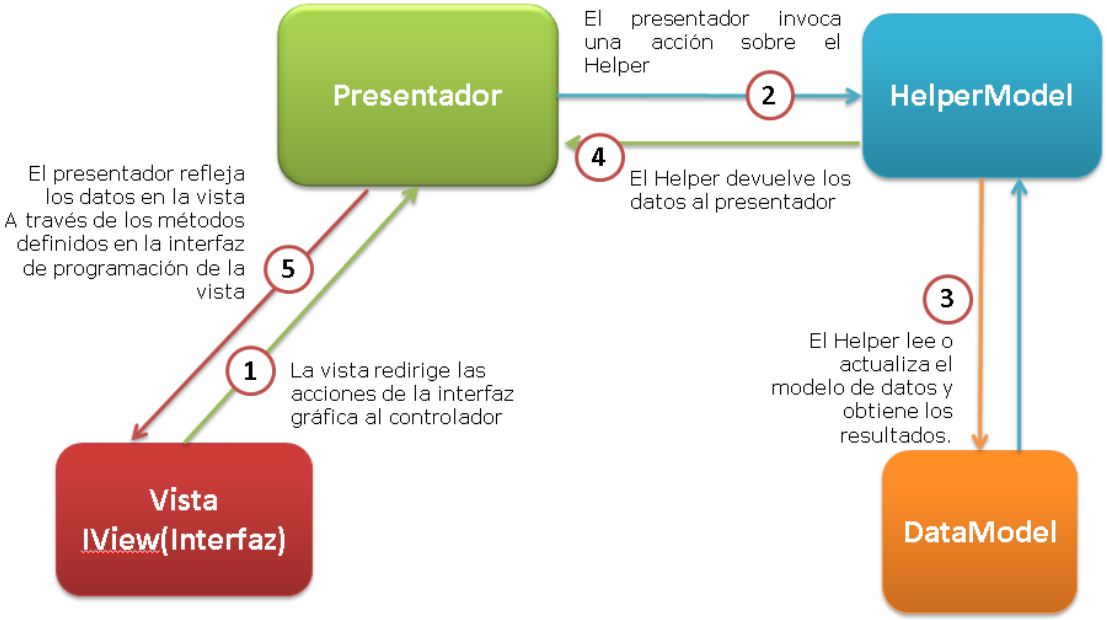


Figura 23: Interacción entre los distintos componentes de la implementación del patrón Modelo Vista Presentador

En el desarrollo de la aplicación se ha hecho uso de distintas "tríadas" MVP para permitir la separación de los subsistemas de la aplicación. El esquema planteado se detalla en la siguiente figura.

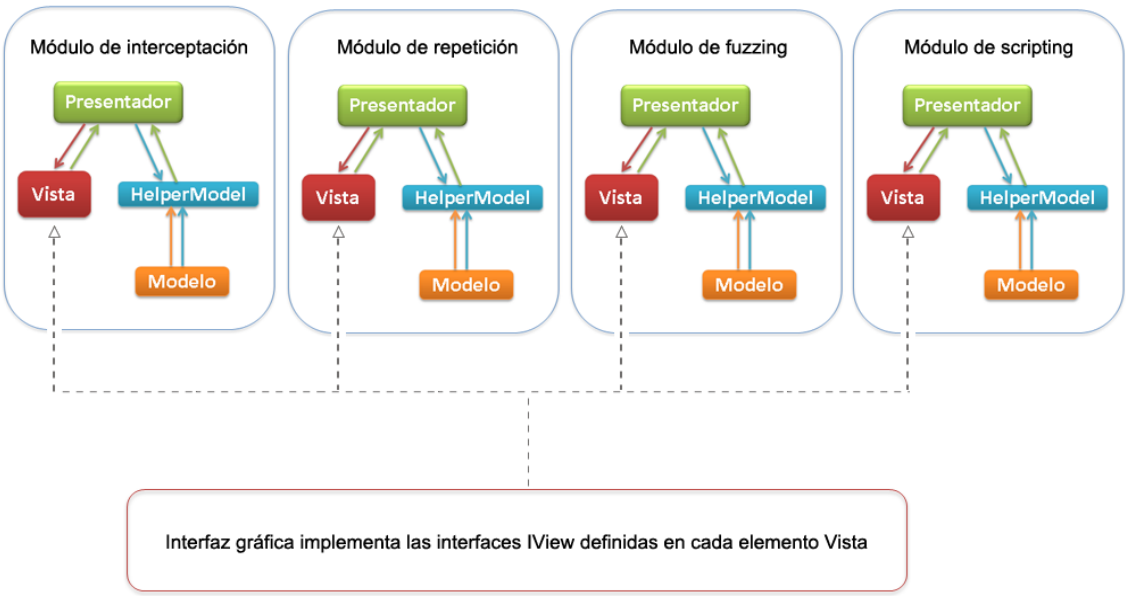


Figura 24: Múltiples tríadas MVP

El presentador de cada triada MVP gestiona la lógica de presentación de su subsistema e invoca las funcionalidades de lógica de negocio asociadas a su subsistema concreto.

A continuación se analiza el diseño de la aplicación en cada una de las iteraciones descritas en el ciclo de vida de desarrollo del Software. Cada iteración representa en un subsistema de la aplicación, y se encarga de llevar a cabo un grupo de tareas específicas.

5.3 Iteración 1.

El diseño de la primera iteración comprende el conjunto de clases que reconstruyen desde una perspectiva orientada a objetos las funcionalidades de la API HTTPCore exportadas a través de una biblioteca dinámica. Las clases de esta primera iteración abstraen los detalles a bajo nivel del acceso a la biblioteca dinámica.

Con el objetivo de aportar mayor claridad al diseño se han omitido el conjunto de métodos que permiten el acceso a las funciones de la API HTTPCore a través la tecnología P/Invoke que permite la realización de llamadas a bibliotecas de enlace dinámico. Estas características se detallaran en profundidad en el "*Capítulo 6: Implementación*" y el anexo "*Manual Técnico*"

Las clases de esta primera iteración permitirán el cumplimiento de los requisitos RF1 hasta RF10.

Para describir las responsabilidades de las distintas clases que se incorporan en cada una de las iteraciones, se empleará la técnica diseño de tarjetas CRC (Clase-

Responsabilidad-Colaboración) que permiten describir a alto nivel el comportamiento de cada una de las clases representadas en el diseño.

La estructura será la siguiente:

Nombre de la clase	
Responsabilidad	Clases colaboradoras

El objetivo de clases definidas en esta iteración es el de proporcionar las capacidades de comunicación de la aplicación a través del protocolo HTTP. Al abstraer los detalles de implementación de las mismas este componente es completamente reutilizable en cualquier otro proyecto.

El inventario de clases para la primera iteración es el siguiente:

HTTP	
Esta clase es responsable de proporcionar el acceso al núcleo de la API HTTPCore, y define las funcionalidades principales: ▪ Inicio y terminación de conexiones HTTP/s ▪ Inicio y terminación del proxy HTTP ▪ definición de opciones del motor de comunicaciones HTTP/s y del proxy. ▪ Envío de peticiones HTTP/s y la recuperación de las respuestas asociadas.	HTTPSession, HTTPRequest, HTTPResponse

HTTPRequest	
Clase permite el acceso y configuración de los elementos de una petición HTTP: Cabeceras, cuerpo de la petición, método HTTP empleado, URL.	

HTTPResponse	
Clase que permite el acceso y configuración de los elementos de una respuesta HTTP: Cabeceras, código de estado HTTP y cuerpo de la respuesta	

HTTPSession	
Clase encargada de mantener la relación entre una petición HTTP y su respuesta correspondiente.	HTTPRequest, HTTPResponse

IHeaders	
Interfaz que permite el acceso a métodos comunes para las clases HTTPRequest y HTTPResponse	HTTPRequest HTTPResponse

Iteración 1

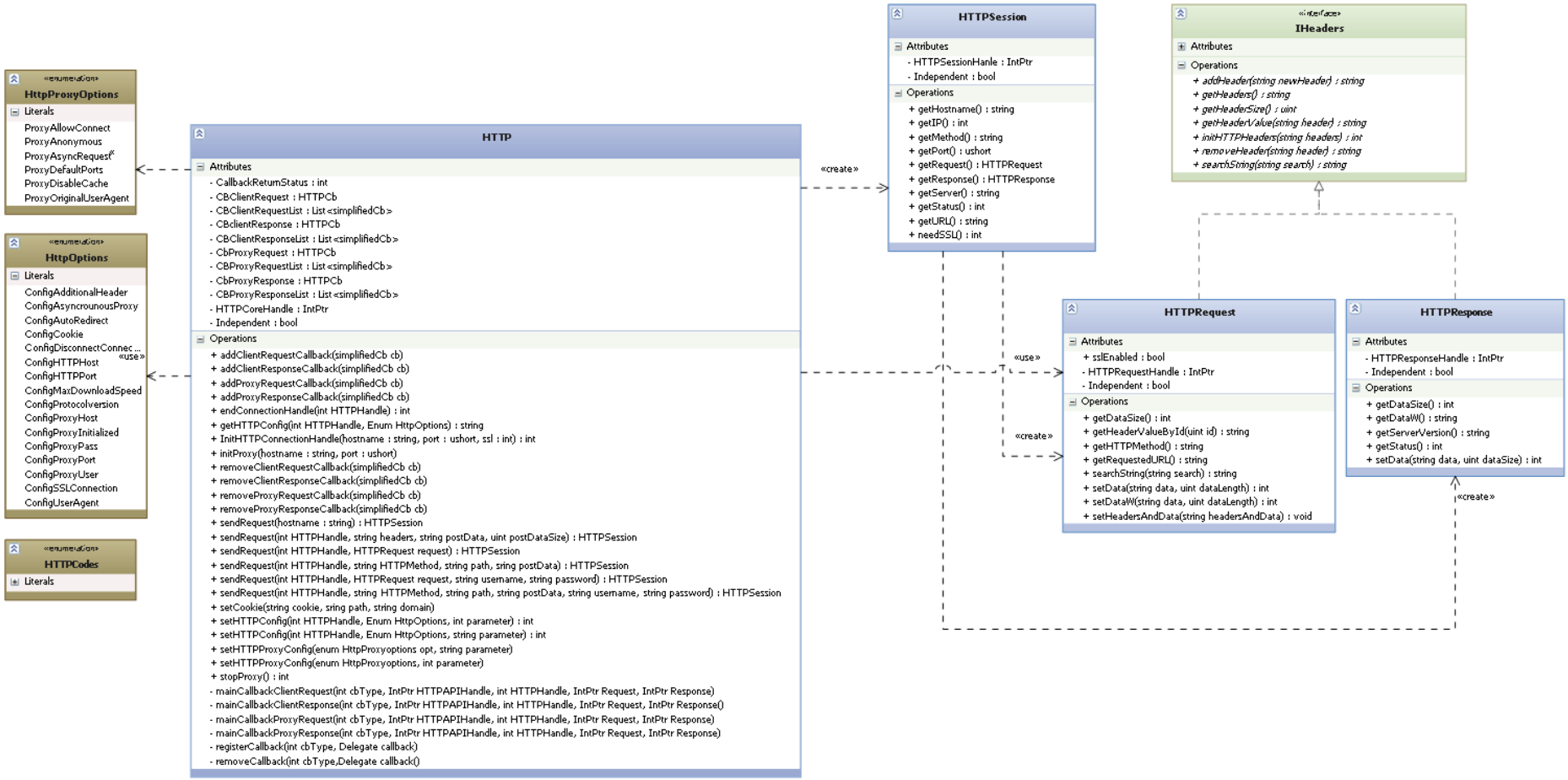


Figura 25: Diagrama de clases de la reconstrucción de las funcionalidades de la API.

5.4 Iteración 2

El diseño de la segunda iteración define una tríada MVP encargada de implementar las funcionalidades del subsistema de interceptación de peticiones HTTP.

Con el objetivo de aportar mayor claridad al diseño de esta iteración se han omitido algunas relaciones existentes con otros componentes de la aplicación (Implementación concreta de la interfaz gráfica, Presentador Fachada y clase SingletonHTTP), sin embargo sí que describe la colaboración existente con dichas clases en las tarjetas CRC. La representación de las relaciones omitidas se detallará en el diagrama de clases global de la aplicación.

InterceptProxyPresenter	
Presentador encargado de implementar la lógica de presentación para el subsistema de interceptación de peticiones HTTP. El objetivo de esta clase es la recepción de los eventos originados en la interfaz gráfica y la invocación de la lógica de negocio asociada a este subsistema a través de la clase Helper (InterceptProxyService).	(Interfaz IInterceptProxyView implementada por el Form principal), InterceptProxyService.

InterceptProxyService	
Clase encargada de implementar la lógica de negocio para este subsistema.	InterceptProxyDataModel, RequestResponse SingletonHTTP

InterceptProxyDataModel	
Clase que representa el modelo de datos para este subsistema. Contiene los parámetros de configuración del proxy, así como la lista de peticiones y respuestas HTTP enviadas a través del mismo.	RequestResponse

IInterceptProxyView	
Interfaz de programación que define los métodos que permiten el acceso a las propiedades de los distintos elementos de la interfaz gráfica asociados a este subsistema. El presentador se comunica con la implementación concreta de la interfaz gráfica a través de este elemento.	

RequestResponse	
Clase encargada de mantener en la aplicación la asociación entre una petición HTTP y su respuesta correspondiente.	

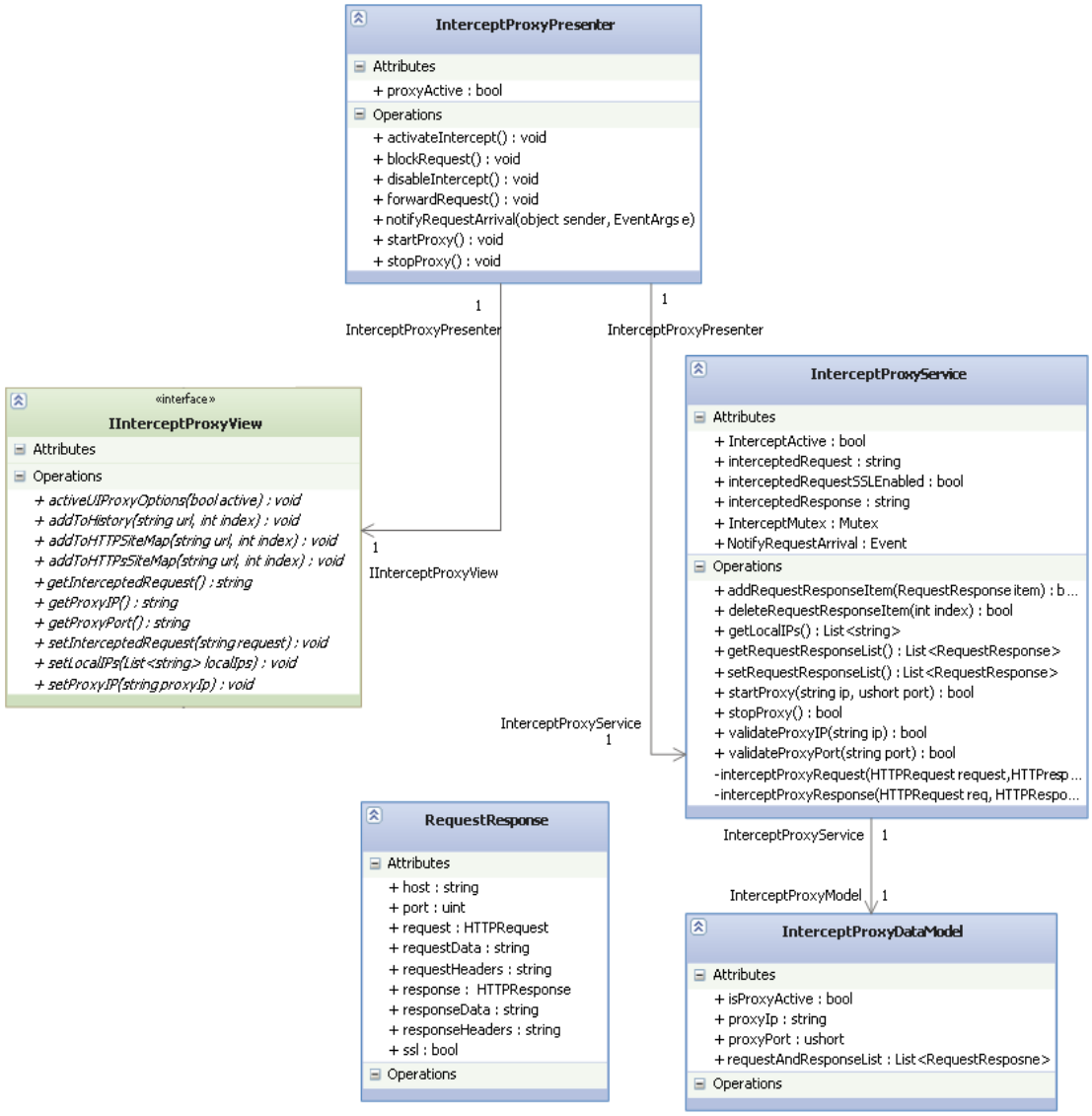


Figura 26: Diagrama de clases del subsistema de interceptación de peticiones HTTP.

5.5 Iteración 3

El diseño de esta iteración define la tríada MVP para el subsistema "módulo de repetición" que permite el envío y construcción de peticiones HTTP con independencia del proxy.

Al igual que en la iteración 2 se han omitido algunas relaciones que serán detalladas en el diagrama de clases global.

El inventario de clases para esta iteración es el siguiente:

RepeaterPresenter.	
Presentador encargado de implementar la lógica de presentación del subsistema.	IRepeaterView, ReapeterService

RepeaterService	
Clase encargada de implementar la lógica de negocio para este subsistema.	RepeaterDataModel SingletonHTTP

RepeaterDataModel	
Clase que representa el modelo de datos para este subsistema. Contiene los parámetros de configuración necesarios para el envío de peticiones HTTP. Definiendo el host destino, puerto, petición (incluyendo cabeceras y cuerpo), respuesta asociada y el parámetro ssl que permite establecer si al comunicación se debe realizar empleando un canal de comunicación seguro HTTPs	

IRepeaterView	
Interfaz de programación que define los métodos que permiten el acceso a las propiedades de los distintos elementos de la interfaz gráfica asociados a este subsistema. El presentador se comunica con la implementación concreta de la interfaz gráfica a través de este elemento.	

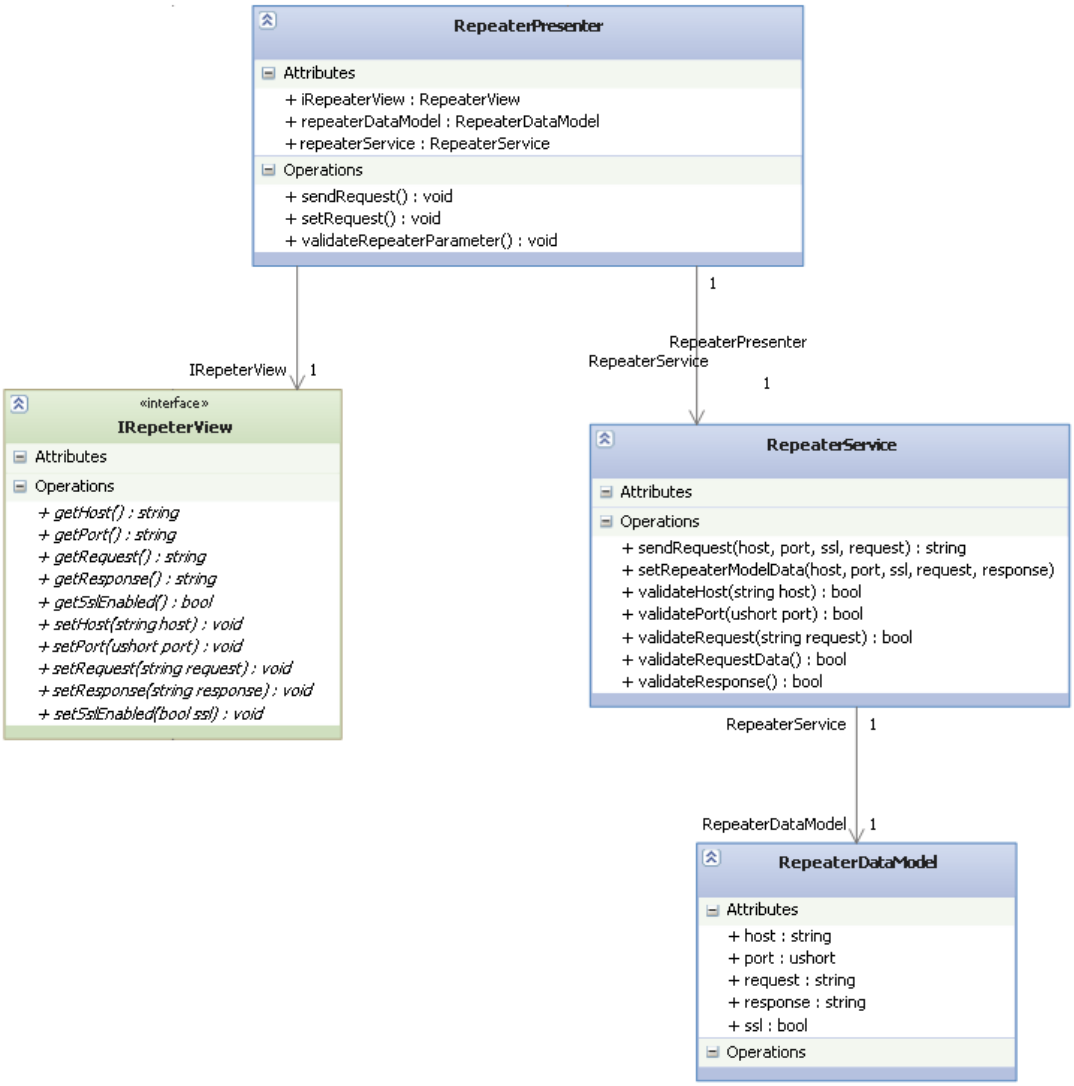


Figura 27: Diagrama de clases para el subsistema "módulo de repetición".

5.6 Iteración 4

En esta sección se describen los elementos de diseño para el subsistema “Módulo de fuzzing”, el cual permite llevar a cabo procedimientos de fuzzing iterativos a través del envío de múltiples peticiones HTTP, reemplazado el valor de ciertos parámetros de una petición previamente configurados por el usuario.

El inventario de clases para esta iteración es el siguiente:

FuzzerPresenter	
Presentador encargado de implementar la lógica de presentación del subsistema. El acceso a distintas interfaces de programación para la interacción con la implementación concreta de la interfaz gráfica es debido a la implementación de distintas vistas en función del tipo de *payload* seleccionado.	Interfaces (IFuzzerView, IFuzzerPayloadTypeNumbers, IFuzzerPayloadList) FuzzerService

FuzzerService	
Clase encargada de implementar la lógica de negocio para este subsistema.	FuzzerDataMode SingletonHTTP

FuzzerDataModel	
Clase que representa el modelo de datos para este subsistema. Incluye las opciones de configuración que permiten definir el host objetivo y los parámetros seleccionados por el usuario.	Interfaz IPayloadParameter

IPayloadParameter	
Interfaz de programación que define atributos y métodos comunes para las clases PayloadParameterList y PayloadParameterNumbers	PayloadParameterList, PayloadParameterNumbers

PayloadParameterList	
Clase responsable de almacenar la lista de payloads que serán sustituidos por los parámetros seleccionados durante el proceso iterativo de fuzzing.	PayloadParameterList, PayloadParameterNumbers

PayloadParameterNumbers	
Clase responsable de almacenar los valores empleados en la generación de la lista de payloads numéricos que serán sustituidos por los parámetros seleccionados durante el proceso iterativo de fuzzing.	PayloadParameterList, PayloadParameterNumbers

IFuzzerView	
Interfaz de programación que define los métodos que permiten el acceso a las propiedades de los distintos elementos de la interfaz gráfica general de este subsistema	IFuzzerPayloadTypeNumbersView, IFuzzerPayloadTypeListView

IFuzzerPayloadTypeNumbersView	
Interfaz de programación que define los métodos que permiten tanto el acceso a las propiedades de los distintos elementos de la interfaz gráfica general como a los elementos propios de la interfaz gráfica específica de los payloads numéricos.	

IFuzzerPayloadListView	
Interfaz de programación que define los métodos que permiten tanto el acceso a las propiedades de los distintos elementos de la interfaz gráfica general como a los elementos propios de la interfaz gráfica específica de los payloads de tipo lista.	

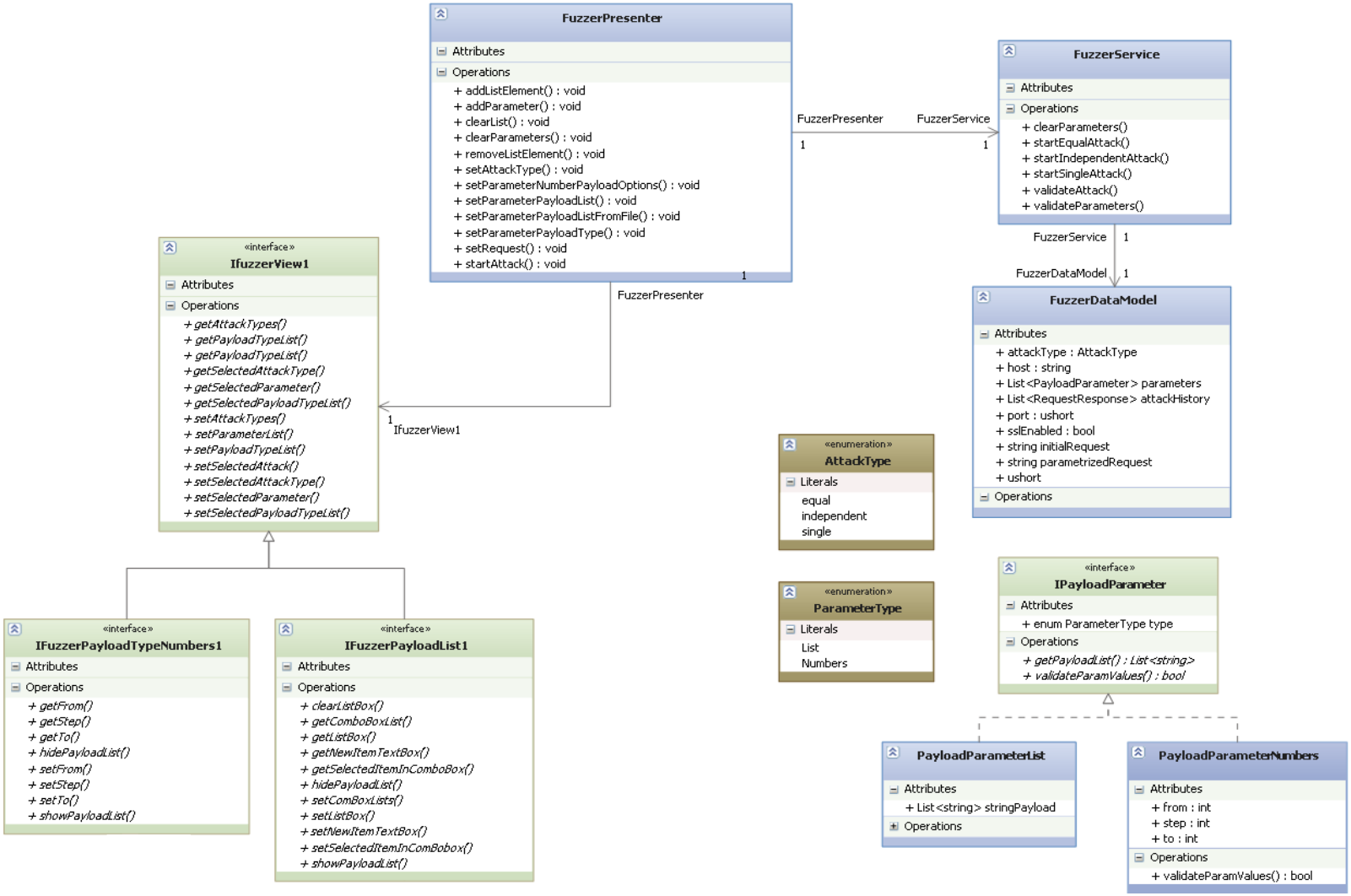


Figura 28: Diagrama de clases del subsistema "Módulo de fuzzing".

5.7 Iteración 5

Esta sección detalla los elementos de diseño del módulo de scripting, el cual permite la creación y edición de scripts, compilación y ejecución de los mismos, así como la activación/desactivación de callbacks que permiten la modificación del comportamiento del motor de comunicaciones HTTP y del proxy.

Este subsistema también especifica las interfaces de programación definidas para la creación de los scripts de auditoría (custom scripts) y los scripts de callback.

El inventario de clases para esta iteración es el siguiente:

ScriptingIDEPresenter	
Presentador encargado de implementar la lógica de presentación del subsistema.	Interfaz IScriptingIDEView ScriptingIDEService

ScriptingService	
Clase encargada de implementar la lógica de negocio para este subsistema. Esta clase implementa las funcionalidades que permiten la activación y desactivación de scripts de callback que permiten modificar el comportamiento del motor de comunicaciones HTTP y del proxy de interceptación. Además define los métodos que permiten el acceso a las funcionalidades de compilación y ejecución de scripts, así como la carga de los scripts en los repositorios de la aplicación.	CustomScriptManager, CallbackScriptManager, ScriptingDataModel

ScriptingManager	
Clase abstracta responsable de proporcionar métodos y atributos comunes a las clases concretas CustomScriptManager y CallbackScriptManager	CustomScriptManager, CallbackScriptManager,

CustomScriptManager	
Clase responsable de proporcionar las funcionalidades de compilación y ejecución de scripts de auditoría (CustomScripts	

CallbackScriptManager	
Clase responsable de proporcionar las funcionalidades de compilación y ejecución de scripts de callback, encargados de modificar el comportamiento del proxy de interceptación.	HTTPResponse

ScriptDataModel	
Clase que representa el modelo de datos para este subsistema. Mantiene listas de los distintos scripts del repositorio, así como de los scripts de callback que se mantienen activos.	CallbackScriptFile, CustomScriptFile

Script	
Clase que representa un fichero script genérico. Define una serie de atributos comunes para las clases CallbackScriptFile y CustomScriptFile. La clase mantiene referencias al nombre del fichero, la ruta del fichero, el objeto compilación (si ha creado una instancia del script tras la compilación), el tipo de instancia del objeto y el tipo de script.	CallbackScriptFile, CustomScriptFile

CallbackScriptFile	
Clase que representa un script de tipo callback	

CustomScriptFile	
Clase que representa un script de auditoria (CustomScript)	

ICustomScript	
Clase abstracta que define una serie de métodos que deben ser implementados por los scripts de auditoría desarrollados por el usuario para que sea posible su compilación.	

ICallbackRequest	
Interfaz de programación que define un método que debe ser implementado por los scripts de callback desarrollados por el usuario. Los scripts que implementen esta interfaz podrán modificar el comportamiento del motor de comunicaciones o del proxy, posibilitando la modificación de las **peticiones** HTTP antes de su envío, así como el bloqueo o redirección de las mismas ante una serie de condiciones definidas en el script.	HTTPRequest

ICallbackResponse	
Interfaz de programación que define un método que debe ser implementado por los scripts de callback desarrollados por el usuario. Los scripts que implementen esta interfaz podrán modificar el comportamiento del motor de comunicaciones o del proxy, posibilitando la modificación de las **respuestas** HTTP tras su recepción.	HTTPResponse.

PresentationScript	
Clase empleado por el presentador para mantener información de presentación acerca de los ficheros de scripts que se encuentran abiertos en la interfaz gráfica.	

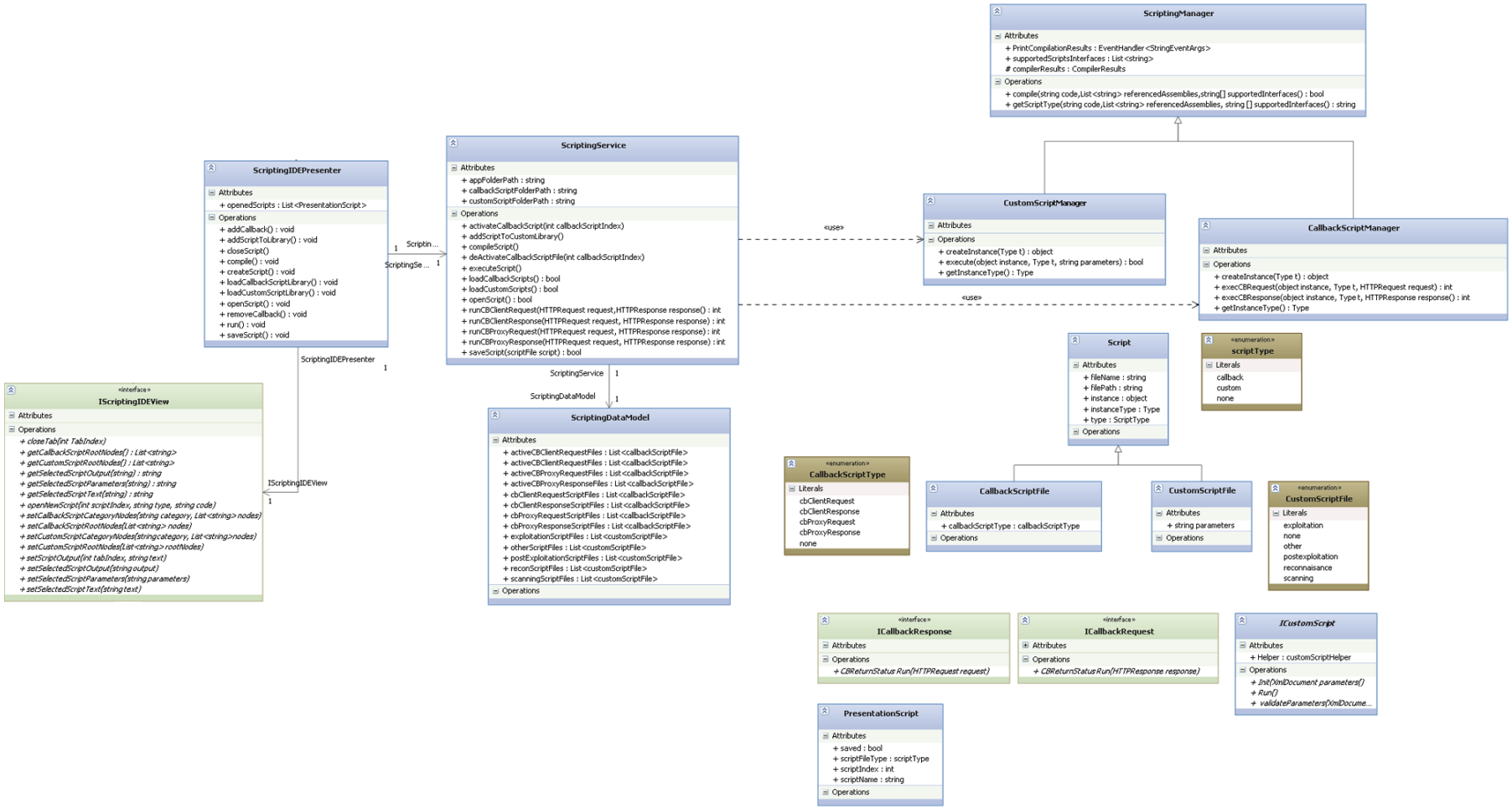


Figura 29: Diagrama de diseño del subsistema "Módulo de scripting".

5.8 Patrones de diseño

Los patrones de diseño definen un conjunto de soluciones a problemas recurrentes durante el desarrollo del software. Los patrones están planteados desde una perspectiva que permita la reutilización de los diseños en distintos escenario.

Pueden ser categorizados en tres grupos (22):

- ***Creacionales***: Los patrones creacionales abordan el problemática asociada a los diferentes mecanismos de creación de objetos. Este tipo de patrón trata la creación de objetos de la forma más adecuada en función de un problema específico.

- **Estructurales:** Los patrones estructurales facilitan el diseño mediante la identificación de la forma más simple de llevar a cabo la relación entre diversas entidades.

- **Comportamiento:** Identifican patrones de comunicación entre objetos permitiendo aportar mayor flexibilidad en la realización de la comunicación.

Los patrones de diseño empleados por la aplicación son los siguientes:

- **Facade(Fachada)** (23)**:** El patrón de diseño fachada es un tipo de patrón estructural que permite definir una interfaz como punto de entrada específico para el acceso a un grupo de objetos o funcionalidades ofreciendo visibilidad de los mismos.

  Si bien existen distintos usos de este patrón, el objetivo de implementación del mismo en la aplicación es el de permitir la visibilidad de los presentadores de los distintos subsistemas a la interfaz gráfica. El patrón de diseño se ha implementado en la clase FacadePresenter.

- **Mediator:** (24) El patrón Mediator es un patrón de comportamiento que define como debe ser la interacción entre un grupo de objetos relacionados. A diferencia del patrón fachada el patrón Mediator permite imponer una política de limitación de visibilidad, encapsulando el modelo comunicación entre objetos y actuando de “Mediador”, como su nombre indica.

  El objetivo de implementar este patrón de diseño es el de permitir la comunicación entre el presentador InterceptProxyPresenter y los presentadores RepeaterPresenter y FuzzerPresenter, limitando qué funcionalidades deben ser expuestas.

- **Proxy** (7)**:** El patrón de diseño proxy es un tipo de patrón estructural cuyo propósito es el de proporcionar un intermediario para llevar a cabo el control de acceso a un objeto. Su utilización es muy común a la hora de proporcionar un punto de acceso a APIs de terceros.

  El objetivo de implementación de este patrón de diseño es permitir abstraer los detalles de implementación del acceso a la biblioteca dinámica HTTPCore.dl. Dado que el acceso a las funcionalidades de la API se realiza a través de las clases proxy cualquier cambio realizado sobre la misma no requerirá la modificación del resto de la aplicación.

  Este patrón de diseño se ha implementado en las clases HTTP, HTTPRequest, HTTPResponse y HTTPSession.

- **Singleton** (25)**:** El objetivo de este patrón es garantizar que solo exista una única instancia de una clase concreta, permitiendo proporcionar un único punto de acceso a un objeto de forma global a toda la aplicación.

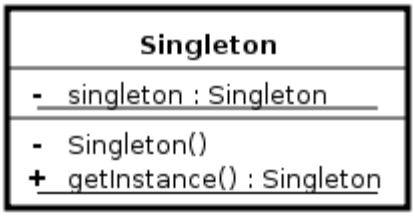


Figura 30: Patrón de diseño Singleton.

Se ha optado por utilizar este patrón diseño para disponer de una única instancia global de la clase proxy HTTP. De esta forma se permite el registro de funciones de callback sobre la misma. Todas las clases que implementan la lógica de negocio de la aplicación acceden a las funcionalidades de la API a través de la clase *SingletonHTTP*

.

5.9 Diagramas de secuencia.

Los diagramas de secuencia son un tipo de diagrama UML que permiten modelar la interacción entre distintos objetos del sistema a lo largo del tiempo. Generalmente un diagrama de secuencia representa un caso de uso, y muestra el modelo de interacción existente entre los objetos necesarios para la implementación de dicho caso de uso.

A través de los diagramas de secuencia representados es posible comprender visualmente como es la interacción entre los distintos elementos que componen una tríada MVP para un subsistema concreto.

A continuación se analizarán los diagramas de secuencia que modelan el comportamiento de las clases asociadas a los casos de uso: “Iniciar Proxy”, “Enviar petición” y “Compilar Script”.

CU-01 Iniciar proxy

Escenario: El caso de uso se ejecuta correctamente.

El diagrama de secuencia se inicia con la invocación del método StartProxy en el presentador, como consecuencia de la generación de un evento en la interfaz gráfica tras pulsar el botón “Start Proxy”.

En primer lugar el presentador “InterceptProxyPresenter” se encarga de validar que el proxy de interceptación haya sido iniciado previamente. Para ello consulta a la clase “InterceptProxyService” responsable de implementar la lógica de negocio del subsistema de interceptación, que a su vez recupera el valor del modelo de datos. Dado que el presentador implementa la lógica de Presentación, será responsable de obtener los datos de la interfaz gráfica, invocando los métodos getProxyIP y getProxyPort definidos en la interfaz de programación InterceptProxyView. Tras la recepción de los valores, se lleva a cabo invocación del método startProxy a través de la clase InterceptProxyService la cual se encargará de validar los datos de entrada y obtener la instancia global de la clase HTTP mediante el acceso a la clase SingletonHTTP, que implementa el patrón de diseño Singleton.

A continuación se iniciará el proxy y se llevara a cabo la actualización del modelo datos. Finalmente el presentador actualizará sus atributos de presentación y reflejará el resultado de la acción en la interfaz gráfica.

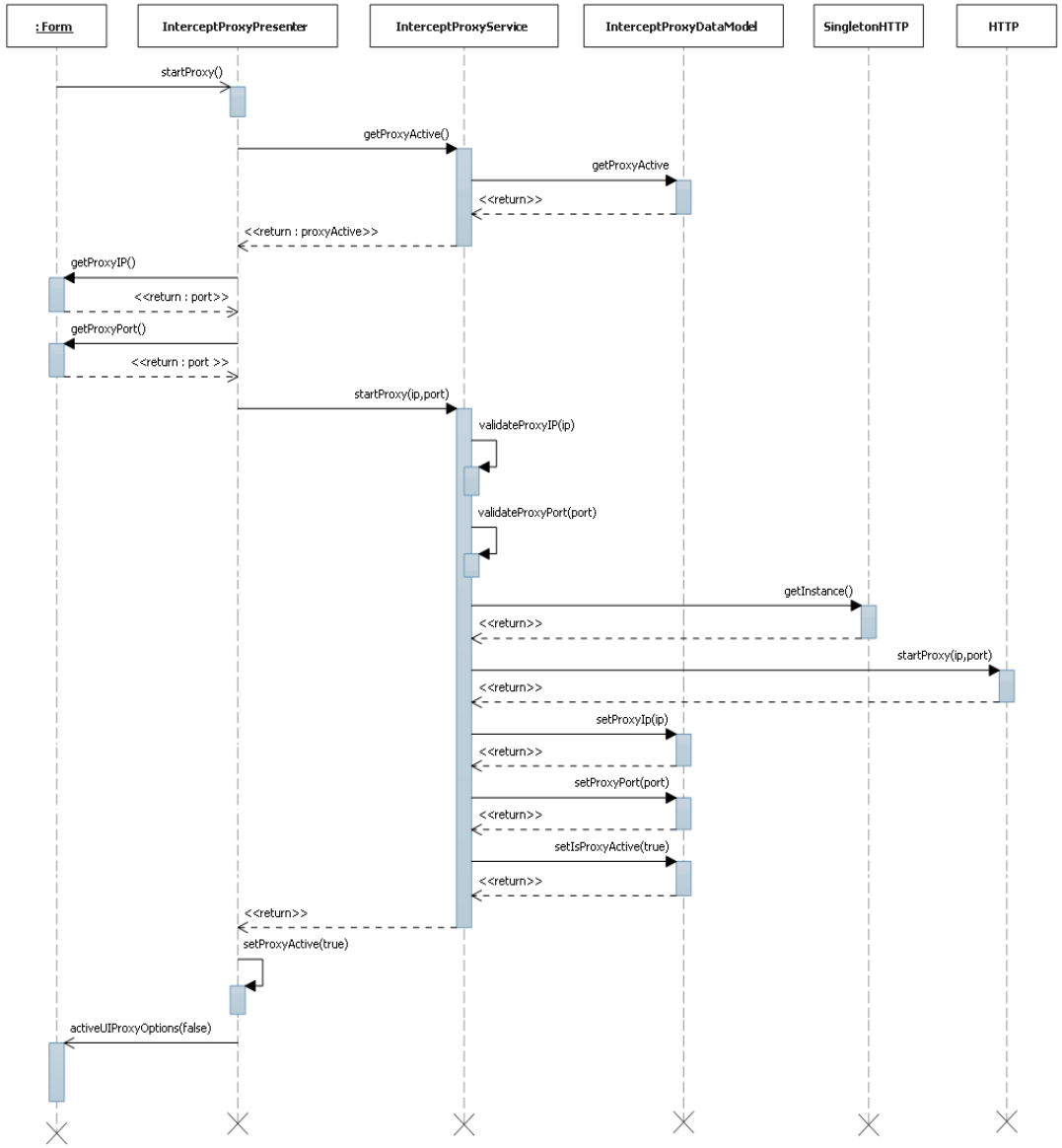


Figura 31: Diagrama de secuencia para el caso de uso iniciar proxy.

CU-12: Reenviar petición HTTP/s

Escenario: El caso de uso se ejecuta correctamente.

En el siguiente diagrama de secuencia es posible visualizar las acciones que se llevan a cabo para la ejecución del caso de uso "Reenviar petición HTTP/s". Las acciones relevantes de este caso de uso son:

- El establecimiento de la conexión con el host objetivo mediante la invocación del método InitHTTPConnectionHandle sobre la clase proxy HTTP.
- La definición de los datos de una petición HTTP a través de la clase HTTPRequest.
- El envío de la petición mediante la invocación del método SendHTTPRequest.
- La recuperación de los datos de la respuesta HTTP a través del acceso a la clase HTTPSession.

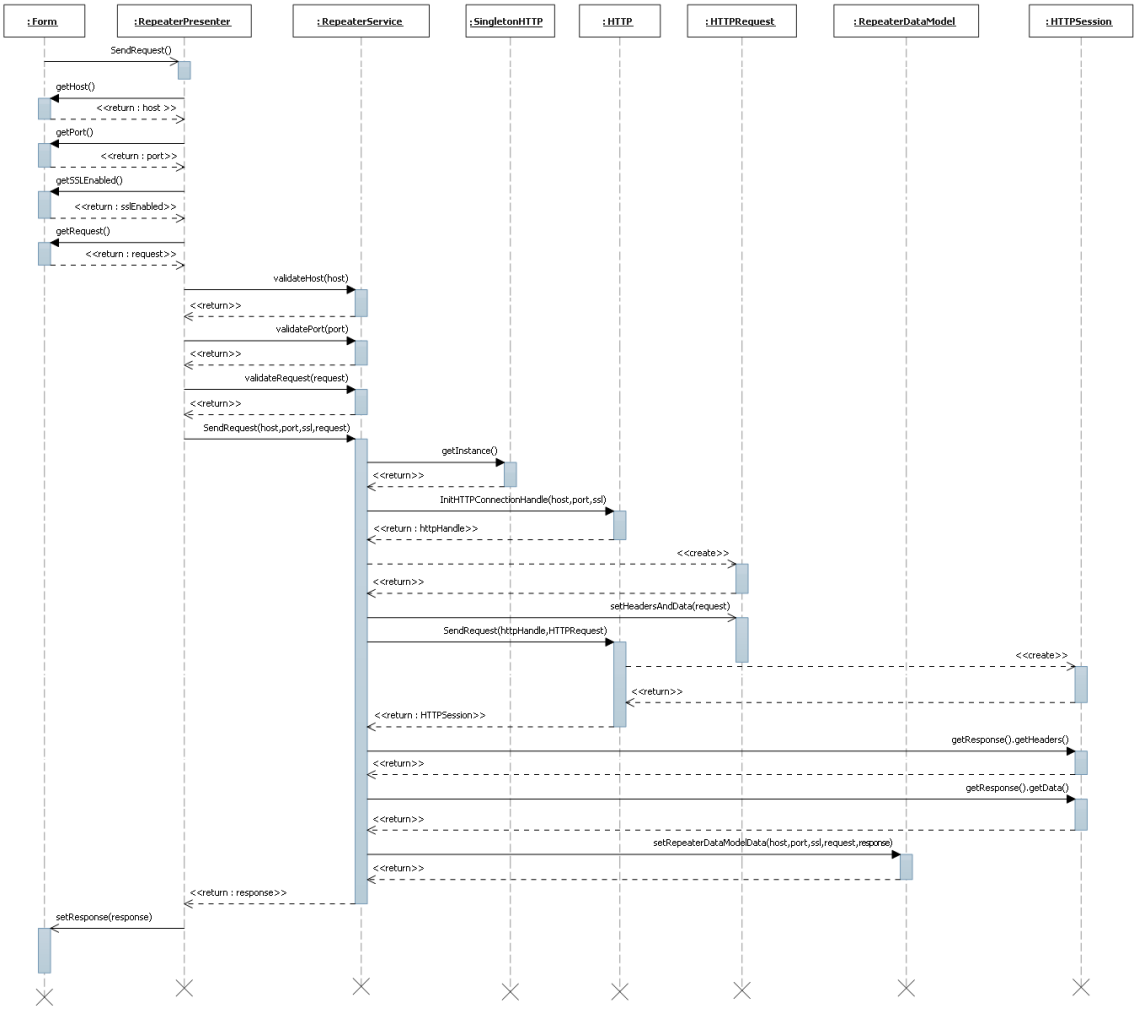


Figura 32: Diagrama de secuencia para el caso de uso "Reenviar petición HTTP/s"

Caso de uso: Compilar Script.

Escenario: El caso de uso se ejecuta correctamente.

El siguiente diagrama de secuencia representa el conjunto de acciones que se realizan para llevar a cabo la compilación de un script de auditoría (CustomScript).

Para ello el presentador invoca el método de compilación sobre el Helper ScriptingService que delega el proceso en la clase CustomScriptManager. Una vez se ha ejecutado la acción, se obtienen los resultados de compilación y se muestran en la interfaz gráfica.

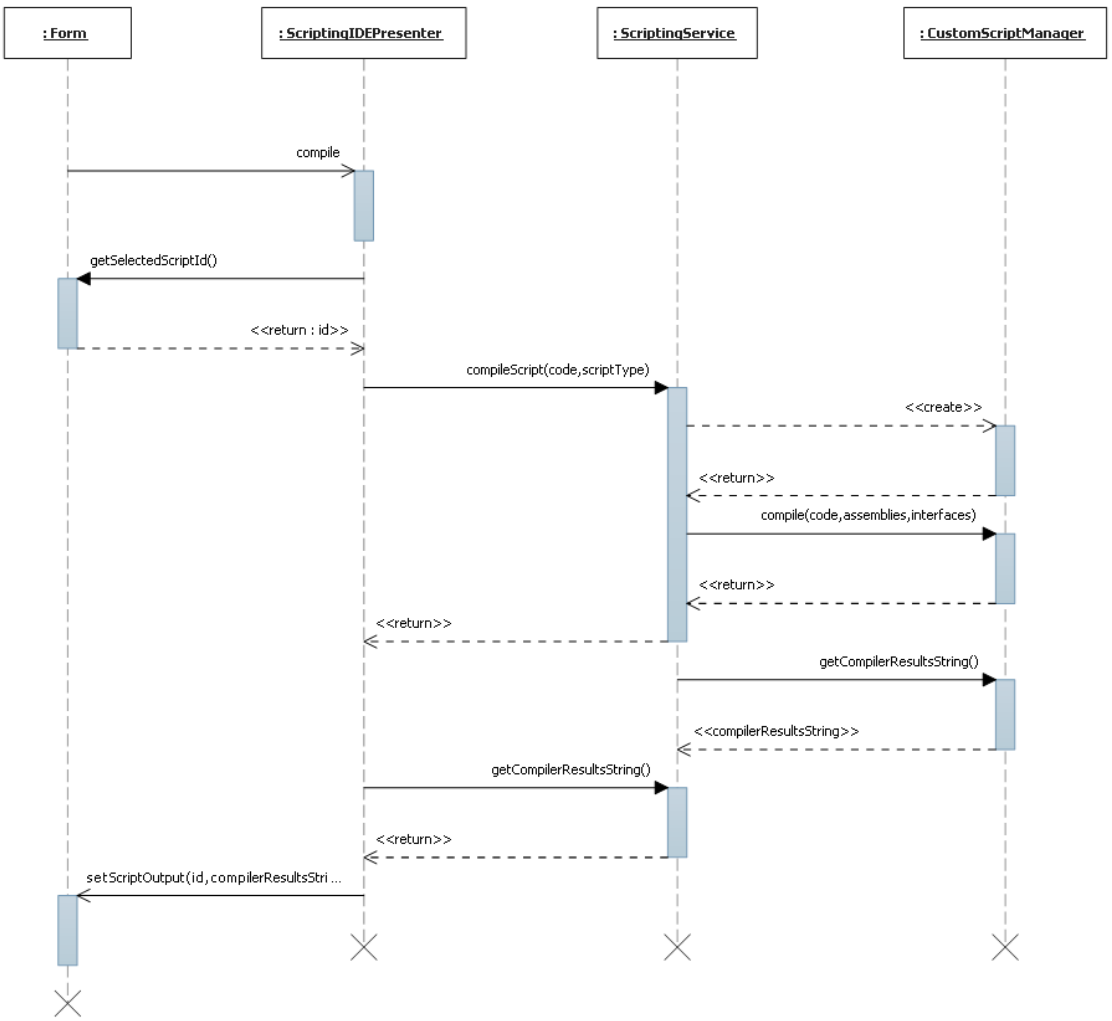


Figura 33: Diagrama de secuencia para el caso de uso "Compilar Script".

6. Implementación.

En este capítulo se detallan los detalles de implementación de la biblioteca dinámica HTTPCore.dll, la cual contiene todas las funcionalidades exportadas de la API.

A continuación se especifican los detalles técnicos asociados al proceso de creación de la biblioteca de vínculo dinámico (DLL) que permitirá ofrecer la interoperabilidad de la API con múltiples lenguajes. Se comenzará analizando la problemática, que ha condicionado la implementación final.

6.1 Consideraciones Iniciales

Inicialmente se dispone de la API HTTPCore, implementada como un conjunto de clases desarrolladas en C++ que implementan toda la lógica que permite establecer un canal de comunicación empleando el protocolo HTTP. Dado que la API está desarrollada siguiendo el paradigma orientada objetos, la interacción original con la misma requiere la instanciación de objetos para a continuación realizar la invocación de los métodos asociados a la clase que define dicho objeto.
Como se comentó en el Capítulo 4 la aproximación implementada para permitir la interoperabilidad de la API, consiste en ofrecer una biblioteca de vínculos dinámicos que exponga todas las funcionalidades definidas en la especificación de requisitos.
A continuación se describen los pasos que se han llevado a cabo para permitir la interoperabilidad de la API y se detallan los distintos aspectos relevantes de su implementación.

1. Identificación de funcionalidades que deben ser exportadas, teniendo en cuenta la especificación de requisitos.
2. Creación del *wrapper*, que expone las funcionalidades de la API a través de funciones básicas de C. La decisión asociada a la implementación de este esquema se detalla en la sección "*Decoración de nombres*"
3. Compilación de la DLL. Las funciones que son expuestas en la biblioteca de vínculos dinámicos son aquellas definidas a través del *wrapper*.
4. Creación de las clases proxy que permiten el acceso a la API a través de las funciones exportadas en la DLL. Las clases proxy pueden ser implementadas en cualquier en lenguaje de programación, aunque en este proyecto han sido desarrolladas en C#.

6.2 Detalles de implementación del Wrapper

Ya que el wrapper expone las funcionalidades a través de funciones básicas de C, la implementación de esta aproximación implica perder el paradigma orientado a objetos con el que fue concebida la API.
Para llevar a cabo la interacción con los objetos de la API, el wrapper debe implementar funciones que permitan:

- La creación de objetos, permitiendo devolver como valor de retorno tras su invocación, un puntero que permita definir una referencia al objeto instanciado.
- La destrucción de objetos, tomando como entrada el puntero que mantiene la referencia de un objeto instanciado previamente.
- La invocación de los métodos de API, tomando como entrada un puntero que mantiene la referencia al objeto instanciado y a través del cual es posible realizar las llamadas a los métodos de las clases C++.

Con el objetivo de aportar mayor claridad a la solución se presenta el siguiente código de ejemplo, que representa una clase C++ “Persona”, que dispone de un atributo edad y dos métodos de acceso a dicho atributo.

```
	// Clase desarrollada en C++
	class Persona{
		private:
			unsigned int edad;
		public:
			Persona();		// Constructor de la clase
			unsigned int getEdad();	// Método getter
			void setEdad(unsigned int edad);// Método setter
	};

	 Persona::Persona(){
		this->edad = 30;
	 }

	 unsigned int Persona::getEdad(){
		return this->edad;
	 }

	 void Persona::setEdad(unsigned int edad){
		this->edad = edad;
	 }

	 // WRAPPER QUE EXPONE LAS FUNCIONALIDADES DE LA CLASE COMO FUNCIONES
BÁSICAS DE C
extern "C"{
	 // Exportación del método constructor.
	 __declspec(dllexport) Persona * newPersona(){
		return new Persona;
	 }
	 // Exportación del método destructor
	 __declspec(dllexport) void destroyPersona(Persona * persona){
```

```
                if ((Persona *)persona != NULL){
                        delete persona;
                        persona = NULL;
                }
        }
        // Exportación del metodo getter getEdad, tomando como entrada
        // un puntero que referencia al objeto sobre el cual se desea
        // invocar este método
        __declspec(dllexport) unsigned int getEdad(Persona * persona){
                if ((Persona *)persona != NULL){
                        return ((Persona *)persona)->getEdad();
                }else{
                        return -1;
                }
        }
        // Exportación del método setter setEdad, tomando como entrada
        // un puntero que referencia al objeto sobre el cual se desea
        // invocar este método y el valor que se desea asignar.
        __declspec(dllexport) void setEdad(Persona * persona, unsigned int
edad){
                if ((Persona *)persona != NULL){
                        ((Persona *)persona)->setEdad(edad);
                }
        }
}
```

Figura 34: Código de ejemplo del proceso de creación del wrapper

Las directivas de compilación__declspec(dllexport) (26) permiten definir las funciones que serán exportadas en la (DLL). La compilación del código generaría la biblioteca de vínculo dinámico Persona.dll.
Como se puede observar en el código de ejemplo se exponen funciones básicas de C para permitir tanto la creación y destrucción de objetos, así como para la invocación de los métodos sobre la referencia de un objeto concreto instanciado previamente.

6.3 Detalles de implementación de las clases Proxy en C#

Las clases proxy son responsables de encapsular los métodos de acceso a las funcionalidades de la API exportadas a través de la DLL, permitiendo abstraer los detalles acceso, además de permitir "reconstruir" el modelo de clases original de la API.
Las clases proxy pueden ser desarrolladas utilizando cualquier lenguaje de programación, por lo tanto permitiendo la interoperabilidad de la API.
A continuación se muestra un código de ejemplo en el que se define una clase proxy en C#, que permite el acceso a la DLL a partir del código de ejemplo de la Figura 31.

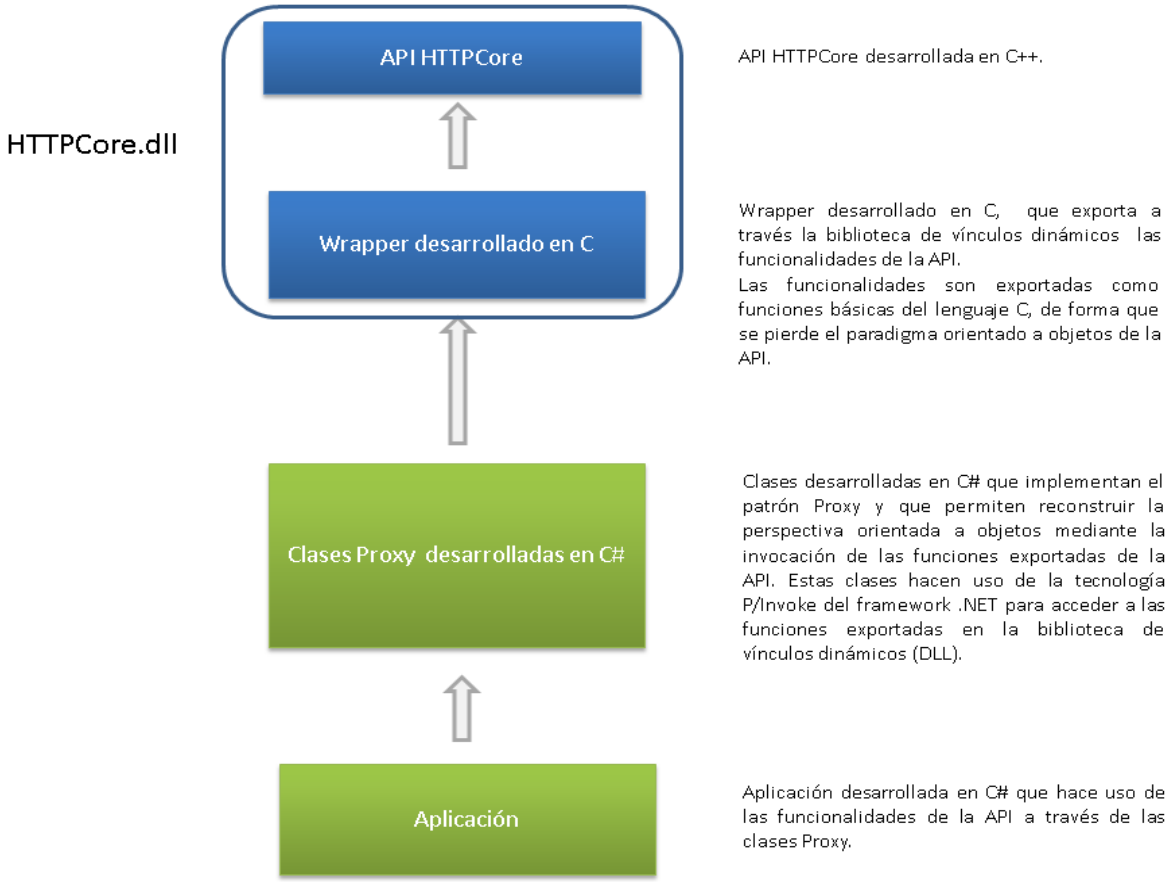
```
    public class Persona : IDisposable
    {

        // Acceso a las funciones exportadas por la DLL a través de la
tecnología P/Invoke del framework .NET

        [DllImport("Persona.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern IntPtr newPersona();
```

```
        [DllImport("Persona.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern void destroyPersona(IntPtr personaRef);

        [DllImport("Persona.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern UInt32 getEdad(IntPtr personaRef);

        [DllImport("Persona.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern void setEdad(IntPtr personaRef,UInt32 edad);

      // Puntero que permite almacenar la referencia a la instancia de un objeto
        private IntPtr personaRef;

        // Métodos getters/setters que acceden a las funciones getEdad y setEdad
        // exportadas por la biblioteca de vínculo dinámico Persona.dll
        public UInt32 edad
        {
            get
            {
                if (personaRef != IntPtr.Zero)
                {
                    return getEdad(this.personaRef);
                }else
                {
                    return 0;
                }
            }

            set
            {
                if (personaRef != IntPtr.Zero)
                {
                    setEdad(this.personaRef,value);
                }
            }
        }
        // Constructor para la clase C# que invoca la función newPersona,
        // exportada en la dll.

        public Persona()
        {
            this.personaRef = newPersona();
        }

        // Metodo destructor de la clase C# que invoca al método dispose.
        public ~Persona()
        {
            Dispose();
        }
        // Método dispose que lleva a cabo la liberación de memoria
        // mediante la invocación de la función de la biblioteca dinámica
        // destroy persona.
        public void Dispose()
        {
            if (this.personaRef != IntPtr.Zero)
            {
                destroyPersona(this.personaRef);
            }
        }
    }
```

Figura 35: Código de ejemplo que define la creación de clases proxy.

6.4 Modelo de acceso a la API HTTPCore

En esta sección se comentará desde una perspectiva de alto nivel como se implementa el modelo de acceso a la API HTTPCore.



A continuación se analizará la problemática, que ha condicionado la implementación final.

6.5 Decoración de nombres

La decoración de nombres o “name mangling” (27) es una técnica empleada por los compiladores con el objetivo de solucionar ciertos problemas causados por la necesidad llevar a cabo una resolución de nombres únicos por parte de entidades de programación.

La decoración de nombres permite solucionar los siguientes problemas:

- La identificación de nombres de clases, estructuras, funciones y tipos de datos en lenguajes de programación que permiten que diferentes entidades dispongan del mismo nombre siempre y cuando pertenezcan a un espacio de nombres (namespace) (28) distintos.
- La identificación de nombres de funciones cuando se implementa la propiedad de sobrecarga en lenguajes de programación orientados a objetos.

Mediante la decoración de nombres es posible llevar a cabo una definición de nombres únicos mediante la codificación de información adicional en el nombre de clases, funciones y estructuras con el objetivo de ofrecer más información semántica a compiladores y enlazadores (linkers).

6.6 Decoración de nombres y bibliotecas de vínculos dinámicos

El proceso de decoración de nombres no está estandarizado y depende principalmente de compilador empleado. Cada compilador implementa el algoritmo de decoración de una forma diferente (27).El problema fundamental asociado a esta situación es que si se desean exportar clases o métodos de una clase a través de una biblioteca de enlace dinámico será necesario hacer referencia a los nombres decorados de clases, métodos o estructuras.
A continuación se muestra un código de ejemplo que permite exportar dos métodos de una clase C++ que implementan la propiedad de sobrecarga.

```
class Test{
      public:
            __declspec(dllexport) int Foo(int n);
            __declspec(dllexport) char Foo(char c);
};

 int Test::Foo(int n){
      return n;
}

 char Test::Foo(char c){
      return c;
}
```

Figura 36: Definición de clase que permite exportar dos métodos sobre los que se ha implementado la propiedad de sobrecarga.

Tras compilar el código es posible visualizar la tabla de símbolos de la DLL resultante, empleado una herramienta como *Dependency Walker* (29) de la Suite de herramientas Sysinternals.

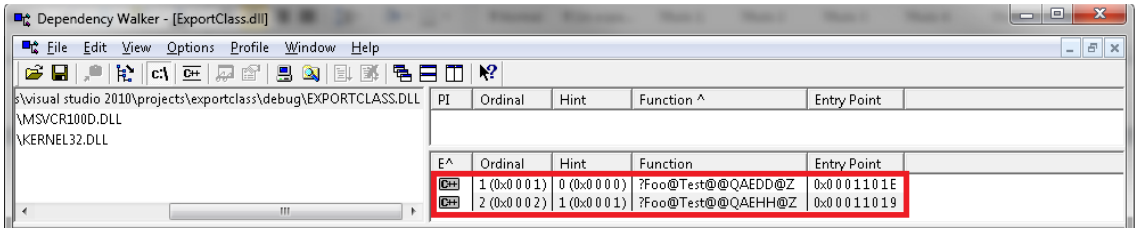


Figura 37: Métodos de una clase exportados en una DLL.

Como se observa en la Figura 23, cada una de los métodos exportaos dispone de un nombre único tras llevar a cabo la decoración de nombres.
Teniendo en cuenta que cada compilador implementa su propio esquema de decoración y que el esquema puede variar entre distintas versiones de un mismo

compilador, se deduce que si se desease invocar estos métodos desde una cualquier aplicación que utilice la DLL sería necesario referenciar los nombres decorados. Esta situación hace que los métodos exportados sean dependientes del compilador empleado.

Solución:

Para la definición de las funcionalidades que deben ser exportadas ha sido necesario implementar otra aproximación ya que los problemas asociados a la decoración de nombres hacen que la solución no sea mantenible.
Para evitar la decoración de nombres se ha hecho uso de la directiva **`extern "C"`**, que permite definir bloques de funciones garantizando que no se produzca la decoración de los identificadores de funciones.

6.7 Fugas de memoria (Memory Leaks).

Las fugas de memoria o (memory leaks como se conoce en inglés) (30) es un tipo de error de programación que ocurre cuando se pierde la referencia de un bloque reservado dinámicamente en memoria, ya que el puntero que mantiene la dirección de un bloque es reasignado sin llevar a cabo una liberación del mismo.
Este tipo de error ocurre principalmente en aquellos lenguajes en los que la administración de la memoria se realiza de forma manual, donde el programador es responsable de realizar tanto la reserva como la liberación de bloques de memoria. Los fallos de fuga de memoria pueden llegar a ser problemáticos cuando suceden frecuentemente ya que el espacio de memoria para un proceso seguirá creciendo indefinidamente hasta que se produzca un agotamiento de los recursos.

Solución:

Como ya se comentó en la sección "*4.3.2 Esquema de interacción entre componentes*" con el objetivo de proporcionar una capa de abstracción para el acceso a la API empleando el patrón "Proxy Pattern", se ha implementado un conjunto clases desarrolladas en C# para permitir el acceso a las funcionalidades de la API HTTPCore ofreciendo una perspectiva orientada a objetos.
Para hacer frente al problema de fugas de memoria todas las clases "proxy" deben implementar la interfaz de programación del framework .NET IDisposable (31) la cual define un método que permite establecer la rutinas de destrucción de un objeto, de forma que, cuando el flujo del programa salga del contexto local en el que fue instanciada la clase proxy, se llevará a cabo la invocación de forma automática de la función que permite la destrucción del objeto y por lo tanto la liberación de memoria.
La creación de los objetos tras invocar el constructor de una clase concreta a través de la función de creación exportada en la biblioteca de vínculos dinámicos, para cada uno de las clases que constituyen la API HTTPCore, genera un puntero que actúa como *handle* (32) y permite referenciar el objeto creado internamente a través del código de la DLL. Este *handle* se almacena de forma privada en cada una de clases proxy y se

pasa como parámetro a los métodos exportados con el objetivo de identificar el objeto sobre el que se tiene que ejecutar la invocación de una método concreto.

```
public class HTTP{

        private IntPtr HTTPCoreHandle;

        // Constructor de la clase C# que reconstruye el paradigma orientado a
objetos de la API HTTPcore.
        public HTTP()
        {
            this.HTTPCoreHandle = newHTTPAPI();
        }


        ~HTTP()   // Destructor de un objeto en C# que es invocado cuando el
flujo del programa sale del contexto local del objeto.
        {
            Dispose();
        }

        public virtual void Dispose()
        {
            if (this.HTTPCoreHandle != IntPtr.Zero) // Validación de puntero
nulo.
            {
                destroyHTTPAPI(this.HTTPCoreHandle); // Invocación de la función
de destrucción de un objeto creado internamente por la biblioteca de vínculo
dinámico
            }
        }
}
```

Figura 38: Ejemplo de definición de handles y contramedida contra fugas de memoria.

6.8 Convenciones de llamada (Calling Conventions)

Las convenciones de llamada (33) son un mecanismo estandarizado que definen como, a bajo nivel se debe llevar a cabo la implementación de funciones y como éstas deben ser invocadas por parte del sistema, permitiendo por lo tanto definir una interfaz de llamada al código a través de la cual se especifica:

- El orden en el que se insertan en la pila (Stack) los argumentos de una función.
- El orden en el que se reserva espacio en la pila para las variables locales.
- Como se devuelven los valores de retorno a la función llamadora (Caller en inglés), es decir aquella función que llama a una subrutina.
- Qué registros del procesador deben ser preservados por la subrutina o función llamada (Callee en inglés) tras la ejecución de la misma.

- Como se dividen las tareas de preparación y restauración de la pila (prólogo de función y epílogo de función respectivamente (33)) entre la función llamadora y la función llamada, tras la ejecución de la misma.

Existen 3 convenciones de llamada utilizadas para código desarrollado en el lenguaje de programación C:

- CDECL (34): Convención de llamada empleada por defecto para código C.
- STDCALL (35): Convención de llamada definida por Microsoft y empleada para el acceso a las funciones de la API Win32.
- FASTCALL (35): Convención de llamada, no estándar. Existen diferencias de implementación entre distintos compiladores, de forma que debe ser utilizada con cautela.

Existen también otros tipos de convenciones de llamada como por ejemplo: THISCALL empleado para el código desarrollado en C++, PASCAL para código desarrollado en Pascal y FORTRAN para código desarrollado en Fortran.

Se ha optado por utilizar la convención de llamada CDECL a la hora de compilar la DLL, por lo tanto cualquier invocación de las funciones exportadas a través de la misma debe ser realizada utilizando CDECL.

6.9 Representación y formato de tipos de datos

El tamaño de tipos de datos entre distintos lenguajes de programación/compiladores puede variar, por lo tanto es necesario definir una especificación que defina el formato y tamaño de los tipos de datos que serán pasados como argumentos a las funciones exportadas, o que son devueltos como valor de retorno tras la invocación de una función.
La DLL ha sido compilada para una arquitectura de 32 bits con soporte UNICODE UTF-16 (36). La definición de tipos de datos, formato y tamaño es la siguiente, para la compilación de la API utilizando el compilador Visual C++ de Microsoft

Tipo de dato en C	Tipo de dato en C#	Tamaño (x86)	Formato
Int/HTTPHANDLE	int	4 bytes	Número entero
unsigned int	uint	4 bytes	Número entero sin signo.
wchart_t*	`[MarshalAs (UnmanagedType.LPWStr)] string`	4 bytes	Apunta a una cadena de caracteres de 2 bytes Unicode

Tipo de dato en C	Tipo de dato en C#	Tamaño (x86)	Formato
			(UTF-16) terminada en byte nulo.
const wchart_t	`[MarshalAs (UnmanagedType.LPWStr)] string`	4 bytes	Apunta a una cadena de caracteres de 2 bytes Unicode (UTF-16) terminada en byte nulo.
int *	IntPtr	4 bytes	Puntero
HTTPAPI *	IntPtr	4 bytes	Puntero
HTTPRequest *	IntPtr	4 bytes	Puntero
HTTPResponse *	IntPtr	4 bytes	Puntero
HTTPSession *	IntPtr	4 bytes	Puntero

La sección **Manual Técnico** especifica los detalles de los métodos de la API HTTPCore, así como las funcionalidades exportadas en la DLL a través del wrapper y los métodos de las clases proxy desarrolladas en C# que permite el acceso a las funcionalidades exportadas.

6.10 Prácticas de desarrollo seguro

Con el objetivo de limitar las posibles vulnerabilidades en el software se proponen un conjunto de prácticas de desarrollo seguro definidas por el "Software Engineering Institute, Carniege Mellon" (37).

1. **Validación de datos**: Es necesario la validación de los datos de entrada de todas las fuentes de datos no confiables. La correcta validación de los datos de entrada permite eliminar la gran mayoría de las vulnerabilidades del software.
2. **Prestar atención a las advertencias del compilador**: La compilación del código se debe realizar empleando el nivel de advertencia más alto disponible por el compilador, y eliminar las advertencias mediante la modificación del código.
3. **Definir políticas de seguridad a nivel de diseño**: Se debe crear una arquitectura software que permite implementar políticas de seguridad robustas.
4. **Principio "Keep it simple"**: El diseño de la aplicación debe ser simple. La implementación de diseños complejos aumenta la probabilidad de errores en el desarrollo.
5. **Definir esquemas de bloqueo por defecto**: Las decisiones de permisos de acceso deben definir un esquema de denegación/bloqueo por defecto en vez de utilizar listas de exclusión en las que la decisión de bloqueo se realiza en base a unos principios definidos.
6. **Principio del mínimo privilegio:** Cualquier proceso creado por el sistema debe ejecutarse con el mínimo privilegio permitido para llevar a cabo su tarea.
7. **Filtrar datos enviados a otros sistemas:** Los datos enviados a otros subsistemas como pueden ser líneas de comando o bases de datos deben ser filtradas correctamente. La aplicación de esta práctica permite evitar la invocación de funcionalidades en otros sistemas evitando ataques de inyección de comandos.
8. **Defensa en profundidad:** Debe ser necesario gestionar los riesgos de seguridad de la aplicación disponiendo de múltiples estrategias defensivas mediante la implementación de una estructura de seguridad en múltiples niveles, de forma que si las defensas establecidas en una capa fallan pueda ser posible mitigar el error en otro nivel.
9. **Emplear técnicas efectivas de control de calidad**: Es necesario definir procedimientos de prueba en el proceso de desarrollo con el objetivo de permitir la identificación y eliminación de vulnerabilidades:
10. **Adoptar un estándar de codificación segura:** Se deben aplicar estándares de codificación segura tanto para el lenguaje de programación empleado como la plataforma empleada en el desarrollo.
11. **Definir requisitos de seguridad:** Debe ser necesaria la identificación de requisitos de seguridad en las fases tempranas del ciclo de vida de desarrollo.

Dadas las características del sistema y de las restricciones temporales asociadas al proyecto los puntos aplicables serán los siguientes:

- **Validación de datos**: Todas las funcionalidades del software dispondrán de rutinas de validación de los datos de entrada. Se prestará especial atención a aquellos módulos del software que permitan la interacción con la API HTTPCore.

- **Prestar atención a las advertencias del compilador**: La compilación tanto de la API como de la herramienta se llevara a cabo utilizando máximo nivel de advertencia disponible por parte del compilador integrado en el entorno de desarrollo Visual Studio 2010

- **Definir políticas de seguridad a nivel de diseño**: Se definirá una arquitectura modular que permite aplicar correctamente las estrategias de seguridad en cada uno de los módulos de la aplicación.

- **Defensa en profundidad:** Las rutinas de validación de datos de entrada se llevará a cabo en múltiples niveles. Tanto por parte de la aplicación como de aquellos módulos que permitan el acceso a la API HTTPcore.

- **Emplear técnicas efectivas de control de calidad:** Con el objetivo de ejecutar esta práctica se llevarán a cabo pruebas de análisis estático y pruebas de análisis dinámico sobre el código de la API HTTPCore y de los módulos que permiten el acceso a las funcionalidades de la misma.

- **Adoptar un estándar de codificación segura:** El estándar aplicado para el desarrollo del proyecto será el definido por los libros: “The Cert C secure coding standard” (38) y “Secure coding in C and C++ (39)”.

Conclusión

La definición de las funciones exportadas a través de la DLL, la convención de llamadas utilizada en la compilación, así como la definición del formato y tamaño de los tipos de datos utilizados como argumentos y valores de retorno de las funciones exportadas, especifican una Interfaz de aplicación binaria o ABI (Application Binary Interface) (17), que define como se debe realizar la invocación de las funciones de la API HTTPCore exportadas a través de la biblioteca de vínculo dinámico.

Por lo tanto conociendo la ABI es posible interactuar con la DLL desde cualquier lenguaje de programación.

7. Pruebas y validación.

El proceso de pruebas juega un papel fundamental en el ciclo de desarrollo de vida del software ya que permite la identificación fallos en la implementación, la comprobación de la calidad del producto así como la validación del cumplimiento de los requisitos planteados. Las pruebas constituyen por lo tanto un método más para permitir verificar y validar el software

A continuación se presenta el proceso de validación de requisitos especificando los procedimientos de prueba ejecutados.

7.1 Pruebas funcionales

RD-1

Título del requisito	La parte grafica de la aplicación ser desarrollada utilizando la tecnología Windows Forms.
Estado	Cumplido
Justificación	La aplicación implementa su parte gráfica utilizando la tecnología Windows Forms.
Procedimiento de prueba	1. Verificar que todos los componentes de la aplicación son implementados con la tecnología Windows Forms.
Resultado esperado.	N/A

RD-2

Título del requisito	La aplicación debe disponer de una arquitectura de diseño que permita reemplazar la tecnología empleada en el desarrollo de la parte gráfica de la aplicación.
Estado	Cumplido
Justificación	La aplicación implementa el patrón de arquitectura Modelo-Vista-Presentador, el cual permite reemplazar la implementación de la parte gráfica de la aplicación, ya que la lógica de presentación se encuentra en el Presentador, y éste accede a las funcionalidades de

	la vista a través de una interfaz de programación definida.
Procedimiento de prueba	1. Verificar que todos los presentadores interactúan con la implementación de la parte gráfica de la aplicación a través de una interfaz de programación.
Resultado esperado.	N/A

RD-3

Título del requisito	La aplicación debe ser desarrollada utilizando la versión 4.0 del framework .NET
Estado	Cumplido
Justificación	El proyecto se ha desarrollado utilizando la versión 4.0 del framework .NET
Procedimiento de prueba	1. Verificar en el entorno de desarrollo que la versión del framework .NET empleada es la 4.0
Resultado esperado.	N/A

RD-4

Título del requisito	Se debe permitir la interoperabilidad de la API HTTPCore con cualquier lenguaje de programación.
Estado	Cumplido
Justificación	Se ha creado una DLL que exporta todas las funcionalidades definidas en la especificación de requisitos.
Procedimiento de prueba	Verificar a través de una herramienta como dumpbin o Dependency Walker que la DLL exporta todas las funcionalidades definidas en la especificación de requisitos.
Resultado esperado.	N/A

RD-5

Título del requisito	El sistema debe hacer uso exclusivo de la API HTTPCore para todas las comunicaciones HTTP/s
Estado	Cumplido
Justificación	Cualquier tipo de comunicación HTTP se realiza utilizando la API.
Procedimiento de prueba	Verificar que no se hace uso de otra API para el establecimiento de comunicaciones HTTP.
Resultado esperado.	N/A

RF-1

Título del requisito	El wrapper debe exportar la funcionalidad de la API que permite el inicio conexiones HTTP/s
Estado	Cumplido
Justificación	La funcionalidad de inicio de conexiones definida en el wrapper, exportada en la DLL y accesible a través de la clase proxy HTTP establece correctamente una conexión HTTP/s
Procedimiento de prueba	Ejecución de pruebas unitarias sobre la funcionalidad. 1. Creación de un objeto HTTP. 2. Invocación del método de inicio de conexión InitHTTPConnectionHandle. 3. Verificación del resultado de la invocación.
Resultado esperado.	La invocación del método devuelve un valor entero distinto de -1

RF-2

Título del requisito	El wrapper debe exportar la funcionalidad que permite terminar conexiones HTTP/s establecidas previamente.
Estado	Cumplido
Justificación	La funcionalidad de terminación de conexiones definida en el

	wrapper, exportada en la DLL y accesible a través de la clase proxy HTTP ejecuta correctamente la terminación de la conexión.
Procedimiento de prueba	Ejecución de pruebas unitarias sobre la funcionalidad. 1. Creación de un objeto HTTP. 2. Invocación del método de inicio de conexión InitHTTPConnectionHandle. 3. Invocación del método de terminación de conexión HTTP endConnectionHandle 4. Verificación del resultado de la invocación.
Resultado esperado.	La invocación del método devuelve un valor entero distinto de -1

RF-3

Título del requisito	l wrapper debe exportar la funcionalidad que permite interactuar con los distintos campos de una petición HTTP.
Estado	Cumplido
Justificación	Las funcionalidades definidas en el wrapper, exportadas en la DLL y accesible a través de la clase proxy HTTPRequest permiten acceder o modificar los campos de una petición HTTP.
Procedimiento de prueba	Ejecución de pruebas unitarias sobre las funcionalidades de la clase HTTPRequest 1. Creación de un objeto HTTPRequest 2. Invocación de los métodos que permiten el acceso, modificación e inicialización los campos de una petición HTTP. 3. Verificación del valor de retorno tras la invocación.
Resultado esperado.	La invocación del método devuelve: • Un valor entero distinto de -1 en las funciones que devuelven un valor numérico. • Un valor distinto de *null* en las funciones que devuelven cadenas de texto.

RF-4

Título del requisito	El wrapper debe exportar la funcionalidad que permite obtener las respuestas de las peticiones HTTP utilizando los métodos de la API.
Estado	Cumplido
Justificación	Las funcionalidades definidas en el wrapper, exportadas en la DLL y accesible a través de la clase proxy HTTPResponse permiten acceder o modificar los campos de una respuesta HTTP.
Procedimiento de prueba	Ejecución de pruebas unitarias sobre las funcionalidades de la clase HTTPResponse 1. Creación de un objeto HTTPResponse 2. Invocación de los métodos que permiten el acceso, modificación e inicialización los campos de una respuesta HTTP. 3. Verificación del valor de retorno tras la invocación.
Resultado esperado.	La invocación del método devuelve: • Un valor entero distinto de -1 en las funciones que devuelven un valor numérico. • Un valor distinto de *null* en las funciones que devuelven cadenas de texto.

RF-5

Título del requisito	El wrapper debe exportar la funcionalidad que permite iniciar un proxy HTTP.
Estado	Cumplido
Justificación	Las funcionalidades definidas en el wrapper, exportadas en la DLL y accesible a través de la clase proxy HTTP permiten iniciar un proxy HTTP.
Procedimiento de prueba	Ejecución de pruebas unitarias sobre las funcionalidades de la clase HTTP 1. Creación de un objeto HTTP 2. Invocación del método InitProxy que permite iniciar un proxy HTTP. 3. Verificación del valor de retorno tras la invocación.

	b. Verificación manual ejecutando el comando *netstat /nab* y comprobando que existe un proxy escuchando en la dirección IP y puerto especificado.
Resultado esperado.	La invocación del método devuelve: • Un valor numérico distinto de -1

RF-6

Título del requisito	El wrapper debe exportar la funcionalidad que permite detener un proxy HTTP establecido previamente.
Estado	Cumplido
Justificación	Las funcionalidades definidas en el wrapper, exportadas en la DLL y accesible a través de la clase proxy HTTP permiten detener el proxy HTTP.
Procedimiento de prueba	Ejecución de pruebas unitarias sobre las funcionalidades de la clase HTTP 1. Creación de un objeto HTTP 2. Invocación del método initProxy que permite iniciar un proxy HTTP. 3. Invocación del método stopProxy que permite detener el proxy HTTP. 4. Verificación del valor de retorno tras la invocación. b. Verificación manual ejecutando el comando *netstat /nab* y comprobando que tras la detención del proxy se ha liberado el puerto en el que estaba escuchando para una IP concreta.
Resultado esperado.	La invocación del método devuelve: • Un valor numérico distinto de -1

RF-7

Título del requisito	El wrapper debe exportar la funcionalidad que permite establecer las opciones de configuración del proxy HTTP.
Estado	Cumplido
Justificación	Las funcionalidades definidas en el wrapper, exportadas en la DLL y accesible a través de la clase proxy HTTP permiten establecer las opciones del proxy HTTP.
Procedimiento de prueba	Ejecución de pruebas unitarias sobre las funcionalidades de la clase HTTP 1. Creación de un objeto HTTP 2. Invocación del método setHTTPProxyConfig especificando los posibles valores de configuración. 3. Verificación del valor de retorno tras la invocación.
Resultado esperado.	La invocación del método devuelve: • Un valor numérico distinto de -1

RF-8

Título del requisito	El wrapper debe exportar la funcionalidad que permite establecer las opciones de configuración del motor de comunicaciones
Estado	Cumplido
Justificación	Las funcionalidades definidas en el wrapper, exportadas en la DLL y accesible a través de la clase proxy HTTP permiten detener el proxy HTTP.
Procedimiento de prueba	Ejecución de pruebas unitarias sobre las funcionalidades de la clase HTTP 1 Creación de un objeto HTTP 2. Invocación del método setHTTPConfig especificando los posibles valores de configuración. 3. Verificación del valor de retorno tras la invocación.
Resultado esperado.	La invocación del método devuelve: • Un valor numérico distinto de -1

RF-9

Título del requisito	El wrapper debe exportar las funciones que permiten interactuar con las funcionalidades de callback definidas en la API.
Estado	Cumplido
Justificación	Las funcionalidades definidas en el wrapper, exportadas en la DLL y accesible a través de la clase proxy HTTP llevar a cabo la activación y desactivación de callbacks.
Procedimiento de prueba	Ejecución de pruebas unitarias sobre las funcionalidades de la clase HTTP **A. Pruebas de callbacks de envío sobre el motor de comunicaciones HTTP.** 1. Creación de un objeto HTTP 2. Creación de una función test de callback que permita modificar los campos de una petición HTTP antes de su envío por parte del motor de comunicaciones HTTP. 3. Verificar que la petición ha sido modificada. **B. Pruebas de callbacks de recepción sobre el motor de comunicaciones HTTP.** 1. Creación de un objeto HTTP 2. Creación de una función test de callback que permita modificar los campos de una respuesta HTTP tras su recepción por parte del motor de comunicaciones HTTP. 3. Verificar que la respuesta ha sido modificada. **C. Pruebas de callbacks de envío sobre el proxy HTTP.** 1. Creación de un objeto HTTP 2. Creación de una función test de callback que permita modificar los campos de una petición HTTP antes de su envío por parte del proxy. 3. Verificar que la petición ha sido modificada **D. Pruebas de callbacks de recepción. sobre el proxy HTTP.** 1. Creación de un objeto HTTP 2. Creación de una función test de callback que permita modificar los campos de una respuesta HTTP antes de su recepción por parte del proxy. 3. Verificar que la respuesta ha sido modificada.

Resultado esperado.	La invocación del método devuelve: • Los callbacks de test realizan correctamente la modificación tanto de peticiones como de respuestas.

RF-10

Título del requisito	La aplicación debe permitir iniciar un proxy de interceptación en una dirección IP y puerto especificado por el usuario.
Estado	Cumplido
Justificación	La funcionalidad de inicio de proxy accesible desde la interfaz gráfica de la aplicación en la sección "Proxy Options" permite el inicio del proxy en una dirección IP y puerto especificado por el usuario.
Procedimiento de prueba	A. 1. Acceder a la sección de opciones de proxy. 2. Seleccionar una IP del equipo. 3. Definir el puerto en el que debe escuchar el proxy. 4. Seleccionar el botón "Start proxy". B. Verificación manual ejecutando el comando *netstat /nab* y comprobando que existe un proxy escuchando en la dirección IP y puerto especificado por el usuario.
Resultado esperado.	• La aplicación inicia el proxy en la dirección IP y puerto especificado por el usuario. • La aplicación devuelve un mensaje de error si no se ha podido establecer el proxy.

RF-11

Título del requisito	La aplicación debe permitir detener un proxy HTTP iniciado previamente.
Estado	Cumplido
Justificación	La funcionalidad de inicio de proxy accesible desde la interfaz gráfica

	de la aplicación en la sección "Proxy Options" permite detener la ejecución del proxy.
Procedimiento de prueba	A. 1. Acceder a la sección de opciones de proxy. 2. Selección del botón "Stop proxy". B. Verificación manual ejecutando el comando *netstat /nab* y comprobando que ya no existe un proxy escuchando en la dirección IP y puerto definido previamente.
Resultado esperado.	• La aplicación detiene el proxy HTTP.

RF-12

Título del requisito	La aplicación debe permitir el registro de las peticiones HTTP que se envían a través del proxy.
Estado	Cumplido
Justificación	La aplicación muestra las peticiones que han sido redirigidas a través del proxy a través de un componente de la interfaz gráfica.
Procedimiento de prueba	1. Verificar que el proxy HTTP está activo. 2. Configurar un navegador web para establecer una comunicación a través del proxy de la aplicación en una dirección IP y puerto en el que se encuentra escuchando el proxy. Enviar una petición HTTP a través del navegador. 3. Verificar que la petición ha sido registrada en la aplicación.
Resultado esperado.	La aplicación registra correctamente la petición HTTP enviada a través del proxy. La aplicación registra correctamente la respuesta HTTP de la petición enviada.

RF-13

Título del requisito	La aplicación debe permitir la activación o desactivación del módulo de interceptación de peticiones HTTP/s.
Estado	Cumplido
Justificación	La aplicación permite activar o desactivar las capacidades de interceptación de peticiones de HTTP enviadas a través del proxy.
Procedimiento de prueba	A. Activación de la funcionalidad de interceptación. 1. Verificar que el proxy HTTP se encuentra iniciado. 2. Seleccionar el botón "Intercept On" accesible a través de la sección "Intercept" en la interfaz gráfica. 3. Configurar un navegador web para establecer una comunicación a través del proxy de la aplicación en la dirección IP y puerto en el que se encuentra escuchando el proxy. 4. Envío de una petición HTTP a través del navegador web. 5. Verificar que la petición se intercepta y es mostrada al usuario para la elección del bloqueo o redirección de la misma. B. Desactivación de la funcionalidad de interceptación. 1. Verificar que el proxy HTTP se encuentra iniciado. 2. Seleccionar el botón "Intercept Off" accesible a través de la sección "Intercept" en la interfaz gráfica. 3. Configurar un navegador web para establecer una comunicación a través del proxy de la aplicación en la dirección IP y puerto en el que se encuentra escuchando el proxy. 4. Envío de una petición HTTP a través del navegador web. 5. Verificar que la petición no es interceptada, sin embargo sí que es registrada.
Resultado esperado.	La aplicación registra las peticiones HTTP cuando la funcionalidad de interceptación está desactivada. La aplicación muestra la petición interceptada al usuario para llevar a cabo la elección de redirección o bloqueo de la misma.

RF-14

Título del requisito	La aplicación debe permitir bloquear/rechazar las peticiones HTTP que han sido interceptadas por el proxy.
Estado	Cumplido
Justificación	La aplicación permite bloquear las peticiones HTTP que se envían a través del proxy, cuando la funcionalidad de interceptación está activa, y se pulsa el botón de bloqueo.
Procedimiento de prueba	1. Seguir los pasos del procedimiento de prueba A definido para el requisito RF-13. 2. Seleccionar el botón "Block". 3. Verificar que el navegador web recibe muestra una pantalla de error indicando que la petición ha sido bloqueada.
Resultado esperado.	La aplicación bloquea correctamente la petición interceptada.

RF-15

Título del requisito	La aplicación debe permitir la redirección de peticiones HTTP enviadas a través del proxy hacia un sistema de destino.
Estado	Cumplido
Justificación	La aplicación permite redirigir las peticiones HTTP que se envían a través del proxy, cuando la funcionalidad de interceptación está activa, y se pulsa el botón "Forward".
Procedimiento de prueba	1. Seguir los pasos del procedimiento de prueba A definido para el requisito RF-13. 2. Seleccionar el botón "Forward". 3. Verificar que la petición es registrada en la aplicación y que el navegador obtiene la página solicitada.
Resultado esperado.	La aplicación redirige correctamente las peticiones que han sido interceptadas por el proxy.

RF-16

Título del requisito	La aplicación debe permitir la modificación de las peticiones HTTP interceptadas.
Estado	Cumplido
Justificación	La funcionalidad de modificación de peticiones interceptadas se ejecuta correctamente.
Procedimiento de prueba	1. Seguir los pasos del procedimiento de prueba A definido para el requisito RF-13. 2. Modificar la URL de la petición interceptada estableciendo un valor erróneo. 3. Seleccionar el botón "Forward". 4. Verificar que la modificación de la petición se ha realizado correctamente, y que la respuesta del servidor web refleja la modificación de la petición.
Resultado esperado.	La petición interceptada es modificada correctamente.

RF-17

Título del requisito	La aplicación debe disponer de un módulo que permita el reenvío de peticiones HTTP a un host y puerto concreto.
Estado	Cumplido
Justificación	La funcionalidad de reenvío de peticiones se ejecuta correctamente, permitiendo el envío de peticiones HTTP construidas por el usuario.
Procedimiento de prueba	1. Acceder a la sección "Repeater". 2. Configurar la dirección IP o nombre de host del sistema objetivo. 3. Configurar el puerto al que será enviado la petición. 4. Definir las cabeceras y cuerpo de una petición HTTP. 5. Seleccionar el botón "Send". 6. Verificar que se visualiza en la interfaz gráfica la respuesta asociada a la petición realizada.
Resultado esperado.	Tras seleccionar el botón "Send" se visualiza a través de la interfaz gráfica la respuesta HTTP de la petición enviada.

RF-18

Título del requisito	La aplicación debe permitir realizar procedimientos de fuzzing sobre peticiones HTTP.
Estado	Cumplido
Justificación	La aplicación permite la ejecución de pruebas de fuzzing mediante el envío iterativo de peticiones HTTP en las que se modifican de forma automática ciertos parámetros seleccionados.
Procedimiento de prueba	1. Acceder a la sección "Fuzzer". 2. Configurar la dirección IP o nombre de host del sistema objetivo en la subsección Target. 3. Configurar el puerto al que será enviado la petición en la subsección Target 4. Definir las cabeceras y cuerpo de una petición HTTP en la subsección "Parameters". 5. Seleccionar una sección de las cabeceras o cuerpo de la petición HTTP y seleccionar el botón "Add Parameter". 6. Seleccionar el botón "Add Parameter". 7. Acceder a la subsección "Payloads". 8. A. Definir un payload de tipo numérico ("Numbers"). B. Definir un payload de tipo lista ("List"). 9. Seleccionar el botón "Start Attack" Repetir el procedimiento para cada uno de los procedimientos de fuzzing definidos en la aplicación.
Resultado esperado.	La aplicación ejecuta correctamente el proceso de fuzzing y muestra al usuario los resultados, visualizando todas las peticiones HTTP enviadas así como sus respuestas asociadas.

RF-20

Título del requisito	La aplicación debe permitir la compilación de scripts.
Estado	Cumplido
Justificación	La aplicación permite la creación y compilación de scripts que utilizan la API HTTPCore a través de las clases proxy que permiten el acceso a las funcionalidades exportadas a través de la DLL.
Procedimiento	1. Acceder a la sección "Scripting IDE".

de prueba	2. Seleccionar el botón "New Script". 3. Crear un script de ejemplo que implemente la clase abstracta "Custom Script". 4. Implementar los métodos de la clase abstracta. 5. Seleccionar el botón "Compile". 6. Verificar que los resultados de compilación son mostrados al usuario. Repetir el procedimiento de prueba creando un script de tipo "Callback" que implemente las interfaces ICallbackRequest y ICallbackResponse.
Resultado esperado.	La aplicación ejecuta la funcionalidad de compilación y muestra un mensaje de error si el código proporcionado no es correcto. La aplicación muestra un mensaje indicando que el resultado de la compilación es correcto, si el código del script no tiene ningún fallo.

RF-21

Título del requisito	La aplicación debe permitir la ejecución de los scripts compilados a través de la aplicación.
Estado	Cumplido
Justificación	La aplicación implementa correctamente la funcionalidad de ejecución de scripts, mostrando al usuario los resultados de ejecución..
Procedimiento de prueba	1. Acceder a la sección "Scripting IDE". 2. Abrir un script seleccionando el botón "Open Script". 3. Compilar el script seleccionado el botón "compile". 4. Pulsar el botón "Run". 5. Verificar que los resultados de la ejecución del script son mostrados a través de la interfaz gráfica.
Resultado esperado.	La aplicación muestra los resultados de ejecución de un script.

RF-22

Título del requisito	La aplicación debe permitir la activación y desactivación de la ejecución de los scripts de callback.
Estado	Cumplido
Justificación	La aplicación implementa correctamente la funcionalidad de activación y desactivación de callbacks.
Procedimiento de prueba	A. *Ejecución de scripts de callback cbClientRequest* 1. Acceder a la sección "Scripting IDE". 2. En el repositorio de script de callbacks seleccionar un script de prueba de tipo (cbClientRequest, que representa aquellos scripts que permiten modificar las peticiones HTTP antes de su envío por parte del motor de comunicaciones). 3. Seleccionar el botón "Activate Script" 4. Verificar que se produce la modificación del comportamiento del motor de comunicaciones. B. *Ejecución de scripts de callback cbClientResponse* 1. Acceder a la sección "Scripting IDE". 2. En el repositorio de script de callbacks seleccionar un script de prueba de tipo cbClientResponse, que representa aquellos scripts que permiten modificar las respuestas HTTP tras su recepción. por parte del motor de comunicaciones). 3. Seleccionar el botón "Activate Script" 4. Verificar que se produce la modificación del comportamiento del motor de comunicaciones. C. *Ejecución de scripts de callback cbProxyRequest* 1. Acceder a la sección "Scripting IDE". 2. En el repositorio de script de callbacks seleccionar un script de prueba de tipo (cbProxyRequest, que representa aquellos scripts que permiten modificar las peticiones HTTP antes de su envío por parte del proxy). 3. Seleccionar el botón "Activate Script" 4. Verificar que se produce la modificación del comportamiento del proxy HTTP. D. *Ejecución de scripts de callback cbProxyResponse* 1. Acceder a la sección "Scripting IDE". 2. En el repositorio de script de callbacks seleccionar un script de prueba de tipo (cbProxyResponse, que representa aquellos scripts que permiten modificar las respuestas HTTP tras su recepción por parte del proxy.).

	3. Seleccionar el botón "Activate Script" 4. Verificar que se produce la modificación del comportamiento del proxy HTTP.
Resultado esperado.	Los scripts de callback modifican tanto el comportamiento del motor de comunicaciones HTTP como del proxy.

RF-23

Título del requisito	La aplicación debe disponer de un repositorio de scripts que permita el almacenamiento de los mismos.
Estado	Cumplido
Justificación	La aplicación implementa dispone de un repositorio de scripts que permite almacenar los scripts desarrollados por parte del usuario.
Procedimiento de prueba	1. Verificar que la interfaz gráfica de la aplicación dispone de un componente que permite visualizar lo scripts del repositorio. 2. Verificar que la aplicación carga los scripts del repositorio al inicio. 3. Verificar que es posible abrir cualquier script del repositorio.
Resultado esperado.	La aplicación implementa el repositorio de scripts, permitiendo cargar los scripts almacenados. La aplicación permite abrir cualquier script del repositorio.

RNF-1

Título del requisito	El idioma de la aplicación debe ser el inglés.
Estado	Cumplido
Justificación	Todos los mensajes de la aplicación se muestran en inglés.
Procedimiento de prueba	Verificar que todos los nombres de componentes gráficos, así como de los mensajes de error devueltos por la aplicación están escritos en inglés.
Resultado esperado.	N/A

RNF-2

Título del requisito	La aplicación debe estar desarrollada en C#.
Estado	Cumplido
Justificación	Toda la aplicación se ha desarrollado utilizando el lenguaje de programación C#.
Procedimiento de prueba	N/A
Resultado esperado.	N/A

RI1

Título del requisito	La interfaz gráfica de la aplicación debe ser adaptable al espacio del escritorio. Igualmente los componentes de la interfaz deben disponer de capacidades de redimensionamiento.
Estado	Cumplido
Justificación	La aplicación hace uso de componentes gráficos que permiten la redimensión de los mismos.
Procedimiento de prueba	1. Verificar que se permite redimensionar los componentes gráficos de la aplicación.
Resultado esperado.	N/A

RC2

Título del requisito	No deben existir fugas de memoria asociados a la creación y no liberación de los objetos creados a través del código no administrado de la API.
Estado	Cumplido
Justificación	Las clases proxy establecen un mecanismo a través del cual se produce la gestión automática de la liberación de memoria.

Procedimiento de prueba	1. Verificar que todas las clases proxy implementan correctamente el mecanismo de liberación de objetos mediante la invocación de llamadas de destrucción de objetos exportadas en la DLL.
Resultado esperado.	La aplicación no dispone de fugas de memoria.

REV1

Título del requisito	La aplicación debe disponer de una arquitectura modular, de forma que permita de forma flexible la integración de nuevas funcionalidades.
Estado	Cumplido
Justificación	La aplicación hace uso de la arquitectura de diseño Modelo-Vista-Presentador, permitiendo la implementación e integración de nuevos módulos de forma sencilla.
Procedimiento de prueba	N/A
Resultado esperado.	N/A

RPR1

Título del requisito	Tanto la aplicación como la documentación deben estar finalizadas y entregadas antes del día 9 de Septiembre de 2013.
Estado	Cumplido
Justificación	La entrega de esta memoria evidencia el cumplimiento del requisito.
Procedimiento de prueba	N/A
Resultado esperado.	N/A

RES1

Título del requisito	El wrapper debe validar correctamente los datos de entrada para cada funcionalidad concreta.
Estado	Cumplido
Justificación	Todas las funcionalidades exportadas a través de la DLL realizan la validación del formato de los parámetros de entrada.
Procedimiento de prueba	Ejecución de pruebas unitarias sobre cada una de las funciones exportadas verificando el formato de entrada de cada uno de los parámetros.
Resultado esperado.	La aplicación valida correctamente los parámetros de entrada proporcionados.

RES2

Título del requisito	El wrapper debe comprobar la validez de los punteros que referencian los objetos creados por la API.
Estado	Cumplido
Justificación	Todas las funcionalidades exportadas a través de la DLL realizan la validación de los punteros que permiten referenciar objetos, de forma que no se permita ejecutar cualquier procedimiento cuando el puntero especificado en la entrada sea nulo.
Procedimiento de prueba	Ejecución de pruebas unitarias sobre cada una de las funciones exportadas verificando el comportamiento de las mismas cuando se pasa como parámetro de entrada un puntero nulo.
Resultado esperado.	La aplicación valida correctamente los parámetros de entrada proporcionados.

7.2 Prueba de seguridad

Debido a la materialización del riesgo RK-05 la duración de las pruebas de seguridad ha tenido que ser limitada con el objetivo de cumplir el requisito temporal de finalización del proyecto. La revisión de las pruebas siguiendo el tiempo estimado inicialmente se propondrá como trabajo futuro.

La realización de las pruebas se centrará en la API HTTPCore y los wrappers al tratarse de código implementado utilizando los lenguajes C y C++. Dado que estos lenguajes permiten el acceso a memoria a muy bajo nivel, los programas desarrollados pueden estar sujetos fallos de corrupción de memoria que pueden suponer vulnerabilidades en el software.

Pruebas de análisis estático

El análisis estático es una técnica que permiten llevar a cabo un análisis del código fuente sin llegar a ejecutar el software. Este tipo de técnicas son ampliamente utilizadas por muchas herramientas y en muchos ámbitos y propósitos diferentes. Un ejemplo claro son los analizadores léxicos y sintácticos o los correctores gráficos en herramientas de edición de texto.

El propósito de la ejecución de este tipo de pruebas en el proyecto se centra principalmente en la identificación de fallos en el software.

Generalmente la ejecución de este tipo de pruebas se realiza mediante la utilización de herramientas automáticas que disponen de una serie de reglas que son verificadas contra el código fuente. Como resultado se obtiene un conjunto de advertencias acerca de posibles debilidades, las cuales deben ser verificadas manualmente, ya que normalmente generan una gran cantidad de falsos positivos.

A continuación se exponen las herramientas empleadas para la ejecución de las pruebas y se comentan los resultados obtenidos:

Pruebas de análisis estático con RATS (Rough Auditing Tool For Security).

RATS (Rough Auditing Tool For Security) (40) es una herramienta automática de análisis estático que permiten la identificación de vulnerabilidades en programas desarrollados en C, C++, Python, Perl y PHP, permitiendo la identificación de fallos típicos como pueden ser los desbordamiento de buffer (41), condiciones de carrera (42), fallos TOC TOU (Time of check Time of Use) (43).

Los resultados de ejecución de RATS sobre el código de la API HTTPCore así como de los wrappers desarrollados en C suponen un total 87 advertencias, de las cuales 85 se tratan de falsos positivos. Por lo tanto se identificaron 2 vulnerabilidades que implican un riesgo de seguridad.

En la siguiente tabla se detallan los fallos identificados y se comenta la solución propuesta para su mitigación.

ID	Tipo de vulnerabilidad	Comentario	Solución
1	Desbordamiento de buffer	Utilización de la función de **`sprintf`** sin verificar el tamaño del buffer destino.	Reemplazo de la función **`sprintf`** por la función **`sprintf_s`** verificando el tamaño del buffer destino.
2	Desbordamiento de buffer	Utilización de la función **`swscanf`** sin verificar el tamaño del buffer destino.	Reemplazo de la función **`swscanf`** por la función **`wscanf_s`** y verificar el tamaño del buffer destino.

Tabla 6: Vulnerabilidades identificadas mediante la ejecución de pruebas de análisis estático.

Pruebas de análisis dinámico.

Las pruebas de análisis dinámico permiten llevar a cabo la identificación de vulnerabilidades mediante la ejecución del software, generalmente utilizando de técnicas de fuzzing. Este tipo de pruebas permiten analizar el comportamiento del software cuando se proporcionan valores de entrada malformados o inesperados.

La ejecución de este tipo de pruebas se centrará sobre la funcionalidad de envío de peticiones HTTP y recepción de respuestas a través del proxy HTTP proporcionado por la API. A continuación se detalla el procedimiento de prueba así como los resultados obtenidos.

Prueba AD-PR1 – La ejecución de esta prueba consiste en la utilización de un script desarrollado a medida, que permite establecer una conexión contra el proxy HTTP proporcionado por la API, para a continuación enviar una gran cantidad de peticiones HTTP contra servidores web aleatorios.

En la ejecución de esta prueba se enviaron 100.000 peticiones llevando a cabo la modificación de los valores de las cabeceras HTTP mediante la variación del tamaño de las mismas. Para la realización de las pruebas se utilizó un script desarrollado en Python por el equipo de desarrollo de Tarlogic Security S.L.

Tras la realización de las pruebas no se identificó ningún fallo en las rutinas de envío y recepción de peticiones.

7.3 Evaluación heurística.

La evaluación heurística es una técnica de análisis de usabilidad que permite analizar la conformidad de una interfaz gráfica de acuerdo a una serie de principios heurísticos establecidos. El objetivo de llevar a cabo este tipo de análisis es medir la calidad de la interfaz gráfica de una aplicación o sistema interactivo con respecto a distintos aspectos como pueden ser la facilidad de uso y aprendizaje. Los resultados de la evaluación heurística permiten por lo tanto identificar problemas de usabilidad.

Para la realización de la evaluación heurística se seguirán los 10 principios propuestos por Jakob Nielsen y Robert Molich (44).

1. **Visibilidad del estado del sistema**

Toda acción realizada por el usuario de la aplicación tiene como resultado una retroalimentación apropiada, ya sea mediante la activación o desactivación de los controles de la interfaz gráfica o por que el resultado de la acción es mostrado en la pantalla. A diferencia de otros proxies de interceptación que disponen de una interfaz recargada y con gran cantidad de elementos se ha optado por la sencillez ofreciendo una interfaz simple e intuitiva.

2. **Similitud entre el mundo real y el sistema.**

A pesar de que las funcionalidades y términos empleados por el sistema requieren un conocimiento previo por parte del usuario (ya que se trata de una herramienta para la realización de auditorías de seguridad) los distintos elementos de la interfaz gráfica disponen de iconos representativos que permiten identificar la acción que se va a realizar.

3. **Control y libertad para el usuario.**

Las funcionalidades de la aplicación disponen de elementos que permiten al usuario salir de un estado indeseado. Un ejemplo claro son las funciones de deshacer y rehacer disponibles en el editor de texto que permite la creación de scripts o los distintos botones que permiten la activación y desactivación de funcionalidades de forma sencilla.

4. **Consistencia y estándares**

Las distintas funcionalidades de la aplicación se encuentran agrupadas en distintas secciones con el objetivo de evitar cualquier tipo de confusión posible por parte del usuario.

5. **Recuperación de errores.**

Todos los posibles mensajes de error de la aplicación se presentan utilizando un lenguaje sencillo, explicando la causa del error y aquellas pasos que debe realizar el usuario para ejecutar la acción de forma correcta.

6. **Prevención de errores**

Con el objetivo de evitar una situación que pueda conducir a un error la interfaz gráfica lleva a cabo la activación o desactivación de los componentes gráficos evitando la interacción con los mismos hasta que el estado del sistema así lo permita. Un ejemplo claro de este principio es la activación sucesiva de elementos gráficos en el formulario que permite definir los parámetros de ejecución del módulo de fuzzing.

7. **Reconocimiento antes de cancelación.**

La visibilidad de todas las secciones de la aplicación evita que el usuario deba recordar información asociada a diálogos anteriores.

8. **Flexibilidad y eficiencia de uso.**

Todos los componentes de la interfaz gráfica están diseñados con el objetivo de permitir su redimensión de forma que el usuario pueda decidir cómo posicionar los componentes en función de sus necesidades. La interfaz gráfica también dispone de formas de interacción que permiten mejorar la eficiencia de uso por parte de usuarios expertos. Un ejemplo claro de este supuesto es la disponibilidad de atajos de teclado para la realización de acciones. El atajo de teclado (Control +F) permite desplegar el dialogo de búsqueda en la vista de edición de scripts.

9. **Estética de diálogos y diseño minimalista**

Todas las secciones de la aplicación disponen de los elementos relevantes para la ejecución de las acciones disponibles para cada sección. Los distintos elementos de la interfaz están bien estructurados y siguen un orden lógico a lo largo de todas las vistas

10. **Ayuda general y documentación.**

La aplicación dispone de secciones de ayuda que permiten informar al usuario acerca de la realización de acciones, o del funcionamiento de las mismas.

8. Conclusiones y trabajo futuro.

8.1 Conclusiones

Se ha logrado de forma satisfactoria el cumplimiento de los objetivos definidos inicialmente. Tras el análisis de las alternativas disponibles para permitir la interoperabilidad de las funcionalidades de la API HTTPCore con cualquier lenguaje de programación, se ha optado por la creación de una biblioteca de vínculo dinámico a través de la cual se permite el acceso a las funcionalidades públicas de la API.

Si bien se ha demostrado la interoperabilidad de la misma desde el lenguaje de programación C# a través del desarrollo de clases que implementan el patrón de diseño "proxy". Es posible seguir el mismo esquema de implementación para permitir la interoperabilidad con cualquier otro lenguaje. La sección de manual técnico especifica todas las funciones que han sido exportadas en la DLL y que es necesario invocar para permitir la interacción con la biblioteca de vínculo dinámico.

El objetivo de creación de una herramienta que permita asistir en la ejecución de pruebas de seguridad durante el desarrollo de una auditoría ha sido satisfactorio y se han logrado implementar todas aquellas funcionalidades definidas. Así mismo se ha logrado definir una arquitectura de diseño modular que permite la integración de nuevas funcionalidades de forma flexible.

La ejecución de las pruebas realizadas demuestra que los requisitos analizados han sido implementados y validados correctamente. La entrega del software y la documentación, así como la validación por parte del "Tarlogic Security S.L" demuestra la consecución de todos los objetivos del proyecto.

8.2 Trabajo futuro

El proyecto ha sido pensado como un punto de partida para otro proyecto de mayor tamaño en Tarlogic Security S.L.

Algunas de las posibles medidas de ampliación y mejora, son las siguientes:

- Integrar un motor de almacenamiento que permita almacenar toda la información recolectada durante la ejecución de una auditoría.
- Proporcionar un esquema colaborativo de la herramienta.
- Inclusión de nuevos tipos de payload.
- Desarrollar scripts que permitan realizar pruebas concretas sobre cada uno de los puntos de la "Guía de pruebas de OWASP" (1).
- Soporte de visualización de peticiones HTTP a través de esquema de grafo a través del cual sea posible ver de forma gráfica como es la interacción entre los distintos recursos de una aplicación web.
- Definir funcionalidades que permitan modificar la codificación de los datos (Base64, URL Encoding).
- Definir funcionalidades que permitan ejecutar de forma rápida operaciones de cifrado a través de alguno de los siguientes algoritmos: MD5, SHA-1.
- Publicar la herramienta en OWASP (Open Web Application Security Project) cuando la herramienta haya superado una fase de pruebas por parte de los usuarios y el proyecto tenga una mayor madurez.
- Llevar a cabo la repetición de pruebas de seguridad, disponiendo de mayor tiempo para la ejecución de las mismas.
- Ofrecer soporte multiplataforma, creando una biblioteca compartida tanto para entornos Linux, como Mac OS X.

ANEXO

A. Manual técnico.

A.1 Especificación de funciones de la API exportadas en la biblioteca de vínculo dinámico a través de los wrappers.

A continuación se detalla el conjunto de funciones que han sido exportados en la biblioteca de vínculo dinámico. El propósito de documentar estas funciones es permitir la implementación de clases proxy que permitan el acceso a las funcionalidades de la API utilizando cualquier lenguaje de programación.

A.1.1 Funciones exportadas a través de la biblioteca de vínculo dinámico asociadas a la clase HTTPAPI de la API HTTPCore.

A continuación se describen todas las funciones de la API HTTPCore exportadas en la biblioteca de vínculo dinámico, así como los métodos de las clases proxy en C# que permiten la interacción con dichas funciones.

Método original de la API desarrollada en C++	`HTTPAPI::HTTPAPI()`
Función de la API exportada en la DLL	`HTTPAPI * newHTTPAPI()` **Valor de retorno:** Devuelve un una referencia a un objeto HTTPAPI.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`new HTTP()`
Descripción	Creación de un objeto HTTPAPI, punto de entrada para la interacción con las funcionalidades de la API HTTPCore.

Método original de la API desarrollada en C++	`HTTPAPI::~HTTPAPI()`
Función de la API exportada en la DLL	`int destroyHTTPAPI(int * HTTPAPI_HANDLE)` **Parámetro de entrada:** El parámetro de entrada a la función es un *handle* o

	referencia a un objeto creado previamente mediante la invocación de la función newHTTPAPI **Valor de retorno:** La función devuelve valor 1 si fue posible llevar a cabo la destrucción del objeto HTTPAPI, o valor -1 si la referencia al objeto no era válida.
Método para la invocación de la funcionalidad desde las clase proxy en C#	~HTTP()
Descripción	Destrucción de un objeto HTTPAPI

Método original de la API desarrollada en C++	`int HTTPAPI::InitHTTPConnectionHandle(`
	`wchar_t * lpHostName,`
	`int port,`
	`int ssl)`

Función de la API exportada en la DLL	`int HTTPInitConnectionHandle`
	`(HTTPAPI * HTTPAPI_HANDLE,`
	`wchar_t * lpHostName,`
	`int port,`
	`int ssl)`
	
	Parámetros de entrada:
	
	El parámetro *lpHostName* indica el nombre del host y se especifica como una cadena de texto “wide char” (2 bytes) con codificación UNICODE.
	El parámetro port permite especificar el puerto destino.
	El parámetro ssl indica si la comunicación se realiza a través de un canal de comunicaciones seguro HTTPs. El valor 1 indica comunicación HTTPs el valor 0 indica comunicación HTTP
	
	Valor de retorno:
	
	La función devuelve valor -1 si no se pudo establecer la comunicación con el host destino o valor distinto de -1 si se pudo establecer la comunicación con el sistema destino.
	Un valor distinto de -1 es referencia o descriptor de la comunicación establecida.

Método para la invocación	`int HTTP:InitHTTPConnectionHandle`

de la funcionalidad desde las clase proxy en C#	`(string hostname,`
	`int port,`
	`int ssl)`

Descripción	Funciones que permiten establecer una conexión HTTP/s con un host destino.

Método original de la API desarrollada en C++	`int HTTPAPI::EndHTTPConnectionHandle(` `HTTPHANDLE)`

Función de la API exportada en la DLL	`int EndHTTPConnectionHandle(`
	`HTTPAPI * HTTPAPI_HANDLE,`
	`int HTTPHandle)`
	
	Parámetros de entrada:
	
	El parámetro HTTPAPIHANDLE es una referencia al objeto HTTPAPI creado mediante la invocación de la función `newHTTPAPI`.
	El parámetro HTTPHandle es una referencia o descriptor de la comunicación establecida previamente mediante la invocación de la función `HTTPInitConnectionHandle.`
	
	Valor de retorno:
	
	La función devuelve valor 1 si la conexión referenciada por HTTPHandle fue terminada correctamente, valor -1 si `HTTPAPI_HANDLE` es una referencia no válida, o valor 0 si el descriptor de fichero de la comunicación establecida (HTTPHandle) es incorrecto.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`int endConnectionHandle(int httpHandle)`
Descripción	Función que permite terminar una conexión HTTP establecida previamente.

Método original de la API desarrollada en C++	`int HTTPAPI::SetHTTPConfig(`
	`int HTTPHandle,`
	`enum HttpOptions opt,`
	`int parameter);`

Función de la API exportada en la DLL	`SetHTTPConfigInt(` `HTTPAPI * HTTPAPI_HANDLE,`

	`int HTTPHandle,`
	`unsigned int options,`
	`int parameter)`
	
	Parámetros de entrada:
	
	El parámetro HTTPAP_IHANDLE es una referencia al objeto HTTPAPI creado mediante la invocación de la función `newHTTPAPI`.
	El parámetro HTTPHandle es una referencia o descriptor de la comunicación establecida previamente mediante la invocación de la función `HTTPInitConnectionHandle.`
	El parámetro *opt* permite especificar el tipo de opción del motor de comunicaciones HTTP que se desea modificar. Los posibles valores son:
	
	`ConfigProxyHost = 0x00,`
	`ConfigProxyPort = 0x01,`
	`ConfigProxyUser = 0x02,`
	`ConfigProxyPass = 0x04,`
	`ConfigAdditionalHeader = 0x08,`
	`ConfigCookie = 0x10,`
	`ConfigUserAgent = 0x20,`
	`ConfigProtocolversion = 0x40,`
	`ConfigMaxDownloadSpeed = 0x80,`
	`ConfigHTTPHost = 0x100,`
	`ConfigHTTPPort = 0x200,`
	`ConfigAsyncronousProxy = 0x400,`
	`ConfigProxyInitialized = 0x800,`
	`ConfigSSLConnection = 0x1000,`
	`ConfigMaxDownloadSize = 0x2000,`
	`ConfigCookieHandling = 0x4000,`
	`ConfigAutoredirect = 0x8000,`
	`ConfigDisconnectConnection = 0x10000`
	
	El parámetro *parameter* permite especificar el valor que se desea asignar a la opción especificada a través del parámetro opt.
	
	Valor de retorno:
	
	La función devuelve valor 1 si la opción se estableció correctamente o valor 0 si la referencia (handle) al objeto HTTPAPI es inválida.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`int setHTTPConfig(int HTTPHandle, Enum HttpOptions, int parameter)`

Descripción	Funcionalidad que permite establecer las opciones de configuración del motor de comunicaciones HTTP.

Método original de la API desarrollada en C++	`int HTTPAPI::SetHTTPConfig(`
	`HTTPHANDLE HTTPHandle,`
	`enum HttpOptions opt,`
	`const wchar_t * parameter);`

Función de la API exportada en la DLL	`SetHTTPConfigString(`
	`HTTPAPI * HTTPAPI_HANDLE,`
	`int HTTPHandle,`
	`unsigned int options,`
	`const wchar_t * parameter)`
	
	Parámetros de entrada:
	
	El parámetro HTTPAPI_HANDLE es una referencia al objeto HTTPAPI creado mediante la invocación de la función `newHTTPAPI`.
	El parámetro HTTPHandle es una referencia o descriptor de la comunicación establecida previamente mediante la invocación de la función `HTTPInitConnectionHandle`.
	El parámetro *opt* permite especificar el tipo de opción del motor de comunicaciones HTTP que se desea modificar. Los posibles valores son:
	
	`ConfigProxyHost = 0x00,`
	`ConfigProxyPort = 0x01,`
	`ConfigProxyUser = 0x02,`
	`ConfigProxyPass = 0x04,`
	`ConfigAdditionalHeader = 0x08,`
	`ConfigCookie = 0x10,`
	`ConfigUserAgent = 0x20,`
	`ConfigProtocolversion = 0x40,`
	`ConfigMaxDownloadSpeed = 0x80,`
	`ConfigHTTPHost = 0x100,`
	`ConfigHTTPPort = 0x200,`
	`ConfigAsyncronousProxy = 0x400,`
	`ConfigProxyInitialized = 0x800,`
	`ConfigSSLConnection = 0x1000,`
	`ConfigMaxDownloadSize = 0x2000,`
	`ConfigCookieHandling = 0x4000,`
	`ConfigAutoredirect = 0x8000,`
	`ConfigDisconnectConnection = 0x10000`

	El parámetro *parameter* permite especificar el valor que se desea asignar a la opción especificada.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`int setHTTPConfig(int HTTPHandle, Enum HttpOptions, string parameter)`
Descripción	Funcionalidad que permite establecer las opciones de configuración del motor de comunicaciones HTTP.

Método original de la API desarrollada en C++	`int HTTPAPI::InitHTTPProxy(`
	`const wchar_t * hostname,`
	`unsigned short port);`

Función de la API exportada en la DLL	`int InitHTTPProxy(`
	`HTTPAPI * HTTPAPI_HANDLE,`
	`wchar_t * hostname,`
	`unsigned short port`
	
	Parámetros de entrada:
	
	El parámetro *HTTPAPI_HANDLE* es una referencia al objeto HTTPAPI creado mediante la invocación de la función `newHTTPAPI`.
	El parámetro *hostname* permite especificar la dirección IP en la que se desea que escuche el servidor proxy HTTP y se especifica como una cadena de texto "wide char" (2 bytes) con codificación UNICODE.
	El parámetro *port* permite especificar el puerto en el que se desea que escuche el servidor proxy.
	
	Valor de retorno:
	
	La función devuelve valor 1 si el servidor proxy se creó correctamente o valor -1 si se produjo algún error.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`int initProxy(string hostname, ushort port)`
Descripción	Funcionalidad que permite iniciar un servidor proxy HTTP/s en una dirección IP y puerto concreto.

Método original de la API desarrollada en C++	`int HTTPAPI::StopHTTPProxy();`

Función de la API exportada en la DLL	`int stopHTTPProxy(`
	`HTTPAPI * HTTPAPI_HANDLE)`
	
	Parámetros de entrada:

	El parámetro *HTTPAPI_HANDLE* es una referencia al objeto HTTPAPI creado mediante la invocación de la función newHTTPAPI. **Valor de retorno:** La función devuelve valor 1 si el servidor proxy HTTP se detuvo correctamente o valor -1 si se produjo algún error.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`int stopHTTPproxy()`
Descripción	Funcionalidad que permite detener un servidor proxy HTTP creado previamente.

Método original de la API desarrollada en C++	`int HTTPAPI::SetHTTPProxyConfig(`
	`HTTPHANDLE HTTPHandle,`
	`enum HttpOptions opt,`
	`const wchar_t * parameter);`

Función de la API exportada en la DLL	`SetHTTPProxyConfigString(`
	`HTTPAPI * HTTPAPI_HANDLE,`
	`unsigned int Proxyoptions,`
	`wchar_t * parameter)`
	
	Parámetros de entrada:
	
	El parámetro *HTTPAPI_HANDLE* es una referencia al objeto HTTPAPI creado mediante la invocación de la función newHTTPAPI.
	El parámetro *opt* permite especificar el tipo de opción del servidor proxy HTTP, que se desea modificar. Los posibles valores son:
	
	`ProxyAllowConnect = 0x00,`
	`ProxyAnonymous = 0x01,`
	`ProxyAsynRequest = 0x02,`
	`ProxyDisableCache = 0x03,`
	`ProxyOriginalUserAgent = 0x04,`
	`ProxyDefaultPorts = 0x05`
	
	El parámetro *parameter* permite especificar el valor que se desea asignar a la opción especificada.
	
	Valor de retorno:

	La función devuelve valor 1 si el servidor proxy se detuvo correctamente o valor -1 si se produjo algún error.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`int setHTTPProxyConfig(` `Enum HttpProxyoptions,string value)`
Descripción	Funcionalidad que permite establecer las opciones de configuración del servidor proxy HTTP.

Método original de la API desarrollada en C++	`int HTTPAPI::SetHTTPConfig(`
	`HTTPHANDLE HTTPHandle,`
	`enum HttpOptions opt,`
	`HTTPCSTR parameter);`

Función de la API exportada en la DLL	`int SetHTTPProxyInt(`
	`HTTPAPI * HTTPAPI_HANDLE,`
	`unsigned int Proxyoptions,`
	`int parameter)`
	
	Parámetros de entrada:
	
	El parámetro HTTPHandle es una referencia o descriptor de la comunicación establecida previamente mediante la invocación de la función `HTTPInitConnectionHandle.`
	El parámetro *opt* permite especificar el tipo de opción del servidor proxy HTTP que se desea modificar. Los posibles valores son:
	
	`ProxyAllowConnect = 0x00,`
	`ProxyAnonymous = 0x01,`
	`ProxyAsynRequest = 0x02,`
	`ProxyDisableCache = 0x03,`
	`ProxyOriginalUserAgent = 0x04,`
	`ProxyDefaultPorts = 0x05`
	
	Valor de retorno:
	
	La función devuelve valor 1 si el servidor proxy se detuvo correctamente o valor -1 si se produjo algún error.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`int setHTTPProxyConfig(` `Enum HttpProxyoptions,int value)`
Descripción	Funcionalidad que permite establecer las opciones de

	configuración del servidor proxy HTTP.

Método original de la API desarrollada en C++	`HTTPSTR HTTPAPI::GetHTTPConfig(`
	`int HTTPHandle,`
	`enum HttpOptions opt)`

Función de la API exportada en la DLL	`wchar_t * HTTPGetHTTPConfig(`
	`HTTPAPI * HTTPAPI_HANDLE,`
	`HTTPHANDLE HTTPHandle,`
	`unsigned int option)`
	
	Parámetros de entrada:
	El parámetro *HTTPAPI_HANDLE* es una referencia al objeto HTTPAPI creado mediante la invocación de la función newHTTPAPI.
	El parámetro HTTPHandle es una referencia o descriptor de la comunicación establecida previamente mediante la invocación de la función HTTPInitConnectionHandle.
	El parámetro *option* permite especificar el tipo de opción del servidor proxy HTTP que se desea consultar.
	
	Valor de retorno:
	La función una cadena de texto "wide char" (2 bytes) con codificación UNICODE, que especifica el valor de la opción.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getHTTPConfig(int HTTPHandle, Enum HttpOptions)`
Descripción	Funcionalidad que permite consultar las opciones establecidas del motor de comunicaciones HTTP.

Método original de la API desarrollada en C++	`HTTPSession* HTTPAPI::SendHttpRequest(` `const wchar_t * Fullurl)`

Función de la API exportada en la DLL	`HTTPSession * SendHTTPRequestString(`
	`HTTPAPI * HTTPAPI_HANDLE,`
	`const wchar_t * hostname)`
	
	Parámetros de entrada:
	El parámetro HTTPAP_IHANDLE es una referencia al objeto HTTPAPI creado mediante la invocación de la función newHTTPAPI.
	El parámetro hostname es una cadena de caracteres wide char (2 bytes) y codificación UNICODE y permite

	especificar el nombre del host al que se va a realizar una petición HTTP. **Valor de retorno:** La función devuelve una referencia (handle) a un objeto HTTPSession a través del cual posible interactuar (a través de los objetos HTTPRequest y HTTPresponse) con los datos de la petición realizada, así como de la respuesta asociada.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`HTTPSession sendRequest(string hostname)`
Descripción	Función que permite enviar una petición HTTP contra el host especificado. La petición se envía utilizando el método HTTP GET.

Método original de la API desarrollada en C++	`HTTPSession* HTTPAPI::SendHttpRequest( const wchar_t * Fullurl)`
Función de la API exportada en la DLL	`HTTPSession* SendHttpRequestByObject( HTTPAPI * HTTPAPI_HANDLE, int HTTPHandle, HTTPRequest* request)` **Parámetros de entrada:** El parámetro HTTPAPI_HANDLE es una referencia al objeto HTTPAPI creado mediante la invocación de la función newHTTPAPI. El parámetro HTTPHandle es una referencia o descriptor de la comunicación establecida previamente mediante la invocación de la función `HTTPInitConnectionHandle.` El parámetro request es una referencia a un objeto HTTPRequest que contiene toda la información relevante de una petición HTTP (Cabeceras HTTP y cuerpo de la petición). **Valor de retorno:** La función devuelve una referencia (handle) a un objeto HTTPSession a través del cual posible interactuar (a través de los objetos HTTPRequest y HTTPresponse) con los datos de la petición realizada, así como de la respuesta asociada.
Método para la invocación	`HTTPSession sendRequest(`

de la funcionalidad desde las clase proxy en C#	`HTTPRequest request,` `int HTTPHANDLE)`
Descripción	Función que permite enviar una petición HTTP contra un host concreto especificando los detalles de la petición a través de un objeto HTTPRequest.

Método original de la API desarrollada en C++	`HTTPSession* HTTPAPI::SendRawHTTPRequest(`
	`int HTTPHandle,`
	`const wchar_t * headers,`
	`wchar_t *  postdata,`
	`size_t PostDataSize)`

Función de la API exportada en la DLL	`HTTPSession* SendRawHTTPRequest(`
	`HTTPAPI * HTTPAPI_HANDLE,`
	`int HTTPHandle,`
	`const wchar_t * headers,`
	`wchar_t * PostData,`
	`size_t PostDataSize)`
	
	Parámetros de entrada:
	El parámetro *HTTPAPI_HANDLE* es una referencia al objeto HTTPAPI creado mediante la invocación de la función newHTTPAPI.
	El parámetro *HTTPHandle* es una referencia o descriptor de la comunicación establecida previamente mediante la invocación de la función `HTTPInitConnectionHandle.`
	El parámetro *Headers* es una cadena de caracteres wide char (2 bytes) con codificación UNICODE, a través de la cual es posible especificar las cabeceras de la petición HTTP.
	El parámetro *PostData* es una cadena de caracteres wide char (2 bytes) con codificación UNICODE, a través de la cual se especifican los datos "Post" de una petición HTTP Post.
	El parámetro *PostDataSize* permite especificar el tamaño de los datos Post.
	
	Valor de retorno:
	La función devuelve una referencia (handle) a un objeto HTTPSession a través del cual posible interactuar (a través de los objetos HTTPRequest y HTTPresponse) con los datos de la petición realizada, así como de la respuesta asociada.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`HTTPSession sendRequest(`
	`HTTPRequest request,`
	`int HTTPHANDLE)`

Descripción	Función que permite enviar una petición HTTP contra un host concreto especificando los detalles de la petición a través de un objeto HTTPRequest.

Método original de la API desarrollada en C++	`int RegisterHTTPCallBack(`
	`unsigned int cbType,`
	`HTTP_IO_REQUEST_CALLBACK cb,`
	`const wchar_t * Description);`

Función de la API exportada en la DLL	`int HTTPRegisterCallback(`
	`HTTPAPI * HTTPAPI_Handle,`
	`int cbType,`
	`HTTP_IO_REQUEST_CALLBACK CallBack)`
	
	Parámetros de entrada:
	El parámetro *HTTPAPI_HANDLE* es una referencia al objeto HTTPAPI creado mediante la invocación de la función `newHTTPAPI`.
	El parámetro cbType permite especificar el punto en el que se ejecutará el callback. Los posibles valores son:
	
	`CBTYPE_CLIENT_REQUEST 0x01:`
	Ejecución del callback antes del envío de una petición HTTP por parte del motor de comunicaciones.
	
	`CBTYPE_CLIENT_RESPONSE 0x02:` Ejecución del callback tras la recepción de una petición HTTP por parte del motor de comunicaciones.
	
	`CBTYPE_PROXY_REQUEST 0x04:` Ejecución del callback antes del envío de una petición HTTP por parte del servidor proxy HTTP.
	
	`CBTYPE_PROXY_RESPONSE 0x08:` Ejecución del callback tras la recepción de una petición HTTP por parte del servidor proxy HTTP.
	
	El parámetro Callback permite especificar la función que será ejecutada. Este parámetro es un puntero a una función y debe disponer de la siguiente estructura:

	`int(__cdecl*HTTP_IO_REQUEST_CALLBACK)(`
	`int cbType,`
	`void *api,`
	`int HTTPHandle,`
	`HTTPRequest * request,`
	`HTTPResponse * response);`
	
	La función de callback puede especificar los siguientes valores de retorno (int):
	
	Valor 0: Funcionamiento normal.
	Valor 1: Bloquear el envío/recepción de una petición o respuesta HTTP.

Método para la invocación de la funcionalidad desde las clase proxy en C#	Método que permite añadir un callback (delegado) que permitirá modificar peticiones HTTP antes del envío de las mismas por parte del motor de comunicaciones HTTP.
	
	`clientRequestEvent.Add(`
	`simplifiedCb callback)`
	
	Método que permite añadir un callback (delegado) que permitirá modificar respuestas HTTP tras la recepción las mismas por parte del motor de comunicaciones HTTP.
	
	`clientResponseEvent.Add(`
	`simplifiedCb callback)`
	
	Método que permite añadir un callback (delegado) que permitirá modificar peticiones HTTP antes del envío de las mismas por parte del servidor proxy HTTP.
	
	`clientProxyRequestEvent.Add(`
	`simplifiedCb callback)`
	
	Método que permite añadir un callback (delegado) que permitirá modificar respuestas HTTP tras la recepción de las mismas por parte del servidor proxy HTTP.
	
	`clientProxyResponseEvent.Add(`
	`simplifiedCb callback)`
	
	El tipo `simplifiedCb` es un delegado (equivalente a

	un puntero a función) con la siguiente estructura
	
	`delegate int simplifiedCb(`
	`HTTPRequest request,`
	`HTTPResponse response);`

Descripción	Funcionalidad que permite registrar una función callback, que será ejecutada en alguno de los siguientes puntos: ▪ Antes del envío de una petición HTTP por parte del motor de comunicaciones. (Esto es, a través de la invocación de funciones de envío de peticiones). ▪ Tras la recepción de una respuesta HTTP por parte del motor de comunicaciones ▪ Antes del envío de una petición por parte del servidor proxy HTTP. ▪ Tras la recepción de una respuesta por parte del servidor proxy HTTP.

Método original de la API desarrollada en C++	`int RemoveHTTPCallBack(`
	`unsigned int cbType,`
	`HTTP_IO_REQUEST_CALLBACK cb);`

Función de la API exportada en la DLL	`RemoveHTTPCallback(`
	`HTTPAPI * HTTPAPI_HANDLE,`
	`unsigned int cbType,`
	`HTTP_IO_REQUEST_CALLBACK cb)`

Método para la invocación de la funcionalidad desde las clase proxy en C#	Métodos que permiten eliminar funciones de callback (delegados) establecidos previamente.
	
	`clientRequestEvent.Remove (`
	`simplifiedCb callback)`
	
	`clientResponseEvent.Remove(`
	`simplifiedCb callback)`
	
	`clientProxyRequestEvent.Remove(`
	`simplifiedCb callback)`
	
	`clientProxyResponseEvent.Remove(`
	`simplifiedCb callback)`

Descripción	Funcionalidad que permite eliminar un callback definido

	previamente.

A.1.2 Funciones exportadas a través de la biblioteca de vínculo dinámico asociadas a la clase HTTPRequest de la API HTTPCore.

Método original de la API desarrollada en C++	`HTTPRequest::HTTPRequest()`
Función de la API exportada en la DLL	`HTTPRequest * NewHTTPRequest()` **Valor de retorno:** Devuelve un una referencia al objeto creado.
Método para la invocación de la funcionalidad desde las clase proxy en C#	new HTTPRequest()
Descripción	Creación de un objeto HTTPRequest que permite interactuar con todos los elementos de una petición HTTP: Cabeceras HTTP y cuerpo de la petición.

Método original de la API desarrollada en C++	`HTTPRequest::~HTTPRequest()`

Función de la API exportada en la DLL	`int DestroyHTTPRequest(`
	`HTTPRequest * request)`
	
	Parámetro de entrada:
	El parámetro de entrada a la función es un handle o referencia a un objeto HTTPSession instanciado previamente.
	
	Valor de retorno:
	El valor de retorno es 1 si el objeto se destruyó correctamente o -1 si la referencia del objeto HTTPRequest es inválida.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`~HTTPRequest()`
Descripción	Destrucción de un objeto HTTPRequest

Método original de la API desarrollada en C++	`HTTPRequest::SetData(`
	`wchar_t *lpData,`
	`size_t DataLength)`

Función de la API exportada en la DLL	`int HTTPRequestSetDataW(HTTPRequest *`
	`request,`
	`wchar_t * lpData,`
	`size_t DataLength)`

	Parámetros de entrada: El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente. El parámetro *lpData* permite especificar los datos del cuerpo de una petición HTTP. Se especifica como una cadena de texto "wide char" (2 bytes) con codificación UNICODE. El parámetro DataLength permite especificar el tamaño de los datos del cuerpo de una petición HTTP. **Valor de retorno:** El valor de retorno es 1 si los datos del cuerpo de la petición HTTP se establecieron correctamente o -1 si se produjo algún error.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`int setDataW(string data, uint DataLength)`
Descripción	Funcionalidad que permite definir los datos del cuerpo de una petición HTTP.

Método original de la API desarrollada en C++	`wchar_t * HTTPRequest::GetData(void)`
Función de la API exportada en la DLL	`wchar_t * HTTPRequestGetData( HTTPRequest * request)` **Parámetro de entrada:** El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente. **Valor de retorno:** El valor de retorno es un puntero a una cadena de texto "wide char" (2 bytes) con codificación UNICODE.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getData()`
Descripción	Funcionalidad que permite obtener los datos del cuerpo de una petición HTTP.

Método original de la API desarrollada en C++	`size_t HTTPRequest::GetDataSize(void)`
Función de la API exportada en la DLL	`size_t HTTPRequestGetDataSize( HTTPRequest * request)` **Parámetro de entrada:** El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente. **Valor de retorno:** El valor de retorno es el tamaño de los datos de una petición HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`int getDataSize()`
Descripción	Funcionalidad que permite obtener el tamaño de los datos del cuerpo de una petición HTTP.

Método original de la API desarrollada en C++	`wchar_t * HTTPRequest::GetRequestedURL()`
Función de la API exportada en la DLL	`wchar_t * HTTPRequestGetRequestedURL( HTTPRequest * request)` **Parámetro de entrada:** El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente. **Valor de retorno:** El valor de retorno es un puntero a una cadena de texto "wide char" (2 bytes) con codificación UNICODE que contiene la URL de la petición HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getRequestedURL()`
Descripción	Funcionalidad que permite obtener la URL de una petición HTTP.

Método original de la API desarrollada en C++	`HTTPCHAR *HTTPRequest::GetHTTPMethod()`
Función de la API exportada en la DLL	`wchar_t * HTTPRequestGetHTTPMethod( HTTPRequest * request)`

	Parámetro de entrada: El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente. **Valor de retorno:** El valor de retorno es un puntero a una cadena de texto "wide char" (2 bytes) con codificación UNICODE que especifica el método HTTP (Ej: GET/POST/HEAD) de la petición.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getHTTPMethod()`
Descripción	Funcionalidad que permite obtener el método HTTP de una petición.

Método original de la API desarrollada en C++	`wchar_t * HTTPHeaders::GetHeaders(void)`
Función de la API exportada en la DLL	`wchar_t * HTTPRequestGetHeaders( HTTPRequest * request)` **Parámetro de entrada:** El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente. **Valor de retorno:** El valor de retorno es un puntero a una cadena de texto "wide char" (2 bytes) con codificación UNICODE que especifica las cabeceras de la petición HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getHeaders()`
Descripción	Funcionalidad que permite obtener las cabeceras de una petición HTTP.

Método original de la API desarrollada en C++	`wchar_t * HTTPHeaders::AddHeader(HTTPCSTR newheader)`
Función de la API exportada en la DLL	`wchar_t * HTTPRequestAddHeader( HTTPRequest * request, const wchar_t * Header)` **Parámetro de entrada:** El parámetro *request* es un handle o referencia un

	objeto HTTPRequest creado previamente. El parámetro Header es una cadena de texto "wide char" (2 bytes) con codificación UNICODE que especifica la cabecera HTTP que se desea añadir al objeto HTTPRequest, el cual representa una petición HTTP. **Valor de retorno:** El valor de retorno es un puntero a una cadena de texto "wide char" (2 bytes) con codificación UNICODE que contiene la cabecera HTTP que fue añadida, o una cadena vacía si se produjo algún error.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string addHeader(string newHeader)`
Descripción	Funcionalidad que permite añadir una cabecera HTTP sobre un objeto HTTPRequest.

Método original de la API desarrollada en C++	`size_t HTTPHeaders::GetHeaderSize(void)`
Función de la API exportada en la DLL	`size_t HTTPRequestGetHeaderSize(HTTPRequest * request)` **Parámetro de entrada:** El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente. **Valor de retorno:** El valor de retorno es el tamaño de las cabeceras de la petición HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`uint getHeaderSize()`
Descripción	Funcionalidad que permite obtener el tamaño de las cabeceras HTTP de una petición, representada por el objeto HTTPRequest.

Método original de la API desarrollada en C++	`void HTTPHeaders::InitHTTPHeaders(` `const wchar_t * header)`

Función de la API exportada en la DLL	`int HTTPRequestInitHTTPHeaders(`
	`HTTPRequest * request,`
	`const wchar_t * header)`

	Parámetro de entrada: El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente. El parámetro header es un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que contiene las cabeceras HTTP. **Valor de retorno:** El valor de retorno es el tamaño de las cabeceras establecidas o -1 si se produjo algún error.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`int initHTTPHeaders(string headers)`
Descripción	Funcionalidad que permite establecer todas las cabeceras de una petición HTTP, representada a través de un objeto HTTPRequest.

Método original de la API desarrollada en C++	`wchar_t * HTTPHeaders::GetHeaderValue(`
	`const wchar_t * value,`
	`int n)`

Función de la API exportada en la DLL	`wchar_t * HTTPRequestGetHeaderValue(`
	`HTTPRequest * request,`
	`const wchar_t *  value)`
	
	Parámetro de entrada:
	El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente.
	El parámetro *value* es un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que contiene el nombre de una cabecera HTTP.
	
	Valor de retorno:
	El valor de retorno es un puntero a una cadena de texto wide char (2bytes) con codificación UNICODE que contiene el valor para la cabecera HTTP especificada

Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getHeaderValue(` `string value)`
Descripción	Funcionalidad que permite obtener el valor de una cabecera HTTP.

Método original de la API desarrollada en C++	`const wchar_t *`

	`HTTPHeaders::Headerstrstr(` `const wchar_t * searchdata)`

Función de la API exportada en la DLL	`const HTTPSTR HTTPRequestHeaderSearch(`
	`HTTPRequest * request,`
	`const wchar_t * searchdata)`
	
	Parámetro de entrada:
	El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente.
	El parámetro *searchdata* es un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que contiene el texto que se desea buscar en las cabeceras HTTP.
	
	Valor de retorno:
	Devuelve un puntero a la primera ocurrencia de la cadena de texto que se desea buscar.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`string searchString(string search)`
Descripción	Funcionalidad que permite buscar una cadena de texto en las cabeceras HTTP de una petición.

Método original de la API desarrollada en C++	`wchar_t * HTTPHeaders::RemoveHeader(` `const wchar_t * oldheader)`

Función de la API exportada en la DLL	`wchar_t * HTTPRequestRemoveHeader(`
	`HTTPRequest * request,`
	`const wchar_t * Header)`
	
	Parámetro de entrada:
	El parámetro *request* es un handle o referencia un objeto HTTPRequest creado previamente.
	El parámetro header es un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que contiene la cabecera HTTP que se desea eliminar.
	
	Valor de retorno:
	El valor de retorno es puntero a una cadena de texto "wide char" con codificación UNICODE que contiene la cabecera que fue eliminada.

Método para la invocación	`string removeHeader(string header)`

de la funcionalidad desde las clase proxy en C#	
Descripción	Funcionalidad que permite eliminar una cabecera HTTP especificada a través del parámetro header.

A.1.3 Funciones exportadas a través de la biblioteca de vínculo dinámico asociadas a la clase HTTPResponse de la API HTTPCore.

Método original de la API desarrollada en C++	`HTTPResponse::HTTPResponse()`
Función de la API exportada en la DLL	`HTTPResponse * newHTTPResponse()` **Valor de retorno:** Devuelve un una referencia al objeto creado.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`new HTTPResponse()`
Descripción	Creación de un objeto HTTPResponse a través del cual es posible interactuar con todos los elementos de una respuesta HTTP: Cabeceras HTTP de la respuesta y cuerpo.

Método original de la API desarrollada en C++	`HTTPResponse::~HTTPResponse()`

Función de la API exportada en la DLL	`int DestroyHTTPResponse(`
	`HTTPResponse * response)`
	
	Parámetro de entrada:
	El parámetro *response* es un handle o referencia un objeto HTTPResponse creado previamente.
	
	Valor de retorno:
	La función devuelve valor 1 si fue posible llevar a cabo la destrucción del objeto, o valor -1 si la referencia al objeto no era válida.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`~HTTPResponse()`
Descripción	Destrucción de un objeto HTTPResponse

Método original de la API desarrollada en C++	`wchar_t * HTTPResponse::GetDataW(void)`

Función de la API exportada en la DLL	`wchar_t * HTTPResponseGetDataW(`
	`HTTPResponse * response)`
	
	Parámetro de entrada:
	El parámetro *response* es un handle o referencia un objeto HTTPResponse creado previamente.

	Valor de retorno: El valor de retorno es un puntero a una cadena de texto "wide char" con codificación UNICODE que contiene los datos del cuerpo de una respuesta HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getDataW()`
Descripción	Funciones que permiten obtener los datos de una respuesta HTTP representada por un objeto HTTPResponse.

Método original de la API desarrollada en C++	`size_t HTTPResponse::GetDataSize(void)`
Función de la API exportada en la DLL	`size_t HTTPResponseGetDataSize( HTTPResponse * response)` **Parámetro de entrada:** El parámetro *response* es un handle o referencia un objeto HTTPResponse creado previamente. **Valor de retorno:** El valor de retorno es el tamaño de los datos del cuerpo de una respuesta HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`int getDataSize()`
Descripción	Funcionalidad que permite obtener los datos de una respuesta HTTP.

Método original de la API desarrollada en C++	`wchar_t * HTTPResponse::GetServerVersion()`
Función de la API exportada en la DLL	`wchar_t * HTTPResponseGetServerVersion( HTTPResponse * response)` **Parámetro de entrada:** El parámetro *response* es un handle o referencia un objeto HTTPResponse creado previamente. **Valor de retorno:** El valor de retorno es un puntero a una cadena de texto "wide char" con codificación UNICODE que contiene el valor de la cabecera Server de una respuesta HTTP.
Método para la invocación	`string getServerVersion()`

de la funcionalidad desde las clase proxy en C#	
Descripción	Funcionalidad que permite obtener el valor de la cabecera HTTP server de una respuesta HTTP.

Método original de la API desarrollada en C++	`unsigned short HTTPResponse::GetStatus()`
Función de la API exportada en la DLL	`unsigned short HTTPResponseGetStatus( HTTPResponse * response)` **Parámetro de entrada:** El parámetro *response* es un handle o referencia un objeto HTTPResponse creado previamente. **Valor de retorno:** La devuelve el código de estado de una respuesta HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`int getStatus()`
Descripción	Funcionalidad que permite obtener el código de estado de una respuesta HTTP. (Ej 200 OK).

Método original de la API desarrollada en C++	`void HTTPResponse::SetData( void *lpData, size_t Datalength)`
Función de la API exportada en la DLL	`int HTTPResponseSetData(HTTPResponse * response, void * lpData, size_t Datalength)` **Parámetro de entrada:** El parámetro *response* es un handle o referencia un objeto HTTPResponse creado previamente. El parámetro lpData es un puntero a los datos del cuerpo de la respuesta de una petición HTTP. Los datos deben ser una cadena de texto wide char (2 bytes) con codificación UNICODE. El parámetro DataLength especifica el tamaño de los datos del cuerpo de una respuesta HTTP. **Valor de retorno:** La función devuelve el tamaño de los datos del cuerpo de una respuesta HTTP.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`int setData(string lpData, uint dataSize)`
Descripción	Funcionalidad que permite establecer los datos del cuerpo de una respuesta HTTP, representada por un objeto HTTPResponse.

Método original de la API desarrollada en C++	`HTTPHeaders::InitHTTPHeaders(`
	`const wchar_t * header,`
	`size_t length)`

Función de la API exportada en la DLL	`int HTTPResponseInitHTTPHeaders(`
	`HTTPResponse * response,`
	`const wchar_t * header)`
	
	Parámetro de entrada:
	El parámetro *response* es un handle o referencia un objeto HTTPResponse creado previamente.
	El parámetro header es un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que contiene las cabeceras de una respuesta HTTP.
	
	Valor de retorno:
	El valor de retorno es 1 si las cabeceras de la respuesta HTTP se establecieron correctamente, o -1 si se produjo algún error.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`int initHTTPHeaders(string headers)`
Descripción	Funcionalidad que permite establecer las cabeceras de una respuesta HTTP representada por un objeto HTTPResposne.

Método original de la API desarrollada en C++	`HTTPCHAR* HTTPHeaders::GetHeaders(void)`

Función de la API exportada en la DLL	`const wchar_t * HTTPResponseGetHeaders(`
	`HTTPResponse * response)`
	
	Parámetro de entrada:
	El parámetro *response* es un handle o referencia un objeto HTTPResponse creado previamente.
	
	Valor de retorno:
	El valor de retorno es un puntero a una cadena de texto

	"wide char" con codificación UNICODE que contiene las cabeceras de una respuesta HTTP, representada por un objeto HTTPResponse.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getHeaders()`
Descripción	Funcionalidad que permite obtener las cabeceras de una respuesta HTTP, representada por un objeto HTTPResponse.

Método original de la API desarrollada en C++	`size_t HTTPHeaders::GetHeaderSize(void)`
Función de la API exportada en la DLL	`size_t HTTPResponseGetHeaderSize( HTTPResponse * response)` **Parámetro de entrada:** El parámetro *response* es un handle o referencia un objeto HTTPResponse creado previamente. **Valor de retorno:** La función devuelve el tamaño de las cabeceras de una respuesta HTTP, representada por el objeto HTTPResponse.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`uint getHeaderSize()`
Descripción	Funcionalidad que permite obtener el tamaño de las cabeceras de una respuesta HTTP.

Método original de la API desarrollada en C++	`wchar_t * HTTPHeaders::GetHeaderValue( const wchar_t * value, int n)`
Función de la API exportada en la DLL	`wchar_t * HTTPResponseGetHeaderValue( HTTPResponse * response, const wchar_t * value)` **Parámetro de entrada:** El parámetro *response* es un handle o referencia un objeto HTTPResponse creado previamente. El parámetro value es un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que contiene el nombre de una cabecera HTTP.
Método para la invocación	`string getHeaderValue(string value)`

de la funcionalidad desde las clase proxy en C#	
Descripción	Funcionalidad que permite obtener el valor asociado a una cabecera de una respuesta HTTP.

Método original de la API desarrollada en C++	`wchar_t * HTTPHeaders::AddHeader( const wchar_t * newheader)`
Función de la API exportada en la DLL	`wchar_t * HTTPResponseAddHeader( HTTPResponse * response, const wchar_t * Header)` **Parámetro de entrada:** El parámetro *response* es un handle o referencia un objeto HTTPRequest creado previamente. El parámetro Header es una cadena de texto "wide char" (2 bytes) con codificación UNICODE que especifica la cabecera HTTP que se desea añadir al objeto HTTPResponse. **Valor de retorno:** El valor de retorno es un puntero a una cadena de texto "wide char" (2 bytes) con codificación UNICODE que contiene la cabecera HTTP que fue añadida, o una cadena vacía si se produjo algún error al ejecutar la función.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string addHeader(string header)`
Descripción	Funcionalidad que permite añadir una cabecera HTTP sobre una respuesta, representada por un objeto HTTPResponse.

Método original de la API desarrollada en C++	`wchar_t * HTTPHeaders::RemoveHeader( const wchar_t * oldheader)`
Función de la API exportada en la DLL	`wchar_t * HTTPResponseRemoveHeader( HTTPResponse * response, const wchar_t * Header)` **Parámetro de entrada:** El parámetro *response* es un handle o referencia un

	objeto HTTPResponse creado previamente. El parámetro header es un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que especifica la cabecera HTTP que se desea eliminar. **Valor de retorno:** El valor de retorno es puntero a una cadena de texto "wide char" con codificación UNICODE que contiene la cabecera que fue eliminada.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string removeHeader(string header)`
Descripción	Funcionalidad que permite eliminar una cabecera de una respuesta HTTP representada por un objeto HTTPResponse.

Método original de la API desarrollada en C++	`const HTTPSTR HTTPHeaders::Headerstrstr( const wchar_t * searchdata)`
Función de la API exportada en la DLL	`const HTTPSTR HTTPResponseHeaderSearch( HTTPResponse * response, const wchar_t * searchdata)` **Parámetro de entrada:** El parámetro *response* es un handle o referencia un objeto HTTPRequest creado previamente. El parámetro *searchdata* es un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que especifica el texto que se desea buscar en las cabeceras HTTP. **Valor de retorno:** Devuelve un puntero a la primera ocurrencia de la cadena de texto que se desea buscar.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string searchString(string search)`
Descripción	Funcionalidad que permite buscar una cadena de texto en las cabeceras HTTP de una respuesta.

A.1.4 Funciones exportadas a través de la biblioteca de vínculo dinámico asociadas a la clase HTTPSession de la API HTTPCore.

Método original de la API desarrollada en C++	`HTTPSession::HTTPSession()`
Función de la API exportada en la DLL	`HTTPSession * newHTTPSession()` **Valor de retorno:** Devuelve un una referencia al objeto creado.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`new HTTPSession()`
Descripción	Creación de un objeto HTTPSession, que contiene la referencia de una petición HTTP (representada por un objeto HTTPRequest) y su respuesta correspondiente (representada por un objeto HTTPResponse).

Método original de la API desarrollada en C++	`HTTPSession::~HTTPSession()`

Función de la API exportada en la DLL	`int DestroyHTTPSession(`
	`HTTPSession * session)`
	
	Parámetro de entrada:
	El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente.
	
	Valor de retorno:
	La función devuelve valor 1 si fue posible llevar a cabo la destrucción del objeto, o valor -1 si la referencia al objeto (especificada a través del parámetro sesión) no era válida.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`~HTTPSession()`
Descripción	Destrucción de un objeto HTTPSession.

Método original de la API desarrollada en C++	`HTTPSession::response`

Función de la API exportada en la DLL	`HTTPResponse * HTTPSessionGetResponse(`
	`HTTPSession * session)`
	
	Parámetro de entrada:
	El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente.

	Valor de retorno: La función devuelve un puntero a un objeto HTTPResponse.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`HTTPResponse getResponse()`
Descripción	Funcionalidad que permite obtener la referencia a un objeto HTTPResponse.

Método original de la API desarrollada en C++	`HTTPSession::request`
Función de la API exportada en la DLL	`HTTPRequest * HTTPSessionGetRequest( HTTPSession * session)` **Parámetro de entrada:** El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente. **Valor de retorno:** La función devuelve un puntero a un objeto HTTPRequest.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`HTTPRequest getRequest()`
Descripción	Funcionalidad que permite obtener la referencia a un objeto HTTPRequest

Método original de la API desarrollada en C++	`int HTTPSession::IsValidHTTPResponse(void)`
Función de la API exportada en la DLL	`int HTTPSessionIsValidHTTPResponse( HTTPSession * session)` **Parámetro de entrada:** El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente. **Valor de retorno:** La función devuelve valor 1 si tras el envío de una petición se obtuvo una respuesta válida. La función devuelve valor 0 si no se obtuvo una respuesta válida tras el envío de una petición HTTP.
Método para la invocación de la funcionalidad desde	`int isValidHTTPResponse()`

las clase proxy en C#	
Descripción	Funcionalidad que permite verificar que, tras el envío de una petición HTTP se obtuvo una respuesta válida.

Método original de la API desarrollada en C++	`int HTTPSession::HasResponseData(void);`
Función de la API exportada en la DLL	`int HTTPSessionHasResponseData( HTTPSession * session)` **Parámetro de entrada:** El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente. **Valor de retorno:** La función devuelve valor 1 si tras el envío de una petición se obtuvo una respuesta HTTP con datos en el cuerpo de la misma. La función devuelve valor 0 si no se obtuvo una respuesta sin datos en el cuerpo de la misma.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`int hasResponseData()`
Descripción	Funcionalidad que permite verificar que, tras el envío de una petición HTTP se obtuvo una respuesta que contiene datos.

Método original de la API desarrollada en C++	`HTTPSession::ip`
Función de la API exportada en la DLL	`int HTTPSessionGetIP( HTTPSession * session)` **Parámetro de entrada:** El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente. **Valor de retorno:** La función devuelve la dirección IP (especificada como un número entero) del host contra el que se realizó una petición HTTP. La función devuelve valor -1 si se produjo algún error.
Método para la invocación de la funcionalidad desde	`int getIP()`

las clase proxy en C#	
Descripción	Funcionalidad que permite obtener la dirección IP del sistema contra el que se realizó una petición HTTP.

Método original de la API desarrollada en C++	`HTTPSession::port`

Función de la API exportada en la DLL	`int HTTPSessionGetPort(`
	`HTTPSession * session)`
	
	Parámetro de entrada:
	El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente.
	
	Valor de retorno:
	La función devuelve el puerto (especificado como un número entero) del host contra el que se realizó una petición HTTP.
	La función devuelve valor -1 si se produjo algún error.

Método para la invocación de la funcionalidad desde las clase proxy en C#	`int getPort()`
Descripción	Funcionalidad que permite obtener el puerto del sistema contra el que se realizó una petición HTTP.

Método original de la API desarrollada en C++	`HTTPSession::NeedSSL`

Función de la API exportada en la DLL	`int HTTPSessionGetNeedSSL(`
	`HTTPSession * session)`
	
	Parámetro de entrada:
	El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente.
	
	Valor de retorno:
	La función devuelve el valor 1 si el envío de una petición HTTP se realizó a través de un canal de comunicaciones seguro (HTTPs).
	La función devuelve valor 0 si la comunicación se realizó a través de HTTP.
	.

Método para la invocación de la funcionalidad desde las	`int needSSL()`

clase proxy en C#	
Descripción	Funcionalidad que permite verificar si una petición HTTP se realizó a través de un canal de comunicaciones seguro (HTTPs).

Método original de la API desarrollada en C++	`HTTPSession::url`
Función de la API exportada en la DLL	`wchar_t * HTTPSessionGetUrl( HTTPSession * session)` **Parámetro de entrada:** El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente. **Valor de retorno:** La función devuelve un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que contiene la URL contra la que se realizó una petición HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getURL()`
Descripción	Funcionalidad que permite obtener la URL contra la que se realizó una petición HTTP.

Método original de la API desarrollada en C++	`HTTPSesssion::Method`
Función de la API exportada en la DLL	`wchar_t * HTTPSessionGetMethod( HTTPSession * session)`
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getMethod()` **Parámetro de entrada:** El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente. **Valor de retorno:** La función devuelve un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que contiene el método HTTP de una petición realizada.
Descripción	Funcionalidad que permite obtener el método HTTP especificado cuando se realizó el envío de una petición

Método original de la API desarrollada en C++	`HTTPSession::hostname`
Función de la API exportada en la DLL	`wchar_t * HTTPSessionGetHostname( HTTPSession * session)` **Parámetro de entrada:** El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente. **Valor de retorno:** La función devuelve un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que contiene el nombre del host contra el que se realizó una petición HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`string getHostname()`
Descripción	Funcionalidad que permite obtener el nombre del host contra el que se envió de una petición HTTP.

Método original de la API desarrollada en C++	`HTTPSession::status`
Función de la API exportada en la DLL	`unsigned int HTTPSessionGetStatus( HTTPSession * session)` **Parámetro de entrada:** El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente. **Valor de retorno:** La función devuelve el código de estado de una respuesta HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	`int getStatus()`
Descripción	Funcionalidad que permite obtener el código de estado de una respuesta HTTP.

Método original de la API desarrollada en C++	`HTTPSession::ContentType`
Función de la API exportada en la DLL	`wchar_t * HTTPSessionGetContentType( HTTPSession * HTTPSession_handle)` **Parámetro de entrada:** El parámetro *session* es un handle o referencia un objeto HTTPSession creado previamente. **Valor de retorno:** La función devuelve un puntero a una cadena de texto wide char (2 bytes) con codificación UNICODE, que contiene el tipo de contenido de una respuesta HTTP.
Método para la invocación de la funcionalidad desde las clase proxy en C#	string getContentType()
Descripción	Funcionalidad que permite obtener el valor de la cabecera HTTP Content-Type de una respuesta HTTP.

A.2 Creación de scripts

La aplicación FHScan permite la creación, edición, compilación y ejecución de scripts. El proyecto contempla la creación de dos tipos de scripts posibles: scripts de auditoría y scripts de callback, los cuales pueden ser ejecutados por parte de la aplicación.

Scripts de auditoría

Los scripts de auditoría permiten mediante la utilización de las clases proxy desarrolladas en C# llevar a cabo el envío de peticiones HTTP y obtener las respuestas asociadas a éstas.

Implementación de scripts de auditoria:

Los scripts de auditoría deben implementarse como clases que heredan de la clase abstracta ICustomScript, y deben proveer una implementación para los métodos:

```
void Run()
```

El método Run define la rutina de ejecución de scripts. Es en la implementación de este método donde se especifica toda la lógica del script.

```
bool ValidateParameters(XmlDocument parameters)
```

El método ValidateParameters es una rutina a través del cual se debe validar que, los parámetros de entrada proporcionados al script son correctos. Los parámetros de entrada se especifican como un documento Xml.

La clase abstracta ICustomScript dispone de una propiedad `customScriptHelper Helper`, que dispone de un método que puede ser utilizado para mostrar los resultados del script por pantalla.

Para mostrar los resultados del script por pantalla se deben seguir los siguientes pasos:

1. Instanciar un objeto de la clase `StringEventArgs`, la cual dispone de la siguiente definición.

```
Public class StringEventArgs
    {
        public string message; // Mensaje que se desea mostrar por pantalla.
        public bool clean;     // Atributo que define se debe limpiar cualquier
dato anterior que haya sido mostrado por pantalla.
        public StringEventArgs(string message,bool clean)
        public StringEventArgs(string message)
    }
```

```
StringEventArgs stringArgs = new StringEventArgs("Mensaje que se desea
mostrar por pantalla.");
```

2. Invocar el metodo PrintLn, para imprimir los datos por pantalla.

```
this.Helper.Print(stringArgs);
```

Scripts de callback

Los scripts de callback permiten modificar el comportamiento del motor de comunicaciones HTTP o del servidor proxy HTTP de la aplicación, mediante la modificación de las peticiones HTTP o de las respuestas.

Como ya se comentó previamente los scripts de callback se pueden ejecutar en cualquiera de estos puntos:

- Antes del envío de una petición HTTP por parte del motor de comunicaciones. (Esto es, a través de la invocación de funciones de envío de peticiones).
- Tras la recepción de una respuesta HTTP por parte del motor de comunicaciones
- Antes del envío de una petición por parte del servidor proxy HTTP.
- Tras la recepción de una respuesta por parte del servidor proxy HTTP.

Los scripts que deseen modificar el comportamiento del motor de comunicaciones HTTP o del servidor proxy antes de llevar a cabo el envío de una petición HTTP deben implementar la interfaz ICallbackRequest, mientras que los scripts que modifiquen el comportamiento del motor de comunicaciones HTTP o del proxy tras la recepción de respuestas deben implementar la interfaz ICallbackResponse.

Implementación de scripts de callback ICallbackRequest

Los scripts de callback de tipo ICallbackRequest deben proveer una implementación siguiente método.

```
CBReturnStatus Run(HTTPRequest Request);
```

El método `Run(HTTPRequest Request)` define la rutina de ejecución de script de callback, la cual puede llevar a cabo la modificación de todas las peticiones HTTP que vayan a ser enviadas por parte del servidor proxy HTTP o del motor de comunicaciones HTTP.
El valor de retorno que debe proporcionar la implementación es una enumeración de tipo `CBReturnStatus`, que permite definir si la petición HTTP debe seguir su curso normal, es decir ser enviada a su destino (`CBReturnStatus.Continue`), o si debe ser bloqueada, y por lo tanto no se llegaría a enviar (`CBReturnStatus.Block`).

A continuación se muestra un código de ejemplo de un script de callback que permite bloquear todas a aquellas peticiones que contengan “google.com” en la URL.

```
public class implementacionScript : ICallbackRequest{

    public CBReturnStatus Run(HTTPRequest request){

        if (request.getRequestedURL().Contains("google.com"))
           return CBReturnStatus.Block;
        else
           return CBReturnStatus.Continue;
    }
}
```

Tabla 7: Código de ejemplo de un script de callback *ICallbackRequest*.

Procedimiento de creación de scripts de callback ICallbackResponse

Los scripts de callback de tipo ICallbackResponse deben proveer una implementación siguiente método.

```
CBReturnStatus Run(HTTPResponse response);
```

El método `Run(HTTPRequest Request)` define la rutina de ejecución del script, a través de la cual es posible modificar el contenido cualquier respuesta HTTP gestionada por parte del motor de comunicaciones HTTP o del servidor proxy HTTP.

El valor de retorno es una enumeración de tipo `CBReturnStatus`, y permite definir si la respuesta HTTP debe siguir su curso normal es decir permitir que sea mostrada (`CBReturnStatus.Continue`), o si debe ser bloqueada, y por lo tanto la respuesta HTTP definida a través del objeto HTTPResponse, no llegaría a ser mostrada a cualquier rutina que permita obtener las respuestas asociadas a peticiones enviadas a través del servidor proxy HTTP o a través del motor de comunicaciones HTTP.

El siguiente código muestra un script ICallbackResponse, que permite eliminar cualquier ocurrencia de la palabra “google.com” en los datos de respuesta.

```
public class ReplaceGoogleString : ICallbackResponse{

    public CBReturnStatus Run(HTTPResponse response){

        data = response.getDataW();
        if (data.Contains("google.com")){
           data.Replace("google.com","");
        }
        return CBReturnStatus.Continue;
```

```
        }
}
```

Tabla 8: Código de ejemplo de un script de callback ICallbackResponse.

B. Manual de usuario.

B.1 Secciones de la interfaz

La interfaz gráfica de la aplicación se divida fundamentalmente en dos secciones principales "**Interceptor**" y "**ScriptingIDE**". La primera de ellas contiene las funcionalidades que permiten llevar a cabo la interceptación de peticiones HTTP/s, el reenvío de peticiones, así como la ejecución de pruebas de fuzzing. La sección ScriptingIDE dispone de aquellas funcionalidades que permiten crear, editar, compilar y ejecutar scripts.

B 1.1 Interceptor

La sección "Interceptor" se muestra por defecto al iniciar la aplicación. En esta sección se agrupan tolas las funcionalidades que nos permitirán interceptar y visualizar peticiones enviadas por una aplicación externa, así como construir y reenviar peticiones manualmente, o de forma automática (fuzzer) a través de procedimientos iterativos.

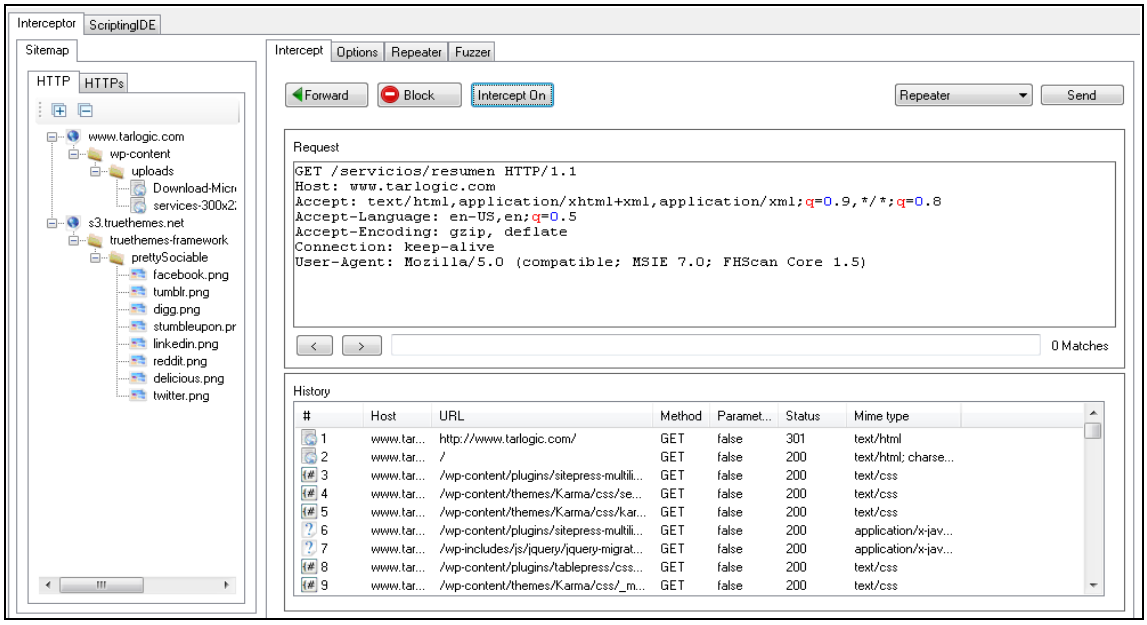


Figura 39: Vista general de la sección "Interceptor".

B.1.1.1 Configuración del servidor proxy HTTP

Para interceptar las peticiones HTTP/s enviadas por parte de una aplicación externa como puede ser una navegador web, es necesario configurar tanto el servidor proxy HTTP proporcionado por la aplicación FHScan, como la aplicación externa que haga uso del proxy redirigiendo las peticiones HTTP a través del mismo.

FHScan inicia por defecto el servidor proxy en la dirección de loopback (127.0.0.1) y puerto 8090. Sin embargo es posible establecer el servidor proxy en una dirección IP concreta, y puerto específico accediendo a la sección "**Options**".

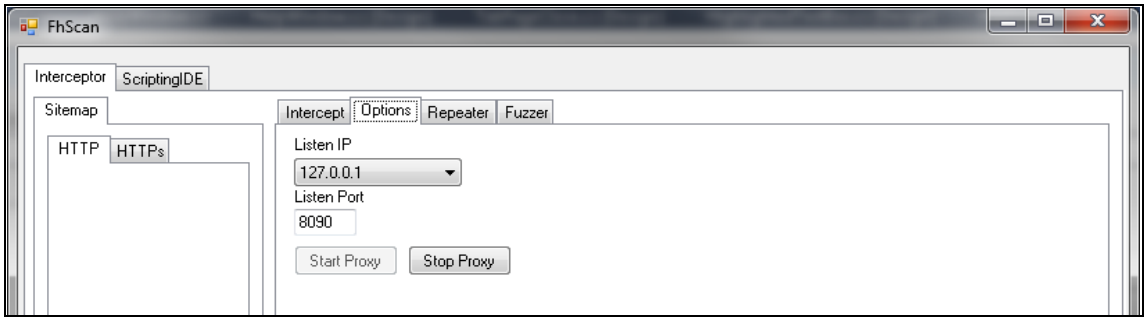


Tabla 9: Configuración del servidor proxy HTTP.

En la sección "**Options**" es posible especificar tanto una dirección IP del equipo, como el puerto en el que se desee que escuche.

B.1.1.2 Configuración del navegador web Firefox para redirigir peticiones a través del proxy.

A continuación se detalla cómo configurar el navegador web Firefox para permitir la redirección de peticiones HTTP a través del proxy.

En primer lugar es necesario abrir el panel de opciones del navegador, y seleccionar la pestaña "**Network**". Una vez seleccionada la pestaña, es posible definir las propiedades de conexión haciendo "clic" sobre el botón "**Settings**", como se muestra en la siguiente figura.

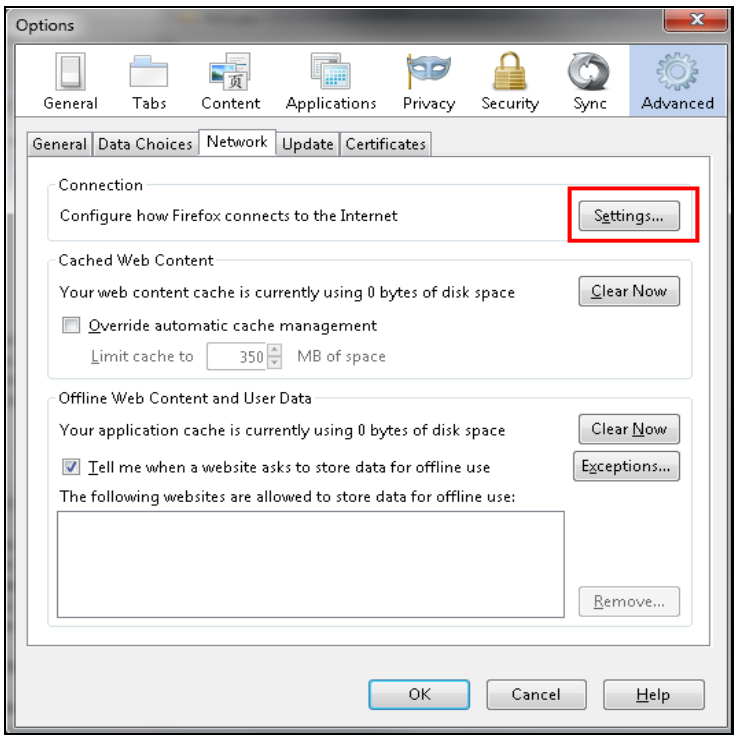


Tabla 10: Configuración de propiedades de conexión en el navegador web Firefox.

Tras hacer clic sobre el botón "**Settings**" se abrirá la siguiente ventana, a través de la cual podremos definir la dirección IP del servidor proxy, así como el puerto.

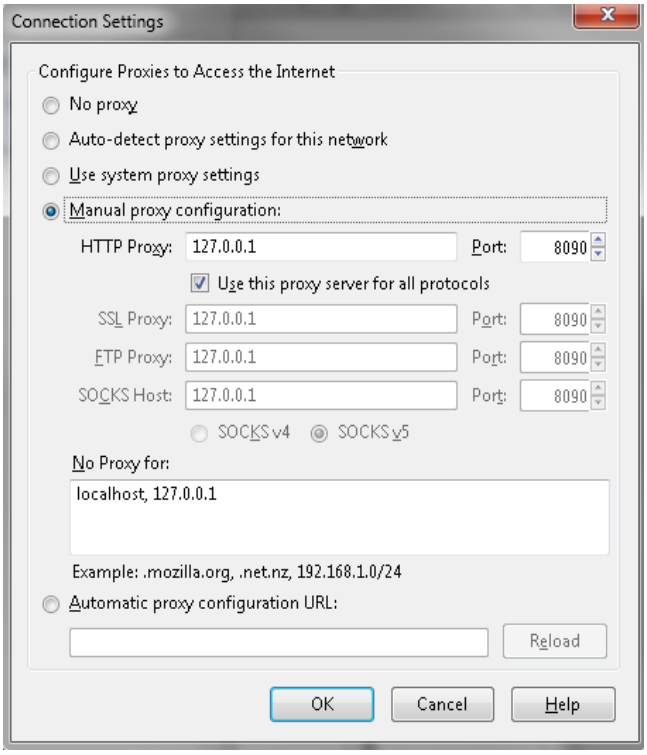


Tabla 11: Configuración de conexión en Firefox para redirigir las peticiones a través de un servidor proxy.

Para finalizar solo será necesario hacer clic sobre el botón "Ok".

B.1.1.3 Envío de peticiones a través del proxy.

Una vez configurada tanto la aplicación como el navegador web, será posible llevar a cabo la interceptación de peticiones HTTP. Todas las peticiones que sean enviadas por parte de la aplicación externa que haga uso del proxy podrán ser visualizadas en la En la sección "Intercept".

Cuando se solicite una página en el navegador web, los datos de la petición HTTP/s serán visualizados por pantalla.

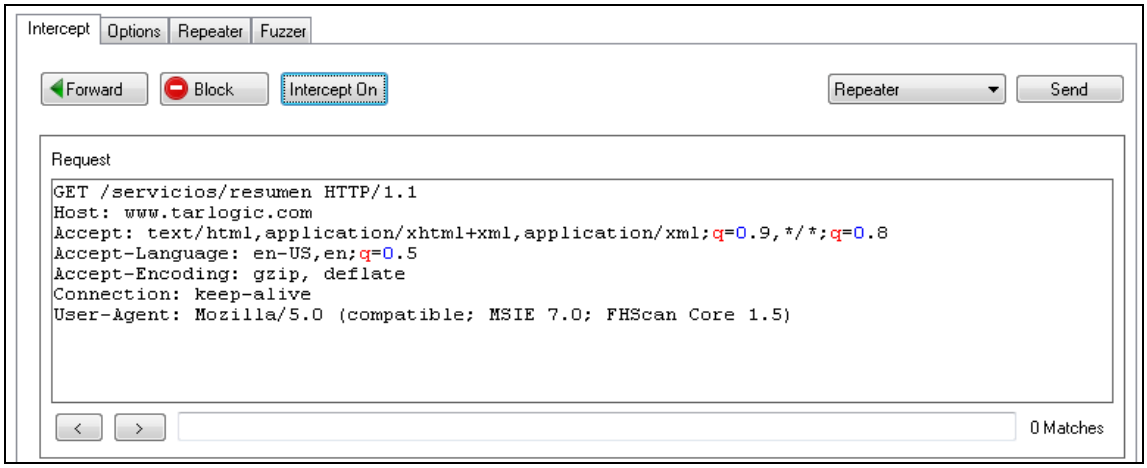


Figura 40: Petición HTTP enviada desde el navegador.

Como se muestra en la figura, la aplicación FHScan muestra los datos de la petición interceptada y permite al usuario realizar dos acciones: redirigir la petición al sistema destino (si hace clic sobre el botón "Forward"), o por el contrario bloquear la petición (Botón "Block"), y por lo tanto la petición HTTP no llegará a ser enviada.

Cuando una petición es bloqueada el proxy enviará un mensaje de error indicando la situación al usuario:

Figura 41: Mensaje de error indicando que la petición HTTP enviada por el navegador fue bloqueada.

El módulo de interceptación dispone de dos modos de funcionamiento "**Intercept On**" e "**Intercept Off**". Cuando la interceptación esté activa las peticiones HTTP serán mostradas por pantalla. Si la interceptación está desactivada las peticiones serán redirigidas de forma automática.

B.1.1.4 Visualización de peticiones enviadas

Todas las peticiones enviadas a través del proxy pueden ser visualizadas posteriormente. Para ello la aplicación dispone de dos modos de visualización. El primero de ellos es el Sitemap, que muestra una organización jerárquica de los recursos que han sido solicitados a un sistema host concreto.

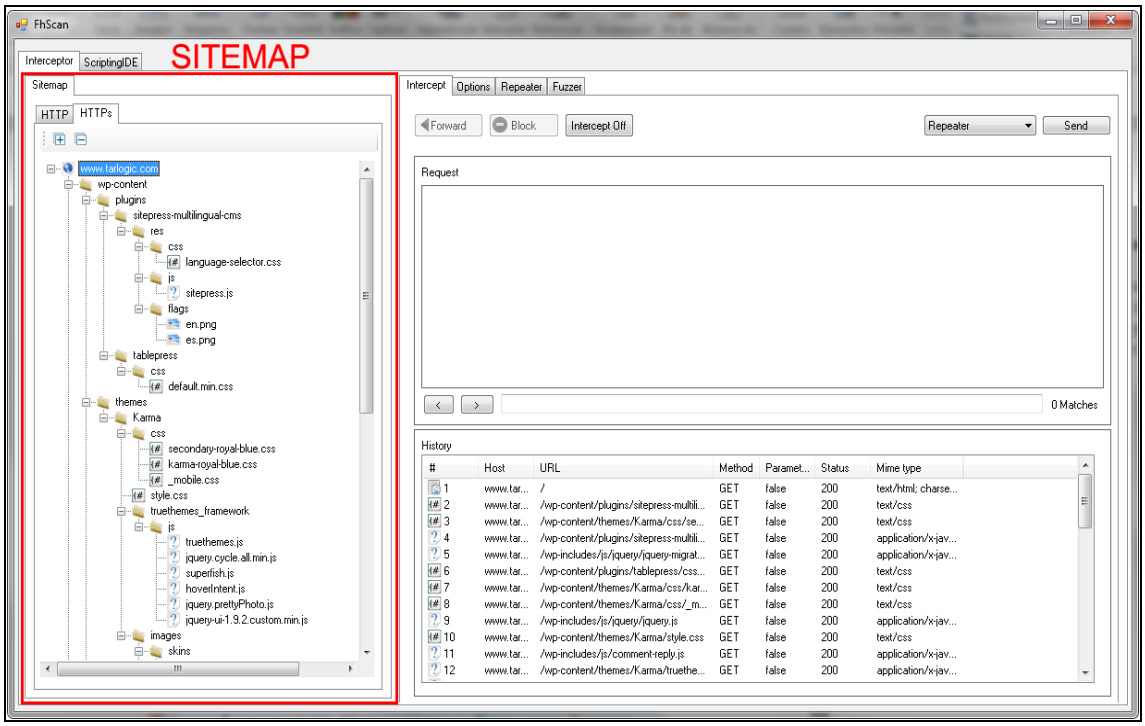


Figura 42: Visualización jerárquica de los recursos solicitados (Sitemap).

Otra forma de visualizar las peticiones es a través del Historial ("History") situado en la sección inferior derecha. El historial proporciona mayor información de las peticiones realizadas, permitiendo visualizar el nombre del host al que se envió la petición, la URL solicitada, el método empleado, el código HTTP de respuesta y el tipo mime de la respuesta.

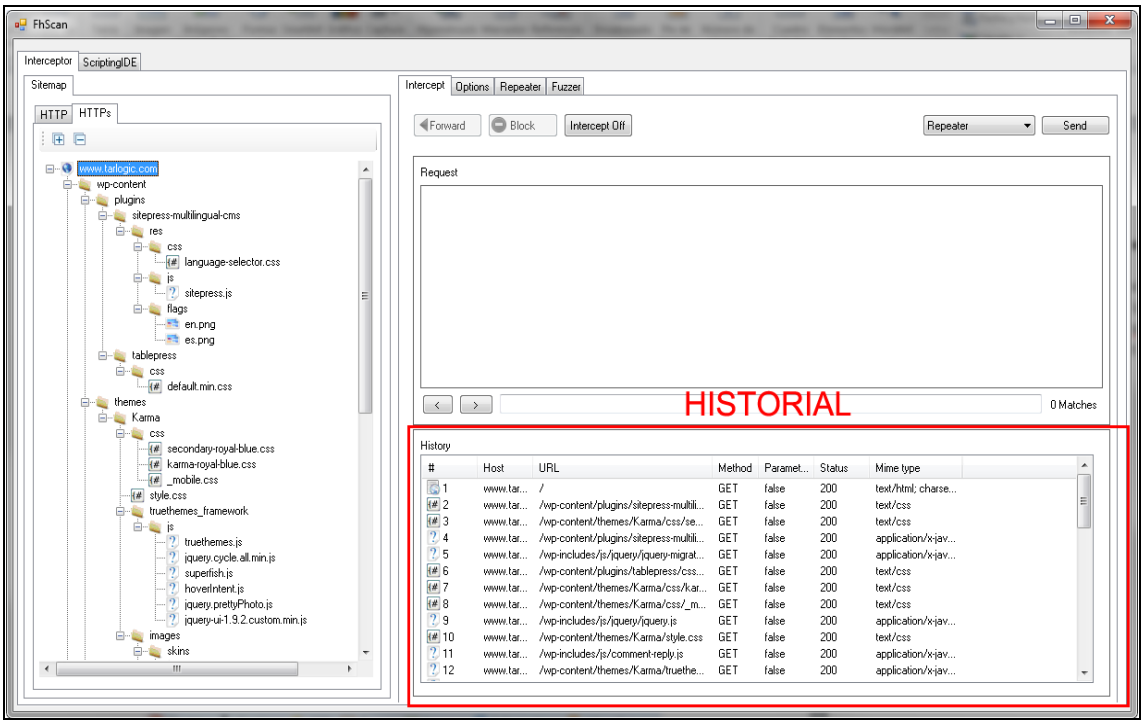


Figura 43: Historial de peticiones realizadas.

Si se selecciona una petición del historial se abrirá una ventana en la cual será posible visualizar tanto los datos de la petición HTTP como de la respuesta asociada.

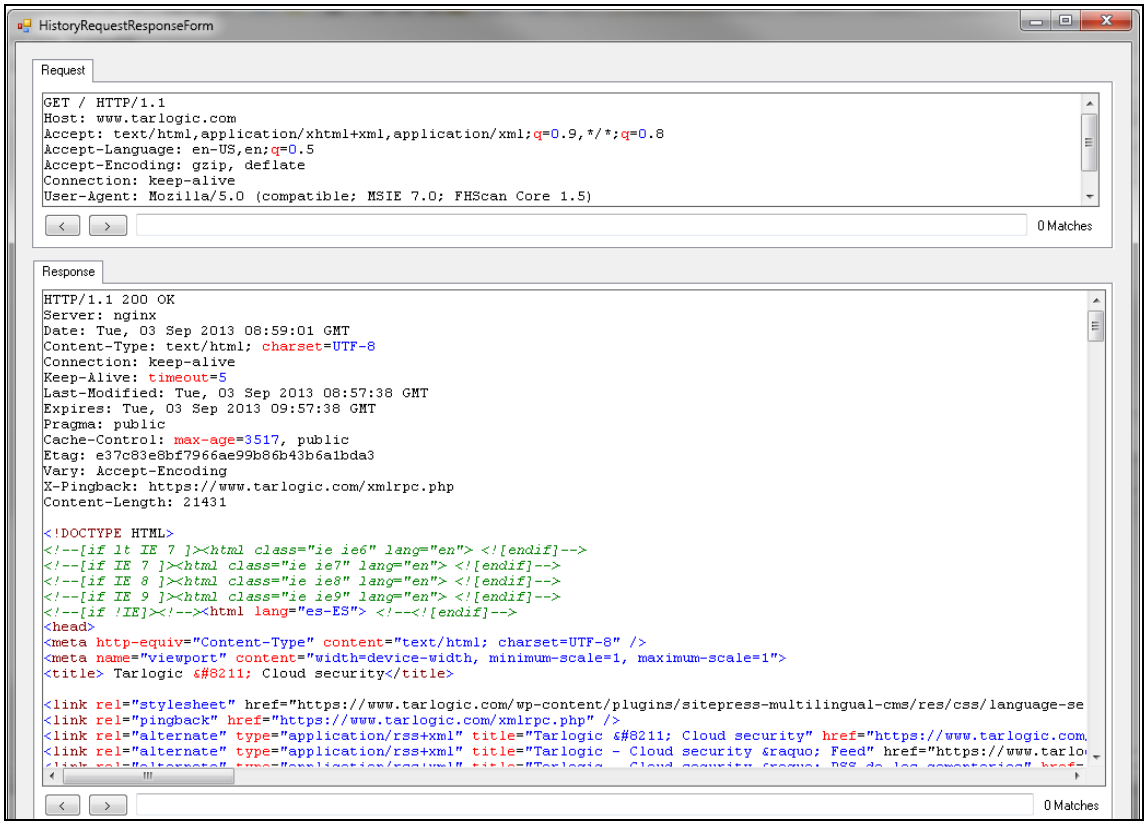


Figura 44: Visualización de datos de una petición HTTP así como su respuesta asociada.

B.1.2 Reenvío de peticiones HTTP (Repeater).

FHScan permite la construcción de peticiones HTTP de forma manual, a través de la funcionalidad de repetición, disponible en la pestaña "**Repeater**" de la sección "**Interceptor**"

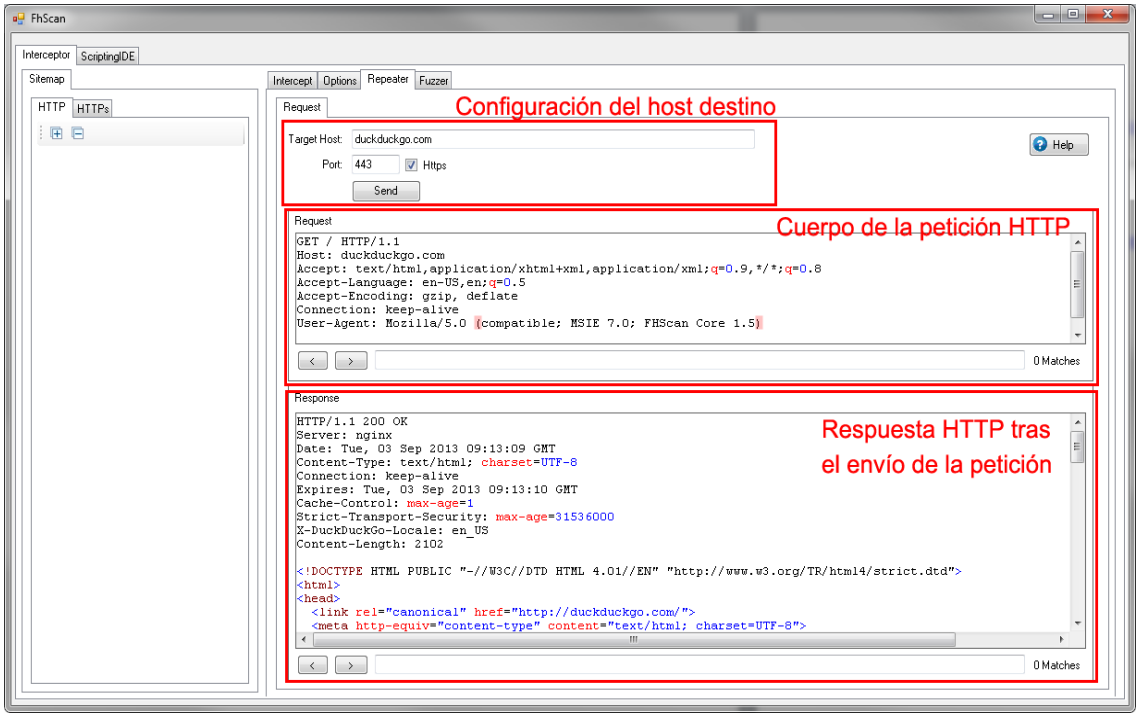


Figura 45: Módulo de reenvío de peticiones (Repeater).

Para construir una petición será necesario especificar tanto el nombre del host, el puerto y si la comunicación se debe realizar a través de un HTTP o HTTPs, así como el cuerpo de la petición HTTP que se desea enviar.

Una vez establecidos los parámetros será posible enviar la petición haciendo "Clic" sobre el botón "**Send**". Tras el envío de la petición los datos de respuesta podrán ser visualizados por el usuario, como se observa en la figura.

B.1.3 Fuzzer

El módulo de fuzzer permite realizar procedimientos iterativos de fuzzing mediante el envío de peticiones HTTP de forma automática. La ejecución de pruebas de fuzzing ayuda a verificar el comportamiento de una aplicación web ante distintos escenarios, permitiendo seleccionar partes de una petición HTTP que serán reemplazadas por los valores (Payloads) configurados por el usuario.

B.1.3.1 Tipos de ataques de fuzzing

La aplicación dispone de 3 tipos de ataques de fuzzing:

Single: El ataque de fuzzing "**Single**" permite realizar pruebas de fuzzing, seleccionando distintos parámetros de una petición HTTP, que serán reemplazados de forma independiente por un único valor de la lista de payloads. Por lo tanto se enviarán (n x m) peticiones HTTP, donde n es el número de parámetros y m es el número de valores de payload definidos.

Equal: El ataque de fuzzing "**Equal**" permite realizar pruebas de fuzzing, seleccionando distintos parámetros de una petición HTTP, que serán reemplazados al mismo tiempo por cada uno de los valores definidos en la lista de Payloads. Por lo tanto se enviaran m peticiones HTTP, donde m es el número de valores de payload definidos.

Independent: El ataque de fuzzing "**Independent**" permite asignar distintos valores de payload a distintos parámetros, de forma que, cada parámetro puede ser reemplazado por una lista definida para dicho parámetro de forma independiente.

B.1.3.2 Selección de host objetivo

Al igual que en el módulo de repetición será necesario definir el host objetivo al que serán enviadas las peticiones. Para definir el host objetivo es necesario acceder a la pestaña Target de la sección Fuzzer.

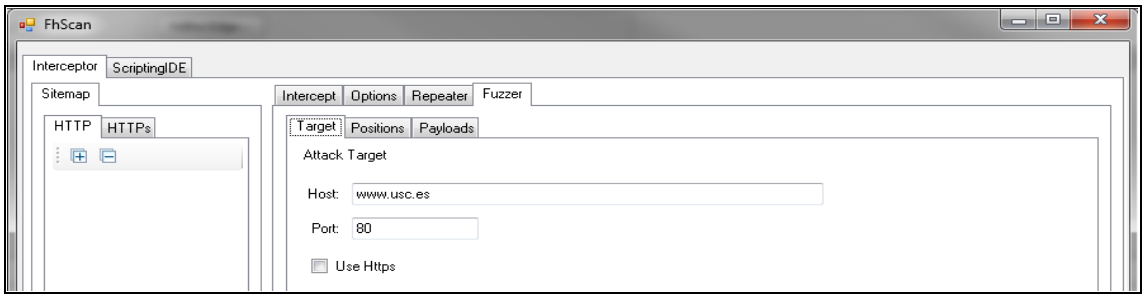


Figura 46: Selección de host objetivo en el módulo de fuzzing.

Una vez se ha accedido a la pestaña “Target” será posible definir el nombre del host objetivo, el puerto, así como definir si la comunicación se debe realizar a través de HTTP o HTTPs.

B.1.3.3 Selección de parámetros

Para seleccionar los parámetros de una petición HTTP que serán reemplazados es necesario acceder a la pestaña “**Parameters**” de la sección Fuzzer.

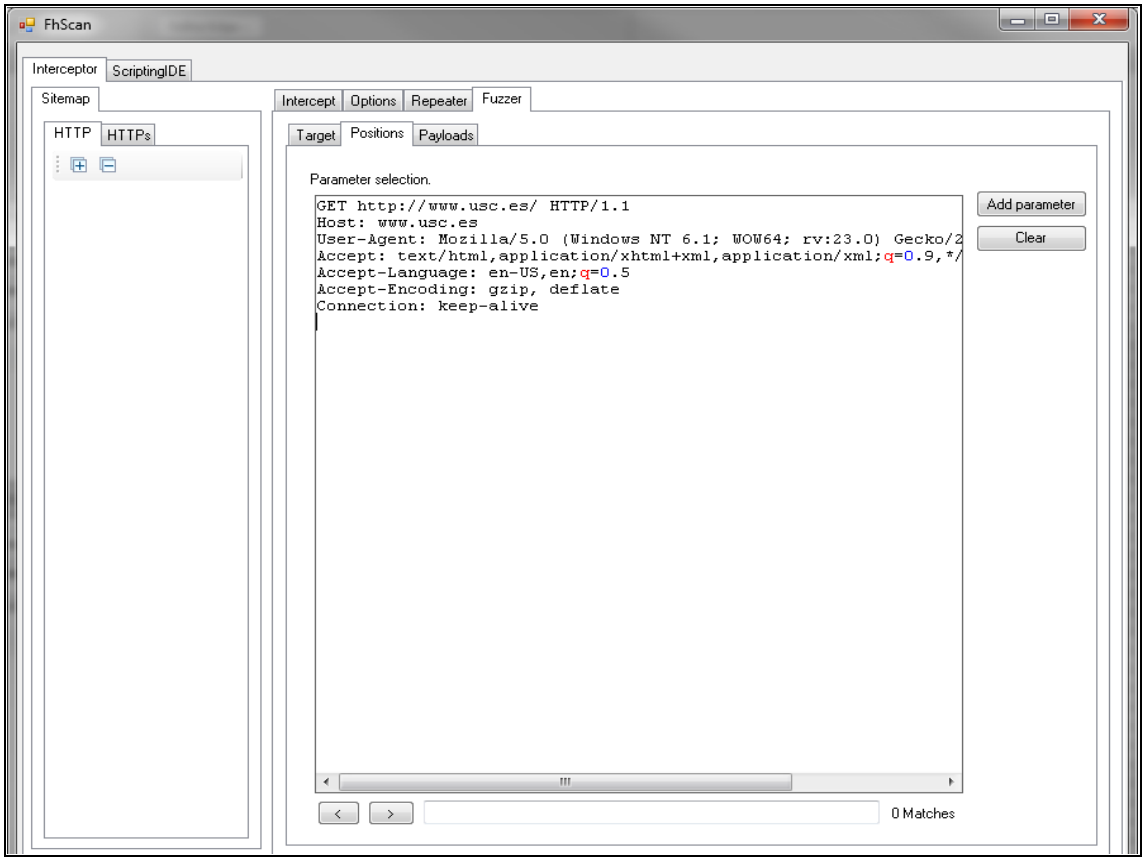


Figura 47: Selección de parámetros en el módulo de fuzzing.

Para añadir un parámetro se deberá seleccionar el texto de una petición HTTP que se desea añadir como parámetro y a continuación hacer “clic” sobre el botón “**Add parameter**”.

Cuando un parámetro ha sido seleccionado, éste será marcado en la petición utilizando el carácter “¶”. Como se muestra en la siguiente figura.

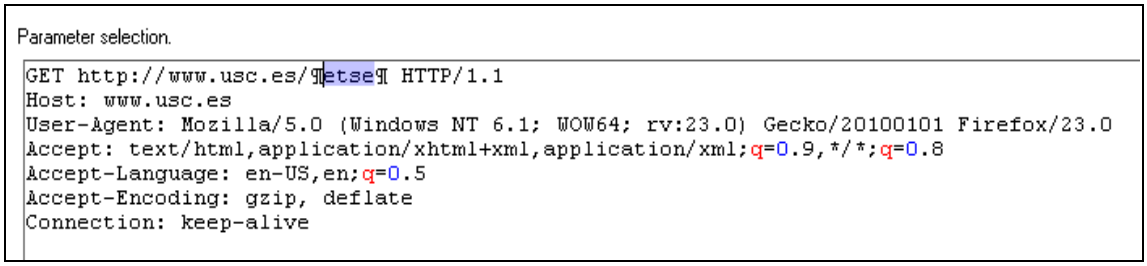


Figura 48: Selección de un parámetro.

B.1.3.4 Asignación de payloads

Las listas de payloads definen aquellos valores que serán reemplazados por los valores seleccionados como parámetros. La herramienta contempla dos tipos de valores payload.

Lista de valores numéricos: Las listas de valores numéricos permiten definir un rango de valores numéricos que serán reemplazados por los parámetros seleccionados en una petición. Para llevar a cabo la asignación de un payload numérico será necesario acceder a la pestaña "Payloads" y, en primer lugar seleccionar el índice del parámetro al que se desea asignar un Payload. (El índice define el parámetro sobre el que se realizará la asignación del payload).

Una vez se ha seleccionado el índice del parámetro, será posible definir el tipo de payload desplegando la lista "Payload Type" y seleccionando el valor "Numbers".

Figura 49: Selección de payload numérico.

Tras la selección del tipo de payload, la aplicación mostrará un panel en el que será posible definir la lista de valores numéricos que serán reemplazados por el parámetro seleccionado.

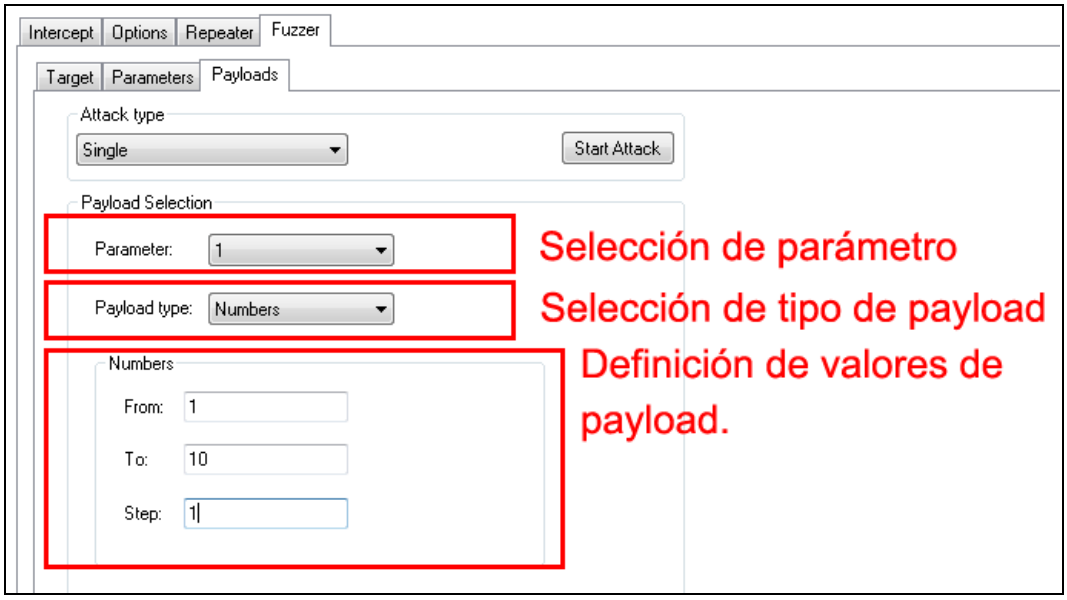


Figura 50: Definición de valores de payload numéricos.

Para definir los valores numéricos se deberá especificar el valor numérico inicial (a través de la opción "**From"**). El valor final (opción "**To"**) y el "**Step"** que permite indicar cuál es el valor de salto que se sumará al valor definido en la opción **From** al realizar cada iteración.

Los valores se definen como los parámetros de un "bucle for" en programación.

Lista de valores textuales:

FHScan permite definir listas de payload textuales en las cuales se definen un conjunto de cadenas de texto que reemplazarán al parámetro seleccionado. Para definir una lista textual será necesario seleccionar tipo de payload "List".

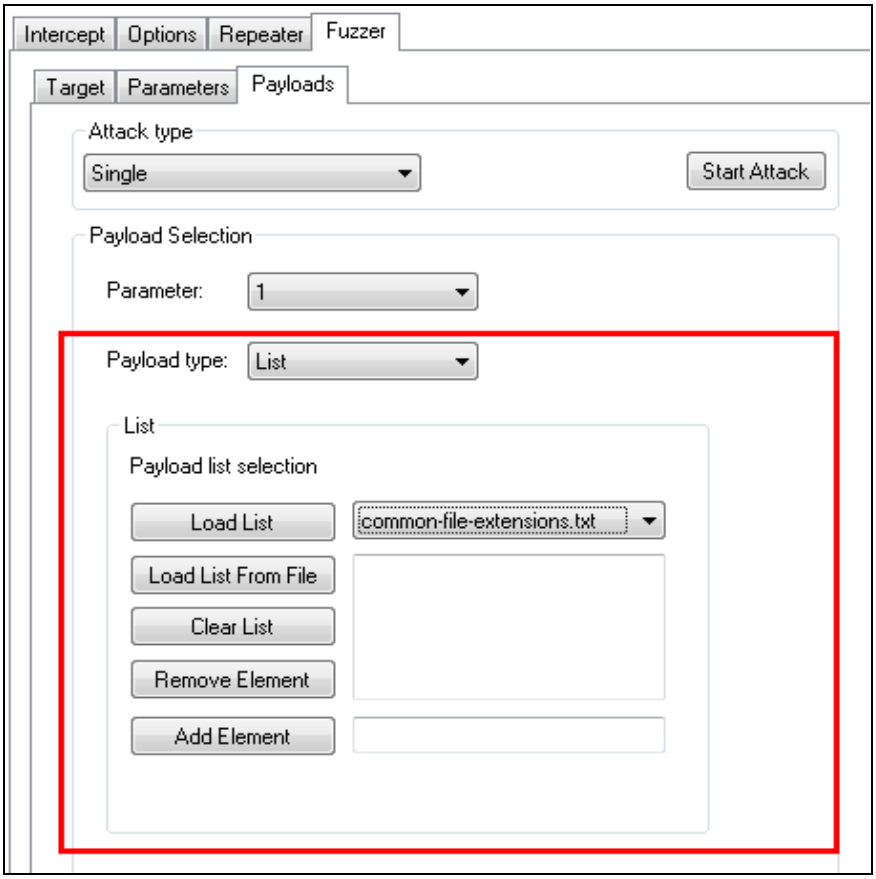


Figura 51: Listas de payload textuales.

La aplicación dispone de un conjunto de listas predefinidas, que contienen valores que permitirán llevar a cabo la ejecución de pruebas concretas. Algunas de las listas definidas son:

- **Common-file-extensions**: Define un conjunto de extensiones de ficheros (Ej: txt, html, php). Esta lista de valores permite probar extensiones de ficheros. La lista de utilidad en pruebas de descubrimiento de recursos.

- **XSS**: Define un conjunto de valores que permiten realizar pruebas para la identificación de vulnerabilidades "Cross-Site-Scripting" (XSS), las cuales permiten inyectar código javascript en una página web cuando ésta no realiza una validación correcta de los parámetros de entrada.

- **SQL Injection**: Define un conjunto de valores que permiten realizar pruebas para la identificación de vulnerabilidades de inyección SQL, basadas en la inyección de directivas SQL (a través de los parámetros de entrada de la aplicación) en las consultas realizadas contra la base de datos, cuando la aplicación no realiza una correcta validación de los datos de entrada.
  Este tipo de vulnerabilidades pueden permitir la extracción, modificación y borrado de datos de una base de datos.

Para seleccionar una lista predefinida será necesario seleccionar el nombre de la lista en el desplegable de listas, y pulsar el botón "**Load List**".

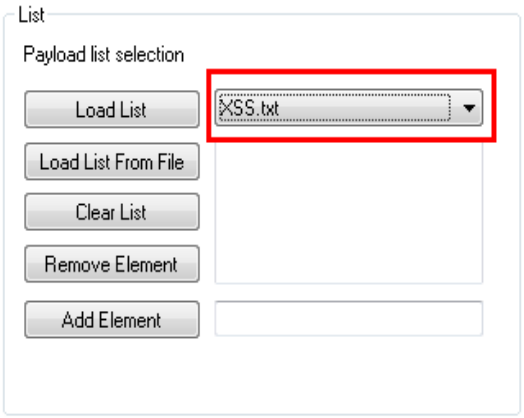


Figura 52: Selección de listas textuales predefinidas.

Una vez cargada la lista será posible visualizar los valores de la misma.

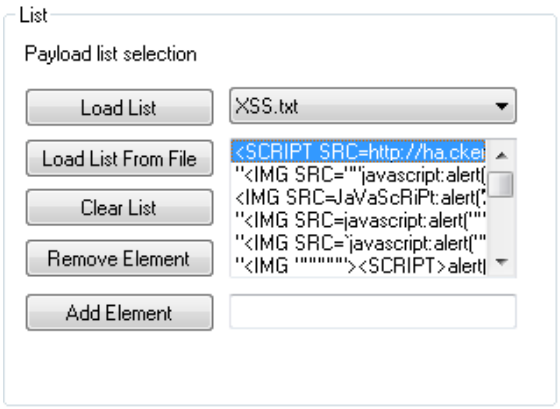


Figura 53: Visualización de valores de lista de payload textual.

La aplicación permite además añadir valores payloads a la lista (**Add element**), eliminar elementos de la lista (**Remove element**), así como cargar listas de payloads desde un fichero (**Load List From File**).

B.1.3.5 Ejecución de ataques de fuzzing.

Para iniciar un ataque de fuzzing será necesario haber definido el host objetivo, haber seleccionado los parámetros que se desean reemplazar, haber asignado el tipo de payload específico, y definir el tipo de ataque de fuzzing que se desea realizar:

Figura 54: Selección de tipo de ataque.

Una vez establecidas todas las opciones es posible ejecutar el ataque/prueba de fuzzing seleccionado el botón "**Start Attack**".

Tras haber hecho clic sobre el botón "**Start Attack**" la aplicación comenzará a enviar las peticiones HTTP y mostrará una ventana en la que será posible consultar tanto las peticiones realizadas como las respuestas asociadas.

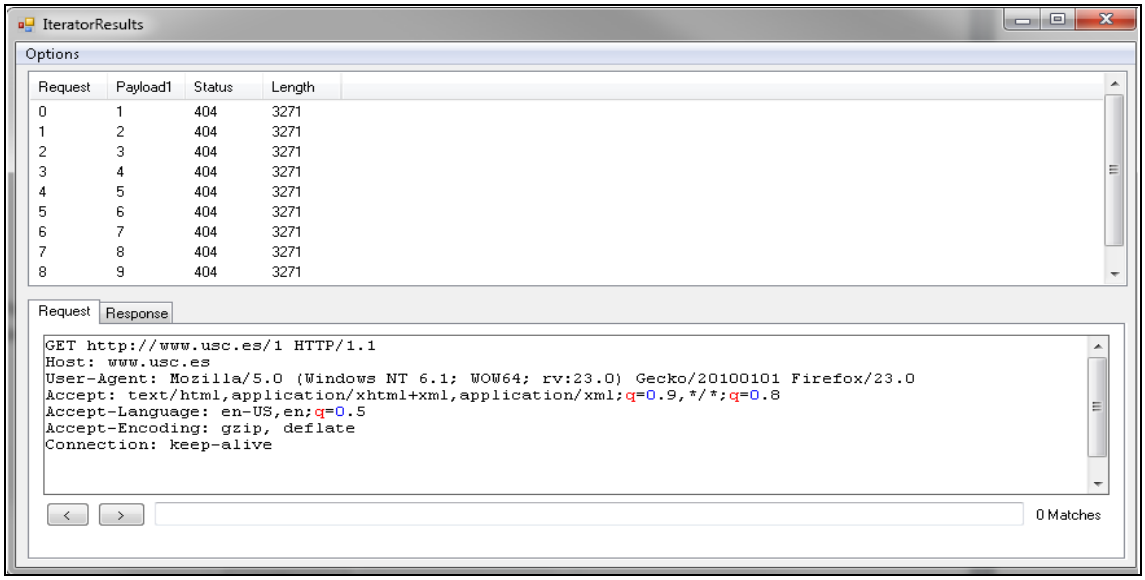


Figura 55: Resultados de pruebas de fuzzing.

B.4 Scripting IDE

La sección "**ScriptingIDE**" permite la creación, edición, compilación y ejecución de scripts. Esta sección dispone de 4 partes diferenciadas:

- **Ventana de edición de scripts**, a través de la cual es posible editar el código de un script.

- **Ventana de selección de parámetros de entrada para un script,** a través de la cual es posible proporcionar los valores de entrada para aquellos scripts que así lo requieran (**pestaña Script XML parameters**)

- **Ventana de visualización de salida**: que permite mostrar los resultados de compilación y ejecución de scripts (**pestaña Output**).

- **Repositorio de scripts de auditoria:** El repositorio de scripts de auditoria permite almacenar los scripts desarrollados con el objetivo de permitir su carga y ejecución de forma rápida.

  El repositorio define 5 categorías principales que se corresponden con las fases de un test de intrusión:

  - **Reconnaissance:** Contiene aquellos scripts que permiten ejecutar acciones de descubrimiento, como pueden ser la identificación del servidor web de un host o la identificación de recursos de un sistema.
  - **Scanning:** Contiene aquellos scripts que permiten la identificación de vulnerabilidades.
  - **Exploitation:** Contiene scripts que permiten explotar vulnerabilidades.
  - **Post-Exploitation:** Contiene scripts que permiten mantener el acceso a un sistema tras la explotación de una vulnerabilidad.
  - **Other:** Contiene aquellos scripts que no se corresponden con ninguna categoría anterior.

- **Repositorio de scripts de callback:** El repositorio de scripts de callback almacena los scripts de callback que permiten modificar el comportamiento del motor de comunicaciones HTTP y del servidor proxy.
  El repositorio define 4 categorías principales que se corresponden con los puntos en los que es posible llevar a cabo la modificación del comportamiento:

  - **cbClientRequest**: Antes del envío de una petición HTTP por parte del motor de comunicaciones. (Esto es, a través de la invocación de funciones de envío de peticiones).
  - **cbClientResponse**: Tras la recepción de una respuesta HTTP por parte del motor de comunicaciones

- **cbProxyRequest**: Antes del envío de una petición por parte del servidor proxy HTTP.
- **cbProxyResponse**: Tras la recepción de una respuesta por parte del servidor proxy HTTP.

Figura 56: Subsecciones del módulo "Scripting IDE".

B.4.1 Creación de un script.

Para crear un nuevo script es necesario seleccionar el botón "New Script" de la barra de herramientas.

Figura 57: Botón "New Script" Marcado en rojo.

Tras pulsar el botón se desplegara una ventana en la cual podremos especificar el nombre del script que se va a crear, el tipo de script y la categoría del script para el tipo seleccionado.

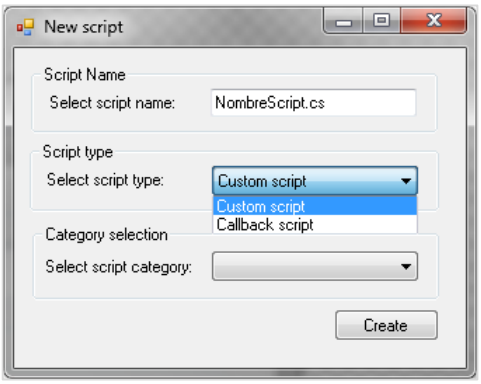


Figura 58: Creación de un script.

Una vez seleccionados estos parámetros, pulsando el botón "Create", se creará una nueva pestaña que permitirá editar el script.

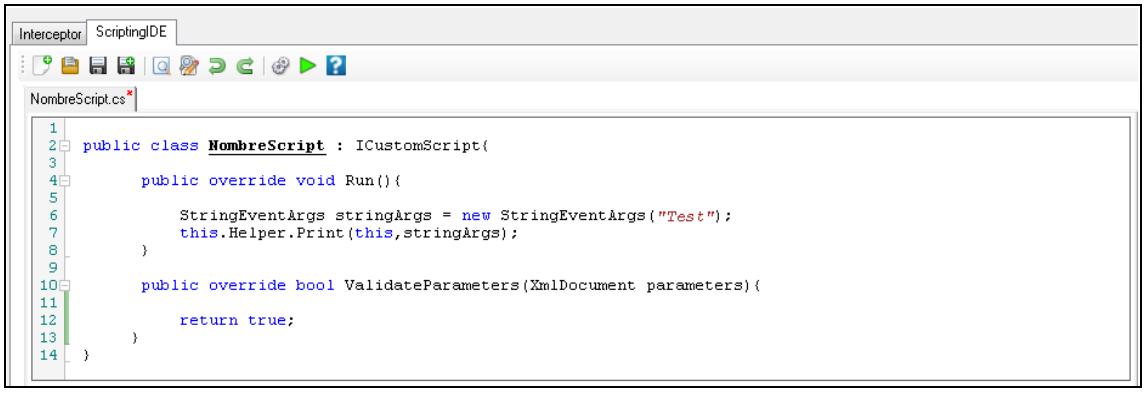


Figura 59: Edición de script una vez creado.

B.4.2 Añadir un script al repositorio

Para añadir un script al repositorio de scripts se debe: en primer lugar seleccionar la pestaña del script que se desee añadir al repositorio y a continuación seleccionar el botón "Add Script to library",

Figura 60: Añadir script al repositorio.

Tras seleccionar el botón se desplegará un menú igual al que se muestra en el proceso de creación de script a través del cual podremos definir el nombre con el que se deseará guardar el script, así como el tipo y la categoría concreta.

B.4.3 Compilar script

Para compilar un script será necesario es necesario seleccionar el botón "Compile" en la barra de herramientas.

Figura 61: Compilación de un script.

Una vez ejecutada la compilación del script se mostrarán los resultados en ventana de salida "output".

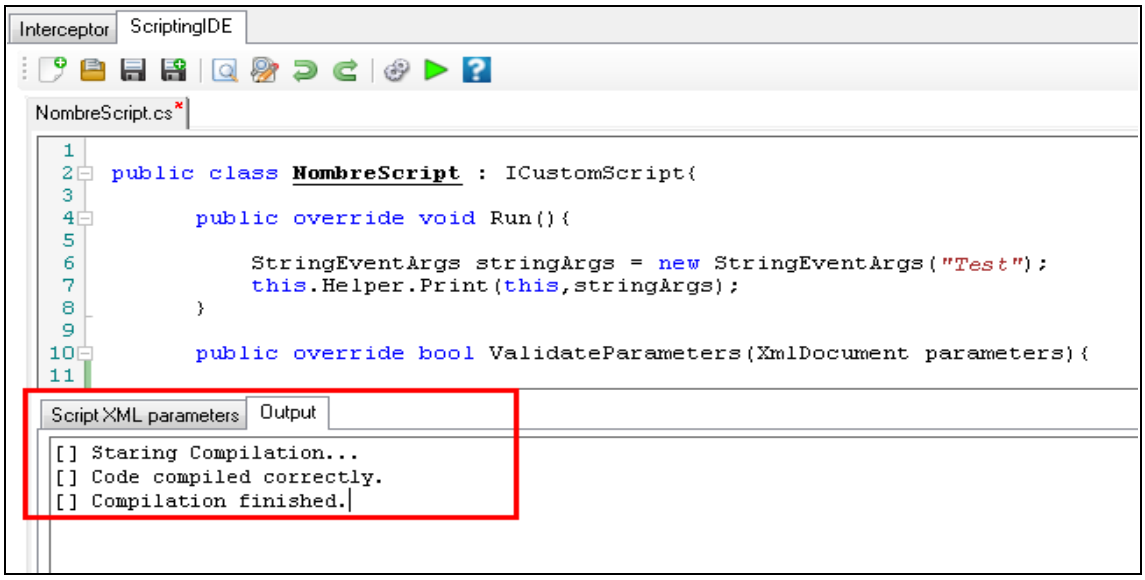


Figura 62: Resultados de la compilación.

B.4.4 Ejecución de scripts

Para ejecutar un script de auditoría ("CustomScript") se debe hacer clic sobre el botón "Run".

Figura 63: Ejecución de scripts de auditoría.

Al igual que en el proceso de compilación los resultados se mostrarán en la ventana de salida "output".

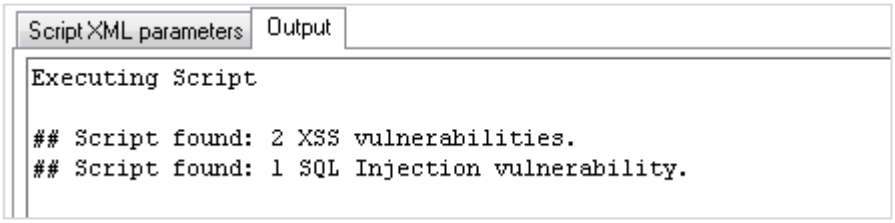


Figura 64: Resultados de ejecución de un script de auditoría "CustomScript".

Si la ejecución del script requiere parámetros de entrada, es posible proporcionarlos a través de la ventana de selección de parámetros de entrada (Script XML Parameters). Los parámetros de entrada deben de ser definidos mediante un documento XML. Para aportar mayor flexibilidad a la creación de scripts, el formato de los parámetros de entrada debe ser definido por el script concreto.

Figura 65: Definición de parámetros de entrada para el script.

B.4.5 Activación y desactivación de scripts de callback.

Los scripts de callback a diferencia de los scripts de auditoría se deben activar en alguno de los puntos comentados anteriormente. Para llevar a cabo la activación de un script éste debe estar incluido en el repositorio de scripts de callback.

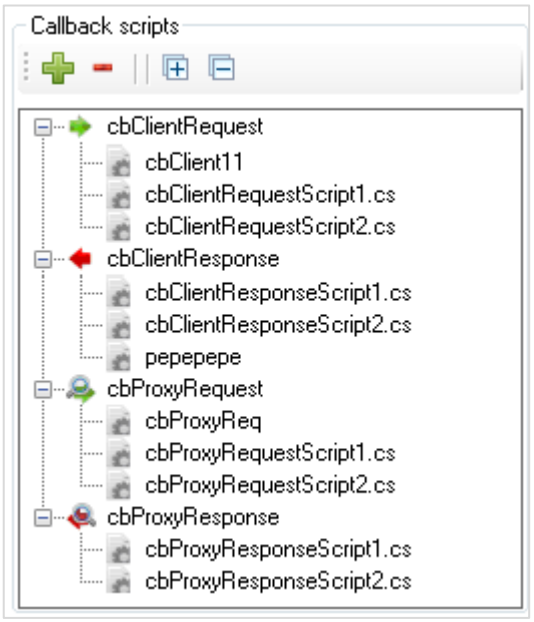


Figura 66: Repositorio de scripts de callback.

La activación de un script requiere su selección en la lista mostrada en el repositorio y a continuación hacer clic en el botón "**Activate Callback Script**"

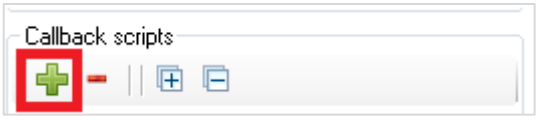


Figura 67: Botón de activación de script de callback.

Una vez activado, el script se marcará como activo, como se puede ver en la siguiente Figura.

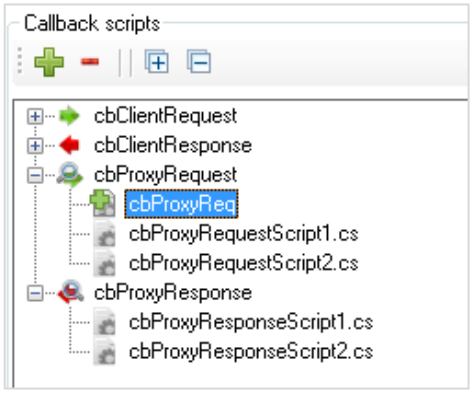


Figura 68: Script de callback activo.

Para desactivar un script activado previamente será necesario seleccionar el script activo y pulsar el botón "**Deactivate callback script**".

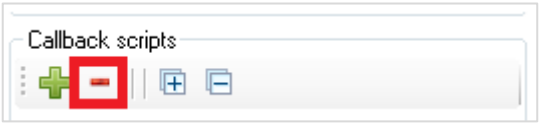


Figura 69: Botón de desactivación de script de callback.

Tras la desactivación del script, el comportamiento del motor de comunicaciones HTTP/s o del servidor proxy volverá a su estado original.

Bibliografía

1. **OWASP.** Testing Guide V3. [Online] 2008. https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf.
2. **Tarascó, Andrés.** HTTPCore. [Online] http://www.tarasco.org/security/fhscan_core_library_api/index.html.
3. **Wikipedia.** Callback. [Online] http://es.wikipedia.org/wiki/Callback_(inform%C3%A1tica).
4. **Michael Sutton, Adam Greene, Pedram Amini.** *Fuzzing: Brute Force Vulnerability Discovery.* s.l. : Addison Wesley, 2007.
5. **MSDN, Microsoft.** Código Administrado. [Online] http://msdn.microsoft.com/en-us/library/windows/desktop/bb318664(v=vs.85).aspx.
6. —. Código no administrado. [Online] http://msdn.microsoft.com/en-us/library/0e91td57.aspx.
7. **Martin, Robert C. Martin & Micah.** *Agile Principles, Patterns, and Practices in C#.* s.l. : Prentice Hall, 2006.
8. **OMG.** UML. [Online] http://www.uml.org/.
9. **MSDN, Microsoft.** CLR - Common Language Runtime . [Online] http://msdn.microsoft.com/es-es/library/8bs2ecf4.aspx.
10. **Wikipedia.** CIL - Common Intermediate Language. [Online] http://es.wikipedia.org/wiki/Common_Intermediate_Language.
11. **Tarlogic Security S.L.** [Online] https://www.tarlogic.com.
12. **IEEE.** 830-1998 - Recommended Practice for Software Requirements Specifications. [Online] http://standards.ieee.org/findstds/standard/830-1998.html.
13. **Develop, Mono.** [Online] http://monodevelop.com/.
14. **MSDN, Microsoft.** Interfaz IDisposable. [Online] http://msdn.microsoft.com/es-es/library/system.idisposable.aspx.
15. **PMI.** *Fundamentos para la dirección de proyectos (Guía del PMBOK).* 2009.
16. **Tanenbaum, Andrew S.** *(Sistemas operativos modernos, 3ª Edición) pag 39,117-145.* s.l. : Pearson, Prentice Hall, 2009.
17. —. *(Sistemas Operativos Modernos, 3ª Edición), pag 596-597.* s.l. : Pearson, Prentice Hall., 2008.
18. **Wikipedia.** ABI (Application Binary Interface). [Online] https://en.wikipedia.org/wiki/Application_binary_interface.
19. —. Lenguaje de programación C. [Online] https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n).
20. **MSDN, Microsoft.** P/Invoke (Platform Invoke). [Online] http://msdn.microsoft.com/en-us/library/aa288468(v=vs.71).aspx.
21. **Fowler, Martin.** GUI Architectures, MVP. [Online] http://martinfowler.com/eaaDev/uiArchs.html.
22. —. *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002.
23. **Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley Professional, 1994.

24. **Martin, Robert C. Martin & Micah.** *Agile Principles, Patterns and Practices in C#, pag 325,326.* s.l. : Prentice Hall, 2006.
25. —. *Agile Principles, Patterns and Practices in C#, pag 327.* s.l. : Prentice Hall, 2006.
26. —. *Agile Principles, Patterns and Practices in C#, pag 331,333.* s.l. : Prentice Hall, 2006.
27. **MSDN, Microsoft.** Exporting from a DLL using __declspec(dllexport). [Online] http://msdn.microsoft.com/en-us/library/a90k134d.aspx.
28. **Contributors, Wikiibook.** *x86 Disassembly, pag 61.* 2011.
29. **Wikipedia.** Namespace. [Online] http://en.wikipedia.org/wiki/Namespace_(C%2B%2B)#In_Programming_Languages.
30. **Russinovich, Mark.** Dependency Walker. [Online] http://www.dependencywalker.com/.
31. **Graham, Mike McShaffry & David "Rez".** *Game Coding Complete 4ª Edición, Memory Leaks pag 835-839.* s.l. : Course Technology Cengage Learning, 2013.
32. **Wikipedia.** Handle. [Online] http://es.wikipedia.org/wiki/Handle.
33. **Contributors, Wikibook.** *x86 Dissasembly, pag 57.* 2011.
34. —. *x86 Dissasembly, pag 58.* 2011.
35. —. *x86 Dissasembly, pag 59.* 2011.
36. **Wikipedia.** Unicode. [Online] https://en.wikipedia.org/wiki/Unicode.
37. **Seacord., Robert C.** Top 10 Secure Coding Practices. [Online] https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices.
38. **Seacord, Robert C.** *The Cert C Secure Coding Standard.* s.l. : Addison Wesley., 2008.
39. —. *Secure coding in C and C++, 2ª Edición.* s.l. : Addison Wesley, 2013.
40. **Code, Google.** Rough Auditing Tool For Security. [Online] https://code.google.com/p/rough-auditing-tool-for-security/.
41. **Seacord, Robert C.** *Secure Coding in C and C++, pag 53-54, 70, 118-120.* s.l. : Addison Wesley., 2013.
42. —. *Secure Coding in C and C++, 2ª Edición, pag 362-364.* s.l. : Addison Wesley, 2013.
43. **Mark Dowd, John McDonald, Justin Schuh.** *The Art of Software Security Assesment, pag 680.* s.l. : Addison Wesley, 2007.
44. **Nielsen, Jakob.** 10 Usability Heuristics for User Interface Design. [Online] http://www.nngroup.com/articles/ten-usability-heuristics/.
45. **Seacord, Robert C.** *Secure Coding in C and C++, 2ª Edición.* s.l. : Addison Wesley., 2013.