



UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

Análisis del tráfico de red para la detección de ataques Informáticos

Alumno: Gregorio Carazo Maza

Tutor: Prof. D. Manuel José Lucena López
Dpto: Departamento de Informática

Febrero, 2017



Universidad de Jaén
Escuela Politécnica Superior de Jaén
Grado en Ingeniería Informática

Don Manuel José Lucena López, tutor del Trabajo Fin de Grado titulado: Análisis del tráfico de red para la detección de ataques Informáticos, que presenta Gregorio Carazo Maza, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Febrero de 2017

El alumno:

Gregorio Carazo Maza

El Tutor:

Manuel José Lucena López

Índice

1. Propósito del proyecto.....	5
2. Introducción.....	5
2.1 Descripción del problema	5
2.2 Descripción de la solución.....	6
2.2.1 Qué es un IDS.....	6
2.2.2 Para qué se usa	7
3. Metodología	7
4. Aplicación.....	9
4.1 Análisis de requisitos.....	9
4.1.1 Necesidades del usuario	9
4.1.2 Ataques a detectar	10
4.1.2.1 Basado en TTL	10
4.1.2.2 Basado en RST	12
4.1.2.3 Basado en SYN	13
4.1.2.4 Fragmentación u Overlapping.....	15
4.1.3 Planificación.....	16
4.1.3.1 Presupuesto	18
4.1.3.1.1 Hardware	19
4.1.3.1.2 Software.....	19
4.1.3.1.3 Instalación y configuración	20
4.2 Diseño del sistema.....	20
4.2.1 Librerías utilizadas y lenguaje de desarrollo.....	20
4.2.2 Estructura de la aplicación	22
4.2.3 UML y diagrama de clases	26
4.2.3.1 Proyecto_MainWindows	28
4.2.3.2 Analyst.....	33
4.2.3.3 Cleaner.....	36
4.2.3.4 jDialogError	37
4.2.3.5 jDialogConfig	38
4.2.3.6 errorAnalyst	40
4.2.3.7 jDialogTableRoute	41
4.2.3.8 jDialogActivateUpdateTable.....	42
4.2.3.9 jDialogHelp	43
4.2.3.10 updateTableClass	45
4.2.3.11 jDialogData	46

4.2.3.12	jDialogInsertDate.....	46
4.2.3.13	jDialogSelectDevice	48
4.3	Implementación.....	48
4.3.1	Decisiones de diseño	49
4.3.1.1	Modularidad.....	49
4.3.1.2	Uso de hilos.....	49
4.3.1.3	Inclusión de la opción “ <i>stop update</i> ”	49
4.3.1.4	Introducción de la opción “ <i>insert table routing</i> ”	50
4.3.1.5	Ejecución del hilo <i>cleaner</i>	50
4.3.1.6	Un solo hilo <i>sniffer</i>	50
4.3.1.7	Espera de la finalización del hilo <i>sniffer</i>	50
4.3.1.8	Selección del servicio de red a monitorizar.....	51
4.3.1.9	Inserción de la fecha de borrado.....	51
4.3.1.10	Utilización de varios UpdateTable	51
4.3.1.11	Restauración de los mensajes	51
4.3.2	Problemas encontrados y solución adoptada	52
4.3.2.1	Almacenado en la base de datos.....	52
4.3.2.2	Comunicación entre el <i>sniffer</i> y el analizador.....	52
4.3.2.3	Comprobación de la fragmentación	52
4.3.2.4	Limitación del envío de la señal “updateTable”	53
4.3.2.5	Modularidad.....	53
4.4	Pruebas y resultados.....	53
5.	Conclusiones.....	64
6.	Anexos	65
7.1	Anexo I.....	65
7.2	Anexo II.....	67
7.3	Anexo III.....	69
7.4	Anexo IV	71
	Bibliografía	75

1. Propósito del proyecto

Este proyecto tiene como objetivo la creación de una aplicación que permita proteger una red de ordenadores frente a ataques externos, destinados a comprometerla.

Los sistemas por excelencia para realizar esta función son los IDS, y dado que estos sistemas son muy utilizados, existen muchas soluciones comerciales. El principal problema de estas es que se basan en la búsqueda de patrones dentro de las cadenas de caracteres de los paquetes, teniendo en cuenta este defecto, se han desarrollado ataques capaces de saltarse los sistemas que utilizan esta técnica de detección.

Los ataques para saltarse los IDS, consisten en la utilización de varios paquetes que camuflan el ataque, y por tanto lo hacen indetectable para las soluciones comerciales que se basan en la búsqueda de patrones, por lo que nosotros proponemos este programa como suplemento a los IDS tradicionales. Además debe cumplir los siguientes objetivos:

- Profundizar en el conocimiento de las técnicas de ataque a servidores o sistemas informáticos.
- Ser capaz de gestionar el tráfico de la red para detectar ataques que involucren uno o varios paquetes de datos.
- Creación de una aplicación multiplataforma que detecte posibles ataques a una red.
- Creación de una aplicación extensible.

2. Introducción

2.1 Descripción del problema

Con el paso de tiempo, tanto los medios para archivar la información como los medios para proteger dicha información han ido cambiando. En el pasado la forma de proteger los datos era colocándolos en una habitación con un candado, siendo

una persona la que se encargaba de vigilar que nadie sin permisos accediera a dichos datos.

Pero la tecnología ha avanzado, y con la proliferación de las TIC (Tecnologías de la Información y Comunicación), la información pasó de estar en archivadores dentro de una habitación guardada por una persona, a estar en ficheros digitales, manipulables desde cualquier lugar del mundo. Además actualmente la información debe ser accesible para varios usuarios con permisos simultáneamente. Ante esta perspectiva, la seguridad tuvo que avanzar, ya que de lo contrario toda la información sería accesible para cualquier persona.

2.2 Descripción de la solución

Con la aparición de internet y su tráfico de paquetes, apareció la figura del *encargado de seguridad virtual* llamado **sistema de detección de intrusiones** (o **IDS** por sus siglas en inglés *Intrusion Detection System*), este sistema tiene como misión la detección de accesos no autorizados a un sistema informático.

2.2.1 Qué es un IDS

Como hemos dicho, el IDS tiene la misión de detectar accesos no autorizados a sistemas informáticos, y así proteger la información que almacenan dichos sistemas. Esta información puede ser desde datos personales (nuestra dirección o número de DNI) hasta datos bancarios (cuentas corrientes, etc.), pasando por cualquier otra información sensible. La protección se realiza a través de sensores que permiten a los IDS detectar conexiones potencialmente peligrosas.

Existen principalmente cinco tipos de IDS:

- NIDS (*Network IDS*): Un IDS basado en red, que se instala en un segmento de la red y detecta los ataques dirigidos al segmento de la misma.
- HIDS (*host IDS*): Trabajan con la información recogida de un único host, por lo que si queremos monitorizar varios host necesitamos varios HIDS.
- DIDS (*Distributed IDS*): Son parecidos a los NIDS pero monitorizan varios puntos de entrada a la red para. Si se produce una alerta, la enviará a un sistema principal.

- Supervisores de archivos: Detectan y analizan patrones en los archivos del registro del sistema.
- Verificadores de integridad de los ficheros: Detectan alteraciones en los archivos (principalmente ejecutables y ficheros de configuración) del sistema.

2.2.2 Para qué se usa

Los IDS se pueden crear con el fin de detectar ataques, pero también se pueden implantar para realizar las siguientes funciones:

- Informar sobre posibles ataques.
- Permitir el bloqueo de un ataque y proteger el sistema.
- Proporcionar datos para perseguir a los atacantes.
- Almacenamiento de datos históricos
- Acumulación de evidencias.
- Automatizar la búsqueda de nuevos patrones de ataque.
- Monitorizar y analizar la actividad de los usuarios.
- Auditar las configuraciones y vulnerabilidades de un sistema
- Automatizar tareas de configuración y mejora del sistema.

3. Metodología

La metodología de desarrollo en cascada ha sido la elegida para desarrollar el proyecto. El modelo de desarrollo en cascada es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase.

Fases de la metodología:

1. Análisis de requisitos: En esta etapa debemos analizar las necesidades del usuario con el fin de obtener los objetivos del sistema. Tras la cual surge la especificación de los requisitos de forma completa, y de lo que

debe hacer la aplicación, sin entrar en vicisitudes internas de la misma. Debemos tener en cuenta que la fase debe ser consensuada, para que el *software* cubra las necesidades requeridas por el usuario.

2. Diseño del sistema: Descompone y organiza el sistema en elementos que puedan elaborarse por separado, aprovechando las ventajas del desarrollo en equipo. Tras esta fase debemos obtener una descripción total del *software*, así como una descripción detallada de las distintas áreas que componen el sistema.
3. Implementación: Como su propio nombre indica, en esta etapa debemos poner en código todo lo establecido en las fases anteriores. Obviamente debemos tener en cuenta el lenguaje de programación, así como todas las librerías seleccionadas en las fases anteriores, además de código reutilizable (al ser un trabajo fin de grado no hemos incluido código reutilizado).
4. Pruebas: Una vez implementado el sistema lo sometemos a las distintas condiciones en las que debe funcionar el sistema, con el fin de observar las reacciones del mismo y los resultados obtenidos en dichas pruebas, así como el cumplimiento de los objetivos.
5. Mantenimiento: Una vez desarrollado el sistema, el cual consiste en la realización del 75% del trabajo final, debemos mantenerlo con las actualizaciones pertinentes, así como cualquier otro aspecto que involucre a nuestro sistema, con el fin de que mantenga los objetivos iniciales del proyecto.

Ventajas:

- Realiza un buen funcionamiento en equipos débiles y productos maduros, por lo que se requiere de menos capital y herramientas para hacerlo funcionar de manera óptima.
- Es un modelo fácil de implementar y entender.
- Está orientado a documentos.
- Es un modelo conocido y utilizado con frecuencia.
- Promueve una metodología de trabajo efectiva: Definir antes que diseñar, diseñar antes que implementar.

Desventajas:

- ✓ El proceso de creación del *software* tarda mucho tiempo ya que debe pasar por el proceso de prueba y hasta que el *software* no esté completo no se opera. Esto es la base para que funcione bien.
- ✓ Cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo.
- ✓ Una etapa determinada del proyecto no se puede llevar a cabo a menos que se haya culminado la etapa anterior.

4. Aplicación

En este epígrafe vamos a proceder a explicar el trabajo realizado atendiendo a la metodología escogida, y agrupada según las fases de la misma.

4.1 Análisis de requisitos

En este apartado vamos a fijar elementos importantes para el cumplimiento de los objetivos establecidos en el apartado 1.

4.1.1 Necesidades del usuario

Este proyecto es un prototipo funcional de IDS, y por lo tanto su funcionamiento está habilitado, pero también es un programa diseñado para seguir desarrollándolo, y por tanto debemos diferenciar dos tipos de usuario, un usuario estándar y un desarrollador que quiera modificar el código para la introducción de ataques que puedan ser detectados.

Las necesidades de los usuarios estándar son:

- Sistema de bajos recursos: El programa debe optimizar el uso de la memoria y demás recursos del sistema, pero sin dejar de lado la funcionalidad, es decir, que el usuario pueda examinar los distintos paquetes que recibe.
- Multiplataforma: Tiene que ser un sistema capaz de ejecutarse sobre cualquier sistema operativo.

- GUI: El sistema se podría realizar con una línea de comandos, pero para el usuario estándar es más fácil el uso de una interfaz gráfica.
- Almacenamiento de ataques detectados: Una parte importante del sistema es el almacenamiento de aquellos paquetes que se catalogan como peligrosos, para un análisis más exhaustivo por parte del administrador del sistema, o del usuario.
- Limpieza del sistema: Con el fin de optimizar el espacio debemos dotar al usuario de una funcionalidad que pueda borrar los registros antiguos de la base de datos.

Para los desarrolladores debemos tener en cuenta:

- Sistema abierto al desarrollo: Con el fin de seguir el desarrollo hemos intentado crear un sistema fácil, que de la capacidad al usuario de detectar cualquier tipo de ataque, proporcionándole los campos necesarios para realizar la detección.

También hemos tenido en cuenta alguna funcionalidad común para ambos usuarios.

- Diversos grados de configuración: Con el fin de proporcionar una mayor personalización de parte del sistema, se han incluido varios grados de configuración.

4.1.2 Ataques a detectar

Basándonos en los objetivos del trabajo, y teniendo en cuenta que los IDS tradicionales no son capaces de detectar determinados ataques, hemos seleccionado los siguientes ataques capaces de saltarse los IDS tradicionales, para que nuestro sistema los detecte.

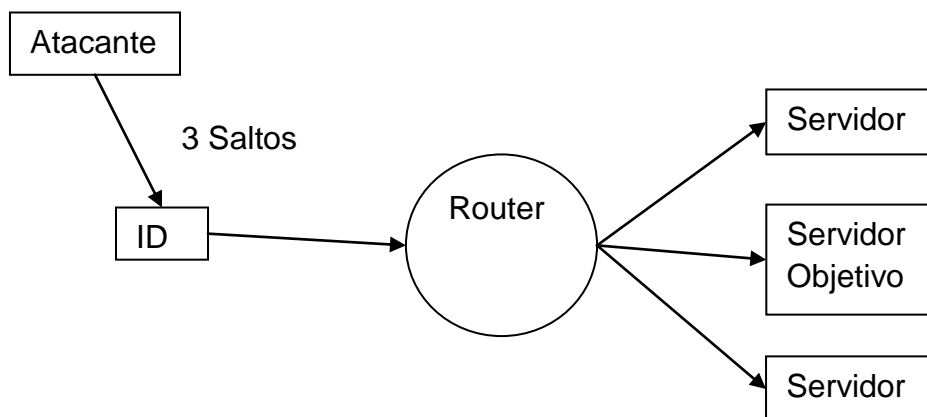
4.1.2.1 Basado en TTL

Este ataque se basa en la utilización del campo TTL de los paquetes que viajan por la red. Este campo consta de 8 bits que indican el número máximo de saltos que puede dar en la red, evitando que el paquete permanezca circulando para siempre. Cada nodo al que llega el paquete resta 1 al campo TTL del mismo, de forma que

cuando este campo llega a 0 el paquete en cuestión es descartado, enviando un mensaje ICMP de tipo *Time Exceeded Message* al origen del mismo.

El ataque consiste en enviar paquetes iguales, en los cuales cambia el TTL y el contenido para que ambos paquetes lleguen al IDS pero no al servidor, y por tanto el IDS no podrá reconstruir el mensaje completo.

Estructura de ejemplo:



Dibujo 4.1.2.1

Como vemos si enviamos paquetes con TTL 4 serían descartados por el *Router*, y por tanto no llegarían al objetivo. Si enviamos paquetes con TTL 5 si llegarían al servidor objetivo, e imposibilitaríamos al IDS reconstruir el mensaje.

Tablas de mensajes:

Imaginemos que queremos enviar un mensaje con el contenido: `/etc/passwd`

Contenido	TTL	Paquete
/	5	1
T	4	
e	5	2
Y	4	

t	5	3
U	4	
c	5	4
X	4	
.	.	.

Tabla de ejemplo 4.1.2.1

Como vemos los paquetes con contenido de letras mayúsculas son los que se quedarían en el *router* mientras que, los que tienen letras minúsculas son los que llegan al servidor objetivo.

Resultado:

Red(TTL)	IDS(TTL)	Router(TTL)	Servidor(Contenido)
3	2	1	/
2	1	Descartado	
3	2	1	e
2	1	Descartado	
3	2	1	t
2	1	Descartado	
3	2	1	c
2	1	Descartado	
.	.	.	.

Tabla de resultados 4.1.2.1

Esta sería la secuencia de mensajes y como van llegando, hasta completar el mensaje.

4.1.2.2 Basado en RST

Este ataque sirve para impedir que el IDS siga monitorizando una comunicación. Para ello se manda un mensaje con el *flag* RST activado, el cual indica la finalización de la conexión, pero con el *checksum* inválido (el *checksum* es una cadena que se crea en función del contenido del mensaje y de la cabecera, con el fin de detectar errores o alteraciones en los mensajes), con lo que el IDS finaliza

la conexión, mientras que el servidor objetivo descarta ese mensaje por *checksum* inválido.

Tanto el ataque basado en RST y SYN hacen uso del *checksum* de la cabecera TCP, en el Anexo I se explica cómo se realiza este proceso.

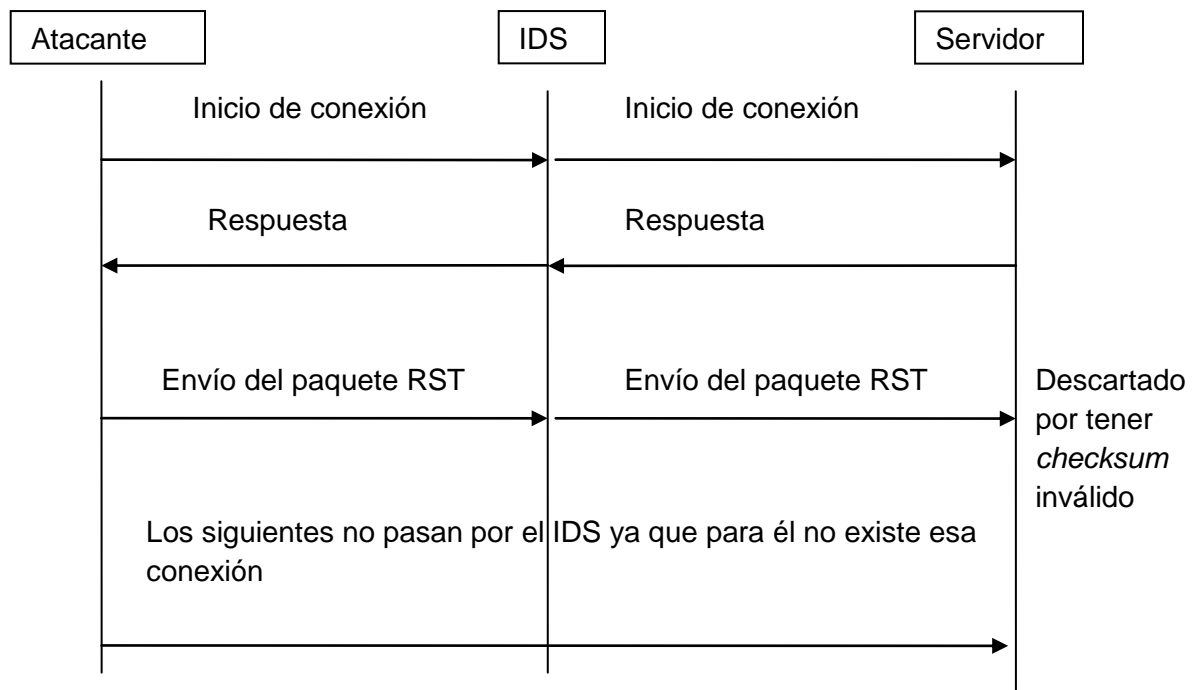


Figura 4.1.2.2

4.1.2.3 Basado en SYN

Como todos los ataques tienen el objetivo de impedir la monitorización por el IDS, este tipo consiste en impedir dicha monitorización desde el inicio de la conexión. El *flag* SYN se utiliza para sincronizar los números de secuencia para iniciar la conexión. Para comprender como funciona este ataque debemos introducir el funcionamiento del protocolo TCP.

El protocolo TCP funciona en tres fases:

- Establecimiento de conexión
- Transferencia de datos
- Cierre de la conexión

La primera fase consiste en establecer la conexión, ¿Cómo se realiza esto?
Para establecer la conexión seguimos los siguientes pasos:

- El cliente envía un paquete SYN, en el cual especifica la secuencia “x”, la cual se envía en el campo de secuencia de la cabecera TCP.
- El host recibe dicho paquete y comprueba si en la dirección ip y el puerto especificado existe un socket escuchando. Si no se diese esta situación respondería con un paquete RST (rechazo de la conexión), en el caso de que exista alguien escuchando entonces se responde con un paquete SYN-ACK, en el cual se envía en el campo *ack* “x + 1” y, en el de secuencia se envía “y”.
- El cliente responde con un paquete ACK en el cual: el *ack* es “y + 1” y el número de secuencia es “x + 1”.

Para una mayor comprensión observa la figura 2.1.2.3

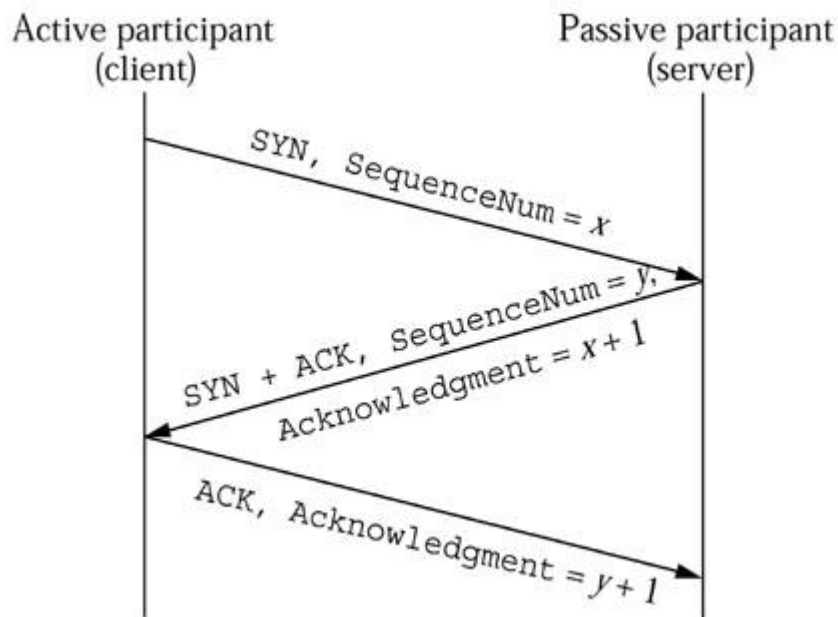


Figura 4.1.2.3

La segunda fase sirve para el envío de los datos y la tercera para finalizar la conexión. Vamos a centrarnos en la primera la cual es la que nos interesa, ya que el ataque se realiza en esta etapa.

El atacante envía un SYN inicial antes de iniciar la conexión real, pero con un *checksum* inválido, lo cual implica que si el IDS no chequea el *checksum* TCP y además, ignora varios paquetes SYN seguidos, entonces el IDS se quedaría monitorizando una conexión irreal mientras que, el objetivo ignoraría esa conexión.

Una vez realizado esto se envía un nuevo SYN con *checksum* válido que sería ignorado por el IDS pero sí aceptado por el objetivo.

4.1.2.4 Fragmentación u Overlapping

Consiste en la utilización de las políticas de re-ensamblaje para atacar al objetivo, ¿Cómo se realiza esto? Como sabemos las redes tienen una MTU o unidad máxima de transferencia. Esta indica el tamaño máximo de los paquetes que viajan por la red, y por tanto los mensajes que superen la MTU tienen que ser fragmentados para poder transmitirse.

TCP/IP no asegura que el orden de llegada de los paquetes sea el correcto, es decir, que el primer fragmento puede llegar el último, por tanto las cabeceras TCP tienen campos para ayudar a la reconstrucción de mensaje con los fragmentos. Actualmente no existe una política estándar de re-ensamblado, las más usadas son:

- BSD: Tienen preferencia los datagramas (paquetes) con offset (indica la porción de datos enviados en este paquete en relación con el mensaje completo) menor, a igual offset, se coge el que llegó primero.
- Linux: Igual que BSD pero a igual offset, se toma el que llegó el último.
- First (Windows): Acepta siempre el primer valor que llega.

Los IDS tienen que realizar el re-ensamblaje de la misma forma que los servidores que tienen detrás, para impedir que nadie envíe un mensaje que dependiendo del tipo de re-ensamblaje obtenga un resultado en el IDS, que no haga saltar ninguna regla, mientras que en el servidor objetivo dé otro resultado que si sea un ataque.

Imaginemos que un atacante quiere enviar la cadena “/etc/passwd”, obviamente esta cadena está definida como ataque en el IDS, por lo que el mensaje que la contenga sería descartado.

/	e	t		c	p	a	s	s	w	
	a	b	c	/		d	e	f	g	D

Tabla de llegada 5.1.2.4

	Paquete 1
	Paquete 2
	Paquete 3
	Paquete 4
	Paquete 5
	Paquete 6

Tabla de paquetes 5.1.2.4

El orden de llegada ha sido: 1, 4, 2, 3, 5, 6.

Dependiendo del tipo de política de re-ensamblaje obtendremos:

- Con BSD: “/etc/passwd”.
- Con Linux: “/etc/pdefgd”.
- Con First: “/etccpasswd”.

Como podemos ver, si el IDS tiene el sistema Linux o First implementado y el servidor tiene BSD, se produciría un ataque no detectado por el IDS.

4.1.3 Planificación

Siguiendo las necesidades del usuario, atendiendo a los ataques a detectar, y a los objetivos, hemos establecido los siguientes plazos para desarrollar el proyecto:

- Semana 1, **definición de la idea del proyecto**: Al ser un proyecto consensuado debemos definir los objetivos del proyecto. Horas dedicadas: 48.
- Semana 2, **elección del lenguaje de programación y las librerías necesarias**. Horas dedicadas: 40.
- Semana 3, **selección del entorno de programación e instalación**. Horas dedicadas: 40.

- Semana 4, **instalación de las librerías y primeras pruebas de funcionamiento de las mismas**. Horas dedicadas: 70
- Semana 5, **definición de las necesidades del usuario**. Horas dedicadas: 30.
- Semana 6, **definición de la estructura de la aplicación**. Horas dedicadas: 48.
- Semana 7, **definición inicial de la interfaz, implementación y prueba**. Horas dedicadas: 50.
- Semana 8, **implementación de la estructura de la aplicación**. Horas dedicadas: 52.
- Semana 9, **implementación del *sniffer***. Horas dedicadas: 50.
- Semana 10, **implementación de la comunicación entre hilos y prueba**. Horas dedicadas: 65.
- Semana 11, **implementación del analizador de paquetes**. Horas dedicadas: 70.
- Semana 12, **implementación de la base de datos**. Horas dedicadas: 30.
- Semana 13, **implementación de las gráficas**. Horas dedicadas: 40.
- Semana 14 y 15, **prueba de la aplicación**. Horas dedicadas: 60.
- Mes de Junio (semanas 16, 17, 18 y 19), **resolución de problemas de la aplicación**. Horas dedicadas: 280.
- Semana 20, **redefinición de la estructura de la aplicación**. Horas dedicadas: 35.
- Semana 21, **re-implementación de la aplicación**. Horas dedicadas: 70.
- Semana 22, **redefinición e implementación de la base de datos**. Horas dedicadas: 42.
- Semana 23, **reestructuración del sistema de ficheros**. Horas dedicadas: 70.
- Semanas 24 y 25, **prueba de la aplicación**. Horas dedicadas: 64.
- Semana 26, **re-implementación del *sniffer***. Horas dedicadas: 48
- Semana 27, **prueba**. Horas dedicadas: 48.
- Meses de octubre, noviembre y diciembre, **elaboración de la documentación**. Horas dedicadas: 126.

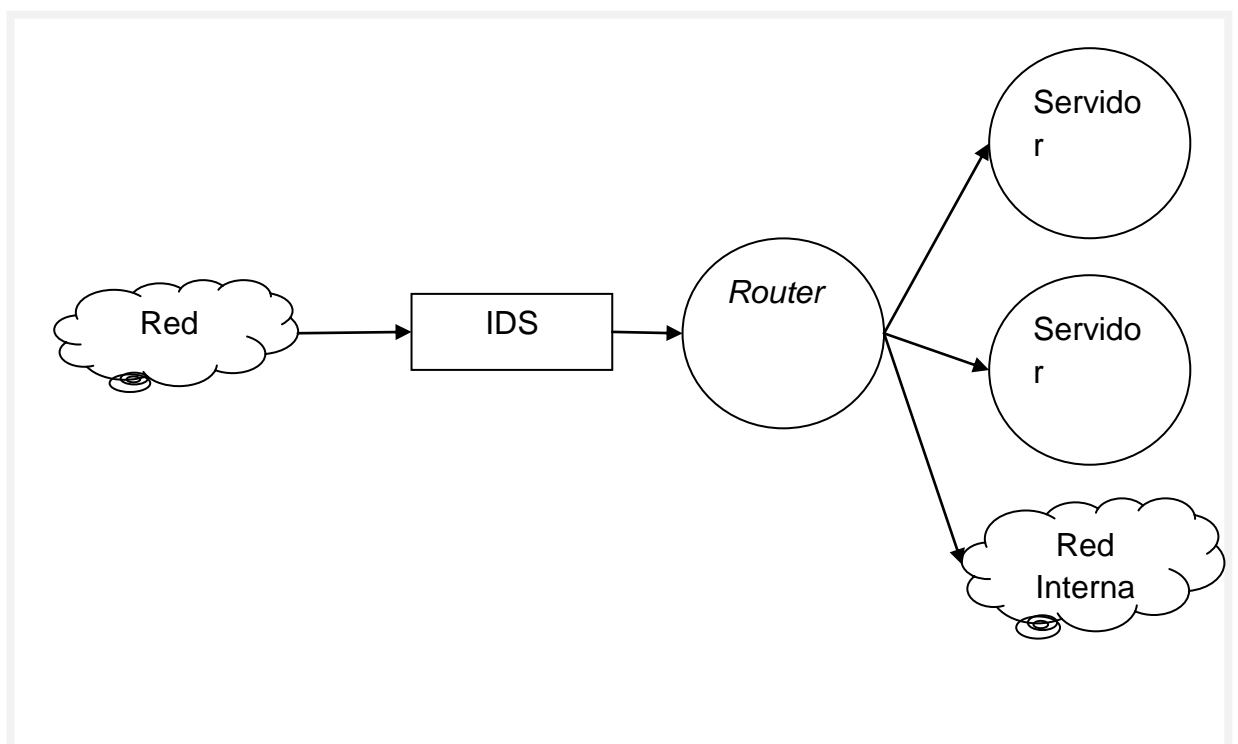
El total de horas dedicadas a la elaboración de este proyecto es: 1.476 horas.

4.1.3.1 Presupuesto

En este apartado vamos a desglosar el coste económico de la instalación y configuración del sistema, así de cómo el desglose económico del desarrollo del sistema.

El presupuesto de desarrollo del sistema se fundamenta en las horas trabajadas y estas son: 1.476, cuyo desglose se encuentra expuesto en el apartado de planificación. El importe por hora es de 60€/h, en consecuencia el coste es de 88.560€

El presupuesto económico que vamos a exponer en presente apartado servirá para la inclusión del IDS en un sistema como el siguiente:



Los requisitos del sistema que soportará se agrupan en **hardware**, **software** e **instalación y configuración**.

4.1.3.1.1 Hardware

El sistema sobre el que se va a ejecutar el IDS requiere de los siguientes componentes mínimos:

- **Placa base:** Asus H110M-D Coste 63,00 €
- **Procesador:** Intel Core i3-6100 3.7GHz Coste 111,00 €
- **Memoria RAM:** Corsair Value Select DDR3 1333 PC-10600 2 GB CL9 Coste 21,00 €
- **Tarjeta de red:** TP-LINK DGE-Tarjeta de Red Gigabit 10/100/1000 Coste 13,75 €
- **Disco duro:** WD Blue 500 GB 2.5" 5400 RPM SATA3 Coste 42,95 €
- **Ventilador de la CPU:** Cooler Master Dream i117 Coste 13,95 €
- **Fuente de alimentación:** Tacens APII500 Fuente Alimentación 500W Coste 14,95 €
- **Ratón:** Approx Optical Mouse USB Negro Coste 4,95 €
- **Teclado:** L-Link LL-KB-628M Teclado multimedia PS/ 2 Negro Coste 6,95 €
- **Pantalla:** Acer V196HQLAb 18.5" LED Coste 69,00 €
- **Torre:** Tacens Anima AC016 USB 3.0 Coste 19,25 €

El coste total de todos los componentes es: 380,75 €

4.1.3.1.2 Software

Además de los componentes físicos el sistema requiere de componentes *software*, y esto son:

- **Sistema operativo:** Linux, distribución **Ubuntu 16.04.1 LTS** (actualmente esta es la disponible, pero las anteriores también se podrían utilizar) sin coste.
- **Python:** Versión 2.7.13, sin coste.
- Librerías (todas las no incluidas en esta lista, se instalan por defecto con *python*):
 - *PyQt4*: Sin coste.
 - *Pcap*: Sin coste.

- *Matplotlib*: Sin coste.
- *Netifaces*: Sin coste.
- *Numpy*: Sin coste.
- *Geoip*: Sin coste.

Como se puede apreciar en este subapartado el coste del software que se necesita es: 0,00 €

4.1.3.1.3 Instalación y configuración

La instalación del sistema *Ubuntu* y de las librerías necesarias requiere de 3 horas, a un importe de de 100 €/h.

La configuración es opcional debido a que requiere de la configuración de los distintos sistemas que compongan la red. Si el usuario quisiese una configuración requeriría de un trabajo de otras 3 horas por el mismo importe.

Total del coste del sistema: 380,75 € + 0,00 € + 300 € (+ 300 € opcionales) = 680,75 € o 980,75 €.

4.2 Diseño del sistema

Ahora vamos a proceder a la descripción del diseño de la aplicación.

4.2.1 Librerías utilizadas y lenguaje de desarrollo

Procedemos a explicar las decisiones tomadas en lo concerniente al lenguaje de programación y las librerías utilizadas.

De entre todos los lenguajes de programación hemos decidido utilizar *Python*. ¿Por qué? Principalmente por su carácter multiplataforma, ya que como uno de los principales objetivos es crear un IDS para cualquier dispositivo, este lenguaje era perfecto, ya que se puede ejecutar en cualquier tipo de máquina. Además de esta característica principal y necesaria, tiene otras características que nos ayudaron a decidimos por este y no por otro lenguaje, las características son:

- Simple: Es un lenguaje fácil de leer y por tanto comprensible, lo cual es vital para dejar el sistema abierto a modificaciones posteriores.

- Propósito general: Este nos sirve para crear tanto páginas, como programas de todo tipo.
- *Open Source*: Como hemos expuesto al ser de código abierto ha sido modificado para poder utilizarse en distintas plataformas, pero no solo en los distintos sistemas operativos como Linux, Windows o Macintosh, sino también en sistemas como VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, entre muchos otros.
- Lenguaje orientado a objetos: Soporta la creación de objetos que combinan datos y funcionalidades.
- Lenguaje de alto nivel: Al ser de alto nivel no hay que preocuparse por la gestión de memoria del programa, aunque esto no implica que el programa sea eficiente en la gestión de memoria, ya que es un campo vital para nuestro proyecto.
- Extensas librerías: Contiene gran cantidad de librerías, que se pueden utilizar en los distintos sistemas, para poder ejecutar el programa en los sistemas más diversos.

Librerías utilizadas, dentro de todas las disponibles para *python* hemos decidido usar las siguientes:

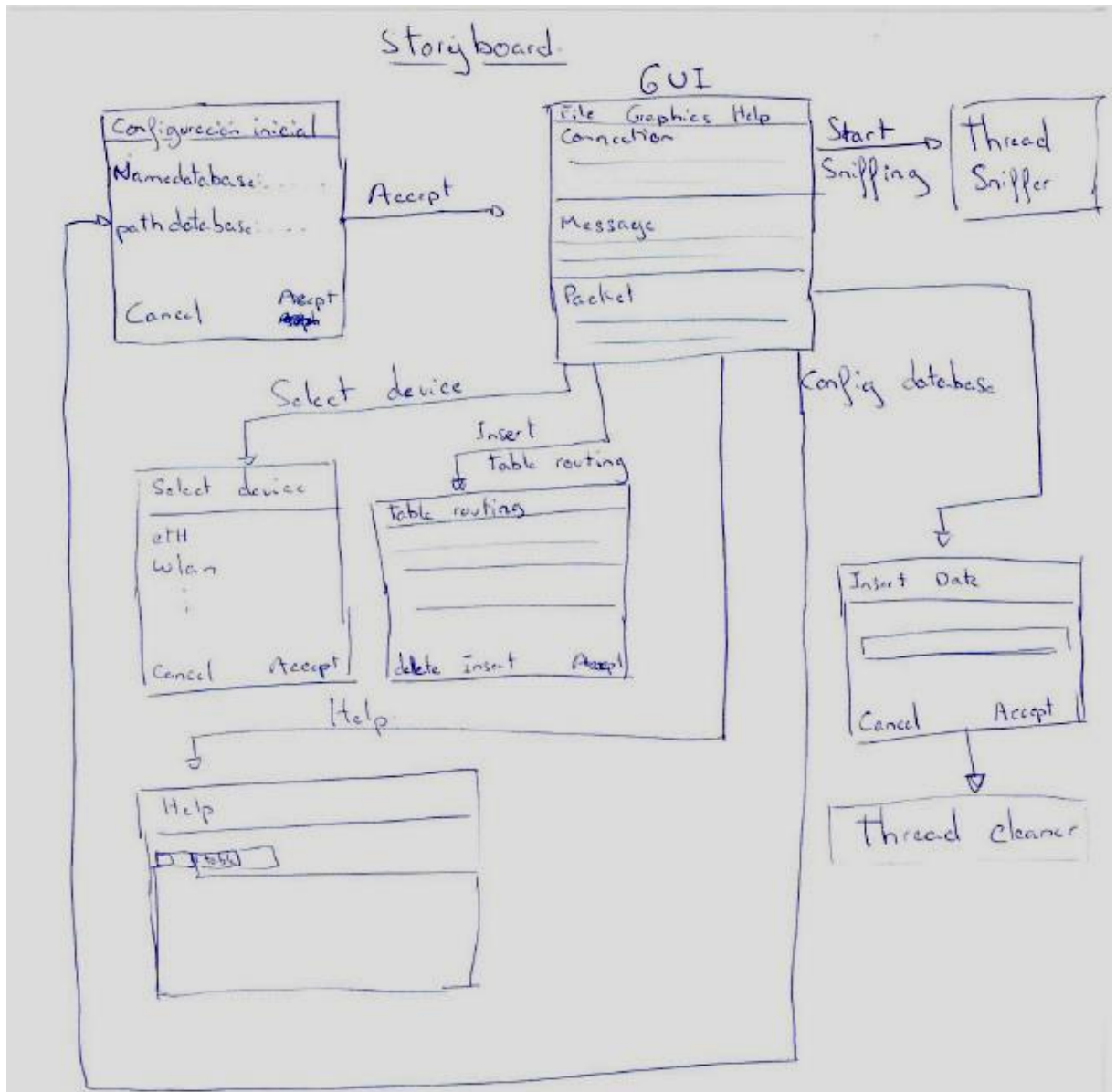
- *PyQt4*: Esta es una librería que nos proporciona la capacidad de crear un entorno gráfico para un programa. El motivo por el cual la hemos utilizado es para crear la interfaz que permita el uso del proyecto.
- *Struct, binascii y socket*: Esta librería la utilizamos para desempaquetar los paquetes que navegan por la red, y extraer una cabecera legible, la librería socket nos permite transformar las direcciones IP y MAC, en el formato habitual.
- *Sys, os.path y shutil*: Se utilizan para la gestión del sistema de ficheros.
- *Threading*: Esta librería, al igual que en los demás lenguajes de programación, se utiliza para la gestión de hilos en *python*.
- *Time, datetime*: Se utiliza para la gestión de fechas.
- *Pcap*: Esta es la librería que permite la captura de los paquetes que viajan por la red, debemos de tener en cuenta que solo se puede

monitorizar un servicio de red, y si queremos hacerlo con varios debemos tener varios hilos de esta librería.

- *Operator*: Esta librería tiene implementada las operaciones de un objeto *list*, lo cual es muy útil para ordenar una lista en función de un entero.
- *Geoip*: Se utiliza para geolocalizar una dirección IP y poder guardar su ubicación en la tabla.
- *Sqlite3*: Esta librería nos permite definir nuestra base de datos, la cual se almacena en un archivo cuyo nombre y ruta la decide el usuario.
- *Matplotlib*, *tkinter* y *numpy*: Estas librerías se utilizan en la generación de las gráficas.
- *Netifaces*: Esta librería nos permite saber si el servicio de red seleccionado está conectado a la red.

4.2.2 Estructura de la aplicación

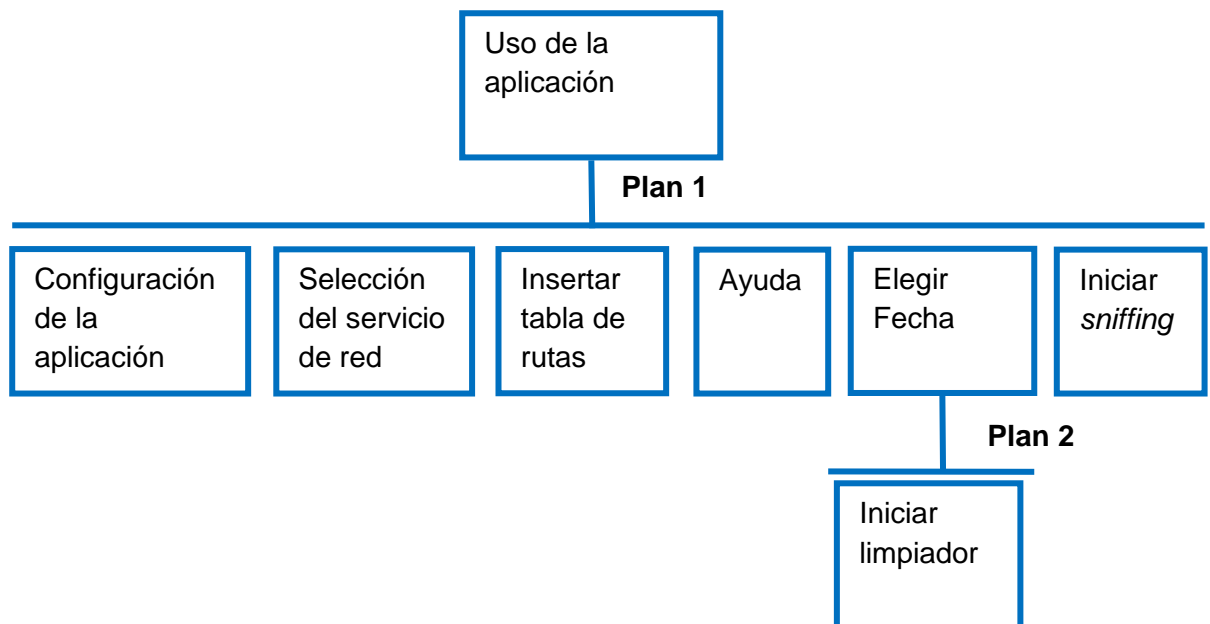
Teniendo en cuenta las necesidades de los usuarios, hemos decidido crear la siguiente estructura mostrada en el storyboard:



Dibujo 4.2.2

Como apreciamos en el *storyboard* todas las opciones parten de la GUI, desde la que se va accediendo a las distintas pantallas de la aplicación. Hemos modificado el inicio de la aplicación, para que cuando esta sea iniciada por primera vez, puedas configurarla, antes de la iniciación de la pantalla principal.

Una vez configurada se inicia la interfaz principal y puedes acceder a las distintas opciones mostrada en la asignación jerárquica de tareas.



Plan 1:

- Cambiar la configuración de la aplicación.
- Seleccionar un servicio de red que monitorizar.
- Insertar la tabla de rutas del *router* para detectar ataques.
- Acceder a la ayuda.
- Elegir una fecha para ejecutar el limpiador.
- Iniciar el *sniffing* del servicio seleccionado.

Plan 2:

- Iniciar el limpiador de la base de datos con la fecha seleccionada.

En plan 1 como se ve, tiene acceso a la mayoría de las opciones, las cuales permiten introducir datos necesarios para la ejecución del sistema.

La selección del servicio de red permite al *sniffer* saber cuál es la interfaz de red que va a monitorizar. Un dato importante es que debe ser lanzado con permisos de administrador, ya que si no se hiciera así, no se podría acceder a las distintas interfaces de las que dispone el sistema. Una vez seleccionada se envía al *sniffer*, para que este la monitorice. El cambio de interfaz de red se realizará cuando el

sniffer esté parado, ya que si se realiza cuando está monitorizando el cambio no se aplicaría durante la ejecución.

Insertión de la tabla de rutas. Como se ha explicado en esta documentación, para la detección de ataques basados en TTL se necesita acceder a una tabla de rutas, la cual tiene el *router*, para poder detectar estos ataques, por lo que es necesario que el usuario configure esta zona de la aplicación antes de lanzar el *sniffer*.

Introducir la fecha permite al proceso *cleaner*, saber a partir de qué fecha debe limpiar la base de datos. Como se verá en la ventana de introducción de la misma muestra el formato en el que debe introducirse la fecha.

Iniciar *sniffing* nos permite lanzar el proceso de monitorización y análisis de paquetes, para la detección de aquellos paquetes potencialmente peligrosos para el sistema.

La opción de ayuda nos da acceso a una pequeña interfaz de ayuda en la cual indica las opciones que tiene el sistema y para qué sirven dichas opciones.

Esta aplicación ha sido diseñada para la utilización de hilos, como máximo en cualquier instante dado de la ejecución de la aplicación. Solo puede haber dos hilos en ejecución, estos hilos son:

- Hilo principal o GUI: Este es el que gestiona la interfaz gráfica, y por tanto este es el que lanzaría todos los demás hilos.
- Hilo *sniffer*: Este hilo ejecuta el *sniffer* y el analizador de los paquetes, y cuando son calificados como peligrosos los introduce en la base de datos, guardando así el contenido del paquete.
- Hilos *cleaner*: Este hilo es el que se encarga de ejecutar la limpieza de la base de datos, pero no puede ejecutarse al mismo tiempo que el *sniffer*.

La comunicación entre el hilo principal y el analizador se realiza con la base de datos, lugar donde se almacena la información, y una clase llamada "*updateTable*", la cual envía una señal a la GUI para que esta actualice las tablas que componen la interfaz, y muestre la información que la base de datos contiene.

No existe comunicación entre el proceso *cleaner* y el *sniffer*, ya que no es necesaria ninguna comunicación y, al igual que con el *sniffer*, el hilo *cleaner* tampoco tiene comunicación con la interfaz hasta que termine, momento en el cual manda una señal a la interfaz para que actualice su información de la base de datos.

Además de estos hilos la aplicación tiene los siguientes elementos:

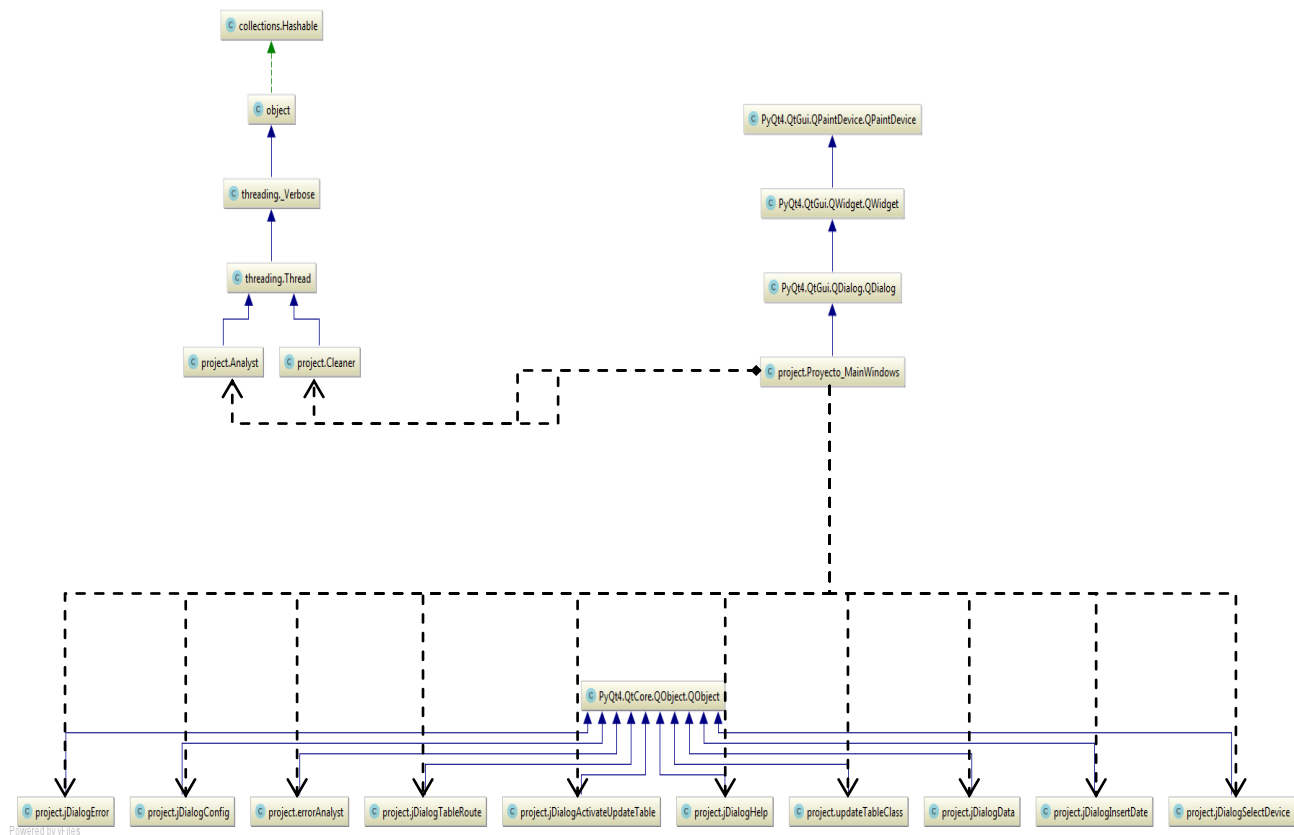
- Base de datos: Aquí se guardan los paquetes clasificados como peligrosos, los cuales se mostrarán en la interfaz gráfica de la aplicación, esta es una base de datos que se almacena en un fichero, y por tanto tiene características limitadas.
- GeoLite2-City: Este fichero nos permite obtener la localización de una dirección IP.
- Sistema de ficheros: Es una estructura de ficheros que permiten guardar el contenido de los paquetes almacenados en la base de datos, haciendo posible mostrarlo a los usuarios cuando estos lo requieran. Este sistema colgará de la carpeta “*pathDataOfPacket*” indicada en la ruta configurada por el usuario, o la predefinida.

4.2.3 UML y diagrama de clases

En este apartado vamos a proceder a explicar las relaciones entre las distintas clases que contiene código fuente, así como los componentes de las mismas.

Nota: En el apartado 4.3, que trata sobre la implementación, se especifica un cambio en la estructura, que repercute en el diagrama de clases. Vamos a exponer el diagrama definitivo pero en los subapartados del 4.3 se explica el cambio, y el porqué del mismo.

Dado que hay algunas clases con una gran cantidad de métodos y de variables, esta descripción se expondrá en un sub-apartados. El diagrama de clases es el siguiente:



Como podemos apreciar la mayoría de las clases heredan de *QObject*, la cual contiene la funcionalidad necesaria para poder ejecutar las interfaces gráficas.

La relación que existe entre la ventana principal o “*Proyecto_MainWindows*” y las demás ventanas de la interfaz, es una relación de dependencia, ya que estas ventanas no pueden ser lanzadas desde ninguna otra forma y permiten la introducción de datos necesarios para poder ejecutar el sistema.

La relación con la clase “*cleaner*” y la clase “*analyst*” es de agregación por valor, y es que para poder lanzarlas es necesario que la interfaz principal esté en funcionamiento y aunque esta se ejecute, como hemos explicado en capítulos anteriores, en un hilo diferente al de la interfaz principal.

En este punto vamos a pasar a explicar cada clase por separado, a fin de exponer los métodos y funciones de las respectivas clases. Hemos de indicar un factor común de diferentes clases, por el ejemplo los métodos *setupUi* y

retranslateUi los cuales son métodos que se heredan de la clase *QtCore.QObject*, en los que se incluye la definición de los elementos que componen dicha clase. Y también los métodos *init* que se heredan de la clase *threading.Thread*, la cual se utiliza en los hilos.

4.2.3.1 Proyecto_MainWindows

Como hemos expuesto esta es la clase principal del sistema, la cual nos permite lanzar las ventanas que permiten establecer la configuración del sistema, así como otras adicionales para mostrar la información mediante gráficas, o una ventana de ayuda que nos permite saber para que se utilizan las distintas opciones que nos permite configurar el sistema. La clase *Proyecto_MainWindows* es la siguiente (la hemos partido en tres imágenes debido a su extensión):

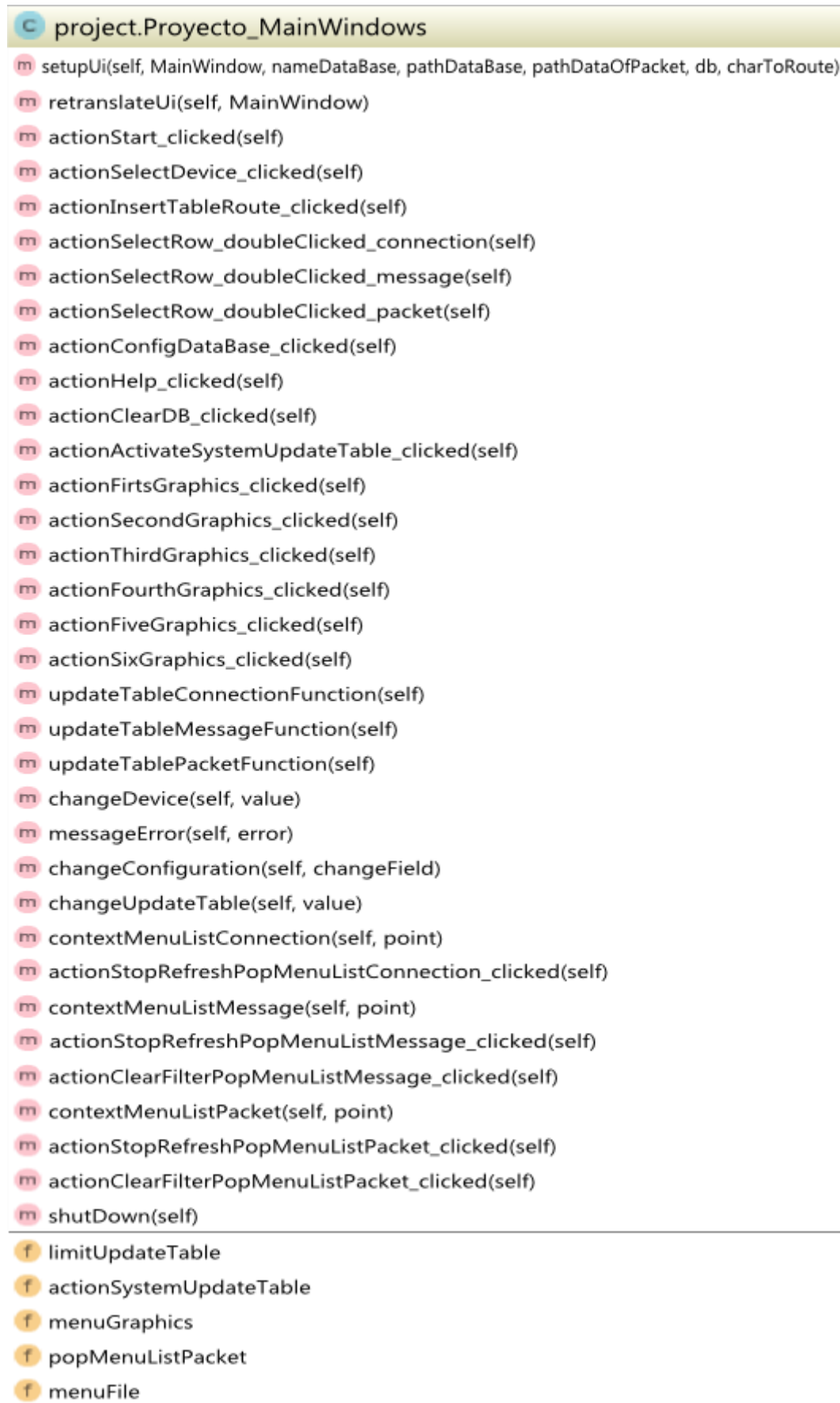


Imagen 4.2.3.1.1

- f pathDataBase
- f actionThirdGraphics
- f listMessage
- f modelListConnections
- f popMenuListConnection
- f numberUpdateTable
- f modelDetailMessage
- f actionConfigDataBase
- f actionClearDB
- f actionFiveGraphics
- f actionInsertTableRoute
- f actionExit
- f popMenuListMessage
- f pathDataOfPacket
- f centralwidget
- f actionClearFilterPopMenuListMessage
- f updateTableInstance
- f actionSelectDevice
- f dialogHelp
- f analystProgram
- f actionHelp
- f actionStopRefreshPopMenuListMessage
- f listConnections
- f dialog
- f listPacket
- f dialogUpdateTable
- f dialogError
- f dialogInsertDate
- f deviceSelected
- f actionStopRefreshPopMenuListConnections
- f errorAnalystInstance
- f actionStopRefreshPopMenuListPacket
- f dialogConfig
- f actionSecondGraphics
- f lastDateUpdateTable
- f actionStart
- f nameDataBase
- f charToRoute
- f menubar
- f activateSystemUpdateTable
- f actionFourthGraphics

Imagen 4.2.3.1.2

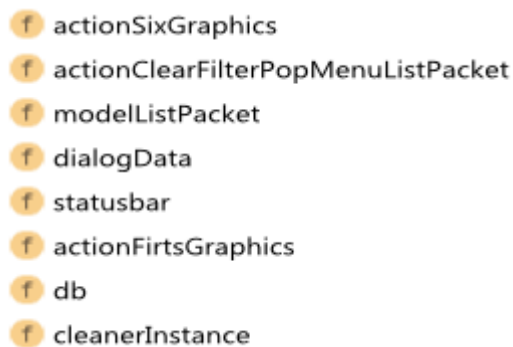


Imagen 4.2.3.1.3

Vamos a proceder a explicar los aspectos más importantes de las variables de la clase “*Proyecto_MainWindows*”.

Como muestran las imágenes, las variables (llevan la letra “f” escrita en el círculo) que comienzan por la palabra “*action*” corresponden a las acciones de la barra superior, a si por ejemplo, si activamos la acción “*actionStart*” lanza el proceso analizador, el cual se encarga de capturar los paquetes, analizar su contenido y posteriormente, si cumple alguna de las reglas o condiciones que lo caracterizan como un paquete sospechoso, almacenarlo en la base de datos para poder mostrar, si el usuario así lo requiere, su contenido o el contenido del *string*, en caso de que sea un paquete que llegó fragmentado.

Nótese, que no existe ninguna variable *action* que se llame “*actionStop*” o algo parecido, y esto es debido a que se reutiliza la misma variable (*actionStart*) para dicha función.

Todas las variables que empiecen por la palabra “*dialog*” son las variables que se instancian según las clases recogidas en el diagrama, para lanzar las ventanas de las que dispone la interfaz, así como la variable “*analystProgram*” es una instancia de la clase *analyst* que declaramos para lanzar el hilo del analizador. Igual que lo anterior es la variable “*cleanerInstance*”, la cual es una instancia de la clase “*Cleaner*”, la cual nos permite hacer la limpieza de la base de datos cuando el usuario así lo decida.

Las variables que comienzan por: “*model*” son aquellas que nos permiten conectar las tablas que se muestran en la interfaz, con las tablas de la base de datos, para que cuando cualquiera de ellas sea actualizada, solo debamos realizar un *select*, para obtener la tabla correctamente. Las que empiezan por “*list*” son las tablas que se muestran, las cuales están enlazadas con los modelos para mostrar el mismo.

Las que contienen la palabra “*Graphics*” son aquellas que activan la interfaz, y los elementos necesarios para mostrar las gráficas correspondientes. En el menú se dice que muestra la gráfica.

La variable “*pathDataBase*”, tanto como la variable “*charToRoute*”, nos permiten localizar la base de datos en la estructura de ficheros del sistema operativo, así como establecer la ruta de los nuevos paquetes que hayamos decidido guardar en la base de datos por calificarlos de peligrosos. El valor de la variable “*charToRoute*” depende del sistema operativo sobre el que se esté ejecutando el sistema, puede ser “/” para Linux y Mac Os, y “\” para Windows. El usuario puede configurar la ubicación de la base de datos, y esta ruta se establecerá también para los ficheros de los paquetes almacenados.

La variable “*updateTableInstance*” es la instancia de la clase “*updateTable*” que, como se explicó en el apartado sobre la estructura, se encarga de comunicar a la GUI que debe actualizar las tablas.

En este punto vamos a explicar las funciones más importantes de la clase.

Al igual que para las variables, las funciones que comienzan por la palabra “*action*” son las que corresponden a las acciones de la barra superior, salvo aquellas que contienen las palabras “*popMenu*” las cuales corresponden al menú contextual de sus respectivas tablas, ya que con ellas podemos limpiar el filtro que se realiza con los *actions* de doble *click* sobre cualquiera de las tablas y para el refresco de la tablas.

La función *messageError* nos muestra cualquier error que ocurra en la ejecución del *analyst*, el cual manda un mensaje a *mainWindows* y esta lo muestra en una ventana de error.

Y por último la función *shutDown* nos cierra la aplicación. Esta se activa al pulsar en la acción *exit* del *menuBar* ya que no existe ninguna función *actionExit*, si estuviese activado el proceso *analyst* debería pedir que este fuese cerrado, y si fuese el proceso *cleaner* debería esperar a la finalización del proceso.

4.2.3.2 Analyst

En este epígrafe explicaremos los métodos y variables más importantes de la clase *Analyst*, la cual se encarga de analizar los paquetes y capturarlos, queda de la siguiente forma:



Imagen 4.2.3.2

Al igual que en el apartado anterior, en este vamos a proceder a explicar primero las variables más relevantes, y después los métodos con el fin de proporcionar un mayor conocimiento del sistema.

Las variables relevantes para el uso de la base de datos, así como para el sistema de ficheros que crea la aplicación con el fin de almacenar el contenido de los paquetes son:

- *pathDataOfPacket*: Esta contiene como bien indica su nombre la ruta a los ficheros que almacenan el contenido de los paquetes, un ejemplo de la misma sería: "C:\User\Gregorio\Proyecto\pathDataOfPacket"
- *dataBase*: Esta variable nos permite acceder a la base de datos, y se compone de la ruta + el nombre de la base de datos que vamos a utilizar.
- *cursorDataBase*: Para poder utilizar las bases de datos de *sqlite* debemos definir un cursor, el cual se encarga de la operaciones en la base de datos (*Insert*, *Select*, etc).
- *dataConnection*: Esta variable nos permite crear el cursor para poder acceder a la base de datos.
- *dbIP*: Esta es una base de datos que lleva integrado el sistema, la cual nos permite obtener la localización de la ip que nos envía dichos paquetes.

Las variables que comienzan por la palabra "*number*" se utilizan para la base de datos, las cuales registran el número de paquetes basándose en:

- El protocolo.
- Si ha sido marcado como un paquete peligroso.

Una vez se quiera finalizar la ejecución del proceso Analyst, estas variables son almacenadas, y posteriormente usadas para la creación de las gráficas, cuando el usuario pulse la opción correspondiente.

Las variables que comienzan por "*nextID*" las utilizamos para colocar dicho número en el nombre de los directorios y ficheros que vayamos a almacenar. Obviamente si ya existe una conexión no creará otro directorio para la misma conexión, sino que almacenará en el directorio de la conexión existente el mensaje que se quiera guardar.

Por último destacar variables importantes como son:

- ✓ *updateTable*: Esta nos permite mandar a la GUI la señal necesaria para que actualice las tablas, por haber habido una modificación. Esta

variable es definida en Proyecto_MainWindows, pero es pasada a esta clase para que haga uso de la variable.

- ✓ *is_alive*: Esta variable se utiliza como condición, para que cuando el usuario lo pida se finalice la ejecución del Analyst. Como hemos expuesto en otras variables, sobre todo las correspondientes a las que contienen “*number*” expuesta anteriormente, no nos sirve una finalización abrupta del hilo ya que si no lo finalizamos correctamente, no podremos almacenar datos que luego utilizaremos para las gráficas.
- ✓ *Device*: La GUI le manda al Analyst cuál es el servicio de red que tiene que *sniffar*.

Con respecto a los métodos más importantes, el que mayor relevancia posé es “*run*”. Esta función es un método abstracto que se hereda de la clase “*Thread*” la cual contiene el código de debe ejecutar el hilo, que es instancia de la clase. En esta función se realiza la captura de los paquetes, así como su desempaquetado en los distintos campos que pueden observarse en el Anexo II y III.

Por último, esta función es la que contiene todas las reglas para poder clasificar los paquetes como potencialmente peligrosos, y si algún desarrollador quisiera seguir con la ampliación de este proyecto, debería introducir en dicha función los ataques que quiera que detecte el sistema.

La función cuyo nombre es “*saveAttack*”, es otro método de importante relevancia ya que, se encarga de guardar un paquete marcado como sospechoso en la base de datos, además de crear la estructura de ficheros sobre la que se guarda el fichero con el contenido del paquete. Para poder proceder de forma correcta con esta acción, debemos pasar a dicha función la información relevante que se almacenará en la base de datos.

Otra función de gran peso es “*restoreStringOfMessage*”, esta nos permite, una vez guardado todos los paquetes de un mensaje fragmentado, reconstruir el *string* del paquete para proporcionar los elementos necesarios para poder hacer un análisis a nivel de mensaje, como se explica en el apartado 4.3.1.11.

Además de las funciones expuestas anteriormente, también hay que tener en cuenta para el análisis de los paquetes los siguientes métodos:

- *checkNumber*: Como se expone en el anexo I, el cálculo del campo *checksum* se realiza con números hexadecimales de 16 bits, pero estos no pueden tener una extensión mayor a la descrita y por tanto, para el cálculo de dicho campo, necesitamos de una función que se encargue de sumar el acarreo para que el número no sea superior a 16 bits.
- *ethAddr*: Esta función nos permite obtener la dirección *Ethernet* de la dicha cabecera, para poder almacenarla en la base de datos si fuese necesario.

Por último las dos funciones restantes, la primera *changeDevice* nos permite cambiar el dispositivo escogido por el usuario, y la segunda *stopSniffer* finaliza la ejecución del proceso Analyst cuando el usuario lo requiera.

4.2.3.3 Cleaner



Imagen 4.2.3.3

En este apartado procedemos a explicar la clase *Cleaner*, la cual se encarga de la limpieza de la base de datos, así como de la estructura de ficheros que almacena el contenido de la misma, cuando el usuario así lo establezca.

Para ello utiliza las siguientes variables, muchas de ellas similares a las ya explicadas en las clases anteriores:

- ❖ *charToRoute*: Tiene la misma función que en las distintas clases ya expuestas en los apartado precedentes.
- ❖ *dataBase*: contiene el nombre de la base de datos que va a limpiarse, es la misma que utiliza la GUI para mostrar las tablas.
- ❖ *cursorDataBase*: Este nos permite realizar las operaciones en la base de datos.
- ❖ *dataSelect*: Esta variable almacena la fecha a partir de la que se borrara todo registro más antiguo.
- ❖ *Executed*: Esta nos permite saber que este hilo está ejecutándose, para impedir que el *Analyst* pueda ejecutarse al mismo tiempo que proceso *cleaner*.
- ❖ *ConnectionDataBase*: Establece la conexión a la base de datos, al igual que en los demás casos.
- ❖ *updateTable*: Al igual que el proceso *Analyst*, esta indica a la interfaz que debe actualizar las tablas para tener el contenido actualizado.
- ❖ *pathFile*: Esta establece la ruta hasta la carpeta de la cual cuelga todo el sistema de ficheros de la aplicación.

El método más importante es *run*, el cual es un método heredado de la clase *thread*, y que incluye el código que debe ejecutar este proceso.

4.2.3.4 jDialogError

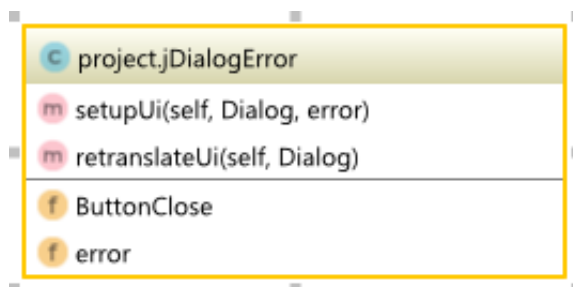


Imagen 4.2.3.4

Esta clase muestra los errores que puedan surgir al lanzar alguno de los procesos que componen la estructura.

Para ello solamente necesita la variable *error*, la cual contiene el mensaje del error que va a mostrar, y la variable *buttonClose* la cual cierra la ventana de error.

Los únicos métodos son los que definen la ventana para poder mostrarla en la pantalla, ya que no ha sido necesaria la inclusión de ningún método para mostrar un mensaje de error.

4.2.3.5 jDialogConfig

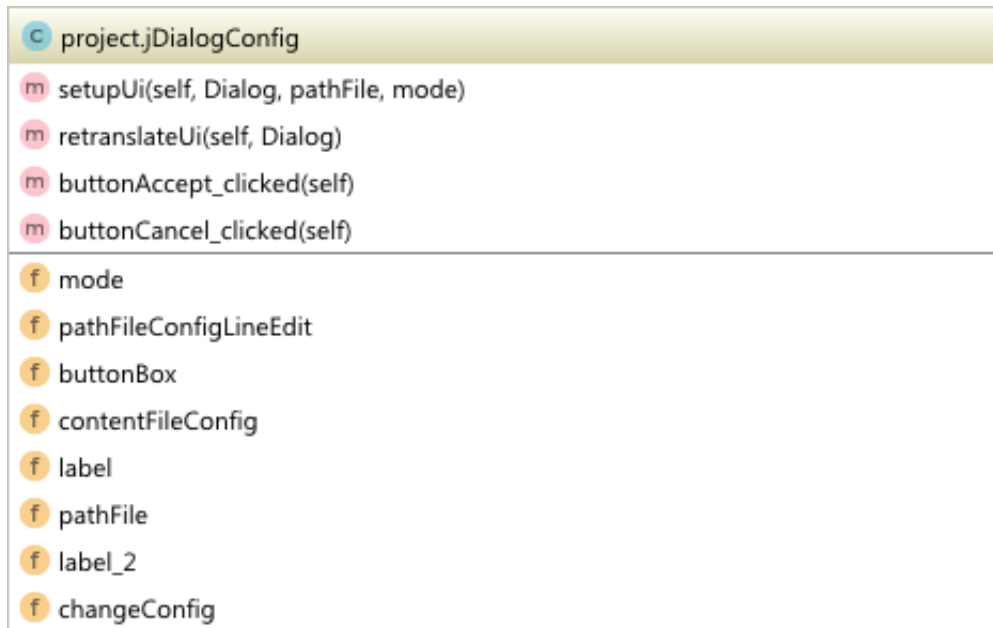


Imagen 4.2.3.5

Este se encarga de mostrar la configuración del fichero “config.txt”, el cual almacena:

- El nombre de la base de datos.
- La ruta de la base de datos.
- La ruta a la estructura de ficheros que almacena el contenido de los paquetes guardados en la base de datos.

Para realizar esta tarea hemos creado las siguientes variables:

- *Mode*: debemos tener en cuenta que cuando el programa se inicia por primera vez pide al usuario que lo configure, o deje la configuración por defecto, y esto se realiza con esta clase, por lo que debemos distinguir

entre estas dos formas de ejecución. De esto se encarga esta variable, indicar en qué modo fue iniciado y el resultado que tendrá, ya que son diferentes en función del modo de ejecución.

- *pathFileConfigLineEdit*: Esta variable, como bien indica la parte final de la misma, define un objeto de la GUI en el cual se muestra la ruta al fichero de configuración que se está manipulando.
- *buttonBox*: Es un objeto de *PyQt* que define una caja con dos botones, uno de aceptar y otro de cancelar, para guardar o no la configuración.
- *contentFileConfig*: Este es el objeto muestra en pantalla la información del fichero de configuración. Una vez modificada la información que se muestra en pantalla, esta será almacenada en el mencionado fichero.
- *pathFile*: Contiene la ruta del fichero que se está modificando, es decir, la ruta al fichero de configuración.
- *changeConfig*: Esta es una variable que permite mandar a la GUI un mensaje, en el cual se indica que la configuración ha sido modificada, para que la GUI se encargue de la modificación del fichero.
- *Label*: Ambas son etiquetas para mostrar información en la pantalla, pero que no tienen ninguna otra función más que esta.

Los métodos más importantes son los siguientes:

- *buttonAccept_Clicked*: Este objeto envía los cambios realizados en *contentFileConfig* a la interfaz principal, para que sepa si debe cambiar el fichero de configuración o no. Si lo hubiese lanzado la función *main*, por no existir el fichero de configuración, guardarían los cambios en el mismo.
- *buttonCancel_Clicked*: Esta función únicamente cierra la interfaz, y dependiendo de quién lo lance finaliza la ejecución del sistema o no.

El resto de funciones han sido creadas automáticamente, de forma que lo único que realizan es la definición de los elementos que definen la interfaz, y que permiten que esta se vea.

4.2.3.6 errorAnalyst

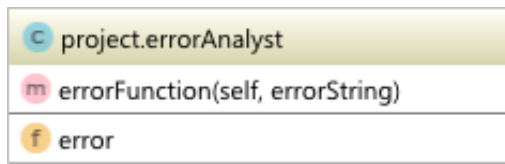


Imagen 4.2.3.6

Esta clase como bien indica su nombre es de error, pero no tiene ninguna función importante salvo la de pasar una cadena a la interfaz, es decir, para comunicarse con la interfaz y mandar los errores que se observen.

Si volvemos al epígrafe que describe la estructura de la aplicación, vemos que el sistema requiere de una clase que se encargue de enviar señales a la interfaz para que actualice las tablas. Pues esta clase tiene la misma función salvo que con la señal envía una cadena para que la GUI la muestre al usuario en un mensaje de error.

Para ello se utiliza la función *errorFunction* la cual envía la señal a la interfaz, cuando se llame a esta función.

4.2.3.7 jDialogTableRoute

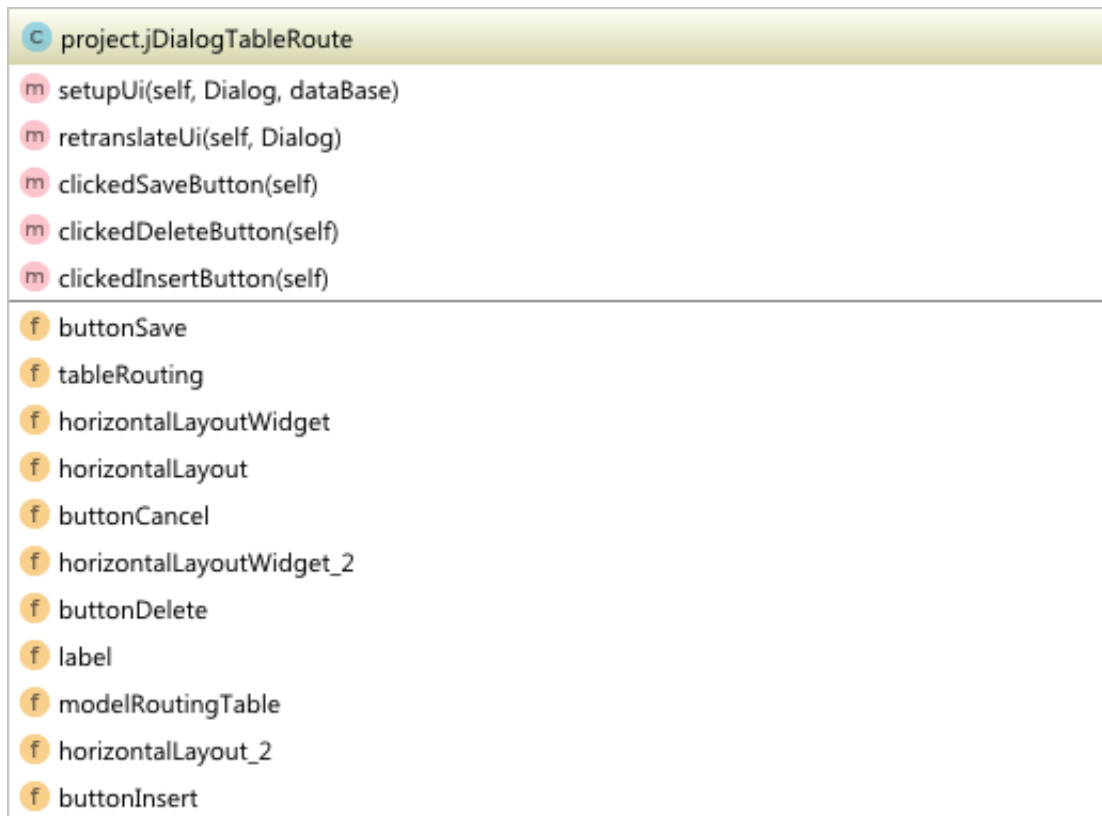


Imagen 4.2.3.7

Esta clase es la encargada de gestionar la tabla de rutas del *router*, para ello utilizamos los siguientes elementos:

- ❖ Los elementos que contienen la palabra *button* realizan la función que viene determinada en el propio nombre del elemento, por ejemplo: el *buttonDelete* se encarga de borrar la tabla seleccionada.
- ❖ *TableRouting*: Esta es la variable que almacena la tabla que se muestra en la pantalla.
- ❖ *ModelRoutingTable*: es la que permite conectar esta tabla con la tabla de la base de datos.
- ❖ Los demás elementos no tienen gran relevancia, debido a que son elementos necesarios para mostrar una interfaz, y por tanto han sido creados de forma automática por el sistema.

En cuanto a las funciones de la clase, solo podemos añadir que el propio nombre es muy descriptivo, y por tanto se utilizan en función del botón que haya sido pulsado por el usuario.

Hemos de indicar que las modificaciones que se realicen sobre esta tabla, solo estarán disponibles en la base de datos, cuando se pulse el botón guardar.

4.2.3.8 jDialogActivateUpdateTable

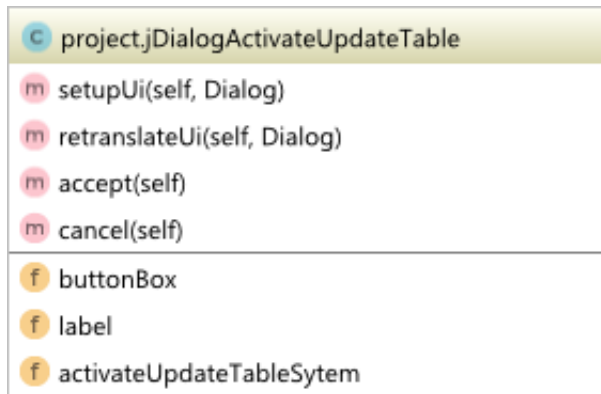


Imagen 4.2.3.8

Esta clase activa un sistema de refresco de las tablas basado en el número de actualizaciones de las tablas y su frecuencia, con el fin de no bloquear la interfaz gráfica cuando el proceso *analyst* pida a la ventana principal que esta se actualice.

Para esto hemos definido esta clase, la cual muestra un mensaje en pantalla y pide al usuario que ratifique o no dicho mensaje, pulsando el botón aceptar o cancelar. Una vez pulsado, se utiliza la variable *activateUpdateTableSystem* para enviar una señal a la GUI con el fin de que active ese sistema de refresco de la interfaz. Si pulsase rechazar entonces no se habilitaría dicho sistema.

Las funciones más importantes por tanto son *accept* y *cancel*, en las cuales se establece el procedimiento cuando el usuario pulsa sobre una u otra opción.

4.2.3.9 jDialogHelp




	project.jDialogHelp
	setupUi(self, Dialog)
	retranslateUi(self, Dialog)
	scrollAreaWidgetContents_2
	label_49
	scrollAreaWidgetContents
	scrollArea
	label_6
	label_5
	tableCTab
	label_20
	label_4
	label_21
	label_3
	label_22
	tab
	label_23
	label_9
	label_24
	label_8
	label_25
	label_7
	tabWidget
	tableMTab
	label_16
	label_18
	label_19
	label_52
	label_53
	label_10
	scrollArea_2
	label_54
	pushButton
	label_11
	label_12
	label_13
	label_14
	label_15
	label_50
	label_51

Imagen 4.2.3.9

Esta clase se encarga de mostrar la ayuda al usuario, con la información más relevante del sistema, para ello se utilizan todos los *label* que pueden apreciarse en la imagen, pero que no tienen ningún tipo de función salvo mostrar un texto.

4.2.3.10 updateTableClass

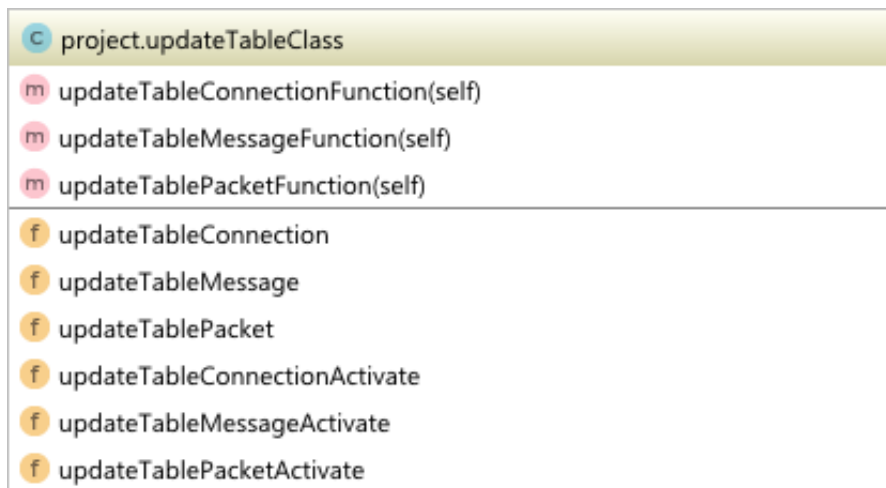


Imagen 4.2.3.10

Esta clase funciona igual que la clase *errorAnalyst* pero esta es la que utilizamos para las tablas, como podemos apreciar, en vez de tener una única función tenemos varias, lo cual permite que actualicemos solo la tabla necesaria, con lo que evitamos que la base de datos se sobrecargue por una gran cantidad de accesos que serían inútiles, ya que la tabla no habría sufrido ninguna modificación.

La variable *updateTable_____Activate* (sustituya _____ por el nombre de la tabla) se encarga de impedir la actualización de las tablas, porque el usuario ha pulsado la opción del menú contextual de una tabla concreta, por eso tenemos una por cada tabla.

Las funciones se encargan de enviar las señales, como en la clase que se ha explicado anteriormente.

4.2.3.11 jDialogData

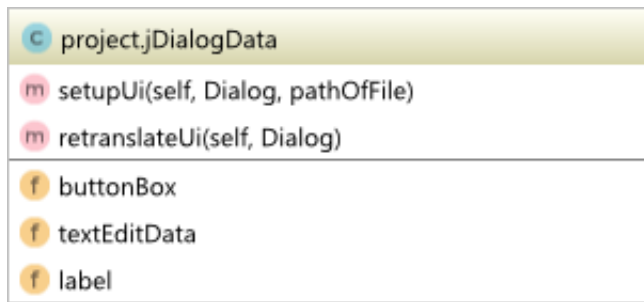


Imagen 4.2.3.11

Esta clase únicamente se utiliza para mostrar el contenido de los ficheros de los paquetes que han sido almacenados, y por consiguiente marcados como peligrosos, en la base de datos.

El contenido del paquete se muestra en el objeto *textEditData*, mientras que el objeto *buttonBox* nos permite cerrar la ventana, en el que se muestra dicho contenido.

4.2.3.12 jDialogInsertDate

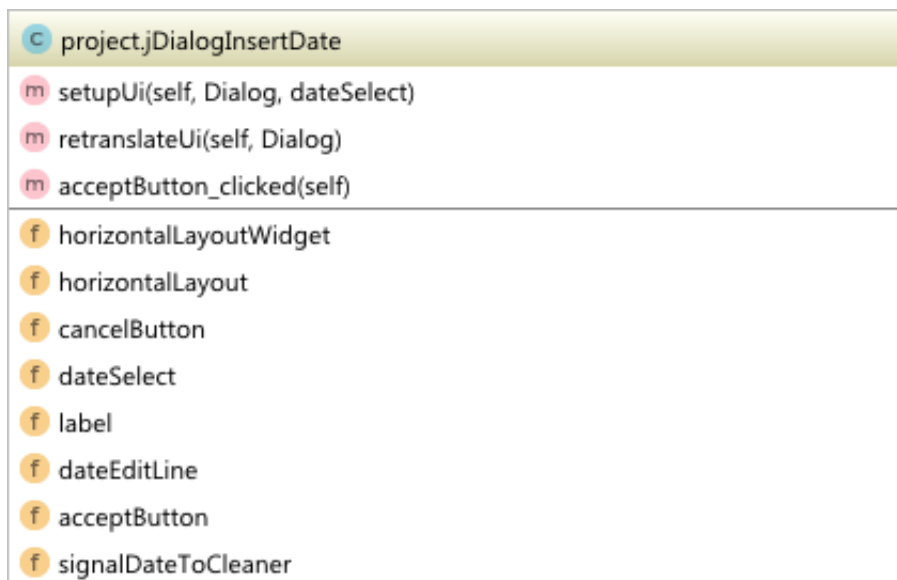


Imagen 4.2.3.12

Esta clase permite la introducción de datos necesarios para el funcionamiento de una parte del sistema, como es el proceso *cleaner*. Cuya función es la limpieza de la base de datos; pero para poder realizar esta tarea, es necesario que el usuario

inserte una fecha, a partir de la cual se borrarán los registros. Este es el objetivo de la presente clase.

Los elementos más importantes, *dateEditLine* el cual es una línea de texto que permite la introducción. En el *placeholder* de dicho elemento podemos observar el formato en el que se debe introducir la fecha, para su correcta utilización.

Para guardar la fecha escogida por el usuario utilizamos la variable *dateSelect*, la cual pasaremos a la interfaz, para que a su vez esta se la remita al proceso demandante.

La variable *signalDateToCleaner* es la señal que utilizamos para enviar a la *main* GUI el dato necesario, se utiliza el mismo proceso que para las señales utilizadas en otras *jDialog*.

Y por último en lo referente a las variables, comentar que los *button* cierran la ventana y, dependiendo cual se pulse, se enviará una fecha o se abortará la ejecución de la ventana. Si nosotros insertásemos una fecha esta sería la que se enviaría al sistema, pero si pulsásemos el botón cancelar entonces se abortaría la ejecución de esta opción. Los elementos *layout* se crean de forma automática para mostrar la interfaz, y el objeto *label* se utiliza para mostrar la información.

Con respecto a las funciones solamente tenemos la función *acceptButton_clicked* la cual se encarga de comprobar que la fecha lleve el formato apropiado, y de mandar la fecha a través de la señal, para su posterior utilización por el proceso *cleaner*.

4.2.3.13 jDialogSelectDevice

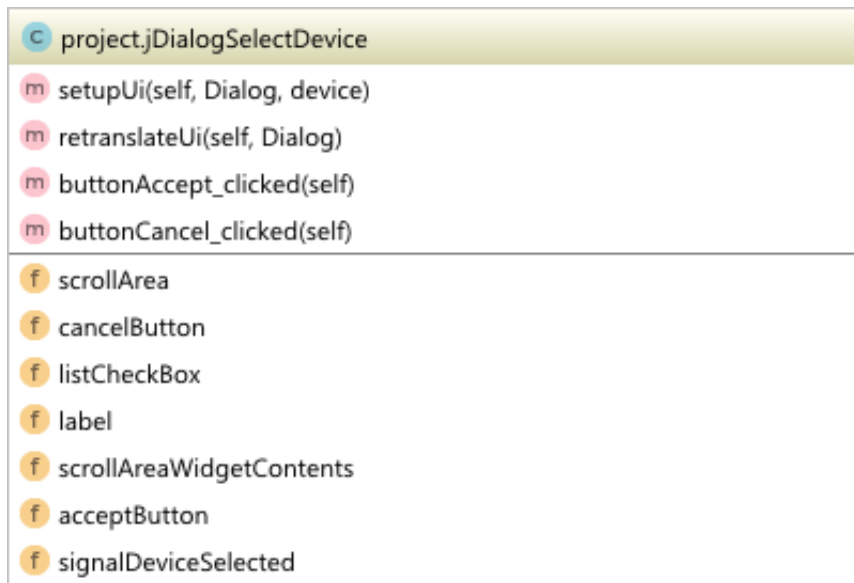


Imagen 4.2.3.13

Los elementos más importantes de esta clase son:

- *ListCheckBox*: En esta variable almacenamos la lista de servicios de red que tiene disponible el sistema, para mostrarlos al usuario, y que este escoja uno.
- *cancelButton*: Es la variable que define el botón de cancelar.
- *AcceptButton*: Es igual que la variable anterior, pero para el botón de aceptar.
- *SignalDeviceSelected*: Esta es la señal que envía el elemento de la lista seleccionado, funciona de la misma forma que sus predecesoras.

De entre las funciones debemos resaltar la función *buttonAccept_Clicked*, la cual extrae el elemento seleccionado de la lista y, se la pasa a la GUI para que esta la envíe al proceso *analyst*.

4.3 Implementación

En este apartado procederemos a describir las decisiones de diseño adoptadas, así como los problemas encontrados y las soluciones adoptadas a dichos problemas.

4.3.1 Decisiones de diseño

Procedemos a explicar las decisiones de diseño y el porqué de dichas decisiones.

4.3.1.1 Modularidad

Hemos tratado de crear un sistema versátil, en el cual se pueda intercambiar una pieza del sistema, siempre que cumplas unas pautas establecidas para el correcto funcionamiento del proyecto. Con ello permitimos un desarrollo más versátil.

Como consecuencia la estructura del proyecto queda con tres procesos que son: Proceso GUI, proceso analizador y proceso *sniffer*.

La parte que hemos establecido para su división y, por tanto susceptible de intercambiarse es: *Sniffer*, para que si un desarrollador prefiere utilizar un sistema diferente para captar los paquetes, puede intercambiar esta parte, lo que da una mayor modularidad.

Importante: Esta decisión de diseño se ha visto alterada. Se expone en la parte de problemas encontrados y solución adoptada, en el apartado 4.3.2.5.

4.3.1.2 Uso de hilos

Hemos considerado necesario el uso de hilos por ser imprescindible para el correcto funcionamiento del sistema.

¿Por qué? Cuando hicimos las primeras pruebas con el *sniffer* y la interfaz gráfica, la cual también incluía el analizador, se demostró que esta opción era incorrecta, ya que el *sniffer* no permitía el uso de la GUI al quedar en un bucle continuo de recepción de paquetes, y por tanto no se podía utilizar ninguna de las opciones que estaban disponibles para la misma, tampoco podíamos dormir el proceso *sniffer* ya que eso implicaba utilizar la interfaz pero esta estaba bloqueada.

4.3.1.3 Inclusión de la opción “*stop update*”

Hemos incluido en las distintas tablas una opción llamada “*stop update*”, con esta acción permitimos al usuario congelar la actualización de una o varias tablas, para que la interfaz no se sobrecargue, e impida su uso por entrar en un bucle de constante actualización de la tablas.

Como se ha explicado en la estructura, al establecer la comunicación entre la GUI y el *sniffer* mediante la clase “*updateTable*” y la base de datos, lo que hace esta opción es impedir el envío de la señal correspondiente al hilo principal.

4.3.1.4 Introducción de la opción “*insert table routing*”

Para poder detectar ataques basados en TTL es necesario el acceso a la tabla de rutas del *router* que se encuentra después del *sniffer*, hemos explorado distintas opciones para la automatización del proceso, pero han sido esfuerzos infructuosos, y por ello incluimos la opción de la inserción manual.

4.3.1.5 Ejecución del hilo *cleaner*

Hemos decidido forzar la ejecución en solitario del hilo *cleaner*, para mantener la integridad de la base de datos.

Sqlite al ser una base de datos de pequeñas dimensiones, no tiene implementado un sistema que permita gestionar el borrado y la inserción de elementos de forma paralela, para no realizar una sobrecarga en los recursos por parte de nuestro sistema.

4.3.1.6 Un solo hilo *sniffer*

Cuando iniciemos la aplicación veremos, que podemos monitorizar cualquier servicio de red, dejando abierta una posible modificación del sistema para la utilización de varios hilos de monitorización.

Nosotros hemos decidido que solo sea capaz de lanzarse un hilo de monitorización de forma concurrente, para poder mantener el objetivo de obtener un IDS que no conlleve un gran consumo de recursos. Aunque se podría modificar el sistema para poder permitir lanzar varios procesos a la vez.

4.3.1.7 Espera de la finalización del hilo *sniffer*

Cuando nosotros pulsemos sobre la opción de parar el *sniffer*, debemos esperar a la finalización de este proceso, ya que para poder lanzarlo otra vez necesitamos sobrescribir las variables porque la librería no nos permite lanzarlo de nuevo, al haber acabado su ejecución, por lo tanto hemos decidido sobrescribir la variable para poder volver a lanzar el proceso.

4.3.1.8 Selección del servicio de red a monitorizar

Cuando sea la primera vez que inicies el hilo *sniffer*, y no hayamos configurado la elección del servicio a monitorizar, se obligará al usuario a elegir, lanzando la ventana de elección del mismo.

Esta elección es requerida para que el proceso *sniffer* sea lanzado.

Importante: Aunque también está incluido en la ayuda del programa, debe ejecutarse en modo administrador, para que el sistema para que pueda acceder a los servicios de red.

4.3.1.9 Inserción de la fecha de borrado

Cuando se quiera lanzar el hilo *cleaner* para la limpieza de la base de datos, hemos considerado que cuando el usuario no quiera incluir una fecha se borre, por lo que no procederemos al borrado, ya que este no ha establecido una fecha.

Hemos tomado esta decisión por si el usuario cambiase de idea, y no quisiese borrar dichos datos, dado que el proceso no puede ser abortado.

4.3.1.10 Utilización de varios UpdateTable

Una vez que se hace necesaria la actualización de las tablas, debido a la introducción de nuevos datos en la base de datos, se envía una señal para la actualización de la tabla correspondiente, ya que hemos definido varios *updateTable*, uno por cada tabla, para que solo se actualice la tabla que deba; es decir, si se marcasse un paquete como peligroso, y esta conexión ya está recogida en la base de datos por otro paquete anterior, entonces solo sería necesario la actualización de la tabla *listMessage* y no de todas las que contiene la GUI.

4.3.1.11 Restauración de los mensajes

Si observamos los objetivos del proyecto, contenidos en el epígrafe 1, podemos observar que este es un proyecto cuya finalidad es crear una aplicación que pueda seguir desarrollándose, para ello debemos de dotar al posible desarrollador del máximo de herramientas para poder hacer el análisis que estime oportunos.

Con este fin, hemos decidido que almacene todos los paquetes fragmentados para realizar una posterior reconstrucción del mensaje, independientemente de si ha

sido marcado o no como paquete peligroso, y así poder analizar el contenido del mismo aunque estuviese fragmentado.

Es importante indicar que, todo paquete fragmentado en el que no se haya detectado ninguna característica del ataque recogido en el apartado 4.1.2.4 tendrá una descripción vacía, y que esta solo se rellenara cuando cumpla alguna condición que lo marque como paquete peligroso.

4.3.2 Problemas encontrados y solución adoptada

4.3.2.1 Almacenado en la base de datos

Cuando pusimos a prueba el sistema el primer paquete que fue a insertarse en la base de datos, dio un error, y es que *sqlite* no puede almacenar cadenas de caracteres que no correspondan a caracteres de la codificación UTF-8 y, los paquetes incluían caracteres que no están incluidos en UTF-8.

La solución adoptada consiste en crear **un sistema de ficheros** que permita el almacenamiento del contenido de los paquetes, además esto permite que la base de datos pueda ser más grande al no tener que almacenar una parte importante del paquete.

4.3.2.2 Comunicación entre el *sniffer* y el analizador

El principal problema fue que la comunicación entre hilos en *python* se realiza mediante una pila o *queue*, pero para poder hacer eso las clases que se comunican deben estar en el mismo fichero, por lo que es imposible si se quiere dar modularidad al sistema.

La solución adoptada consiste en la implementación de un **socket** que permita la comunicación entre los distintos hilos, para poder pasar los paquetes captados al analizador.

Importante: Esta solución ha sido modificada, se explica en el apartado 4.3.2.5

4.3.2.3 Comprobación de la fragmentación

El problema consiste en la llegada de paquetes ip fragmentados, si esto ocurre nosotros no tenemos ninguna herramienta que nos asegure la correcta llegada de los fragmentos.

La solución consiste en la comprobación de la llegada completa del paquete, siempre y cuando llegue un fragmento del paquete ip fragmentado.

4.3.2.4 Limitación del envío de la señal “updateTable”

El problema consiste en que se impide el correcto uso de la interfaz cuando se producen muchas actualizaciones de las tablas de la GUI.

Hemos decidió establecer un **límite para la actualización de las tablas**, debido a que de esta forma se permite el correcto uso de la interfaz.

4.3.2.5 Modularidad

El principal problema encontrado en este apartado es que al usar un socket, se interrumpe la comunicación entre el hilo *sniffer* y el analizador.

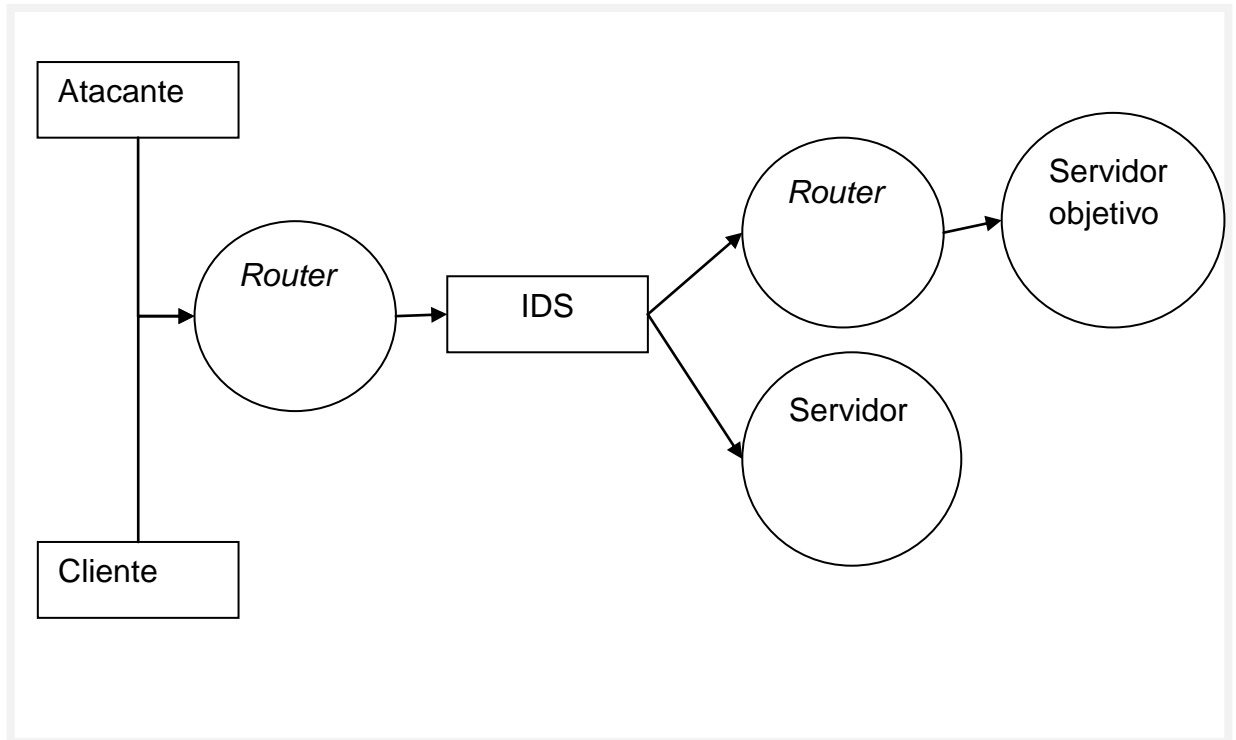
¿Cómo se realiza la comunicación? La comunicación se realizaba estableciendo un servidor y un cliente, para que enviar los paquetes capturados al analizador, al establecer este sistema estamos creando paquetes que se envían entre los hilos, pero dichos paquetes son capturados por el *sniffer* y pasados al analizador como si se tratara de un paquete exterior, pero con la gran diferencia del cambio de ip en algunos paquetes, lo que daba lugar a que la base de datos almacenará paquetes con direcciones imposibles.

Debido a la imposibilidad de poder discretizar dichos paquetes, y a la modificación que en algunos se ejercía, la solución ha consistido en el **cambio en la estructura** para que el *sniffer* se incluya en el analizador y por tanto no sea necesario establecer una comunicación entre el *sniffer* y el analizador.

4.4 Pruebas y resultados

Para probar el sistema hemos tenido que crear otro programa, cuyo nombre es: ***programToSendPacket***, descrito en el anexo IV, el cual se encargaba de realizar el envío de paquete “basura” (paquetes que no tenían ninguna importancia, y cuyo único objetivo era generar tráfico). También puede lanzar los ataques que nuestro IDS es capaz de detectar, dicho programa fue ejecutado sobre el sistema operativo Linux.

La prueba siguiendo la Dibujo 7.1, en la que se han usado 2 ordenadores y 2 instancias distintas del programa para con una enviar paquetes irrelevantes, y con la otra instancia realizar los ataques que detecta nuestro sistema.



Dibujo 4.4.1

Como apreciamos en el ejemplo hemos intentado simular una situación real, en la cual exista una conexión con un de los servidores y que otro servidor esté bajo un ataque. El número de saltos que existe entre los elementos es de 3 saltos.

La configuración del IDS se muestra en las imágenes 4.4.1, 4.4.2 y 4.4.3.

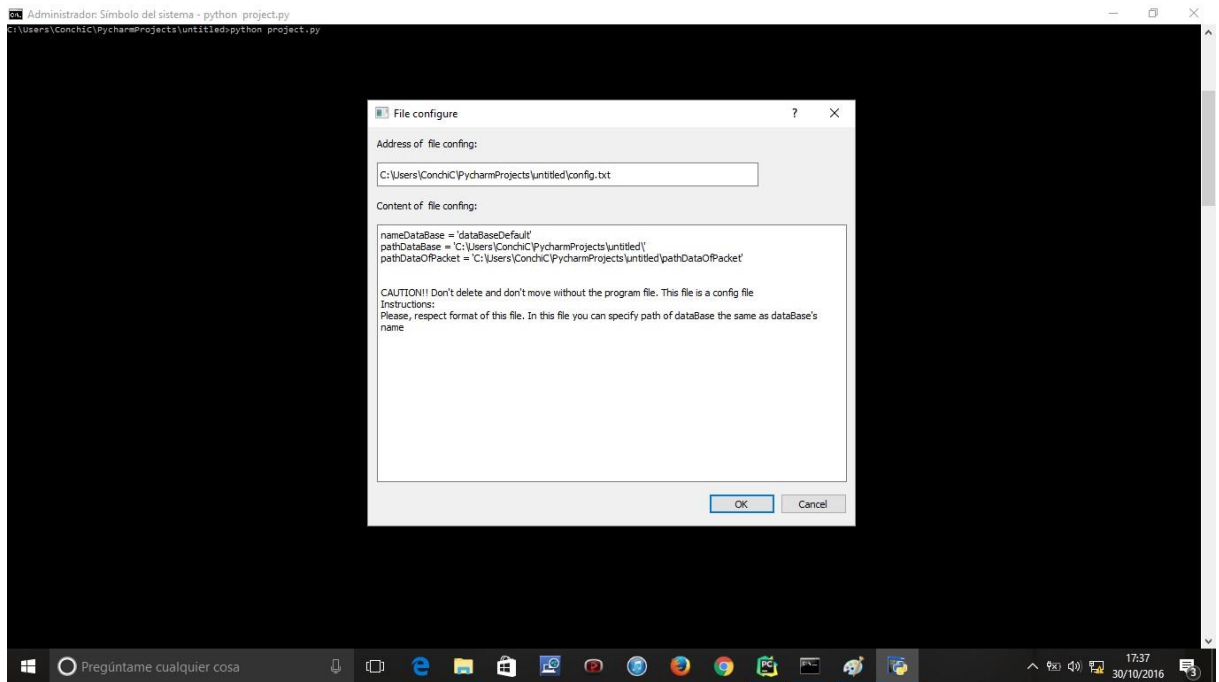


Imagen 4.4.1

En la imagen 4.4.1 podemos apreciar la configuración básica de elementos importantes del sistema, como son el nombre de la base de datos, así como la dirección de la misma y la carpeta donde se almacenan los datos de los paquetes marcados como peligrosos.

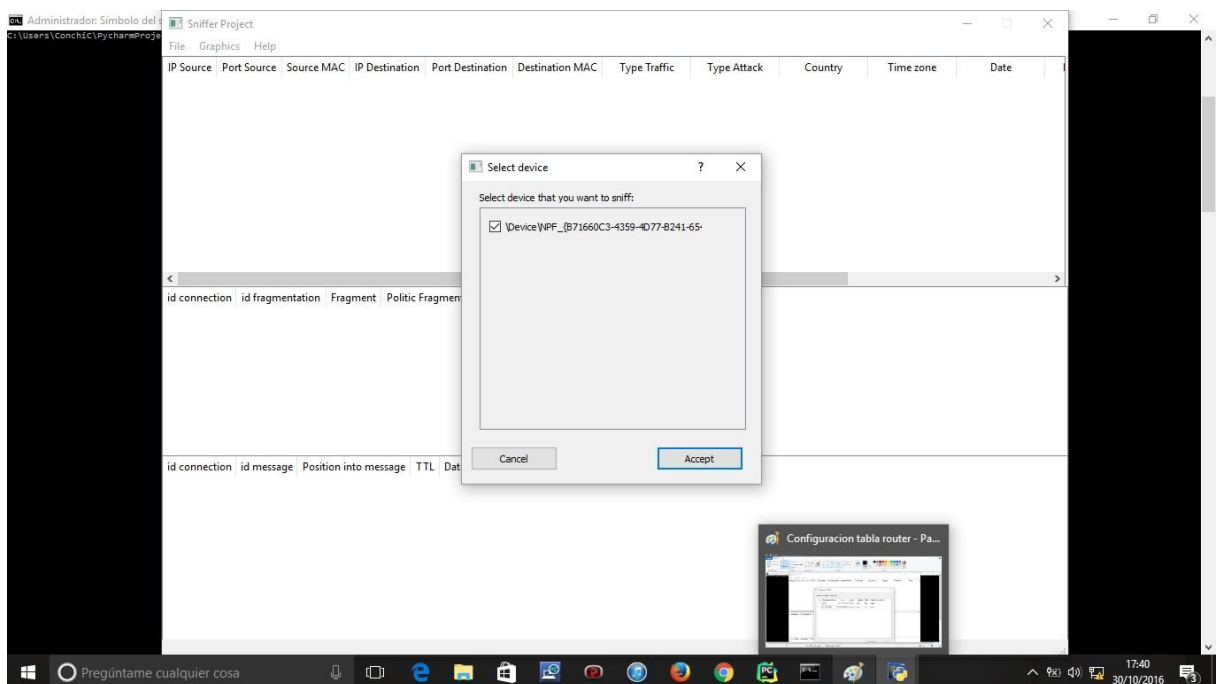


Imagen 4.4.2

En este escogemos el servicio de red que queremos *sniffar*, en caso de que dicho servicio no esté conectado daría un error y no se lanzaría el *sniffer*. Si algún dispositivo no estuviese habilitado por el sistema operativo ni siquiera aparecería en la lista.

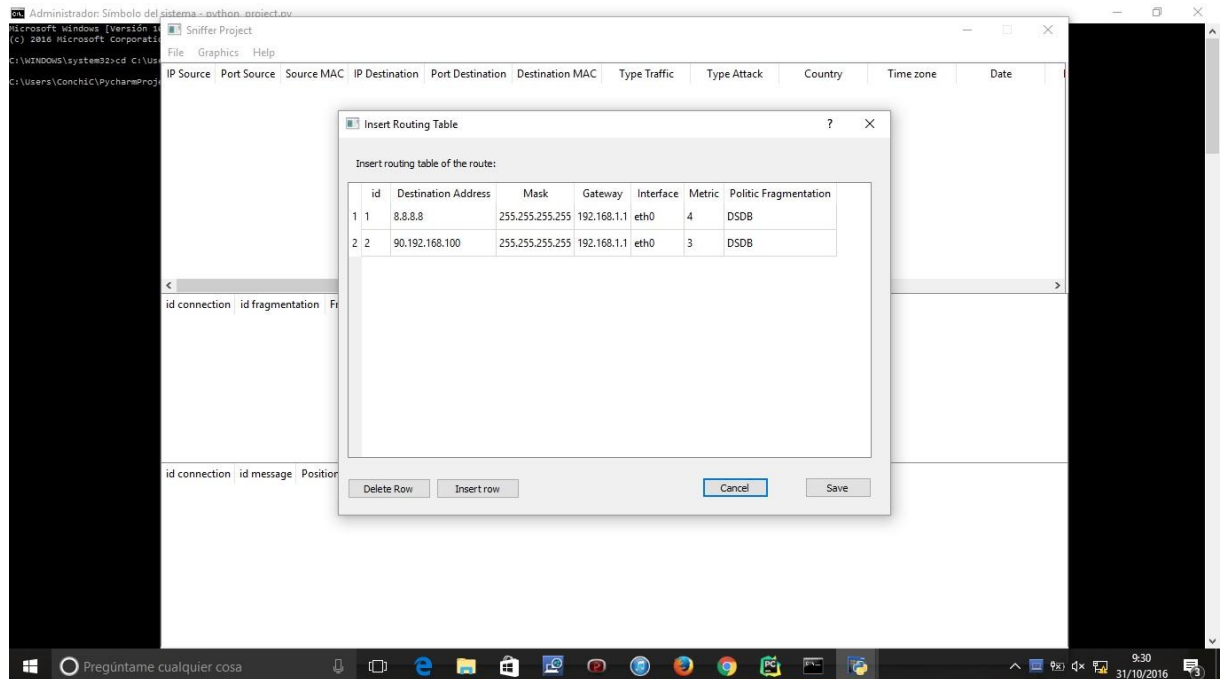
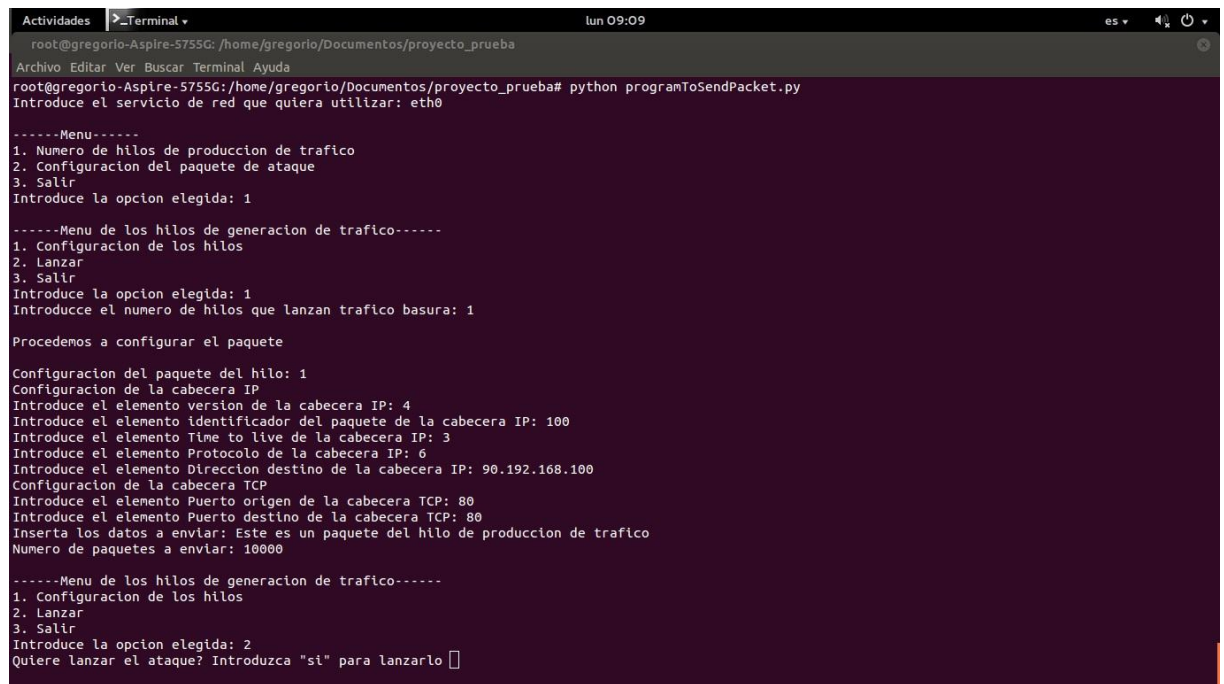


Imagen 4.4.3

En esta tabla debemos introducir la tabla de enrutamiento del *router*, además es una configuración muy importante, ya que si no estuviese rellena la tabla saltaría una advertencia, la cual expondría la imposibilidad de detectar los ataques basados en TTL.

La configuración de todos los paquetes se han realizado rellenando los campos de la cabecera IP y la cabecera TCP, las cuales están explicadas en los Anexos II y III respectivamente, la configuración del hilo que genera tráfico es la siguiente:



```
root@gregorio-Aspire-5755G: /home/gregorio/Documentos/proyecto_prueba
root@gregorio-Aspire-5755G:/home/gregorio/Documentos/proyecto_prueba# python programToSendPacket.py
Introduce el servicio de red que quiera utilizar: eth0

-----Menu-----
1. Numero de hilos de produccion de trafico
2. Configuracion del paquete de ataque
3. Salir
Introduce la opcion elegida: 1

-----Menu de los hilos de generacion de trafico-----
1. Configuracion de los hilos
2. Lanzar
3. Salir
Introduce la opcion elegida: 1
Introduce el numero de hilos que lanzan trafico basura: 1

Procedemos a configurar el paquete

Configuracion del paquete del hilo: 1
Configuracion de la cabecera IP
Introduce el elemento version de la cabecera IP: 4
Introduce el elemento identificador del paquete de la cabecera IP: 100
Introduce el elemento Time to live de la cabecera IP: 3
Introduce el elemento Protocolo de la cabecera IP: 6
Introduce el elemento Direccion destino de la cabecera IP: 90.192.168.100
Configuracion de la cabecera TCP
Introduce el elemento Puerto origen de la cabecera TCP: 80
Introduce el elemento Puerto destino de la cabecera TCP: 80
Inserta los datos a enviar: Este es un paquete del hilo de produccion de trafico
Numero de paquetes a enviar: 10000

-----Menu de los hilos de generacion de trafico-----
1. Configuracion de los hilos
2. Lanzar
3. Salir
Introduce la opcion elegida: 2
Quiere lanzar el ataque? Introduzca "si" para lanzarlo ☐
```

Imagen 4.4.4

Nótese que tanto el campo secuencia, ack y identificador se incrementan de forma automática, ya que si esto no se diese no sería una conexión real debido a que todos los paquetes tendrían el mismo identificador, y esto solo se da cuando el paquete ha sido fragmentado. El campo secuencia tampoco podría ser el mismo ya que este se utiliza para reordenar los paquetes los segmentos TCP.

La configuración de los distintos ataques se ha realizado teniendo en cuenta los campos que deben ser alterados para poder realizar el ataque correspondiente, y la configuración para cada tipo de ataque. Pasamos a mostrar a continuación:

Ataque basado en TTL:

```
root@gregorio-Aspire-5755G: /home/gregorio/Documentos/proyecto_prueba
Archivo Editar Ver Buscar Terminal Ayuda
root@gregorio-Aspire-5755G:/home/gregorio/Documentos/proyecto_prueba# python programToSendPacket.py
Introduce el servicio de red que quiera utilizar: eth0

-----Menu-----
1. Numero de hilos de produccion de trafico
2. Configuracion del paquete de ataque
3. Salir
Introduce la opcion elegida: 2

-----Menu de configuracion de ataque-----
1. Ataque basado en TTL
2. Ataque basado en RST
3. Ataque basado en SYN
4. Ataque de fragmentacion
5. Configurar todo el paquete para atacar
6. Salir
Introduce la opcion elegida: 1
Configuracion del ataque basado en TTL
Introduce el elemento Identificacion de la cabecera IP: 400
Introduce el elemento Time to live de la cabecera IP: 3
Introduce el elemento Direccion destino de la cabecera IP: 8.8.8.8
Introduce el elemento Puerto de origen de la cabecera TCP: 80
Introduce el elemento Puerto de destino de la cabecera TCP: 80
Introduce los datos: Este es el paquete de ataque basado en ttl
PPPPPPEste es el paquete de ataque basado en ttl
Quiere lanzar el ataque? Introduzca "si" para lanzarlo
```

Imagen 4.4.5

En este ataque lo más importante es poner el campo TTL de la cabecera en un número inferior para poder lograr este ataque.

Ataque basado en RST y ataque basado en SYN:

```
root@gregorio-Aspire-5755G: /home/gregorio/Documentos/proyecto_prueba
Archivo Editar Ver Buscar Terminal Ayuda
Introduce el elemento Time to live de la cabecera IP: 3
Introduce el elemento Direccion destino de la cabecera IP: 8.8.8.8
Introduce el elemento Puerto de origen de la cabecera TCP: 80
Introduce el elemento Puerto de destino de la cabecera TCP: 80
Introduce los datos: Paquete de ataque basado en TTL
[PPPPPP] Paquete de ataque basado en TTL
Quiere lanzar el ataque? Introduzca "si" para lanzarlo si
enviado ataque TTL

-----Menu de configuracion de ataque-----
1. Ataque basado en TTL
2. Ataque basado en RST
3. Ataque basado en SYN
4. Ataque de fragmentacion
5. Configurar todo el paquete para atacar
6. Salir
Introduce la opcion elegida: 2
Configuracion del ataque basado en RST
Introduce el elemento Identificacion de la cabecera IP: 401
Introduce el elemento Time to live de la cabecera IP: 4
Introduce el elemento Direccion destino de la cabecera IP: 8.8.8.8
Introduce el elemento Puerto de origen de la cabecera TCP: 80
Introduce el elemento Puerto de destino de la cabecera TCP: 80
Introduce los datos: Este es el paquete de ataque basado en RST
Quiere lanzar el ataque? Introduzca "si" para lanzarlo si
enviado ataque RST

-----Menu de configuracion de ataque-----
1. Ataque basado en TTL
2. Ataque basado en RST
3. Ataque basado en SYN
4. Ataque de fragmentacion
5. Configurar todo el paquete para atacar
6. Salir
Introduce la opcion elegida: 3
Configuracion del ataque basado en SYN
Introduce el elemento Identificacion de la cabecera IP: 402
Introduce el elemento Time to live de la cabecera IP: 4
Introduce el elemento Direccion destino de la cabecera IP: 8.8.8.8
Introduce el elemento Puerto de origen de la cabecera TCP: 80
Introduce el elemento Puerto de destino de la cabecera TCP: 80
Introduce los datos: Paquete de ataque basado en SYN
Quiere lanzar el ataque? Introduzca "si" para lanzarlo si
enviado ataque SYN

-----Menu de configuracion de ataque-----
1. Ataque basado en TTL
2. Ataque basado en RST
3. Ataque basado en SYN
4. Ataque de fragmentacion
5. Configurar todo el paquete para atacar
6. Salir
Introduce la opcion elegida: [ ]
```

Imagen 4.4.6

En la parte superior se puede observar la configuración del ataque RST y en la inferior el de SYN.

Y por últimos el ataque de fragmentación:

```

root@gregorio-Aspire-5755G: /home/gregorio/documentos/proyecto_prueba
Archivo Editar Ver Buscar Terminal Ayuda
-----Menu de configuracion de ataque-----
1. Ataque basado en TTL
2. Ataque basado en RST
3. Ataque basado en SYN
4. Ataque de fragmentacion
5. Configurar todo el paquete para atacar
6. Salir
Introduce la opcion elegida: 4
Configuracion del ataque de fragmentacion
Introduce el numero de paquetes de este ataque: 3
Configuracion comun de los paquetes
Introduce el elemento Identificacion de la cabecera IP: 403
Introduce el elemento Time to live de la cabecera IP: 4
Introduce el elemento Direccion destino de la cabecera IP: 8.8.8.8
Introduce el elemento Puerto de origen de la cabecera TCP: 80
Introduce el elemento Puerto de destino de la cabecera TCP: 80
Introduce el elemento Numero de secuencia de la cabecera TCP: 503
Introduce el elemento Numero ACK de la cabecera TCP: 303
Configuracion del paquete 1
Introduce el elemento flags fragmentation de la cabecera IP del paquete 1: 1
Introduce el elemento offset de la cabecera IP del paquete 1: 0
Introduce los datos del paquete 1: Paquete inicial debe sobrescribirse sobre siguiente
Configuracion del paquete 2
El size del paquete anterior es(en bloques de 64 bits): 13
Introduce el elemento flags fragmentation de la cabecera IP del paquete 2: 1
Introduce el elemento offset de la cabecera IP del paquete 2: 12
Introduce los datos del paquete 2: esta parte se borra, parte intermedia del ataque
Configuracion del paquete 3
El size del paquete anterior es(en bloques de 64 bits): 25
Introduce el elemento flags fragmentation de la cabecera IP del paquete 3: 0
Introduce el elemento offset de la cabecera IP del paquete 3: 25
Introduce los datos del paquete 3: parte final del ataque basado en fragmentacion
Quiere lanzar el ataque? Introduzca "si" para lanzarlo si
E[si]
E[si]
E[si]
-----Menu de configuracion de ataque-----
1. Ataque basado en TTL
2. Ataque basado en RST
3. Ataque basado en SYN
4. Ataque de fragmentacion
5. Configurar todo el paquete para atacar
6. Salir
Introduce la opcion elegida: 6
-----Menu-----
1. Numero de hilos de produccion de trafico
2. Configuracion del paquete de ataque
3. Salir
Introduce la opcion elegida: 3
root@gregorio-Aspire-5755G: /home/gregorio/documentos/proyecto_prueba#

```

Imagen 4.4.7

Cuando se configura un ataque de fragmentación se tiene que definir una parte común de los paquetes, que son las direcciones IP, junto con el TTL y el identificador, ya que este se utiliza para identificar paquetes que componen un mensaje, los cuales han sido fragmentados por no ser capaces de viajar por la red debido a su tamaño. También debemos establecer campos que serán comunes dentro de la cabecera TCP.

Como hemos podido ver en la definición del ataque de fragmentación, el sistema nos proporciona una ayuda indicando el *offset* del segundo paquete del ataque de fragmentación. ¿Qué indica el *offset*? Indica el número de bloque, estos son de 64 bits, que debemos introducir para que el segundo paquete siga al primero.

Para que quede más claro lo expondremos esto con el resultado obtenido en la prueba. Como sabemos el ataque de fragmentación se basa en la utilización de

varios paquetes, como puede apreciarse en el apartado 4.1.2.4, y por eso para la realización de este ataque debemos definir varios paquetes, que compondrán uno mayor.

Tras la definición del primer paquete, el sistema nos indica un número de debemos colocar en el campo *offset* del segundo paquete, este número nos permite manejar la reconstrucción de un paquete en cualquier sistema (servidores, IDS, etc.).

Si nosotros colocásemos el valor 13 en el campo *offset* del segundo paquete, estamos concatenando el paquete de forma correcta, con lo que el resultado sería: “Parte inicial debe sobrescribirse sobre siguiente esta parte se borra, parte intermedia del paquete”, pero si establecemos un número inferior al indicado por el sistema el resultado sería que parte del primer paquete es sobrescrito por parte del segundo, lo que podría dar el siguiente resultado: “Paquete inicial debe sobrescribirse sobre siguiente se borra,”, como apreciamos hay varios caracteres que se han sobrescrito por parte del primer paquete.

Los resultados obtenidos han sido los siguientes:

Para el ataque basado en TTL:

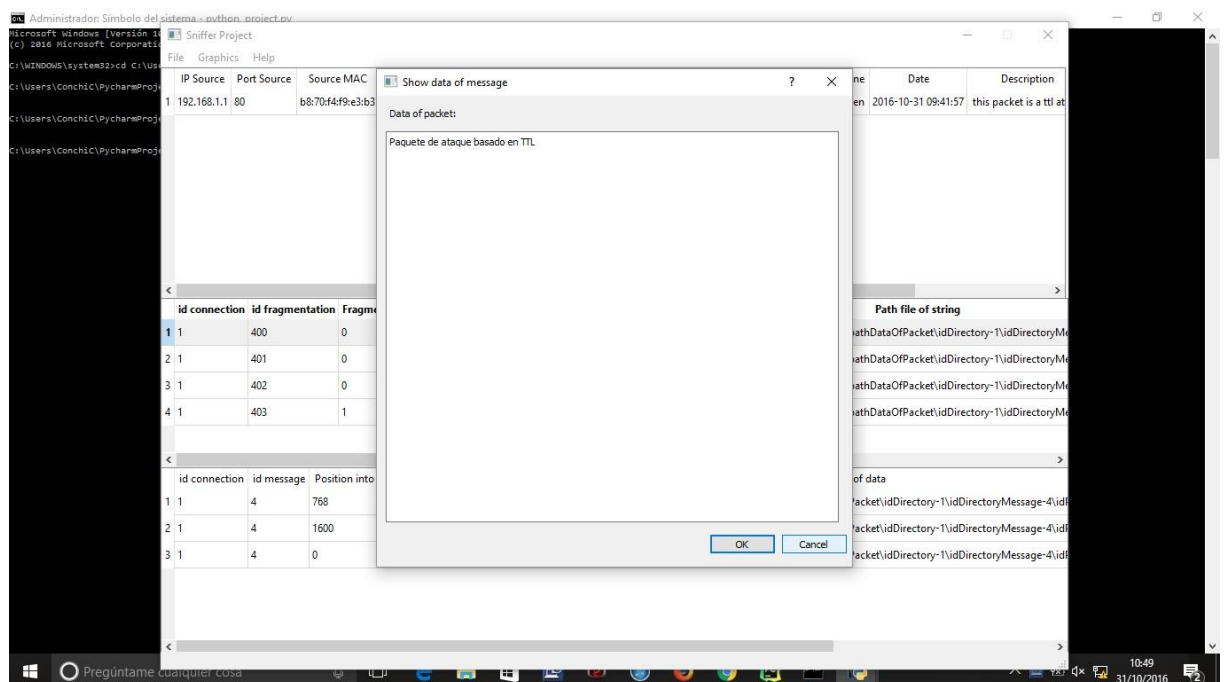
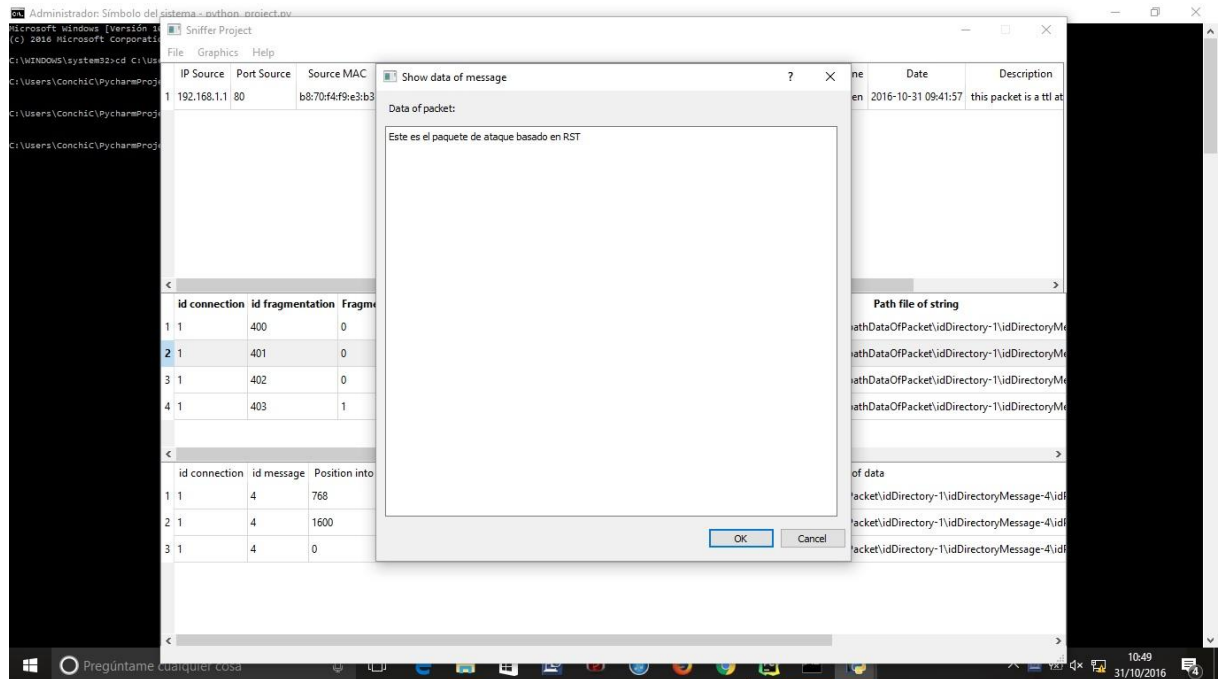


Imagen 4.4.8

Se ha capturado el paquete debido a que cumple las condiciones de un ataque basado en TTL.

Para el ataque basado en RST:

**Imagen 4.4.9**

Para el ataque basado en SYN:

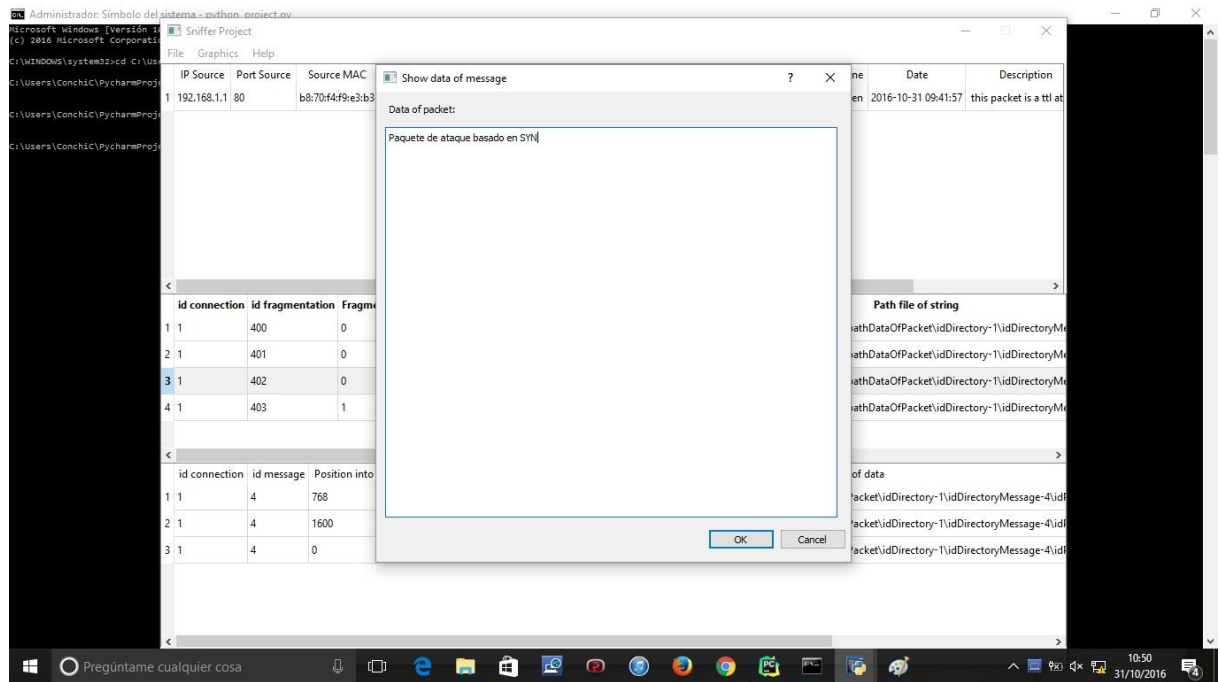


Imagen 4.4.10

Para el ataque de fragmentación:

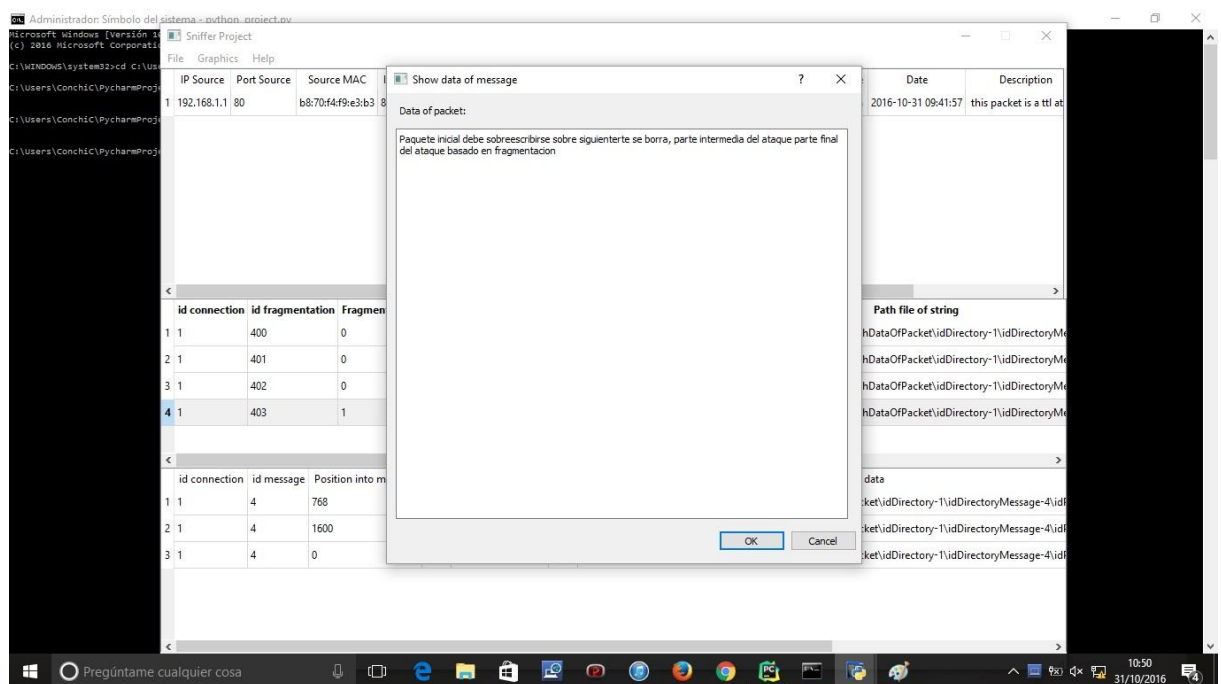


Imagen 4.4.11

Como podemos observar han sido detectados todos los ataques, y en el ataque de fragmentación puede apreciarse que una parte del paquete intermedio ha sido sobrescrita debido a las políticas de re-ensamblado de los paquetes IP.

5. Conclusiones

Como podemos apreciar los IDS son sistemas vitales para la protección de las redes y sistemas informáticos. El sistema propuesto en este trabajo nos permite hacer frente a este desafío, además de ser lo suficientemente extensible para incluir nuevos tipos de ataques a detectar.

En este punto vamos a mirar hacia atrás y comprobar los objetivos marcados en el epígrafe 1 se han cumplido en esta aplicación.

Con respecto al primer punto de los objetivos, hemos debido realizar un análisis profundo de los métodos de ataques con el fin de comprender como se realizan dichos ataques, y lo más importante, cómo podemos detectar ese ataque, para que el usuario sea capaz de dar una respuesta que proteja su sistema.

Atendiendo a la gestión de tráfico de red podemos decir que hemos conseguido una buena gestión, dado que nuestro sistema es capaz de gestionar distintas conexiones, como podemos apreciar en el apartado 4.4, para detectar posibles ataques, que vengan de cualquiera.

También hemos realizado análisis de ataques complejos, es decir, ataques que utilicen más de un paquete para llevar a término dicho intento. Además debemos señalar que proporcionamos al posible desarrollador de este sistema, la capacidad de trabajar a nivel de mensaje, ya que realizamos un re-ensamblado de los paquetes fragmentados siguiendo la política del servidor al que se dirigen los paquetes.

También hemos conseguido crear un sistema multiplataforma debido a la utilización de un lenguaje universal, y que sea capaz de adaptarse al sistema sobre el que corra, como sabemos esta aplicación utiliza un sistema de ficheros para poder guardar la información de los paquetes que han sido marcados como peligrosos, dicho sistema de ficheros se adapta al sistema operativo sobre el que se ejecute la aplicación.

El último punto de los objetivos marcados hacía referencia a una aplicación con bajos recursos y que fuese extensible. La aplicación con bajos recursos lo hemos conseguido intentando minimizar en número de hilos ejecutados en paralelo, por eso

también tomamos la decisión de que solo pueda lanzarse un *sniffer*. Además de ser extensible debido a que puedes realizar una ampliación de la aplicación mediante la inclusión de nuevos ataques a detectar.

También señalar que las pruebas realizadas con este sistema han dado un resultado positivo, lo que nos permite afirmar que este sistema podría instalarse y ser capaz de detectar los ataques descritos en esta documentación.

6. Anexos

7.1 Anexo I

Cálculo del *Checksum* de la cabecera TCP. Para poder realizar este cálculo debemos tener en cuenta los componentes de la cabecera TCP que se encuentran explicados en el anexo III.

Para calcular el *checksum* debemos seguir los siguientes pasos:

- Calcular la suma de los componentes del *pseudo-header*.
- Calcular la suma de los componentes de la cabecera TCP.
- Calcular la suma de los dos anteriores con los datos del paquete (*pseudo-header* + componentesTCP + datos).
- Calcular el complemento a uno del paso anterior.

Vamos a suponer que en este paquete no tenemos datos, por lo tanto los datos son igual a 0, y por consiguiente el tamaño de los mismo también.

Valores del *pseudo-header*, este está compuesto por:

- Dirección IP de origen: $A8B0036C_{16}$
- Dirección IP de destino: $A8B00319_{16}$
- Protocolo: 6_{10} , 6_{16}
- La suma del tamaño de la cabecera TCP y el tamaño de los datos: $1C_{16}$

Para poder exponer mejor como se realizan estos cálculos vamos a poner un ejemplo, Imaginemos un paquete TCP cuyos elementos de la cabecera tienen los siguientes valores:

- Puerto de origen: 17664_{10} , 0593_{16}
- Puerto de destino: 21_{10} , 0015_{16}
- Número de secuencia: 9095835_{10} , $8ACA9B_{16}$
- ACK: 0_{10} , 0_{16}
- Longitud de la cabecera: 7_{10} , 7_{16}
- Reservado: 0_{10} , 0_{16}
- *Flags*: 2_{10} , 2_{16}
- Tamaño de la ventana: 8192_{10} , 2000_{16}
- *Checksum*: Este campo es el que queremos calcular
- Puntero urgente: 0_{10} , 0_{16}
- Opciones(en grupos de 16 bits): 0204_{16} , 0218_{16} , 0101_{16} , 0402_{16}

Ahora vamos a proceder a realizar el primer paso que es calcular la suma de los elementos del *seudo-header*, tenemos que tener en cuenta que la suma debe realizarse en grupos de 16 bits ya que ese es el tamaño del *checksum*.

$$A8B0 + 036C + A8B0 + 0319 + 0006 + 001C = 15807$$

Como podemos ver el número resultante se representa con más de 16 bits, por lo tanto tenemos que coger el bit más representativo y sumarlo como acarreo, con lo que el resultado final sería $5807 + 1 = 5808$

Ahora procedemos a sumar los elementos de la cabecera TCP entre sí, para obtener un número de 16 bits:

$$0593 + 0015 + 008A + CA9B + 0000 + 0000 + 7002 + 2000 + 0000 + 0000 + 0204 + 0218 + 0101 + 0402 = 169EE$$

Al igual que en el *seudo-header* debemos sumar el acarreo $69EE + 1 = 69EF$. Como hemos dicho no tenemos datos, por lo que para este ejemplo los datos son igual 0 pero, en el caso de que tuviésemos datos deberíamos pasarlo a hexadecimal y dividirlo en grupos de 4 números hexadecimales, e ir sumando de la misma forma que en la cabecera TCP o el *seudo-header*.

Por último debemos sumar todos los elementos calculados en los pasos anteriores.

$$5808 + 69EF + 0000(\text{los datos}) = C1F7$$

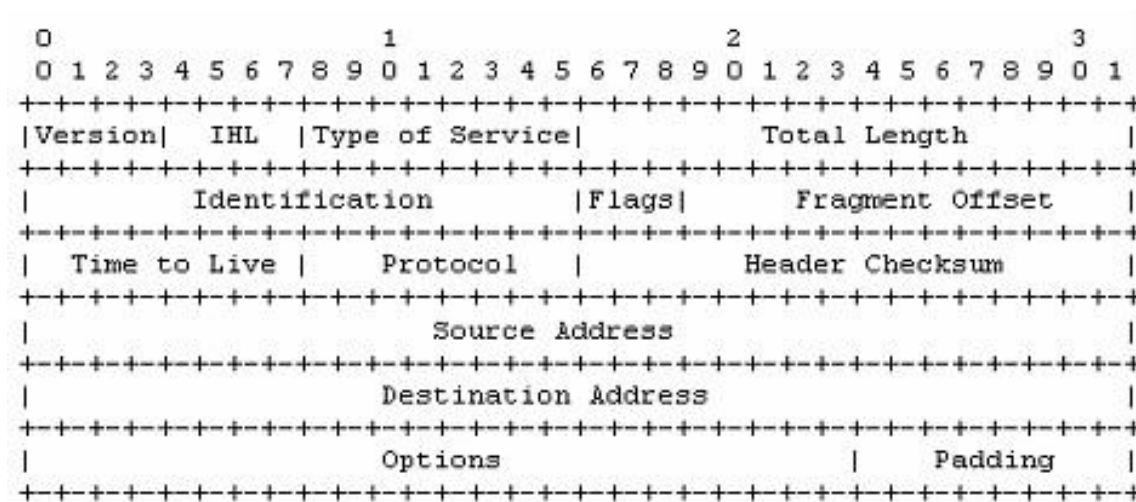
El último paso es hacer el complemento a uno del número obtenido como resultado, el complemento a uno se calcula cambiando los 1 por 0 y los 0 por 1. También podemos calcular el complemento a uno invirtiendo el número (multiplicarlo por -1 en el caso de que sea un decimal), y hacer un and con el número hexadecimal de $FFFF_{16}$.

$$C1F7_{16} = 1100\ 0001\ 1111\ 0111_2$$

El complemento a uno sería $0011\ 1110\ 0000\ 1000_2 = 3E08_{16}$, por lo que el *checksum* es $3E08_{16}$.

7.2 Anexo II

El datagrama de la cabecera IP sigue la estructura marcada por el Dibujo 7.1.



Dibujo 7.2

Los campos del datagrama IP son:

- Versión (4 bits): Este hace referencia a que protocolo de conexión sigue, IPv4 o IPv6.

- IHL (4 bits): Este campo nos indica el tamaño de la cabecera IP, normalmente será de un tamaño de 20 Bytes.
- *Type of Service*(8 bits): Este campo hace referencia a los servicios diferenciados. Este campo se introdujo tras un parcheo del protocolo IPv4 para dar prioridad a unos paquetes frente a otros, contiene:
 - DSCP o *Differentiated Services Code Point* (6 bits): Los 3 bits más significativos indican la clase del *DiffServ*, y los siguientes indican detalles de la clase. Las clases son:
 - DSCP 111xxx y 110xxx: reservado para funciones de red.
 - DSCP 101xxx: corresponde a envío expedito (*Expedited Forwarding*).
 - DSCP 100xxx, 011xxx y 001xxx: corresponde al envío asegurado (*Assured forwarding*).
 - DSCP 000000: envío *Best Effort*.
 - ECN (2 bits): Contiene información sobre la congestión en la ruta.
- *Total length* (16 bits): Indica el tamaño total del datagrama y los datos.
- *Identification* (16 bits): Es un número que nos indica un id único para cada paquete de una conexión, nos permite saber que paquetes corresponden a un datagrama mayor que ha sido fragmentado.
- *Flags*(3 bits):nos sirven para saber si un paquete está fragmentado, los bits que lo componen son(del más significativo al menos significativo):
 - 0: Este es un bit reservado que está siempre a cero.
 - NF: Indica si este paquete se puede *fragmentar* o no (0 lo permite y 1 no lo permite).
 - MF: Nos indica si existen más fragmentos después de este (0 es el último y 1 existen más después de éste).
- *Fragment offset* (13 bits): nos indica la posición en la que empieza el contenido de este paquete, pero teniendo en cuenta que indica bloques de 64 bits. Es decir, si tuviésemos un paquete con *fragment offset* = 2 en el cómputo global ocuparía la posición a partir del bits 128, porque este paquete contiene 2 bloques de 64 bits de datos.
- *Time to live* (8 bits): Contiene el número de saltos que el paquete puede dar para llegar a su objetivo.

- *Protocol* (8 bits): Indica el protocolo de la capa de transporte, que puede ser: TCP, UDP, ICMP etc.
- *Checksum* (16 bits): Este campo se utiliza para la detección de errores en el datagrama.
- Dirección de origen (32 bits): Indica la dirección desde la que procede el paquete.
- Dirección de destino (32 bits): Indica la dirección a la que va el paquete.
- Opciones: Es opcional, no suele usarse para no sobrecargar la cabecera y los *routers*. Siguen la siguiente estructura(8 bits, del menos significativo al más significativo):
 - Número: identificador de opción
 - Clase o tipo de opción, puede ser:
 - 0: control de red
 - 2: depuración y test
 - 1 y 3: reservado para el futuro

7.3 Anexo III

El datagrama del protocolo TCP sigue la siguiente estructura (Dibujo 7.3):

Offsets	Octeto	0								1								2								3							
Octeto	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Puerto de origen																Puerto de destino															
4	32	Número de secuencia																															
8	64	Número de acuse de recibo (si ACK es establecido)																															
12	96	Longitud de Cabecera				Reservado				N	C	E	U	A	P	R	S	F	Tamaño de Ventana														
									S	W	R	E	R	G	K	S	H	T	Y	I	N												
16	128	Suma de verificación																Puntero urgente (si URG es establecido)															
20	160	Opciones (Si la Longitud de Cabecera > 5, relleno al final con "0" bytes si es necesario)																															
...																															

Dibujo 7.3

Campos del datagrama TCP:

- Puerto de origen (16 bits): Identifica el puerto emisor del paquete.
- Puerto de destino (16 bits): Identifica el puerto receptor del datagrama.

- Número de secuencia (32 bits): Indica el byte a partir de donde se debe colocar el fragmento de datos que incluye este paquete, en el inicio de la conexión se utiliza para mandar el ack.
- Número de acuse de recibo o ACK (32 bits): Contiene el valor del siguiente número de secuencia que se espera recibir.
- Longitud de la cabecera (4 bits): Se especifica en palabras de 32 bits.
- Reservado (3 bits): para uso futuro, su valor debe ser 0.
- *Flags* (9 bits, repartidos en un bits para cada uno):
 - NS: *ECN-nonce concealment protection*. Para proteger los paquetes ante cualquier paquete que intente aprovechar el control de congestión de la red, para ganar ancho de banda.
 - CWR: *Congestion Window Reduced*: Se activa cuando se ha recibido un paquete con el *flag* ECE activado y ha respondido con el mecanismo de control de congestión.
 - ECE: Contiene indicaciones sobre la congestión.
 - URG: Indica que el campo puntero urgente es válido.
 - ACK: Indica que el campo ack es válido, todos los paquete de inicio de la conexión llevarán activado este *flag*.
 - PSH: *Push*, El receptor envía los datos de una aplicación tan pronto como sea posible.
 - RST: *Reset*, se activa para reiniciar la conexión, cuando falla esta o cuando se rechazan paquetes.
 - SYN: *Synchronize*, se utiliza para sincronizar los números de secuencia al iniciar una conexión.
 - FIN: Para que el emisor libere la conexión.
- Tamaño de la ventana (16 bits): Se usa para controlar el flujo especificando el número de bytes pendientes de asentimiento.
- *Checksum* (16 bits): Se utiliza para comprobar errores en el paquete.
- Puntero urgente (16 bits): Cantidad de bytes de datos urgentes.
- Opciones: Ocupan el espacio final de la cabecera, pero tienen que tener un tamaño múltiplo de 8 bits

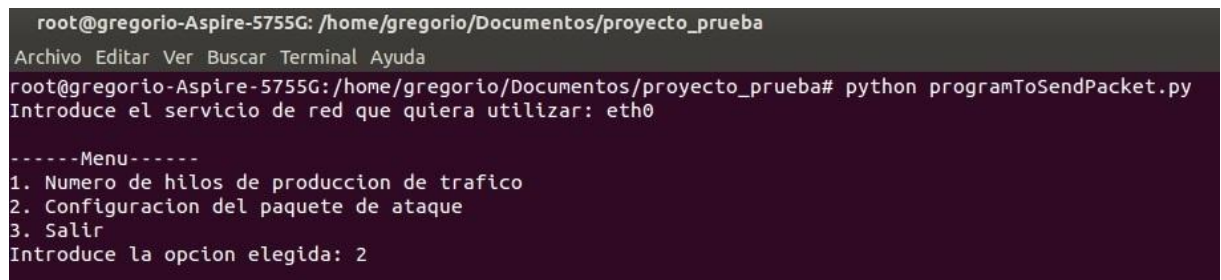
7.4 Anexo IV

En este anexo vamos a explicar el programa ***programToSendPacket***, el cual es un programa capaz de mandar los ataques que nuestro sistema es capaz de detectar, este era necesario para probar el IDS.

Al igual que el IDS, está implementado en *python* y fue ejecutado sobre una máquina Linux para realizar las pruebas.

En este punto vamos a proceder a explicar las distintas opciones que contiene este programa.

El menú se compone de las siguientes opciones:



```
root@gregorio-Aspire-5755G: /home/gregorio/Documentos/proyecto_prueba
Archivo Editar Ver Buscar Terminal Ayuda
root@gregorio-Aspire-5755G: /home/gregorio/Documentos/proyecto_prueba# python programToSendPacket.py
Introduce el servicio de red que quiera utilizar: eth0

-----Menu-----
1. Numero de hilos de produccion de trafico
2. Configuracion del paquete de ataque
3. Salir
Introduce la opcion elegida: 2
```

Imagen 7.4.1

- Número de hilos de producción de tráfico: En esta opción nos permite configurar distintos hilos de producción de tráfico con el fin de saturar la red o, dar al IDS otro tráfico que analizar aparte del peligroso.
- Configuración del paquete de ataque: Esta opción nos permite configurar el paquete de un ataque. Puede ser uno complejo usando fragmentación, o más simples como el basado en TTL.
- Salir: Esta opción cierra el programa.

Si pulsásemos sobre la opción 1 del menú, nos saldrían las siguientes opciones:

```
-----Menu de los hilos de generacion de trafico-----  
1. Configuracion de los hilos  
2. Lanzar  
3. Salir  
Introduce la opcion elegida: 1  
Introduce el numero de hilos que lanzan trafico basura: 1
```

Imagen 7.4.2

- Configuración de hilos: En esta opción configuramos un paquete que será enviado de forma repetida al sistema, también nos pide el número de hilos de tráfico que queremos generar, y el número de paquetes que envía cada hilo.
- Lanzar: Una vez configurados los hilos que queramos, podemos lanzar dichos hilos escogiendo esta acción, debemos tener en cuenta antes de lanzarlos que debemos de configurar los hilos o el hilo.
- Salir: Sale de este submenú.

Y por último, el tercer submenú es:

```
-----Menu de configuracion de ataque-----  
1. Ataque basado en TTL  
2. Ataque basado en RST  
3. Ataque basado en SYN  
4. Ataque de fragmentacion  
5. Configurar todo el paquete para atacar  
6. Salir
```

Imagen 7.4.3

- Ataque basado en TTL: Configura un ataque basado en TTL.
- Ataque basado en RST: Configura un ataque basado en RST.
- Ataque basado en SYN: Configura un ataque basado en SYN.
- Ataque de fragmentación: Configura un mensaje que utilice la fragmentación.
- Configurar todo el paquete para atacar: En esta opción tienes libertad absoluta para configurar todos los elementos de las distintas cabeceras (TCP e IP).
- Salir: Al igual que los demás submenús para salir de este.

En este punto, ya comentado las distintas opciones de los menús, vamos a realizar un análisis de diagrama de clases de este fichero. Diagrama:

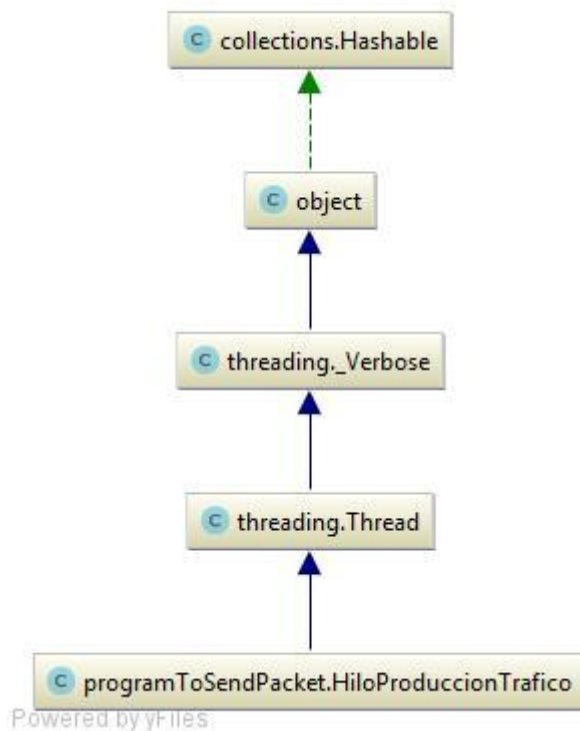


Imagen 7.4.4

Como podemos apreciar este fichero se compone de una clase llamada `HiloProduccionTrafico`, la cual se encarga de gestionar los hilos que enviaran el tráfico basura al IDS.

Las funciones de ataque no forman parte de ninguna clase, son llamadas directamente por la función *main* del fichero.

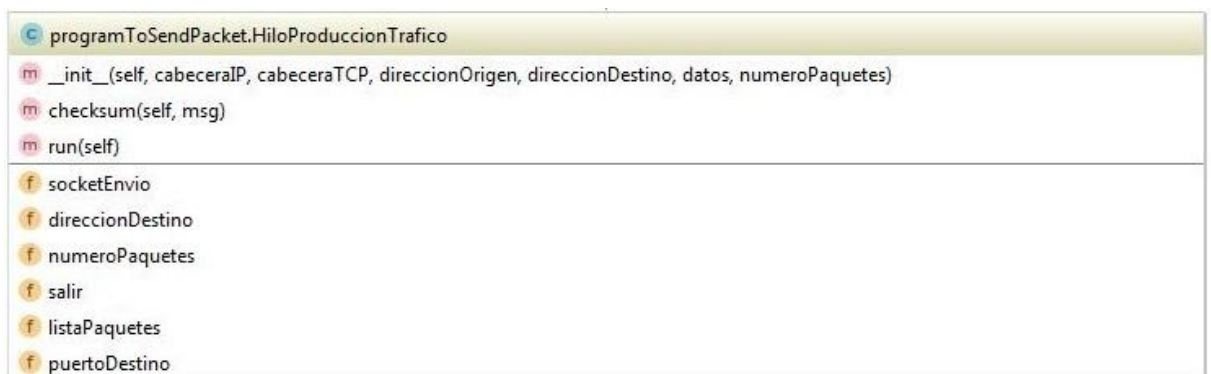


Imagen 7.4.5

Los elementos más destacables de la clase son:

- **SocketEnvio:** Esta variable almacena el *socket* la cual envía los paquetes al sistema.
- **DireccionDestino:** Ésta contiene la dirección IP a la que se le envían los datos.
- **NumeroPaquetes:** Aquí se establecen el número de paquetes que enviará la instancia del hilo.
- **ListaPaquetes:** Aquí almacenamos lo paquetes que se han generado para un posterior envío al sistema.
- **PuertoDestino:** En esta variable tenemos el puerto destino de los paquetes.

Entre los métodos, debemos destacar la función cuyo nombre es “*checksum*”, la cual se encarga de calcular el campo *checksum* de la cabecera TCP, el cálculo de dicho campo se encuentra descrito en el anexo I.

Bibliografía

José Manuel Lucena López, apuntes de la asignatura: *Técnicas avanzadas en seguridad*

Antonio Jesús Rivera Rivas, apuntes de la asignatura: *Programación y administración de redes*

Páginas web consultadas:

https://es.wikipedia.org/wiki/Transmission_Control_Protocol

https://es.wikibooks.org/wiki/Redes_inform%C3%A1ticas/Protocolos_TCP_y_UDP_en_el_nivel_de_transporte

http://notoquesmicodigo.blogspot.com.es/2013_07_01_archive.html

<http://stackoverflow.com/questions/22577327/how-to-retrieve-the-selected-row-of-a-qtableview>

<http://stackoverflow.com/questions/1807299/open-a-second-window-in-pyqt>

<https://docs.python.org/2/library/socket.html#socket.socket.shutdown>

<https://wiki.python.org/moin/PyQt/Handling%20context%20menus>

https://es.wikipedia.org/wiki/Fragmentaci%C3%B3n_IP

<http://www.pedrocarrasco.org/fragmentacion-ip/>