Departamento de computación
Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Sistema de análisis pasivo para auditar redes corporativas utilizando software libre en una Raspberrry Pi

Estudiante: Juan Carlos Otero Freijeiro
Director: Francisco Javier Nóvoa de Manuel

A Coruña, 6 de setembro de 2019.

A ti, que terminaste por comprender que la felicidad es proporcional al esfuerzo dedicado a alcanzar tus metas.

Agradecimientos

- Al profesor Francisco Nóvoa por sus consejos durante el desarrollo del proyecto, y por hacer posible esta idea.
- A mi familia, por su apoyo incondicional en todas mis decisiones.
- A mi compañera de vida, que sufre mis derrotas y celebra mis victorias como si de mi se tratase.
- A mis compañeros de carrera, que han terminado por convertirse en buenos amigos.
- A mis profesores que convirtieron mi pasión, en mi profesión.

Resumen

El la actualidad existe una creciente preocupación por la seguridad de la información en los entornos corporativos. La proliferación de *malware* y el desarrollo de técnicas sofisticadas para implementar amenazas avanzadas permanentes, ha hecho que se incremente el nivel de riesgo en las empresas, puesto que la información es su activo más importante y su robo o manipulación indebida suponen un grave problema.

El objetivo principal de este trabajo de fin de grado es el desarrollo de una herramienta sobre un hardware de bajo coste, escalable y de uso sencillo, que permita analizar la seguridad de una red utilizando técnicas no intrusivas (*passive analysis*).

El software deberá obtener la mayor cantidad de datos posible sin ser detectado, y los procesará para determinar qué información podría obtener un atacante (apoyándose en técnicas de *machine learning*). En base a los datos recogidos mediante el análisis pasivo de la red, se plantean los siguientes objetivos específicos:

- Crear un mapa de red
- Analizar la topología de la red
- Obtener el número de equipos en la red y sus posibles servicios
- Determinar sistemas operativos, servicios o programas instalados.

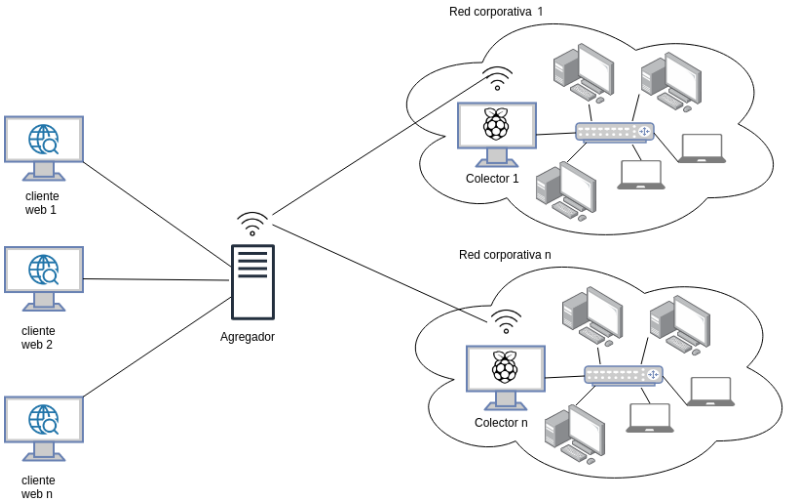

Figura 1: Arquitectura global del proyecto

En la figura 1 se muestra la arquitectura global del proyecto desarrollado en la que se puede observar como uno o varios colectores (dispositivos de escaneo) serán los encargados de realizar la captura de datos en una o varias redes corporativas; el agregador que es el servidor encargado de centralizar las capturas y servir los datos a los administradores de red a través de una interfaz Web.

Palabras clave:

- Raspberry Pi
- Análisis pasivo
- Bosque Aleatorio
- Fingerprinting
- Árbol de decisión

Keywords:

- Raspberry Pi
- Passive Analisis
- Random Forest
- Fingerprinting
- Decision Tree

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

La seguridad de la información, a nivel corporativo sigue, y seguirá siendo, una de las preocupaciones principales a tener en cuenta por una empresa. Cualquier información puede resultar útil para un atacante o para un código malicioso. Por está razón decidimos diseñar un sistema de análisis no intrusivo que permita conocer qué información se publica en la red local y así tomar medidas.

En el presente capítulo expondremos el objeto de este desarrollo. También estudiaremos las diferentes soluciones similares existentes. A continuación seguiremos con la definición de los objetivos que se pretenden cumplir y la exposición de la estructura de esta memoria, será el paso final.

1.1 Motivación

Han pasado mas de 400 años desde que el barón Sir Francis Bacon[1] acuñara la expresión «la información es poder» y, sin embargo, ha sido durante estos últimos años cuando esta frase ha cobrado mayor sentido. Hoy día la ingente cantidad de información que circula en una red corporativa es abrumadora, siendo una ardua tarea evitar que se filtre al exterior información sensible. La obtención de estos datos provoca que la organización sea susceptible a un ataque dirigido. De esta forma, tomando como referencia la cita inicial, a mayor cantidad de información recopilada, mayor será la capacidad de ataque a una infraestructura.[2]

Ante esta situación, las empresas pueden optar por la contratación de diferentes servicios de seguridad o aumentar su plantilla con expertos en la materia. Incluso si dispone del presupuesto necesario se podría reforzar la seguridad con dispositivos *hardware* y *software* específicos.

Todas las opciones citadas anteriormente, aunque útiles, pueden no ser viables a nivel económico para pequeñas y medianas empresas. Por este motivo, principalmente, estas se convierten en los objetivos predilectos de diversas organizaciones cibercriminales.

[1] Filósofo y político inglés, famoso por considerar que la verdad solo podía ser alcanzada por medio del conocimiento

Según datos ofrecidos por la empresa de seguridad «Eset»[3][4], casi una de cada cinco PYMES fue víctima de una amenaza de seguridad externa en los últimos dos años.

Por todo lo expuesto anteriormente, nace la motivación de diseñar esta herramienta, de bajo coste, eficiente, escalable e innovadora por el formato utilizado para auditar el tráfico.

Todas estas características hacen que sea una herramienta perfecta para medianas o pequeñas empresas, pudiendo utilizarse perfectamente en empresas mayores; debido a su diseño escalable solo sería necesario aumentar el número de dispositivos (Raspberrys Pi) repartiéndolos por ejemplo por subred.

1.2 Objetivos Concretos

Teniendo en cuenta la motivación descrita en el apartado 1.1 este trabajo de fin de grado busca desarrollar una herramienta para la monitorización y auditoría de redes corporativas valiéndose para ese cometido de técnicas no intrusivas (técnicas que tratan de analizar tráfico de red legítimo sin interferir en él) para la obtención de datos, y de la utilización de inteligencia artificial (Nos centraremos en el tipo de aprendizaje supervisado pues nuestro objetivo es clasificar sistemas en función de los datos obtenidos) para ayudarnos durante la explotación, siendo escalable y de un coste reducido.

Para considerar este desarrollo como exitoso nos proponemos cumplir los siguientes objetivos:

1.2.1 Creación de mapa de red y análisis de la topología

A partir de los datos que el recolector (o recolectores) haya obtenido, se propone crear un mapa de red donde se representen los nodos (puntos sobre un plano) identificados por IP (*Internet Protocol*) o por MAC (*Medida Access Control*). Además utilizando el campo de la cabecera IP, dirección de destino, estableceremos las conexiones entre los nodos vecinos.

Estas conexiones se representarán en el mapa de red como aristas entre nodos. Teniendo en cuenta el tráfico de red con el que tratamos (normalmente, tráfico de *broadcast* y *multicast*) crear estas conexiones será una tarea compleja al no conocer con certeza el destinatario del paquete. Por este motivo nos centraremos en la paquetería con dirección destino existente en el mapa representado.

Buscaremos representar toda la información posible por cada nodo de la red, obteniendo dirección IP (dirección origen), y direcciones de destino a las que ese *host* envió paquetería. Dirección MAC obtenida del protocolo de red Ethernet (identificador único de red) y el posible sistema operativo (se obtendrán utilizando los campos de la cabecera IP, TTL y DF comentados en capítulos posteriores).

1.2.2 Obtención del número de equipos

Estableceremos contadores en la aplicación web que nos permitan conocer a simple vista cuantos equipos se han descubierto en la red y cuantos se han logrado clasificar por sistema operativo. Para esta clasificación nos basaremos únicamente en dos campos de la cabecera IP (*time to live* y bit de *don't fragment*).

1.2.3 Determinar por equipo el sistema operativo y posibles servicios

Sobre cada representación de equipo se mostrará su dirección IP que será un enlace a una página con un mayor nivel de detalle en la información de cada nodo. En esta nueva página se mostrará el fabricante de la tarjeta de red, puertos y sus posibles servicios (estos servicios se obtienen de IANA [5] y su base de datos de puertos conocidos) y el sistema operativo que esta vez obtendremos utilizando el algorítmo clasificatorio *Random Forest*.

En este último apartado mostraremos la predicción realizada por el algorítmo y su probabilidad así como su nivel de acierto representado en tanto por ciento.

1.3 Estructura de la memoria

En el siguiente apartado describiremos la estructura seguida durante el desarrollo de la memoria:

- **Introducción y Objetivos concretos:** en este capítulo se detalla la motivación de este trabajo, el estado del arte y los objetivos concretos a cumplir.
- **Fundamentos :** se dividirán en teóricos explicando las técnicas utilizadas y tecnológicos donde se hablará del *hardware*.
- **Metodología :** metodología empleada en la realización del proyecto.
- **Resultados obtenidos :** descripción detallada de la aplicación siguiendo cada fase de la metodología utilizada.
- **Conclusiones :** se comentan los resultados y hasta donde se ha podido llegar, en los objetivos especificados.
- **Discusión :** se analiza el trabajo realizado y se proponen posibles líneas de trabajo futuro.
- **Apéndices :** Se incluyen los manuales de instalación, tanto del cliente como del servidor, así como un pequeño manual de usuario.

Capítulo 2
Fundamentos

En los últimos años tanto el hardware como el software, dedicado a la seguridad de la información a evolucionado muy rápidamente. En general este tipo de productos suelen tener precios elevados, dejándolos fuera del alcance a empresas más modestas. Uno de nuestros objetivos es lograr un producto de bajo coste, que sea potente y a la vez escalable.

2.1 Fundamentos teóricos

Para llegar a tener un buen entendimiento del funcionamiento de este desarrollo, es necesario tener en cuenta una serie de conceptos teóricos sobre redes TCP/IP y distintos protocolos de red utilizados.

El modelo TCP/IP es una familia de protocolos de red que permiten la comunicación entre máquinas que pueden estar ubicadas en distintas redes. Este modelo se encuentra formado por dos protocolos de red distintos:

- Protocolo IP (*Internet Protocol*) [6] : es el protocolo que hace posible la comunicación entre distintas máquinas. La paquetería IP contiene la información a transmitir y la dirección del destino (como si se tratase de una carta). Cada máquina tendrá una tarjeta de red con una dirección IP, gracias a ella, sabremos a quien enviar la información (dirección receptora).
- Protocolo TCP (*Transmission Control Protocol*) [7] : el protocolo TCP se encarga de transportar la paquetería IP asegurando que esta no se pierda por el camino (siguiendo con la analogía de las cartas, este protocolo sería nuestro cartero). Nos ofrece seguridad en la transmisión de datos, en el caso de que la información se fragmente por ser demasiado grande, nos asegura que se mantendrá el orden de secuencia.

Además de TCP/IP, utilizaremos otros protocolos (UDP,Ethernet...) que definiremos en secciones próximas.

2.1.1 Análisis de tráfico de red

Es el proceso de interceptar, registrar y analizar patrones de comunicación del tráfico de red con la finalidad de de responder ante amenazas de seguridad. [8]

2.1.1.1 Análisis activo

Como se introdujo en el apartado 2.3.1, el análisis activo se basa en el envío de paquetes de red con el objetivo de analizar la respuesta de las máquinas, siendo esta suficientes para identificar el sistema operativo. Como ventaja principal en este tipo de análisis tenemos la obtención de resultados precisos en poco tiempo. Sin embargo es un tipo de análisis ”ruidoso” fácilmente detectable y bloqueable por dispositivos diseñados para este cometido (IDS: Sistema Identificador de Intrusos).

2.1.1.2 Análisis pasivo

Para este tipo de análisis se realiza la captura del tráfico de la red objetivo de forma pasiva, es decir, sin realizar ningún tipo de interacción sobre la misma. Este será el tipo de análisis empleado por nuestra herramienta con el fin de cumplir con uno de los objetivos principales propuesto tanto en el anteproyecto como en el resumen de este trabajo , evitar ser detectado en la red objetivo.

El tráfico que se logra capturar utilizando esta técnica es más limitado que en el caso de un análisis activo, pero como beneficio principal nos ofrece un alto grado de discreción. En general el tráfico con el que trabajaremos es de tipo *Broadcast* o *Multicast*, por lo que nos centraremos en los protocolos de red que utilizan este tipo de direccionamientos.

La técnica utilizada en la fase de explotación será *Passive OS Fingerpriting*[9], técnica no intrusiva destinada al descubrimiento del sistema operativo que se ejecuta en cada máquina de la red objetivo. Utilizamos la herramienta Wireshark para determinar qué protocolos y opciones existentes en las cabeceras de estos son los más útiles en esta tarea.

2.1.1.2.1 Protocolo IP

Protocolo de comunicación perteneciente a la capa de red (capa 3 según la pila OSI), que permite la transmisión de datos a través de distintas redes físicas. Este es un protocolo no orientado a conexión, es decir, no nos asegura la entrega del dato en el destino (se utiliza conjuntamente con TCP para cumplir con esta premisa).

0-3	4-7	8-15	16-18	19-31
Versión	Tamaño Cabecera	Tipo de Servicio	Longitud Total	
Identificador			Flags	Posición de Fragmento
Tiempo de Vida		Protocolo	Suma de Control de Cabecera	
Dirección IP de Origen				
Dirección IP de Destino				
Opciones				Relleno

Figura 2.1: Formato de la cabecera IP versión 4 y campos utilizados

En la figura 2.1 se muestra los campos de la cabecera del protocolo IP, los bits que utiliza cada uno y se han marcado aquellos que se consideraron más relevantes a la hora de obtener una firma distintiva para cada sistema operativo.

- **Bit Don't Fragment (DF) :** bit encargado de indicar si esta permitida la fragmentación del datagrama (Paquete de datos que constituye el mínimo bloque de información en una red conmutada).

  Este bit se encuentra en el campo *Flags* indicado en la imagen 2.1, y está formado por 3 bits de control, siendo el primero un bit reservado que siempre tienen el valor de 0, el segundo es el bit de *don't fragment* que puede tomar el valor de 1 en el caso de que se encuentre establecido o 0, y el tercero indica si es el último fragmento cuando se encuentra establecido a 0 o si existen más fragmentos cuando su valor es 1 (como es lógico si el bit DF se encuentra establecido a 1, el último bit, *more fragments*, siempre se encontrará a 0).

```
Flags: 0x4000, Don't fragment
    0... .... .... .... = Reserved bit: Not set
    .1.. .... .... .... = Don't fragment: Set
    ..0. .... .... .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0
```

Figura 2.2: Flags de la cabecera IP

Muchos sistemas (Windows 7, 10, XP) comienzan estableciendo el bit de *Don't fragment* en el primer paquete *sync* enviado durante una comunicación en cambio en otros casos (Debian 9, Android 8, Raspbian 4.14) este bit se encuentra deshabilitado. Esta divergencia entre los sistemas que si lo utilizan y los que no, resulta muy útil para nuestra herramienta.

- **Tiempo de vida (TTL) :** este campo establece el número máximo de saltos (routers por los que pasa un paquete) que un paquete puede dar antes de ser descartado (el valor se va disminuyendo por cada salto dado). Esta medida previene que un paquete se quede en bucle en la red sin encontrar destino. El valor varía en función del sistema operativo que envía el paquete, por este motivo se considera uno de los campos más útiles en nuestra aplicación.

Sistema Operativo	TTL
Linux kernel 2.4 - 2.6, 3.1-3.10, 3.11 - nuevas versiones	64
Windows XP, 7, 8, NT	128
Mac - Unix	64
FreeBSD 8, 9	64
OpenBSD 3, 4-5	64
Solaris 6	255

Figura 2.3: Tabla de correspondencia entre sistema operativo y TTL

- **Dirección origen y destino :** como su propio nombre indica, en estos campos se almacenan la dirección ip de origen, es decir quien es el emisor del paquete, y la dirección ip destino, a quien va dirigido el paquete. Se utilizarán estas direcciones para identificar gráficamente los nodos y con quienes se comunican.

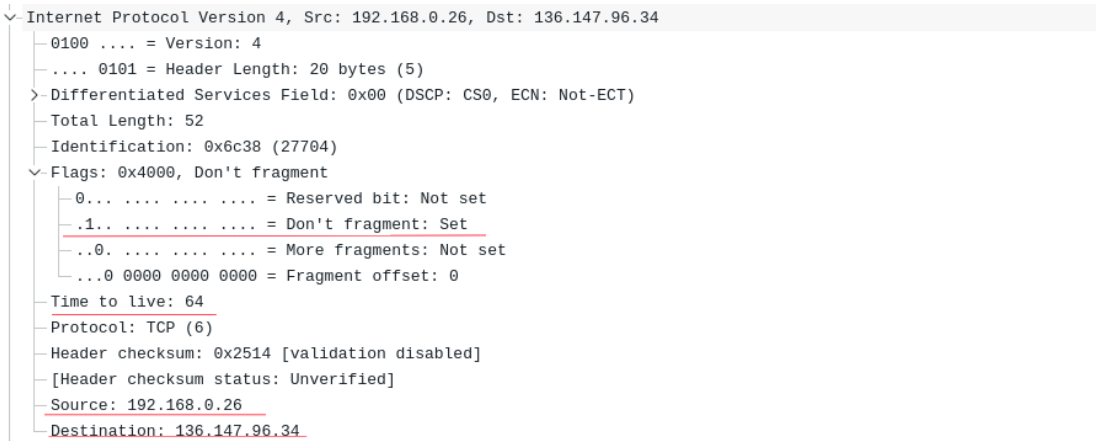

Figura 2.4: Campos de la cabecera IP en una captura de ejemplo

En la imagen 2.4 se observa que no se permite la fragmentación del paquete y que además posee un TTL de 64 indicando de esta forma que probablemente se trate de un sistema Linux. También se muestra la IP origen, que es privada tipo C y la IP destino que en este caso se trata de una IP pública, indicando la comunicación de este host con el exterior.

2.1.1.2.2 Protocolo Ethernet

Este protocolo opera en las capas físicas y de enlace (capas 1 y 2 en la pila OSI), y es la tecnología LAN mas utilizada en el mundo [10] [**?**]. Si nos centramos en la capa de enlace podemos identificar dos subcapas:

1. Control de enlace lógico (LLC): su función principal es la de comunicarse con las capas superiores en la pila y así poder añadir información de la capa de red (generalmente paquetería IPv4) ayudando en el envío del paquete a su destino.
2. MAC (*Media Access Control*): esta subcapa se implemente mediante hardware, por lo general en la NIC (*Network Interfaz Card*) del propio nodo. Su función principal es la encapsulación de datos, construyendo tramas para su envío y desmontandolas a su recepción.

Preámbulo	Delimitador de inicio de trama	MAC de destino	MAC de origen	802.1Q Etiqueta(opcional)	Ethertype (Ethernet II) o longitud (IEEE 802.3)	Payload	Secuencia de comprobación (32-bit CRC)
7 Bytes	1 Byte	6 Byte	6 Bytes	(4 Bytes)	2 Bytes	De 46 (o 42) hasta 1500 Bytes	4 Bytes

Figura 2.5: Estructura de la trama de 802.3 Ethernet

Como se muestra en la imagen 2.5 una trama o *frame* Ethernet está compuesto por 7 campos, dejando al margen el campo Etiqueta por ser opcional y solo utilizado por el protocolo 802.1Q. De todos estos valores el que se considera más interesante para el desarrollo y por tanto el que se va a almacenar, es la dirección MAC origen, indicado con un círculo en la imagen. Esta dirección es un campo de de 48 bits expresado como 12 dígitos hexadecimales que identifica de forma única un dispositivo de red o tarjeta. En el apartado 2.2.2.6.3, se indica la herramienta utilizada para explotar este dato y el valor que aporta al desarrollo.

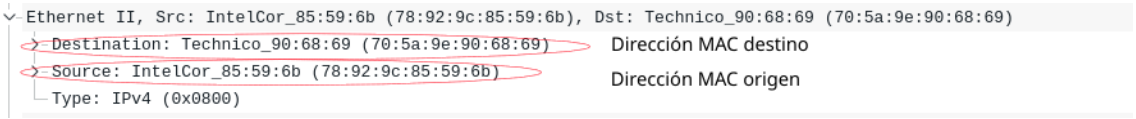

Figura 2.6: Ejemplo del protocolo Ethernet capturado con la herramienta Wireshark

2.1.1.2.3 Protocolo TCP

TCP (*Transmission Control Protocol*), es un protocolo de capa 4, es decir, es un protocolo de transporte que garantiza que los datos enviados llegan a su destino sin errores.

En el caso de nuestra herramienta, el protocolo TCP es el más valioso en cuanto a información utilizada para el reconocimiento de sistemas operativos. Combinando la información contenida en ellos podemos obtener firmas características para cada sistema.

- **Puertos origen y destino :** tanto el puerto origen como el puerto destino, incluidos en la cabecera TCP en dos campos de 16 bits cada uno, serán utilizados por nuestra herramienta para identificar posibles servicios que se encuentren en ejecución en la máquina.

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				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Tamaño de Ventana														
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)																															
...	...	...																															

Figura 2.7: Formato de la cabecera TCP y campos seleccionados

Para la realización de esta tarea, se comparan con una lista de puertos comunes ofrecida por la IANA.

- **Tamaño de vetana (*Window size*) :** campo de 16 bits que define la capacidad de datos (se calcula en bytes tras ser multiplicado por el factor escalar de ventana, se comenta posteriormente) que un receptor es capaz de admitir sin enviar confirmación.

Sistema Operativo	Tamaño de ventana TCP
Linux Debian 9, Centos 7 y Arch 4.14	29.200
Windows XP, NT, 10, Mac OS y FreeBSD	65.535
Linux (kernel 2.4 - 2.6)	5.580
Windows 7, 8	8.192
OpenBSD	16.384
Solaris	32.850

Figura 2.8: Relación entre tamaño de ventana y sistema operativo [1]

Como se puede observar en la imagen 2.8 cada sistema operativo utiliza por defecto un tamaño de ventana TCP específico (obtenido de paquetes SYN), por lo tanto, se tendrá en cuenta para explotarlo *a posteriori*.

- **Campo *Options* :** este campo también existe en la cabecera IP pero se utiliza raramente, en cambio en TCP se utiliza con frecuencia, ocupando hasta el final de la cabecera TCP y siendo su tamaño variable. No todos los sistemas operativos soportan las mismas opciones TCP y cada uno las ordena de forma diferentes.

– ***Maximum Segment Size* (MSS) :** esta opción se utiliza para informar al receptor de la comunicación la MTU (Unidad Máxima de Transferencia) del emisor. Es decir, con esta opción se indica al otro lado de la comunicación el tamaño máximo de datagrama IP que puede enviar a través del enlace al que ambos se encuentran conectados, sin tener que fragmentarlo. El valor de MMS es igual al de MTU menos los tamaños de las cabeceras IP y TCP.

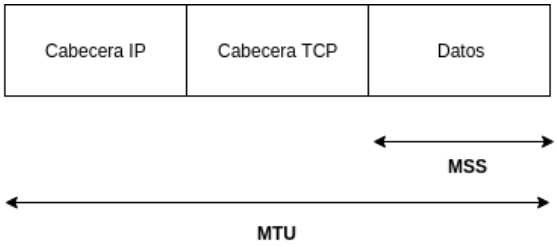

Figura 2.9: Esquema de MTU y MMS

Tecnología de red	MTU (bytes)
Hyperchannel	65535
16 Mbits/sec token ring (IBM)	17914
4 Mbits/sec token ring (IEEE 802.5)	4464
FDDI	4352
Ethernet	1500
IEEE 802.3/802.2	1492
X.25	576
Punto-a-Punto (bajo retraso)	296

Figura 2.10: MTUs y la tecnología de red utilizada

Aunque el valor de MSS depende en gran medida de la tecnología a la que se encuentra conectado el nodo (imagen 2.10), varios sistemas operativos (Windows, Linux, OpenBSD, Mac OS X) calculan su valor de distintas formas [11], siendo un campo interesante para nuestra herramienta.

– ***No Operation* (NOP) :** este campo se utiliza únicamente para establecer un espacio entre las otras opciones. Cada sistema operativo varía la posición y repetición de esta opción situándola entre el resto de opciones de la cabecera TCP, siendo de utilidad para la creación de la firma de cada sistema [11].

– ***Window Scale* (WSCALE) :** como comentamos en el apartado "Tamaño de ventana", este campo es de 16 bits, es decir, tiene un tamaño máximo de $2^{16} = 65.535$ bytes.

El campo WSCALE nace para permitir a un *host* anunciar un tamaño mayor de ventana.

Cuando se utiliza este campo, se indica que TCP esta preparado para realizar el escalado de ventana tanto en envío como en recepción y también comunica el valor de escala que será aplicado en la ventana de recepción.

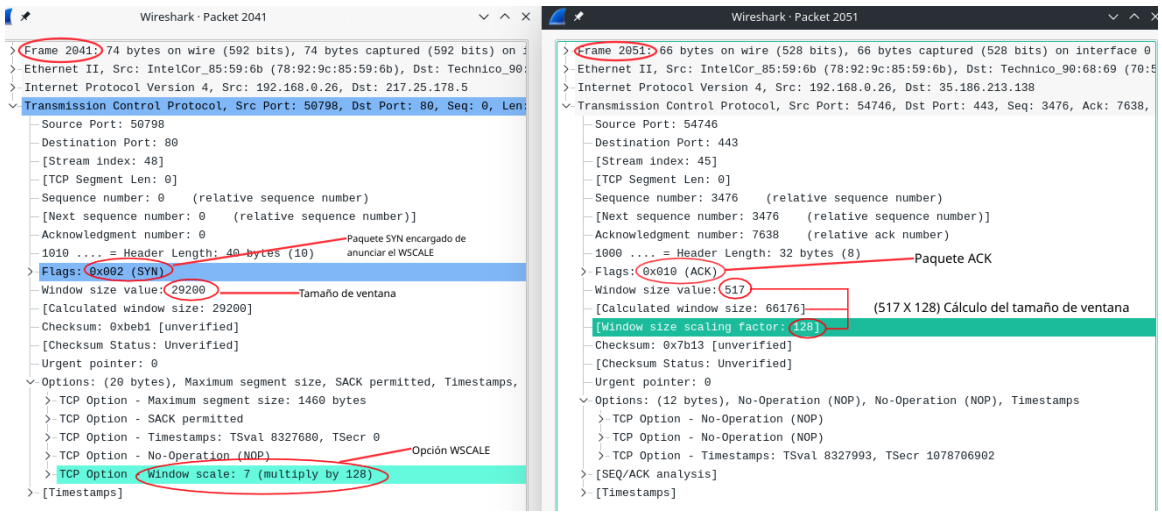

Figura 2.11: Ejemplo de funcionamiento de WSCALE

Como se muestra en la imagen anterior 2.11, la opción *wscale* se establece en paquetes SYN; este es el primer paquete en la comunicación y el encargado de anunciar el uso de la opción, en este caso con un valor de escala de 7bits ($2^7 = 128$). Los siguientes paquetes (ACK) utilizan el factor de escala para multiplicarlo por su valor de ventana y obtener así un tamaño de ventana mayor. En la imagen de ejemplo se muestra como se realiza este producto, $517 * 128 = 66.176$.

Al igual que con la opción MSS esta, no es una opción que utilicen todos los sistemas operativos por lo que nos sirve de igual forma para realizar esta clasificación.

- **Opción *Timestamp* (TS) :** el propósito principal de esta opción es el de estimar el *Round Trip Time* (Tiempo de Viaje, RRT) para identificar cambios en la latencia e identificar situaciones que requieran ajustar el reloj.
  Esta opción a diferencia de MSS o WSCALE es muy común en todos los sistemas, encontrándose formada por dos campos, *Timestamp Value* (TSval) que contiene el valor actual del reloj enviado en la opción TCP y *echo reply Timestamp* que contiene el timestamp más reciente recibido y que solo es válido si el bit ACK se encuentra activo en la cabecera TCP.

  Solo nos centraremos en el primer campo (TSval), en el que cada sistema hace incrementos del reloj de forma distinta. Por ejemplo los sitemas BSD actualizan su reloj cada 500 ms, mientras que los sistemas Linux lo actualizan con mayor frecuencia (cada 10 ms o 1 ms dependiendo del kernel).

El campo *Timestamp* en sistemas Windows no sigue un patrón pero nos sirve igualmente observando la posición que ocupa en el campo *Options*, por lo tanto también será utilizado en la detección del sistema operativo.

- **Opción Sack (*Selective Acknowledgment Data*) [12]:** durante una comunicación TCP los paquetes recibidos deben seguir un orden, en el caso de perdida de paquete el receptor se queda a la espera de que el emisor vuelva a retransmitir el paquete perdido. Con la opción Sack se soluciona este inconveniente, haciendo que el receptor reconozca todos los paquetes al inicio de la comunicación (*handshake*). Es durante este inicio de la comunicación cuando se acuerda entre ambos extremos si se va a utilizar esta opción. Algunos sistemas operativos como OpenBSD, AIX 4.3, os sistemas MAC OS X, no utilizan esta opción y los sistemas que lo utilizan la sitúan en distinta posición en el campo *Options*, por eso la tendremos en cuenta durante el análisis, por ser una característica diferenciadora del sistema.

- **El orden seguido por las opciones (optSort) :** este no es un campo que se encuentre incluido en la cabecera TCP, pero si es un dato importante para la herramienta porque cada sistema operativo establece un orden (posicionamiento) de sus opciones que se puede aprovechar para identificarlos. Se almacenará este campo igual que los comentados anteriormente.

2.1.1.2.4 Protocolo UDP

Como TCP, UDP es un protocolo de la capa de transporte pero que a diferencia de él no nos asegura que los paquetes hayan sido entregados a su destino. La ventaja de UDP frente a TCP es su rapidez gracias a la ausencia de comprobaciones.

Para este protocolo solo se almacenarán aquellos datos en texto claro que se transmitan pudiendo ser útiles para la identificación de servicios que se ejecutan en la máquina (por ejemplo imagen 2.12).

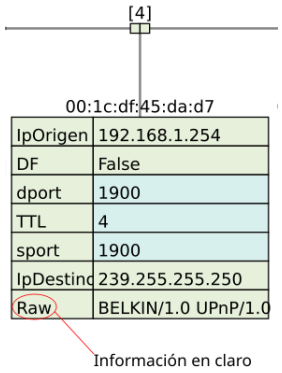

Figura 2.12: Ejemplo de captura con campos UDP realizada por la herramienta

2.1.2 Clasificación del tráfico

2.1.2.1 Técnicas de *Machine Learning*

El objetivo principal durante la explotación de los datos obtenidos es conocer las vulnerabilidades de los nodos analizados. Según la revista estadounidense 'Cybersecurity Magazine' [13] en el top 10 de los riesgos en ciberseguridad se encuentran

- Ataques utilizando *Malware*: hacen referencia a la utilización de *software* malicioso que trata de infectar un ordenador o dispositivo móvil con de extraer información personal, contraseñas, robar dinero o denegar el acceso al usuario legítimo.
- Explotación de vulnerabilidades: hacer referencia al aprovechamiento de fallas de seguridad conocidas (o no conocidas) en sistemas informáticos con la finalidad de obtener acceso al sistema y por tanto a información o datos privados.

Para cada uno de estos riesgos el conocimiento del sistema operativo es imperativo para lograr su objetivo.

Tras la aproximación inicial de descubrimiento del sistema operativo observando y comparando de que forma se configuraba la paquetería de cada nodo, se plantea llevar esta idea un paso más allá, buscando lograr mejores resultados.

Se propone la obtención del sistema operativo mediante el uso de técnicas de *Machine Learning* (de ahora en adelante ML). Este concepto engloba un conjunto de técnicas que permiten a una máquina realizar tareas propias de un humano. Las técnicas que se utilizarán se basan en el paradigma "Aprender de la experiencia", es decir, necesitarán recibir una gran cantidad de información para poder "entrenarse" y así lograr resultados satisfactorios. [14]

Podemos clasificar ML en dos tipos de aprendizaje(existen más pero estos son los más útilizados):

- **Aprendizaje no supervisado:** este tipo de aprendizaje no posee un conocimiento inicial y por tanto no se pueden instruir los algoritmos para basar su respuesta en datos de entrenamiento. Este tipo de aprendizaje agrupa los datos en conjuntos que no se encuentran etiquetados en función de sus características. Al no poseer etiquetas, es difícil evaluar el resultado.
  - ***Clustering* :** como en todo problema de este tipo, el objetivo es encontrar una estructura en una colección de datos no etiquetada. El *clustering* trata de agrupar datos en función de sus similitudes.
- **Aprendizaje supervisado :** técnica utilizada para deducir una función a partir de unos datos de entrenamiento. El objetivo de este tipo de aprendizaje es el de crear una función capaz de evaluar los datos de entrada y por tanto predecir la solución correspondiente en función de unos datos de entrenamiento.

Cubre situaciones no vistas previamente por eso es importante tener un buen conjunto de datos de entrenamiento para ser lo más preciso posible.

* **Clasificación :** los algoritmos de este tipo son útiles cuando el resultado o respuesta a la pregunta planteada, tiene un conjunto de valores finito. De esta forma se evalúa la entrada dada, utilizando el conjunto de datos de entrenamiento y se clasifica en uno de estos conjuntos.
* **Regresión :** este tipo de algorítmos busca establecer un modelo para un cierto número de características y una variable continua. Es decir, se busca obtener una respuesta cuantitativa al problema planteado (por ejemplo predicciones de ventas).

En el ámbito de este desarrollo nos centraremos en el aprendizaje supervisado, pues el problema que queremos tratar es de tipo clasificatorio. Necesitaremos que el algoritmo sea capaz de clasificar un valor de entrada, (paquetería de red), en función de unos parámetros y darnos un resultado que formará parte de un conjunto de valores finitos, siendo estos valores posibles los sistemas operativos incluidos en los datos de entrenamiento.

Este tipo de aprendizaje recibe un gran número de datos de test intentando cubrir todas las opciones posibles, de esta forma, cuando nuestro algoritmo clasificatorio reciba un nuevo caso, aunque no exista explícitamente en nuestro *dataset* (conjunto de datos), podrá determinar que opción se ajusta más a dichos datos.

2.1.2.1.1 Árboles de decisión

Para el aprendizaje de nuestro algoritmo clasificatorio se utilizarán árboles de decisión. Este es uno de los algoritmos de aprendizaje supervisado más utilizado en ML, pudiendo realizar tareas de regresión o clasificación. En este último caso, en el que la variable de salida tiene un conjunto finito de soluciones posibles se denominan áboles de clasificación.

Los árboles de decisión se encuentran formados por un primer nodo raíz del que salen dos ramas (podrían ser más, pero nos vamos a centrar en árboles binarios) donde se plantea una condición que pude ser cierta o falsa, se toma una de las dos opciones y el proceso se repite para una nueva condición dependiente de las anteriormente tomadas. Finalmente se llega al valor resultado que en el árbol se conoce como nodo hoja.

Se muestra un ejemplo reducido de árbol de decisión en la figura 2.13 que toma como parámetros el TTL y DF comentados en la tabla 4.16

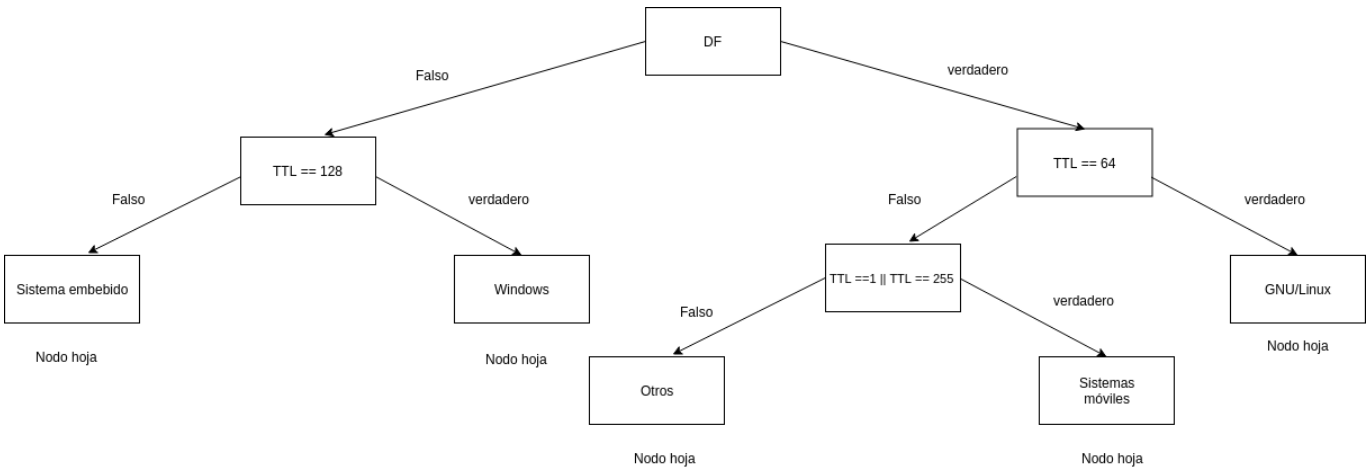

Figura 2.13: Ejemplo de árbol de decisión con atributos DF y TTL

El algoritmo deberá ser capaz de medir de alguna forma las predicciones obtenidas y darles un valor para poder realizar una comparativa. Para dichas tareas utiliza varias funciones, siendo las más conocidas:

- **El índice Gini :** esta función se encarga de medir el grado de impureza de los nodos del árbol, es decir, cuan desordenados quedan los nodos una vez divididos. Se utiliza para atributos con valores continuos y el objetivo principal es minimizarlo pues cuanto mayor sea este indice, mayor será la desigualdad entre ocurrencias produciendo un incorrecto etiquetado. [**?**]
- **La ganancia de información :** mediante este criterio se pretende estimar la cantidad de información aportada por cada atributo categórico (atributos clasificables en categorías, por ejemplo, hombre/mujer). Se utiliza también el concepto de entropía que mide la incertidumbre, y se utiliza para decidir que atributo será el siguiente en ser seleccionado. [**?**]

2.1.2.1.2 Creación del conjunto de datos de entrenamiento

Generaremos un *dataset* que se utilizará como el conjunto de datos de entrenamiento para el algoritmo. Este *dataset* estará formado por los atributos comentados en la sección 2.1.1.2 pero con una nueva columna, ”Sistema operativo” (SO) indicando en cada una de las ocurrencias a que sistema pertenece.

DF	MSS	SakOk	Ts	NOP	Ws.Options	TTL	Wsize	SO
Don't Fragment	Maximum Segment Size	Selective Acknowledgment	Timestamp	Not Operation	Window Scale	Time To Live	Window Size	Sistema Operativo

Figura 2.14: Columnas que conforman el dataset de entrenamiento

Cada una de las columnas que se muestran en la imagen anterior 2.14 (menos SO que será nuestra variable resultado) deben pasar por un proceso de categorización previo para ser evaluadas por el algoritmo. A cada una de las columnas se le asignará un valor numérico dependiendo del contenido de estas o si la opción existe en la ocurrencia evaluada o no.

  - **DF :** opción evaluada de forma booleana, toma los valores 0 cuando es ”False” y 1 cuando es ”True”
  - **TTL :** a esta opción se le asigna un valor numérico en el caso de contener:
    * 64 : se categoriza con el 0
    * 128 : se categoriza con el 1
    * 255 : se categoriza con el 2
    * 1 : se categoriza con el 3

    En el caso de que contenga un valor distinto de los listados, se añadirá una nueva entrada consecutiva de forma automática al diccionario.

- **MSS :** esta opción si no existe en la ocurrencia se categoriza con el valor de 0, en el caso de contener cualquier otro valor, se añadirá una nueva entrada consecutiva al diccionario.
- **SackOk :** opción evaluada de forma booleana, toma los valores 0 cuando no existe y 1 cuando se encuentra contenida en la ocurrencia.
- **Ts (*TimeStamp*):** esta opción si no existe en la ocurrencia se categoriza con el valor de 0, en el caso de contener cualquier otro valor, se añadirá una nueva entrada consecutiva al diccionario.
- **NOP :** opción evaluada de forma booleana, toma los valores 0 cuando no existe y 1 cuando se encuentra contenida en la ocurrencia.
- **WScale :** esta opción si no existe en la ocurrencia se categoriza con el valor de 0, en el caso de contener cualquier otro valor, se añadirá una nueva entrada consecutiva al diccionario.
- **Wsize :** a esta opción se le asignan los siguientes valores numéricos en el caso de contener:
  * 1.444 : se categoriza con el 0
  * 292 : se categoriza con el 1
  * 879 : se categoriza con el 2
  * 350 : se categoriza con el 3

  En el caso de que contenga un valor distinto de los listados, se añadirá una nueva entrada consecutiva de forma automática al diccionario.

Una vez categorizado el conjunto de datos, lo dividiremos en dos bloques que serán, el conjunto de entrenamiento que estará formado por la mayoría de los datos (el 70%), siendo la base de aprendizaje de nuestros modelos (aprenden a entender las relaciones entre los datos). Y el segundo conjunto de datos será el conjunto de validación (30%), porque se utilizará para comprobar que con el entrenamiento realizado, el algoritmo es capaz de ofrecer predicciones acertadas con datos que no ha tratado antes.

2.1.2.1.3 Algoritmo "Random Forest"

Existe un algoritmo que mejora la precisión en la clasificación con respecto a la del árbol de decisión simple, se conoce como *Random Forest*. Esta técnica consiste en combinar varios arboles de decisión simples sobre muestras aleatorias con reemplazamiento (se selecciona una muestra de datos al azar, se extrae del conjunto y tras operar con ella la volvemos a dejar en el conjunto sin modificación) de los datos, se obtiene una predicción de cada uno y se escoge la mejor solución mediante votación.[15][14][16]

Para lograr comprender este algoritmo, se va a detallar paso a paso como funciona:

1. Selecciona una muestra de forma aleatoria en el dataset (conjunto de datos de entrenamiento). Como ya se ha comentado, la selección aleatoria de la muestra se hace con reemplazamiento, ofreciendo la posibilidad de volver a utilizar los mismos o parte de los datos seleccionados esa vez.
2. Se genera un árbol de decisión por cada muestra tomada y se obtiene una predicción por cada uno. La generación de los árboles de decisión se realizan al azar ofreciendo distintos resultados.
3. Se realiza una votación por cada predicción obtenida, realizando una media de los resultados (se queda con el de mayor número de coincidencias).
4. Finalmente se escoge la predicción con mayor votación.

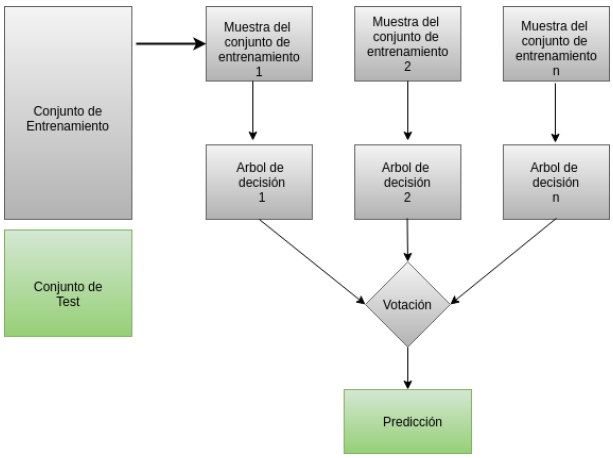

Figura 2.15: Esquema funcional del algoritmo Random Forest

1. Ventajas :
   - Se considera un método muy preciso y robusto gracias a la gran cantidad de arboles de decisión participantes
   - No se observa el problema del sobreajuste (provoca que la máquina falle la predicción por no poseer estrictamente los mismos valores que las muestras de entrenamiento) porque toma el promedio de todas las predicciones.
   - Puedes ser utilizado en problemas de clasificación y de regresión

- Puede obtener la importancia relativa de las características, ayudando a seleccionar las más relevantes.

2. Desventajas :

- Es un algoritmo lento en la generación de las predicciones porque necesita tener múltiples árboles de decisión.
- En comparación con el modelo de árbol de decisión simple este es más difícil de interpretar.

Como *"Random Forests"* aporta más ventajas que desventajas a esta parte del desarrollo, será el algoritmo clasificatorio que se utilice.

2.1.2.1.3.1 Buscando la importancia de las características

"Random Forests" ofrece un buen indicador para la selección y priorización de las características analizadas. Utilizando la librería Scikit-learn, podemos mostrar la importancia relativa o como contribuye cada característica en la predicción. El método *feature_importances* del algoritmo *"Random Forest"* incluído en la librería Scikit-learn realizará el cálculo de la puntuación de relevancia automáticamente de cada característica durante la fase de entrenamiento.

"Random Forest" utiliza el valor de Gini, asignado a cada característica para realizar el cálculo de valoración sobre cada una. Este valor indica la disminución del ajuste o precisión de la característica en el modelo, por eso siempre se busca reducir este valor lo máximo posible. Cuanto mayor es la disminución, más significativa es la variable.

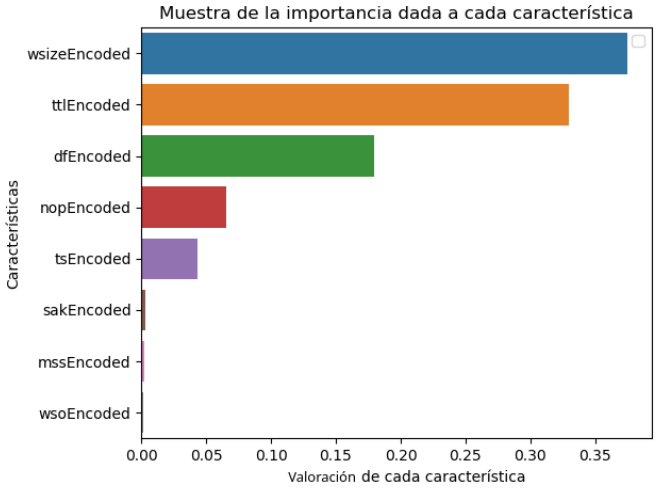

Figura 2.16: Gráfico de valoración de características

2.2 Fundamentos tecnológicos

2.2.1 Material utilizado

Placa de bajo conste con dos *interfaces* para recolectar datos y un servidor web, que incluso podría llegar a escalarse como un servicio en la nube.

El hardware de desarrollo utilizado para los dispositivos dedicados a la obtención de datos de la red será una o varias placas de bajo coste. La elección del servidor dependerá del número de recolectores y de la cantidad de tráfico a procesar, debiendo permitir compatibilidad con la paquetería utilizada y que además tenga navegador web en su instalación. Nuestra estación de trabajo tampoco necesitará unos recursos mínimos relevantes.

2.2.1.1 Dispositivo colector

Para el dispositivo colector (encargado de realizar la captura de tráfico), necesitaremos una placa de coste reducido. Esta es una característica importante para lograr un producto escalable mediante el aumento del número de dispositivos que pueda ser viable para la mayoría de las empresas.

Además, necesitamos un dispositivo de tamaño reducido, que no perturbe el espacio de la red auditada, siendo fácilmente accesible y con mínimas necesidades a nivel energético y de red. En el aspecto de la conectividad es necesario una tarjeta de red secundaria; que podría ser una tarjeta 4G o, en nuestro caso una tarjeta ya integrada en la placa, reduciendo así más los costes.

Nuestro dispositivo debe ser compatible con las herramientas utilizadas para la realización del proyecto, así como con las tecnologías actuales.

Teniendo en cuenta todas las características comentadas, se realiza una comparativa de distintos productos existentes en el mercado con la intención de encontrar el de mayor calidad a un precio reducido.

Marca/Modelo	Raspberry Pi 2B+	Raspberry Pi 3B+	Beaglebone Black	Banana Pi M3
CPU	Cortex A7	Cortex A53	ARM Cortex-A8	Cortex A7
Memoria	1GB DDR2	1GB DDR2	512MB DDR3	1GB DDR3
Ethernet	100Mb	100Mb	100Mb	1GB
Wirless	No	Si	No	No
Precio	36,81€	36,90€	65€	82,99€

Tabla 2.1: Tabla comparativa de productos de bajo coste

Tras el análisis de cada producto por separado se observa que la placa Banana Pi M3 nos ofrece un puerto Ethernet gigabit y una memoria RAM mejor que sus competidoras,con un mayor ancho de banda. La placa "Beaglebone Black" tiene características similares a sus competidoras destacando su procesador. Si nos centramos en la comparativa entre ambas Raspberrys Pi, podemos destacar que por un precio similar y muy reducido, el modelo 3B+ destaca por incorporar una tarjeta *wireless* y poseer un procesador mejor que el modelo 2B+.

Nuestra placa estará dedicada al análisis pasivo de red por lo que no será necesaria una gran potencia de procesamiento ni un elevado ancho de banda. Además parece muy acertado para este desarrollo el que una placa posea dos tarjetas de red, una de ellas *wireless*. Por todas las razones citadas, nos decantamos por la Raspberry Pi modelo 3B+, que por un precio reducido y siendo la única que incorpora dos tarjetas de red, nos ofrece todo lo necesario.

2.2.1.2 Servidor

Además del dispositivo de escaneo, en este proyecto también se desarrollará una aplicación web encargada de mostrar y explotar los datos obtenidos remotamente por nuestra placa de bajo coste.

El servidor web no necesita grandes requerimientos a nivel *hardware*, es más, en el entorno de desarrollo se utilizará una máquina virtual para la instalación del servidor web. Siendo necesarios a nivel físico conexión de red y a nivel lógico, compatibilidad con las herramientas utilizadas.

2.2.1.3 Máquina de desarrollo

La máquina utilizada para el desarrollo será un portátil ASUS k53E con un procesador Intel(R) Core(TM) i5-2450M CPU a 2.50GHz. Posee cuatro núcleos con una compatibilidad de arquitectura x86_64 y tiene seis GB de memoria RAM. Al ser una máquina física poseemos potencia suficiente para este desarrollo.Además de utilizarla para el desarrollo, se encargará de la ejecución de pruebas siendo la anfitriona de la máquina virtual que albergue la aplicación web.

2.2.2 Software utilizado

En este punto se comentará el software utilizado y el motivo de su utilización.

2.2.2.1 Sistemas Operativos

El sistema operativo que se empleará principalmente será linux Debian.

Será el sistema sobre el que se ejecutará tanto en el servidor web como en el colector, que utiliza una distibución de Debian modificada (Raspbian), adaptada a este dispositivo. Debian es un sistema operativo robusto, cualquier actualización del sistema ha sido probada durante un tiempo razonable antes de publicarla en sus repositorios.

Un servidor ha de ser fiable, por esta razón Debian es un sistema operativo idóneo, ofreciendo además un amplio repositorio de utilidades y librerías que serán necesarias para nuestro proyecto, evitando así tener que buscarlas en fuentes externas.

Nuestra estación de trabajo es una máquina física que posee el sistema operativo Manjaro, un sistema que se basa en la distribución Arch linux. Este operativo es muy flexible conteniendo en sus repositorios mantenidos por la comunidad innumerables herramientas muy útiles para el desarrollo del proyecto.

2.2.2.2 Lenguajes de programación utilizados en el desarrollo

El lenguaje principal para el desarrollo del proyecto fue Python por lo versátil y potente que es, siendo además de código abierto, con una licencia denominada *Python Software Foundation License*. Gracias a su popularidad, tiene un gran número de librerías que serán útiles durante el desarrollo. Se eligió la versión 3[17], por contener un número mayor de librerías y una amplia comunidad de usuarios que nos ofrecen todo tipo de ayuda sobre ”buenas prácticas” y resolución de problemas.

Se encuentra especialmente adecuado para este proyecto ya que contiene diferentes librerías destinadas a la manipulación de paquetería de red y es fácilmente ampliable.

Además es uno de los lenguajes más utilizado en proyectos con placas de bajo coste, pudiendo interactuar con la placa incluso a nivel físico.

También se programarán Scripts en Bash para realizar el despliegue de nuestra aplicación de la forma mas sencilla y automatizada posible (Anexo A).

2.2.2.3 Entorno de desarrollo

Para el desarrollo se empleará un IDE (Entorno de Desarrollo Integrado), herramienta que facilita la tarea de codificar y de probar los programas creados.

Desde el primer momento utilizaremos la herramienta PyCharm, que además de facilitarnos la tarea de codificar con auto completado y selector de código, nos ofrece funcionalidades muy útiles como la posibilidad de actualizar nuestro código remoto (Ubicado en nuestra Raspberry pi por ejemplo) en tiempo real. Se obtiene un desarrollo más dinámico evitando tener que migrar el código *a posteriori*.

Esta herramienta a pesar de que no es gratuita, nos ofrece una licencia destinada a estudiantes que nos da acceso a todas la funcionalidades que se ofrecen en la versión de pago. Muchas de estas funcionalidades serán ampliamente utilizadas durante el desarrollo, como por ejemplo la citada anteriormente opción de 'desarrollo en remoto' o las características específicas para el desarrollo web.

2.2.2.4 Secure Shell (SSH)

Protocolo que nos proporciona un canal seguro, por medio del cifrado de la información transmitida, a un equipo remoto. El propio PyCharm utiliza este protocolo con el fin de transmitir la información almacenada en un buffer local, al equipo remoto.

Se ha utilizado de forma continuada este protocolo, tanto para el desarrollo y control de versiones con GIT (Software de control de versiones que nos permite llevar un registro de los cambios realizados), como para la configuración de todas la máquinas, utilizadas en el proyecto.

2.2.2.5 Control de versiones

Para la gestión de cambios realizados en nuestro desarrollo así como para llevar un seguimiento de los avances se utilizará un software para el control de vesiones[18].

Se utilizará la herramienta GIT que nos ofrece este control sin la necesidad de estar sujeto a una conexión a Internet. Podemos gestionar nuestras versiones en local de la misma forma que si tuviésemos acceso al repositorio.

Por supuesto, en cuanto se tiene conexión todos los cambios guardados en local, pueden subirse a un repositorio remoto. Nos aporta flexibilidad en cuento al control de desarrollo en cualquier parte.

Como repositorio remoto se utilizaremos BitBucket por ser gratuito y por poseer cuenta de usuario en el mismo. Gracias a este repositorio externo, se mantuvo 'al día' tanto el dispositivo recolector (Raspberry Pi) como el servidor Web. Además en caso de que se produjese algún problema, se utilizaría este repositorio a modo de copia de seguridad.

El repositorio utilizado así como todos sus cambios se encuentran accesibles al público para su consulta en «https://Juanfreijeiro@bitbucket.org/Juanfreijeiro/tfg.git»

2.2.2.6 Análisis y explotación

Para la realización del análisis y captura de datos así como a su categorización, clasificación, exposición y explotación posterior se utilizaron distintas herramientas ya existentes que se añadieron al código de la aplicación por medio de librerías.

2.2.2.6.1 Análisis pasivo

Para la realización del análisis pasivo se utilizaron distintas librerías y herramientas de código abierto ya existentes y probadas, con las que se obtuvieron unos resultados satisfactorios. Aunque existen un gran número de estas librerías, el mayor problema encontrado ha sido la documentación ofrecida, que suele ser escasa.

Para realizar la capturar pasiva del tráfico se utilizó la librería Scapy[19]. Es una librería que se ejecuta de forma nativa en Linux y que nos permite capturar y manipular el tráfico de red. Scapy es compatible con un gran número de protocolos de red, facilitándonos el trabajo. Se optó por esta librería por que además de ser una herramienta muy potente en el análisis de paquetería de red, nuestro sistema basado en Debian, es totalmente compatible, lo que nos asegura una mejor eficiencia y evitamos problemas futuros que puedan surgir por incompatibilidades.

Con Scapy podemos analizar cada paquete capturado por separado y obtener toda la información relevante para la explotación de datos. Uno de los problemas más comunes en este tipo de análisis es discernir entre la paquetería de la red objetivo y la que se obtiene de otras redes. Para manejar correctamente la paquetería de la red que estamos analizando y descartar el tráfico que no nos interesa se utilizó la librería de python *Ipy*, capaz de diferenciar entre la red analizada y todo lo demás por medio de las ips utilizadas.

Ipy nos permite diferenciar el tráfico de la red local del externo, discriminando las IPs públicas, o fijando la subred sobre la que nos queremos centrar.

Otra de las herramientas utilizadas, aunque no forma parte de la aplicación, fue *Wireshark*. Herramienta especializada en el análisis de tráfico de red que nos permitió analizar un mayor número de protocolos que Scapy, así como realizar estadísticas que nos ayudaron en la toma de decisiones inicial. También fue una herramienta muy útil a la hora de generar un Dataset[1] en un entono controlado para poder relacionar hosts con sistemas operativos específicos. Este Dataset es la base principal en la toma de decisiones del algoritmo clasificatorio implementado.

[1]Colección de datos habitualmente tabulada

2.2.2.6.2 Exposición de los datos obtenidos

Para lograr que los datos obtenidos por nuestros dispositivos de escaneo sean accesibles remotamente por nuestro servidor web se ha creado una API propia. Para esta tarea se utilizó el microframework 'Flask' que nos facilita la creación de endpoints gracias a sus librerías y nos brinda un servidor web interno para exponer nuestros datos.

2.2.2.6.3 Explotación de datos

Para esta parte las herramientas utilizadas se basaron en la necesidad de extraer información coherente de todos los datos capturados en el análisis previo.

Se utilizó el API conocida como 'macvendors' [20]. Un API externa que nos ofrece una gran cantidad de datos sobre dispositivos de red. Realizando una simple petición, en la que se indica la mac (Dirección física de la interfaz, siendo única en cada dispositivo) del nodo obtenido previamente en la captura pasiva de datos, se obtiene el fabricante del dispositivo de red que utiliza.

Esto es posible gracias a que los seis primeros dígitos hexadecimales identifican al fabricante del dispositivo. Información muy relevante en ataques dirigidos.

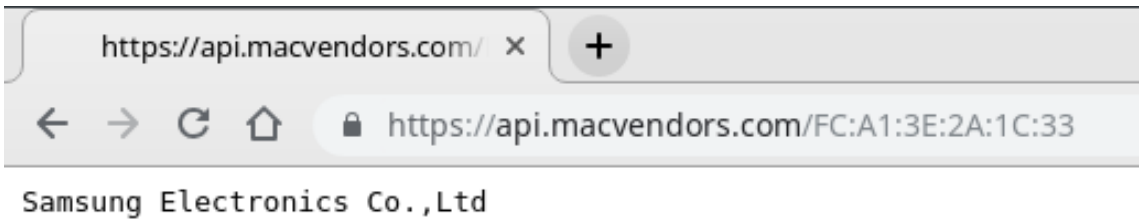

Figura 2.17: Ejemplo de petición al API https://macvendors.com

2.2.2.6.4 Machine Learning

Entre las herramientas utilizadas para implementar las técnicas de *Machine Learning* se encuentran, distintos módulos de Python, como puede ser **Pandas**, librería dedicada al análisis y manipulación de datos, muy útil para esta parte del proyecto, pues es necesario analizar una gran cantidad de datos almacenados en un *dataset* y devolverlo en forma de tabla.

Para la parte de minería de datos y la implementación del algoritmo clasificatorio se utilizó la biblioteca **Scikit-learn**. Libreria de software libre para Python que

incluye varios algoritmos de tipo clasificatorios, regresivos y de análisis de grupos; y entre los que se encuentra *Random Forest*, algoritmo clasificatorio en el que nos vamos a central.

También utilizamos la librería **Matplotlib** de forma puntual para la obtención de gráficas a partir de nuestros datos, pudiendo valorar de forma visual los resultados.

2.2.2.7 Tecnologías web

Para la aplicación web se utilizó el *framework* Django[21], de código abierto y escrito en Python.Este framework sigue un patrón modelo-vista-plantilla o template, aislando los datos, la lógica de la aplicación y lo que se muestra al usuario final.

2.2.2.7.1 Servidor web

El servidor se desarrolló con Python como lenguaje. Las credenciales de acceso a la aplicación se almacenaron directamente en la base de datos facilitada por Django. Por supuesto estas credenciales se almacenaron cifradas utilizando la librería **Argon2** que es un algoritmo seguro de hashing de contraseñas, diseñado para ser configurable en tiempo de ejecución y para tener un mínimo consumo de memoria (este último parámetro es configurable).

2.2.2.7.2 Django

Como la aplicación de análisis se programó en Python, se optó por continuar empleando este lenguaje en el servidor. Para facilitar la labor de diseño, se utilizó Django como framework, debido a sus funcionalidades, por ser de código abierto y por utilizar en todas sus capas el lenguaje Python o pseudo python (capa template, se utilza python en conjunto con html).

Incluye por defecto una base de datos de usuarios que utilizamos para gestionar el acceso a nuestra aplicación y un panel de administración. Además gracias a objetos propios del framework (HttpRequest y HttpResponse) maneja de forma autónoma las peticiones y respuestas HTTP, permitiendo que el desarollador se centre tan solo en la elaboración de la respuesta a las peticiones.

2.2.2.7.3 Cliente web

En la parte cliente de la aplicación web se utilizó Bootstrap. Bootstrap es un conjunto de herramientas diseñadas para el desarrollo web, de código libre que utiliza HTML,CSS y JS para definir el formato de la página web mostrada. Además, gracias a su sistema de rejillas (grids) facilita la creación de páginas web totalmente adaptables al dispositivo o tamaño de pantalla utilizado por el cliente. para poder ejecutar código en la máquina del cliente, se emplea, JavaScript como código base.

2.2.2.7.4 Plotly

Plotly[22] es una empresa de computación que desarrolla herramientas para el análisis de datos y su visualización gráfica. Se utilizó está herramienta para representar nuestras capturas de forma gráfica. Se obtienen los datos y se construyen los gráficos utilizando Python y nuetro framework Django, y se muestra en el lado cliente utilizando JavaScript, HTML y CSS.

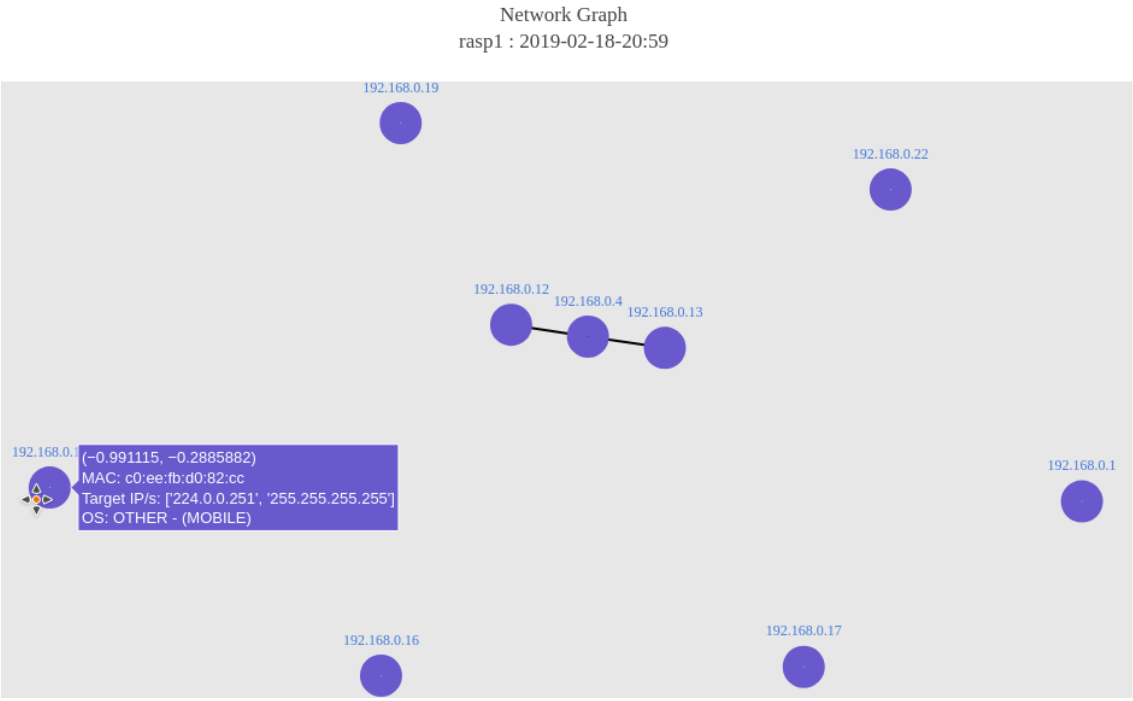

Figura 2.18: Mapa de red generado con Plotly

2.2.2.8 Almacenamiento de la información

La información obtenida durante el escáneo, será almacenada en fichero JSON. Tendremos que trabajar con las librería *json* o *csv*, así como estructuras del tipo diccionarios o listas en Python, para crear, gestionar y almacenar la información recogida.

También utilizaremos la libraría *os* para la gestión de ficheros en Linux y *datetime* para la generación del nombre de los ficheros a partir de la fecha.

Gracias a que el escaneo se ejecutará en un *thread* (hilo de ejecución) distinto, se podrá generar los ficheros JSON en tiempo de ejecución (durante la captura). La librería utilizada para crear los threads se llama *threading*, y contiene la clase *Thread* que será la que se encargue de realizar esta acción.

2.3 Estado del arte

Hoy día se pueden encontrar multitud de escáneres de red, muy completos y potentes. El problema principal de estas aplicaciones es que se centran en la cantidad de funcionalidades sacando protagonismo a la sencillez. Estos se tornan complejos para el usuario común, obteniendo de esta forma el resultado inverso al buscado (si posibilitamos que la gestión de la seguridad sea autónoma por parte del usuario final, lograremos que este no obvie tratarla como algo importante por ser demasiado compleja).

Además el grueso de estas aplicaciones de seguridad se basan en el conjunto de la red y no en cada uno de los nodos que la conforman. Debemos poner un mayor énfasis en asegurar cada una de las piezas, pues, «Una cadena es tan fuerte como su eslabón más débil».

2.3.1 Análisis pasivo en redes Ethernet

Existen principalmente dos tipos de escáneres de red. El activo, que se basa en el envío de paquetes de red a los equipos de la red objetivo y analizar su respuesta. Este tipo de escáner obtiene unos resultados muy precisos en un espacio corto de tiempo. Como contrapartida, si la red analizada utiliza un IDS (sistema de detección de intrusos) es muy probable que el escáner sea descubierto y bloqueado.

Por otro lado se encuentran los escáneres de tráfico de red pasivos, que no generan ningún tipo de tráfico durante la captura de datos. Estos escáneres análizan el tráfico existente entre el *host* que escánea y la red objetivo. Un ejemplo cotidiano del funcionamiento de la técnica sería una tercera persona que escucha una conversación ajena en una cafetería.

En esta última técnica es en la que nos vamos a centrar durante todo el proyecto. Además de la ventaja ante dispositivos detectores de intrusiones, el análisis de estos datos capturados, puede realizarse *offline* utilizando técnicas pasivas de *OS fingerprinting* (descubrimiento del sistema operativo que tiene cada nodo de la red).

2.3.1.1 Nmap

Nmap o Network Mapper, es una utilidad *open source* (de código abierto) para la exploración o auditoría de seguridad de red. Este escáner es del tipo activo siendo uno de los más populares a día de hoy por su rápidez y múltiples opciones que ofrece. Una de las habilidades más preciadas en este escáner es la de detección de sistema operativo (fue el primero en implementar esta técnica). Aunque posee opciones para realizar escáneos silenciosos, estos siguen siendo activos (por tanto interactúa con la red objetivo), teniendo la posibilidad de ser descubiertos.[9]

Nuestra herramienta es un escáner pasivo y aunque también utiliza técnicas de *OS fingerprinting* (pasivo), estas se realizan sin interacción alguna con la red. Gracias a está técnica nos mantendremos en el anonimato y evitaremos los bloqueos de *firewalls*. Además es fácilmente escalable pudiendo realizar escanéos en distintas redes de diferentes tamaños.

2.3.1.2 Xprobe2

La herramienta Xprobe2 también entra en la clasificación de escáner activo siendo mucho más silencioso que Nmap, gracias a la técnica que utiliza. Esta técnica se basa en el envío de un pequeño número de paquetes ICMP válidos (a diferencia de Nmap, no se envían mal formados) evitando ser descubiertos. Xprobe2 utiliza árboles de decisión binarios con los que obtienen una predicción del sistema operativo a partir de las respuestas obtenidas de la red objetivo. [9][23]

Nuestra herramienta también utilizará árboles de decisión binarios con el fin de obtener predicciones del sistema operativo de cada nodo, pero a diferencia de Xprobe2, nuestra herramienta obtendrá el conjunto de datos utilizado para esta tarea análisis pasivo. Nuestra herramienta obtendrá el tráfico pasivamente (a diferencia de Xprobe2) y se generará a partir de estas capturas el conjunto de datos que se utilizará para la generación de los árboles de decisión binarios. Nuevamente las ventajas de nuestra herramienta se centran en la no detección y en la escalabilidad.

2.3.1.3 p0f

La herramienta p0f (passive operation system fingerprinting) es un escaner de red pasivo implementado por el Michael Zalewski en 2006 que tenía como objetivo la detección de sistemas operativos sin ser detectado. Esta herramienta utiliza una base de datos de firmas (formadas por los distintos campos TCP,IP) para cada sistema y las compararlas con los datos capturados para obtener el sistema operativo.[9][12]

Nuestra herramienta, al igual que p0f será un escáner pasivo de red y utilizará una base de datos de firmas para cada sistema operativo. La diferencia principal es que para el reconocimiento del sistema se utilizará el algoritmo de aprendizaje supervisado *Random Forest*

Capítulo 3

Metodología

Para el desarrollo de este proyecto se utilizará una metodología ágil. Este tipo de metodología es especialmente útil en proyectos con un alto grado de incertidumbre por el uso de nuevas tecnologías o por cambios continuos.

En concreto, el modelo seguido se basa en la metodología *Scrum*. *Scrum* es un método para trabajar en equipo a partir de iteraciones o *sprints* teniendo como objetivo principal la planificación de proyectos con un gran volumen de cambios de última hora.

A diferencia de las metodologías clásicas, *Scrum* se suele planificar por semanas y al final de cada iteración se revisa el trabajo y se válida el de la semana anterior, pudiendo adaptar las decisiones futuras en el momento. La principal característica de esta metodología es que el desarrollo es incremental dejando de lado el desarrollo completo del producto o servicio por el que se caracterizaban las metodologías clásicas. Se establecen metas a corto plazo, haciendo más sencilla la corrección de la planificación en caso de que surja algún problema.

Por tanto, en el caso de este proyecto, se opta por una metodología ágil ante la posibilidad de que el software libre que se utilice deje de tener soporte o la escasa documentación de los mismos supongan un retraso inesperado. También se ha tenido en cuenta el desconocimiento de algunas tecnologías por parte del alumno y curva de aprendizaje asociada, pudiendo trastocar la planificación.

3.1 Scrum

La palabra inglesa *Scrum* se traduce como melé, termino del mundo del rugby que se refiere al conjunto de jugadores de un mismo equipo que avanzan a la vez, uniendo sus fuerzas para ganar terreno.[24]

3.1.1 Glosario Scrum

- **Reunión diaria *Scrum* :** una reunión diaria de 15 minutos donde cada miembro del equipo responda a las siguientes preguntas:
  - ¿Qué hice desde la última reunión?
  - ¿Qué voy a hacer para la próxima reunión?
  - ¿Qué impedimentos surgen en la realización de mi trabajo?
- **Impedimentos :** Cualquier cosa que impida al equipo o a un miembro de este, la realización de trabajo lo más eficiente posible.
- ***Backlog* del producto :** Este contiene los requerimientos del sistema, expresados en forma de lista y ordenado de forma priorizada. El *Product Owner* es el único responsable de dicha priorización.
- ***Sprint* :** Es una iteración del trabajo mediante el cual se implementa un incremento de la funcionalidad del producto.
- **El *backlog* del *sprint* :** Define el trabajo del *sprint*, representado por un conjunto de tareas que deben completarse para cumplir los objetivos. Los propios miembros del grupo de trabajo, son los encargados de decidir que tareas se realizan y cuando.
- **Tareas :** Se denomina así al trabajo que con su finalización ayuda al avance hacia el objetivo. Es una unidad de trabajo generalmente entre 4 y 16 horas

3.1.2 Roles de Scrum

Existen tres roles principales en cualquier proyecto de Scrum :

- ***Product Owner* :** figura que representa al cliente, no tiene porque ser el cliente real, puede representarla una persona del equipo que conozca bien el objetivo final y conozca las prioridades del cliente real.
- ***Scrum Master* :** persona que se encarga de resolver cualquier tipo de problema que represente una amenaza para el correcto desarrollo del proyecto. También se encarga de facilitar la comunicación entre el cliente y el equipo.
- ***Developer Team* :** grupo de profesionales auto gestionados que se encargan de realizar las tareas propuestas por el *Product Owner*.

Respecto a los roles *Scrum*, se incluyen figuras que no tienen por que estar presentes, siendo en ocasiones imprescindibles. Para estos casos existen roles auxiliares como por ejemplo:

- ***Stakeholders* :** Personas que no se encuentran directamente involucradas en el desarrollo del proyecto, pero que sí participan con su trabajo y por tanto se encuentran interesadas en su correcta conclusión.
- **Cliente :** Destinatarío final del resultado del proyecto.

3.1.3 El proceso

En *Scrum* un proyecto se ejecuta en ciclos temporales de corta duración. A estos ciclos los llamamos iteraciones, y cada una de ellas proporciona un resultado completo incrementando el valor del producto final, haciendo posible la entrega al cliente final en cualquier momento.

El cliente (*product owner*) es el encargado de priorizar las tareas a realizar por medio del *backlog* de esta forma las tareas se reparten en iteraciónes y entregas.

3.1.3.1 Planificación de la iteración

Cada iteración de *Scrum* se puede dividir en dos partes:

- **Selección de requisitos :** Es el *product owner* el encargado de presentar al equipo la lista de requisitos priorizada (*backlog*). El equipo pregunta dudas al cliente y selecciona los requisitos más prioritarios que creen pueden realizar en esa iteración.
- **Planificación de la iteración :** El equipo elabora la lista de tareas necesarias para el desarrollo de los requisitos seleccionados. Los propios miembros del equipo se auto asignan las tareas y se auto organizan para resolver los objetivos marcados.

3.1.3.2 Ejecución de la iteración

Cada día el equipo realiza una reunión de sincronización. En esta se inspecciona el trabajo realizado y lo que queda por hacer para alcanzar el objectivo de la iteración con cambios o adaptaciones según se requiera. Finalmente, cada miembro del equipo responde a tres preguntas (comentadas anteriormente en el punto 3.1.1).

3.1.3.3 Inspección y adaptación

Para finalizar la iteración se realiza una reunión de revisión que consta de dos partes:

- **Revisión :** el equipo presenta al cliente los requisitos completados y el cliente realiza adaptaciones desde la primera iteración en función del resultado mostrado y del objetivo a alcanzar.
- **Retrospectiva :** el equipo se auto analiza con la intención de mejorar su productividad en próximas iteraciónes. El facilitador o *Scrum Master* se encarga de eliminar o escalar los obstáculos encontrados.

3.2 Aplicando Scrum a este proyecto

Varias de las características de Scrum no son aplicables a un Trabajo Fin de Grado por requisitos previos impuestos por la normativa o por ser inviable las reuniones diarias entre el Product Owner (Director de proyecto) y el Developer Team(Alumno).

- **Anteproyecto :** La obligación de ceñirse a unos objetivos inamovibles especificados en el anteproyecto es contrario a la filosofía Scrum, basada en la flexibilidad de los objetivos.
- **Equipo (Developer Team) :** Aunque es posible la realización de un Trabajo Fin de Grado en Equipo, este no es el caso, siendo imposible la repartición de tareas por iteración y teniendo toda la carga de las mimas un solo integrante. Como antes esto es contrario a la filosofía de las metodologías ágiles que como mínimo recomiendan tres miembros.

3.2.1 Roles aplicados a este proyecto

- **Product Owner :** El cliente en este caso fue representado por el director Francisco Javier Nóvoa Manuel proponiendo en las reuniones nuevos requisitos sobre el producto y mejoras sobre los requisitos realizados previamente.
- **Scrum Master :** Esta figura la desempeñó también el director de este proyecto Francisco Javier Nóvoa Manuel que proporcionó soluciones a los problemas surgidos durante el desarrollo del producto.
- **Developer team :** Como se comentó anteriormente 3.2, en este proyecto el equipo de desarrollo fue asumido una sola persona, el alumno Juan Carlos Otero Freijeiro

3.2.2 Reuniones

- **Diaria :** Por incompatibilidades horarias entre el alumno (encontrándose trabajando en empresa, en el momento de realización de este trabajo) y el director se hace imposible seguir una de las máximas de la metodología Scrum, las reuniones diarias. Sin embargo en caso de que surja algún obstáculo o problema el alumno podrá contactar fuera de la planificación con el director.
- **Planificación del Sprint :** Se sigue el modelo de Scrum, estableciendo tareas y objetivos al inicio de cada Sprint.
- **Retrospectiva :** En el se revisan todas las tareas realizadas en cada Sprint y se comentan los objetivos conseguidos, los que no se alcanzaron y como mejorarlos en el próximo Sprint. Es una oportunidad para el equipo Scrum de auto evaluarse y crear un plan de mejoras.

3.3 Sprints

En este capítulo se pretenden explicar los objetivos realizados en cada iteración. Pasando a una explicación en mayor profundidad en el capítulo 4. Se pretendía utilizar esta metodología siendo rigurosos con las fechas y los tiempos, pero como se comentó anteriormente por motivos laborales no se pudo cumplir este deseo. Se enumerarán los Sprints realizados a modo de esquema del trabajo.

3.3.1 Product Backlog: Definición de tareas

Inicialmente se decide el proyecto a realizar y se establecen una lista de tareas priorizadas (Backlog) para la realización del proyecto. Inicialmente definiremos los objetivos específicos más globales que se encuentran recogidos en el anteproyecto, siempre con la posibilidad de aumentar o mejorarlos en cualquier iteración.

- **Construir un mapa de red :** Desarrollo de una solución que muestre la red auditada, de forma gráfica.
- **Analizar la topología de red :** Auditar cada uno de los protocolos de red que consideremos más útiles a la hora de aportar información relevante.
- **Recavar información sobre los equipos de la red :** La prioridad del cliente es obtener la máxima información posible de cada nodo de red y conocer que vulnerabilidades expone para intentar corregirlas

Este es el esquema principal del proyecto, pero son objetivos ambiciosos par conseguirlos en un solo Sprint. Por este motivo nos marcamos como objetivo inicial obtener información sobre las distintas placas existentes en el mercado y utilizar la que más se amolde a nuestros objetivos iniciales. Además nos proponemos buscar información y herramientas para tomar la decisión de que medio de transmisión de datos vamos a tratar. Vía WIFI y centrarnos en la captura de datos que viajan por el aire o por cable y centrarnos en hosts conectados directamente a la toma de red.

3.3.2 Sprint 1: Decisiones iniciales, preparación del entorno y análisis del tráfico (20/11/2017)

La placa que elegímos es la Raspberry Pi por lo económica que es, la potencia que nos aporta y la conectividad que nos ofrece de fábrica teniendo una antena WIFI y una toma de red incorporada. Sobre el medio en el que vamos a trabajar, nos decantamos por el cableado por ser un producto dedicado a auditar entornos empresariales, se considera que se obtendrán datos más relevantes en este medio.

Como entorno de desarrollo principal se utiliza el framework PyCharm. Se instala y se prepara para poder desarrollar directamente en la Raspberry Pi, sin necesidad de trasladar el código cada vez. En esta iteración se toma una de las decisiones principales en este trabajo. Escoger la paquetería encargada de realizar el escaneo de la red.

Se emplea la librería Scapy que se ejecuta de forma nativa en linux (sistema operativo elegido) y en python (lenguaje elegido para el desarrollo). Scapy es una herramienta diseñada para la manipulación de paquetería de red. Las conexiones remotas, tanto hacía la Raspberry Pi como hacía el resto de entornos se realizarán por medio del protocolo SSH. Para la diferenciación de los paquetes recibidos que no pertenecen a la red auditada de los que si, se utilizó la librería iPy, capaz de diferenciar entre las ips públicas de dispositivos externos a la red y las ips privadas que se asignan a los dispositivos de una intranet [1].

3.3.3 Sprint 2: Captura y análisis inicial del tráfico (18/01/2018)

Inicialmente se decidió realizar escaneos en diferenctes redes utilizando la herramienta Wireshark, con la intención de obtener información sobre los protocolos de red utilizados así como conocer cuales pueden ser capturados de forma pasiva y nos ofrecen información útil. Tras la revisión de estos datos, se toma en esta iteración la decision de que campos de los diferentes protocolos capturados deben ser almacenados por la información que aportan.

3.3.4 Sprint 3: Implementación de escaneo pasivo (02/08/2018)

En esta iteración se implementan los requisitos fijados en el Sprint previo sobre el desarrollo de de la aplicación de análisis pasivo, centrándonos en el escaneo de los protocolos seleccionados y en la captura y almacenamiento de sus opciones (las más relevantes).

Respecto al almacenamiento persistente de las opciones capturadas para cada protocolo, se decide utilizar una base de datos relacional (PostgreSQL) en la propia Raspberry Pi. En base de datos nos proponemos almacenar, usuarios y los análisis realizados por cada uno, paquetes capturados en cada análisis y cada una de sus opciones, los nodos a los que pertenece cada paquete capturado, y para cada nodo los sistemas operativos obtenidos.

En las pruebas realizadas inicialmente tanto el escaneo como el almacenamiento presentan un funcionamiento correcto.

3.3.5 Sprint 4: Pruebas intensivas y correcciones del análisis pasivo (07/08/2018)

Se realizan pruebas en el laboratorio de forma mas intensiva, dejando la Raspberry realizando escaneos durante días con el objetivo de probar la aplicación en una red mayor a las auditadas hasta el momento y utilizando duraciones superiores.

[1] Red no pública (normalmente empresarial) que se basa en las tecnologías utilizadas por los proveedores de servicios de internet

Las pruebas realizadas no resultan satisfactorias provocando que la aplicación falle por un excesivo consumo de recursos. Tras analizar las causas del problema y consultarlo con el Product Owner y Scrum Master (Director), nos decidimos por desechar la idea del almacenamiento en base de datos. Nuestro dispositivo aunque potente no suple las necesidades de memoria RAM(Random Access Memory) necesarias para la captura de paquetes y realizar lecturas y escrituras en tiempo de ejecución sobre la base de datos creando bloqueos en la aplicación y abortando la prueba.

Se decide realizar el almacenamiento persistente de la información en ficheros JSON. El cambio, aunque no es menor, resulta poco traumático gracias a que la aplicación ya utilizaba diccionarios JSON en memoria. Se realizaron los cambios necesarios para que estos datos se almacenasen de forma física en ficheros. También se decidió que debían dividirse estos ficheros en tramos de tiempo para una misma captura, facilitando así el transporte y la obtención de datos por parte del servidor web, en bloques de información más pequeños.

Después de los cambios propuestos por el Product Owner, la aplicación responde de forma satisfactoria a escaneos intensivos realizando la división de los ficheros de forma automatizada y almacenándolos en la propia Raspberry.

3.3.6 Sprint 5: Explotación de los datos mediante algoritmo clasificatorio Random Forest (21/09/2018)

En esta fase, inicialmente se propuso la explotación de datos utilizando comparaciones entre las configuraciones de opciones posibles que fueron capturadas durante el análisis pasivo. La propuesta de realizar el reconocimiento de sistemas operativos de forma manual resultó poco atractiva para el Product Owner por lo que tras reunirse con él, nos planteó la idea de la utilización de inteligencia artificial para el reconocimiento de firmas.

En esta iteración se utilizó un tiempo considerable en obtener información y en el aprendizaje de técnicas de *"Machine learning"*[25][26][27]. Nos servimos de la herramienta Scikit-learn, creada para la minería y análisis de datos y que además cuenta con un conjunto de algoritmos de aprendizaje supervisado clasificados por tipo.

Se genera un "Dataset" que recopila datos de las firmas de los distintos sistemas operativos obtenidos en varios análisis pasivos realizados en distintas redes. Este conjunto de datos será utilizado para "entrenar" a nuestro algoritmo, pudiendo reconocer el sistema opertivo de una firma que nunca hubiese recibido antes, utilizando datos previos almacenados para inferirla.

3.3.7 Sprint 6: Creación de la aplicación Web y comunicación con la Raspberry (11/12/2018)

En este Sprint se estableció como objetivo principal, el desarrollo de la aplicación web colgando del mismo varios subobjetivos. Se utilizó Django como framework y servidor web, además también se utilizó su propia base de datos para la gestión de usuarios sobre la aplicación.

En la aplicación web se muestran dos desplegables. En el primero se pueden elegir entre distintos dispositivos de escaneo (Raspberry), y el el segundo aparecen las capturas realizadas con el dispositivo elegido previamente. En la pantalla principal se muestra el mapa de red obtenido por el análisis pasivo. Bajo este mapa aparecen contadores indicando que tipo de sistema se encontró y el total de nodos. En el gráfico sobre cada nodo se muestra un link con la ip de cada uno, que nos aporta información detallada del nodo concreto.

Con la intención de no contaminar el análisis pasivo con tráfico de red no legítimo, se utilizó la antena Wifi de la Raspberry para realizar la comunicación entre Raspberry y servidor Web.

3.3.8 Sprint 7: Añadir algoritmo Random Forest a la aplicación web (20/03/2019)

Se añade la funcionalidad de predicción del sistema operativo por medio del algoritmo predictivo *Random Forest* a la aplicación web. Esta se ejecuta en cuanto se solicita una información mas detallada del nodo pulsando sobre el link que tiene cada uno representado en el gráfico. El análisis nos aporta uno o varios resultados con una probabilidad de acierto.

3.3.9 Sprint 8: Prueba conjunta con el módulo de explotación (12/06/2019)

Previamente se realizaron varias pruebas en distintos entornos pero sin incluír esta nueva funcionalidad. Como esta funcionalidad explota los datos ya capturados previamente por el análisis pasivo, no fue necesario realizar nuevos análisis, pues se contaba con los datos obtenidos en la red del laboratorio de la FIC y con datos de redes domésticas.

3.4 Diagrama de Gantt

A partir de la metodología seguida *Scrum*, creamos un diagrama de Gantt basándonos en los sprint para generar las tareas. Utilizaremos el diagrama de Gantt para estimar mediante una línea de tiempo lo que nos llevaría realizar este proyecto, si contásemos con un equipo de profesionales.

Tras la división de los sprints nos quedaremos con 18 tareas marcando como fecha de inicio del proyecto el 30 de septiembre de 2019, calcularíamos la fecha de finalización.

Para la realizar este proyecto, pensamos en un equipo formado por siete integrantes que trabajaran a jornada completa (8 horas) y se les remunerará siguiendo lo estipulado en el BOE [28].

- **Jefe de proyecto :** Será el encargado de realizar todas las gestiones del proyecto y del equipo, repartiendo el trabajo entre los integrantes y realizando seguimiento del mismo.
- **Desarolladores :** Se dedicarán a la programación del código, y su puesta a punto.
- **Analista de sistemas :** Se dedicará al diseño de métodos para el desarrollo del proyecto, estudiará las necesidades del software y propondrá soluciones para suplirlas.
- **Administradores de test :** Serán los encargados de probar la aplicación y de transmitir a analistas y desarolladores sus opiniones y las posibles mejoras.

TÍTULO	%	PROPIETARIO	DURACIÓN	FECHA DE INICIO	FECHA DE FINALIZACIÓN
TF1-T1 Elección de placa de bajo coste	0	Juan Carlos	1 days	08-30-2019	08-30-2019
TF1-T3 Documentarse sobre los protocolos de red	0	Juan Carlos	3 days	08-30-2019	09-01-2019
TF1-T2 Preparación del entorno de desarrollo e instalación de paquetería	0	Juan Carlos	2 days	08-31-2019	09-01-2019
TF1-T4 Capturas y revisión de datos	0	Juan Carlos	3 days	09-02-2019	09-04-2019
TF1-T5 Toma de decisión de que campos capturar en los protocolos análizados	0	Juan Carlos	2 days	09-05-2019	09-06-2019
TF1-T6 Desarrollo de escaner pasivo	0	Juan Carlos	30 days	09-07-2019	10-06-2019
TF1-T7 Implementación de base de datos PostgreSQL	0	Juan Carlos	15 days	10-07-2019	10-21-2019
TF1-T11 Información sobre técnicas de Machine Learning	0	Juan Carlos	7 days	11-01-2019	11-07-2019
TF1-T13 Creación de aplicación web	0	Juan Carlos	30 days	11-04-2019	12-03-2019
TF1-T8 Pruebas inciales de escaner pasivo	0	Juan Carlos	3 days	10-22-2019	10-24-2019
TF1-T9 Pruebas intensivas en laboratorio FIC	0	Juan Carlos	2 days	10-23-2019	10-24-2019
TF1-T12 Generación de Dataset	0	Juan Carlos	7 days	11-08-2019	11-14-2019
TF1-T10 Implementando nuevo método de almacenamiento	0	Juan Carlos	15 days	10-25-2019	11-08-2019
TF1-T16 Implementación de tabla informativa por nodo	0	Juan Carlos	2 days	12-04-2019	12-05-2019
TF1-T15 Creación de API para comunicar los datos con el servidor web	0	Juan Carlos	3 days	12-04-2019	12-06-2019
TF1-T17 Añadidiendo Random Forest a la aplicación web	0	Juan Carlos	20 days	12-06-2019	12-25-2019
TF1-T14 Creación de mapa de red	0	Juan Carlos	8 days	12-07-2019	12-14-2019
TF1-T18 Pruebas finales	0	Juan Carlos	2 days	12-26-2019	12-27-2019

Figura 3.1: Tareas del diagrama de Gantt

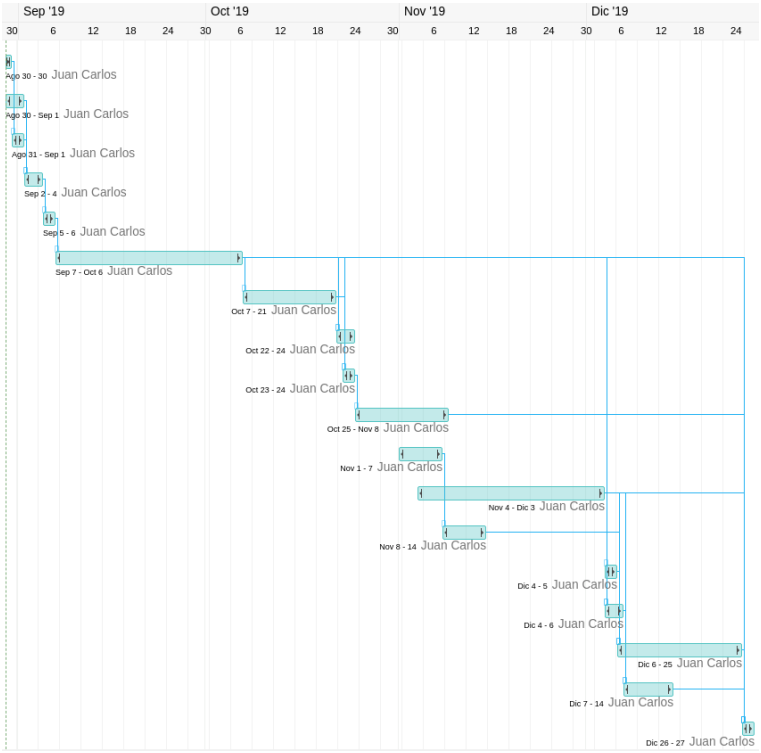

Figura 3.2: Representación del diagrama de Gantt

En la imagen 3.1 se listan todas la tareas que se seguirán durante todo el proyecto.

La siguiente imagen 3.2 muestra el diagrama de Gantt con una linea de tiempo que indica cuando estaría prevista la finalización. Contando con un equipo de 6 personas (jefe de proyecto, 2 desarrolladores, analista de sistemas y 2 testers) y ajustando muy poco el tiempo, se tardaría un total de 86 días.

3.4.1 Coste de personal

Se realiza el cálculo de costes de personal utilizando el boletín oficial del estado [28] y las tablas salariales incluidas en él . Para hallar el cálculo mensual se divide el salario total a percibir en 14 pagas.

Personal	Salario base	Plus del convenio	Salario total	Meses	Coste
Jefe de proyecto	21.555	1.408	22.993	3	4.927
2 Desarrolladores	15.442 x 2	10.84 x 2	16.531 x 2	3	7.084
Analista de sistema	21.969	1.536	23.505	3	5.036
2 Administradores de Test	11.773 x 2	828 x 2	12.601 x 2	3	5.400

Tabla 3.1: Tabla salarial BOE 6/03/2018

El total del proyecto ascendería a 22.447 €

Capítulo 4

Trabajo desarrollado

Aunque en el capítulo previo 3, se indicó que durante el desarrollo se utilizó una metodología ágil, intentar desarrollar la explicación de este trabajo de forma cronológica podría aumentar la complejidad para el entendimiento del lector.

Por tanto, se creyó conveniente utilizar para una explicación más detallada una metodología clásica en el desarrollo del software, organizando la explicación en fases de la metodología en cascada [29] por cada uno de los *sprints* seguidos. Se distinguen cuatro fases en esta metodología secuencial: análisis de requisitos, diseño, implementación y pruebas.

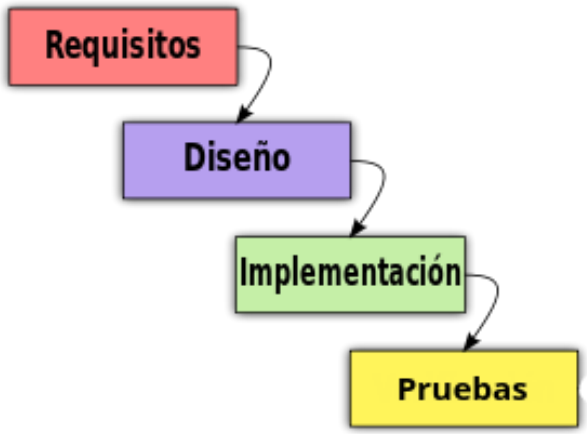

Figura 4.1: Metodología utilizada para la explicación del proyecto

4.1 Diseño arquitectónico

Se definirá la arquitectura del sistema, describiendo cada una de las partes que lo conforman y demostrando la escalabilidad propuesta en los objetivos iniciales.

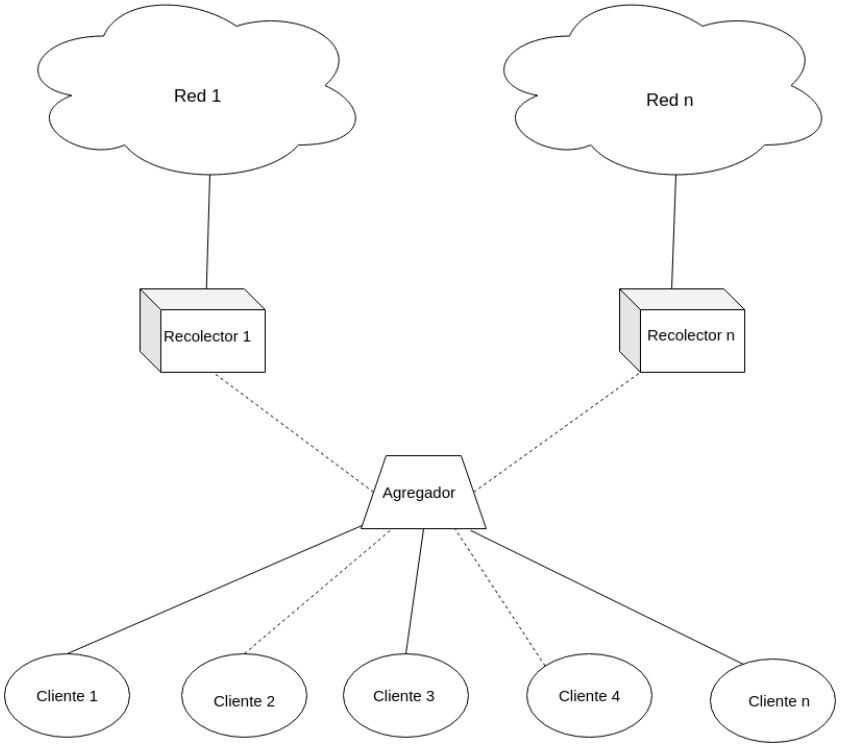

Figura 4.2: Arquitectura del sistema

En la figura 4.2 podemos observar la representación de dos redes locales (pudiendo ser más) y a cada una de ellas se encuentra conectado un dispositivo al que llamaremos *recolector*. Este dispositivo será el que se encargue de realizar las capturas pasivas de tráfico utilizando una de sus tarjetas (se encontrará conectado directamente a la red objetivo). Tendremos tantos dispositivos *recolectores* como redes o subredes queramos analizar cumpliendo con el objetivo de escalabilidad propuesto.

El siguiente elemento representado en la figura 4.2 es el *agregador*, dispositivo que será el encargado de centralizar todos los datos recavados por los *recolectores*. El *agregador* también realizará la tarea de minado de los datos obtenidos y ofrecerá resultados en base a su procesamiento. La transferencia de datos entre *recolectores* y *agregador* se realizará por medio de una interfaz dedicada a esta tarea, pudiendo ser un dispositivo 3G o como en el caso de este desarrollo, una tarjeta *Wifi*.

El *agregador* será el responsable de servir los datos a los clientes autorizados, mostrándolos de forma gráfica e intuitiva.

4.2 Análisis de requisitos

En este punto se describirán los requisitos que ha de cumplir el desarrollo haciendo un breve resumen de cada uno de ellos en líneas generales.

Nuestro sistema tendrá como objetivo principal capturar el tráfico de una red objetivo y enviar estos datos a un servidor capaz de analizarlos y representarlos utilizando para dicho envío una conexión y un medio distinto al del análisis.

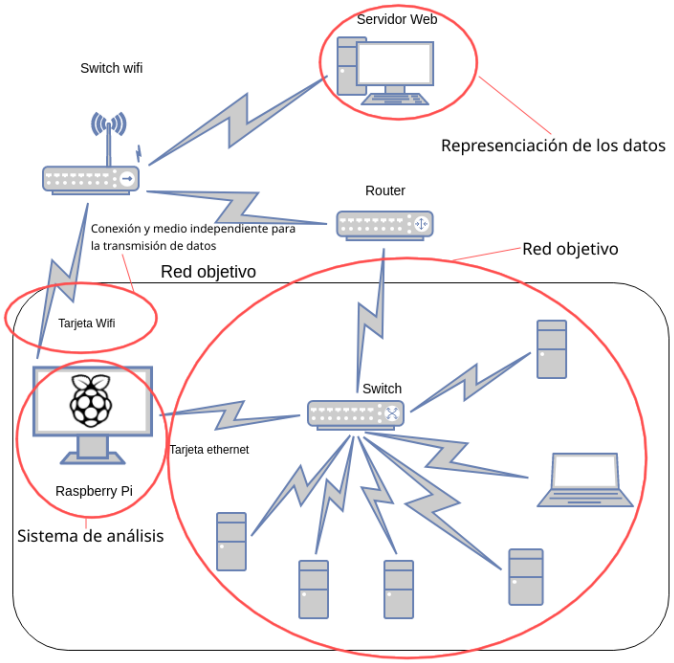

Figura 4.3: Identificación de partes de nuestro sistema

- **Explotación y representación de datos:** Esta tarea será realizada por un servidor web que podría encontrarse en la propia red objetivo o no, siendo esto indiferente. La aplicación web se encargará de mostrar los datos obtenidos de forma gráfica y de explotarlos para obtener información detallada de cada nodo de la red.

  Para acceder a la aplicación web, es necesario que el usuario se autentique previamente en el portal utilizando su usuario y contraseña,como ya se mencionó en la sección 2.2.2.7.1. Las credenciales de los usuarios se almacenan en la base de datos de usuarios que nos facilita Django de forma cifrada utilizando la librería Argon2[30].

  La aplicación que se ejecuta en el servidor web será la encargada de obtener los datos de los colectores. Además, también tendrá la capacidad de explotar los datos recabados por dichos dispositivos utilizando para esta tarea diferentes técnicas.

El portal que se ofrece al usuario mostrará inicialmente un desplegable en el que se escogerá el dispositivo de escaneo del que queremos obtener información, pues podrían ser distintos colectores conectados. Tras realizar esta selección, se mostrará un nuevo desplegable en el que se podrá seleccionar una captura realizada por el dispositivo elegido. Se obtendrá un mapa de red con todos los dispositivos encontrados con su IP asignada sobre cada nodo y pasando el ratón sobre alguno de estos nodos se mostrará un *pop-up* con un resumen de la información obtenida (MAC, IPs a las que se ha envíado paquetes y Sistema operativo).

Si se selecciona uno en concreto, se abrirá una nueva pestaña en el navegador ofreciendo más información sobre el nodo escogido. Tal como "sistema operativo", obtenido utilizando dos parámetros de la paquetería de red obtenida (TTL, DF), "Vendor" o fabricante del dispositivo de red, puertos detectados tanto del dispositivo origen como los puertos a los que envía paquetes, si hay paquetería UDP con texto en claro. Se mostrará en el apartado "otra información" (RAW) y por último se aplicará el algorítmo predictivo "*Random Forest*" que es de tipo árbol de decisión para a partir de datos de muestra poder decidir que sistema operativo se ejecuta en el nodo. La forma en la que se obtienen estos parámetros se explica en las secciones siguientes.

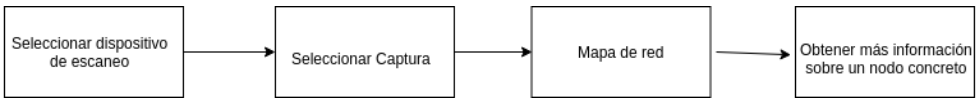

Figura 4.4: Proceso de acceso a la información de la aplicación web

- **Red objetivo:** red o redes analizadas de forma no intrusiva. Este sistema se plantea como escalable, y se desarrolla con la idea de poder disponer de distintos dispositivos de análisis para diferentes redes objetivo a la vez.
- **Transmisión de datos:** para evitar la contaminación durante la captura de datos, se utiliza una tarjeta de red y un medio para transmitir los datos al servidor web distinto al utilizado en el escaneo.
- **Sistema de análisis:** nuestro sistema se encarga de recoger y almacenar de forma pasiva los datos de red más relevantes, para su posterior análisis. Además permite mediante una API exponer estos datos para alimentar nuestra aplicación web.

4.2.1 Historias de usuario

Este concepto es utilizado por la metodología ágil Scrum para identificar los requerimientos del usuario y ser capaz de comunicarnos con él mediante lenguaje común, muy similar al natural. Las historias de usuario describen de forma simple una funcionalidad del producto, normalmente desde la perspectiva del usuario final. Siguen siempre un esquema predefinido, dado respuesta a ¿Quíen soy?, ¿Qué es lo que quiero? y ¿Para qué lo quiero?.

- **Cómo** [usuario de la aplicación]
- **Quiero** [ver un gráfico de la red y mostrar información relevante para cada equipo]
- **Para** [descubrir posibles vulnerabilidades de la red y conocer el número de equipos que la compone]
- **Condición de completitud:**
  * Visualización del mapa de red y conexiones entre equipos
  * Visualización de contadores de equipos
  * Se ofrece información detallada por equipo incluyendo sistema operativo, fabricante de red, puertos y servicios.

- **Cómo** [usuario de la aplicación]
- **Quiero** [realizar análisis en redes de gran tamaño o distintas de forma simultánea]
- **Para** [obtener la mayor cantidad de información sobre todas la redes que desee]
- **Condición de completitud:**
  * Permitir la selección de distintos dispositivos de análisis
  * Permitir la selección de distintas capturas por dispositivo

- **Cómo** [usuario de la aplicación]
- **Quiero** [poder transmitir los datos capturados a un servicio web]
- **Para** [poder visualizarlos]
- **Condición de completitud:**
  * Poder transmitir datos por una interfaz distinta a la de captura
  * Permitir al servidor web el acceso a los datos

- **Cómo** [usuario de la aplicación]
- **Quiero** [analizar el tráfico de red sin ser descubierto]
- **Para** [encontrar vulnerabilidades sobre los sistemas que posee la red]
- **Condición de completitud:**
  * Recolección de datos con técnicas de análisis pasivo
  * Permitir acceso al servidor web de los datos capturados
  * Almacenar por fecha de captura los datos almacenados.

4.3 Diseño

Previo a la fase de implementación se diseña cada uno de los requisitos analizados en el paso 4.2.

4.3.1 Sistema de análisis

Como se especificó en el análisis de requisitos nuestro sistema utiliza únicamente sobre la red objetivo la técnica del análisis pasivo y técnicas pasivas de descubrimiento del sistema operativo (*Passive OS Fingerprinting*)[12][23], buscando ser lo menos intrusivo posible.

Para realizar este análisis no es necesario que el dispositivo obtenga una dirección IP de la red auditada, siendo tan solo la tarjeta *Wifi* (tarjeta utilizada para el envío de los datos capturados al servidor web) la única que necesitará una IP.

Nuestro sistema también se encargará del almacenamiento de los datos que se consideran más relevantes en ficheros JSON dividiéndolos mediante un intervalo de tiempo definido, con la intención de evitar un fichero de datos único difícil de manejar.

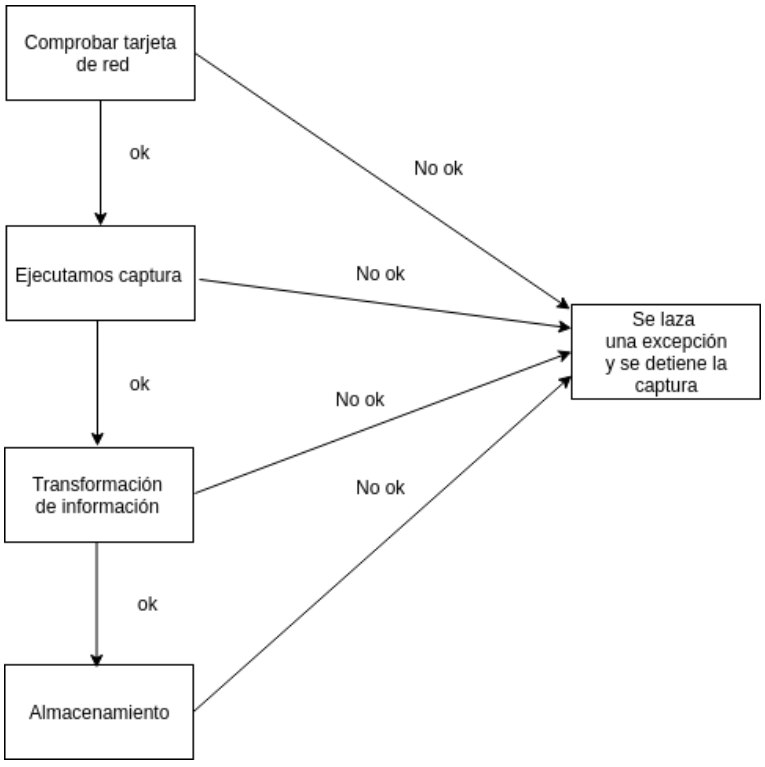

Figura 4.5: Diagrama de flujo seguido por el módulo PassiveFingerprinting.py

En la figura 4.5 se muestra los pasos que seguirá nuestra aplicación para llevar a cabo la captura de datos:

- Durante la primera fase se obtendrá la tarjeta de red que se utilizará para realizar la captura y se realizará la comprobación del modo promiscuo (modo utilizado

para capturar todo el tráfico que fluye por la red)[31]. En caso de que la *interfaz* de red no se encuentre en este modo, se intentará establecerlo. Si por cualquier motivo, no se encuentra la *interface* o no se logra establecerla en el modo indicado se detendrá el proceso lanzando una excepción y se registrará el error en un fichero de log.

- Si la primera fase tiene éxito, se ejecutará la segunda fase con la que se iniciará el escáneo de la red. Se creará una clase *Sniffer* que recibirá como parámetro la tarjeta de red en modo promiscuo. Si el paso previo se ha realizado de forma correcta, comenzará la captura. Al igual que en el paso previo ante cualquier error, se detendrá el proceso lanzando una excepción que además será registrada.
- Este paso, aunque se representa de forma secuencial, se realizarán durante la captura en tiempo de ejecución. Se filtrarán los campos más relevantes para nuestro análisis almacenándolos en memoria en estructuras de Python.
- En el último paso se define el almacenamiento de datos también se realizará de forma simultánea a la captura y filtrado de datos. Se realizará el almacenamiento en ficheros JSON que serán truncados siguiendo una configuración de tiempo establecida en el despliegue de la aplicación de captura.

4.3.1.1 Patrón de diseño de la aplicación de captura

Vamos a hablar del patrón de diseño que se seguirá durante la creación de la aplicación de escáneo. Utilizaremos el patrón de diseño «Modelo-Vista-Controlador» o MVC [32]. Este es un patrón de arquitectura de software que se centra en separar los datos y la lógica de negocio de la representación de los resultados, así en la capa modelo definiremos todas las funciones que vayan a leer o escribir datos, en la capa controladora, se establecerán las funciones encargadas de operar con esos datos y de servir los resultados a la capa vista que será la encargada de mediante diferentes funciones facilitar los datos al usuario.

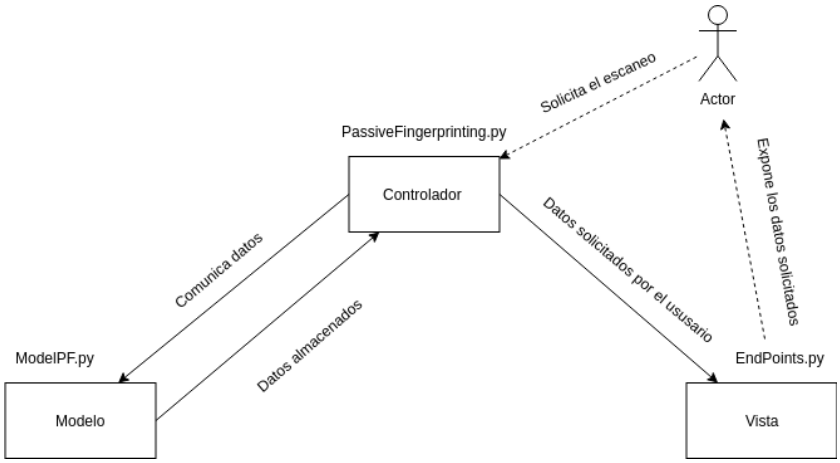

Figura 4.6: Esquema MVC de la aplicación de captura

Nos serviremos de la imagen 4.6 para explicar como funcionará la aplicación en base a los requerimientos establecidos para la captura.

Nuestra aplicación web (de la que se hablará en próximos capítulos) hará el papel de actor que solicita el inicio del escáneo y recibe los datos almacenados en un formato concreto. Así nuestro controlador, que albergará la función de análisis, iniciará la captura, filtrado y almacenamiento del tráfico.

Este almacenamiento será tratado por las funciones contenidas en el modelo. Entre estas funciones se encontrarán varias para generar de forma autónoma los nombres de los ficheros, división de ficheros, escritura en disco, y por supuesto también lectura.

El controlador que hace el papel de intermediario facilitará los datos obtenidos en la capa modelo a la vista, que se encargará de ofrecer los resultados solicitados al servidor web (nuestro cliente en este caso)

4.3.1.2 Almacenamiento y exposición de los datos obtenidos

Durante el escaneo, la información obtenida se irá almacenando en ficheros JSON. Como se comentó en la sección 3.3.5 *sprint* 4, se elige este sistema de almacenamiento por ser ficheros ligeros para su envío y por que inicialmente ya se trabajaba en memoria con estructuras JSON haciendo sencillo su volcado en ficheros.

Para evitar crear ficheros exageradamente grandes, se implementará la división de los mismos estableciendo un intervalo de tiempo previo al escaneo. Igualmente importante es la forma en que se nombrarán estos ficheros, se utilizará siempre la misma estructura, año-mes-día-hora:minuto.json. Además para evitar confundir ficheros de distintas capturas, se utilizará otro fichero JSON a modo de ”mapa” (imagen 4.7).

El dispositivo de escaneo dispondrá de una API REST [33] con su propio servidor web para exponer todos los datos obtenidos. Se crean dos *Endpoints* (URL que ofrece una funcionalidad del API), uno para obtener el fichero de metadatos anteriormente citado (imagen 4.7) y otro para acceder a los datos de cada uno de los ficheros creados.

1. **http://192.168.0.4:5002/metadata :** Con una consulta GET (Una de las cuatro operaciones básicas del protocolo REST utilizada para leer u obtener datos) sobre este *Endpoint* se obtienen los datos de las capturas realizadas, así como cual fue el primer fichero generado y el último para dicha captura
2. **http://192.168.0.4:5002/capture/<nombreFichero> :** A este *endpoint* es necesario indicarle el fichero del que queremos obtener los datos. De esta manera mediante una petición GET sobre la *url* se obtienen todos los datos almacenados en el fichero indicado.

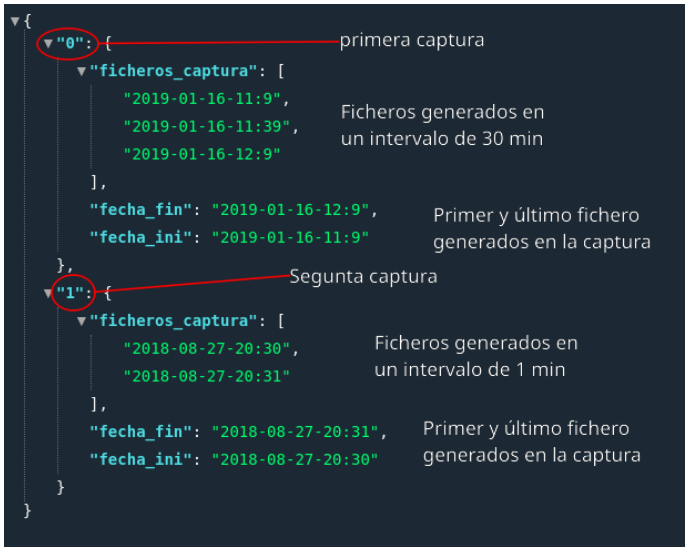

Figura 4.7: Ejemplo de fichero metadata.json con información de las capturas realizadas

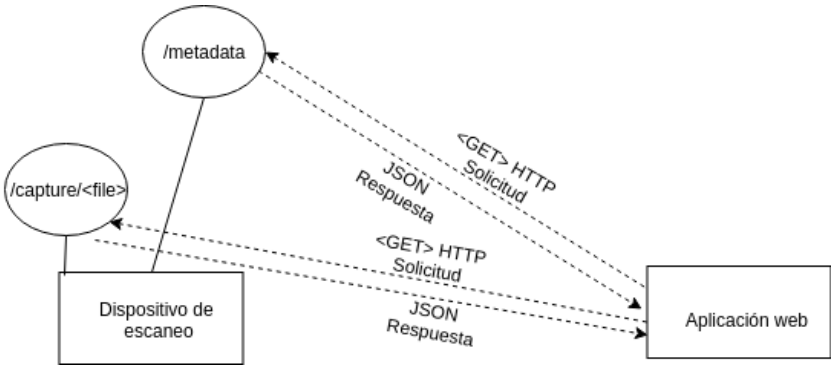

Figura 4.8: Endpoints publicados por el API de la herramienta

Figura 4.9: Ejemplo del endpoint Capture sobre el fichero 2019-02-18-20:59

Toda la configuración de la aplicación de escaneo, se realiza sobre un fichero ".py" separado para abstraerla lo máximo posible de su configuración variable, facilitando así la tarea de realizar modificaciones de la configuración sin riesgos.

En el se incluye el intervalo de tiempo en el que se realiza la división de los ficheros JSON especificado en centésimas de segundo (cs), la dirección del servidor web del API y su puerto.

```
#!/usr/bin/env python3

client ={'interval': 600,
        'apiIp':'192.168.0.4',
        'apiPort': '5002'}
```

Figura 4.10: Ejemplo de configuración del API y del intervalo de tiempo

4.3.1.3 Diagramas de secuencia de la aplicación de captura

Mediante diagramas de secuencia representaremos las funcionalidades de la aplicación dedicada a la captura, almacenamiento y exposición de datos.

4.3.1.3.1 Captura pasiva de datos

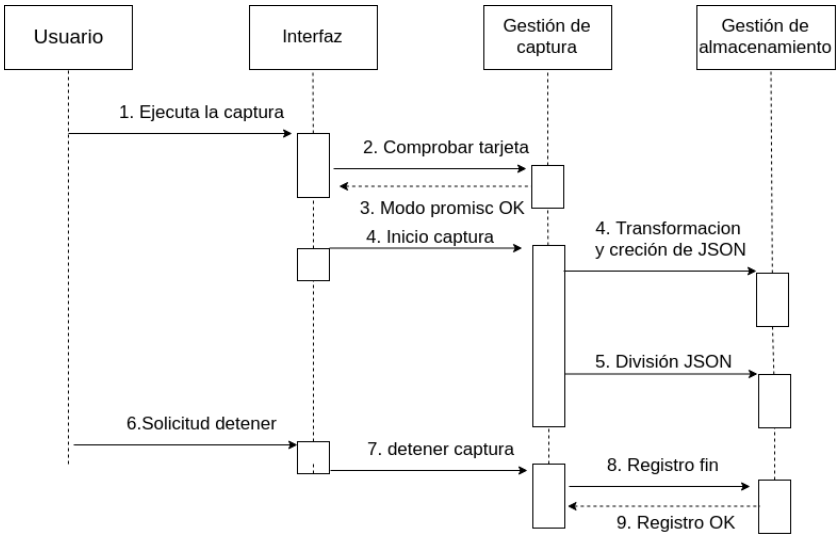

Figura 4.11: Diagrama de secuencia del proceso de captura

En la figura 4.11 representamos la funcionalidad captura, donde siguiendo el orden indicado,se lograrían generar datos, almacenandolos en ficheros json. También se registraría en el fichero "metadata.json" el registro de fin de captura.

4.3.1.3.2 Exposición de datos capturados

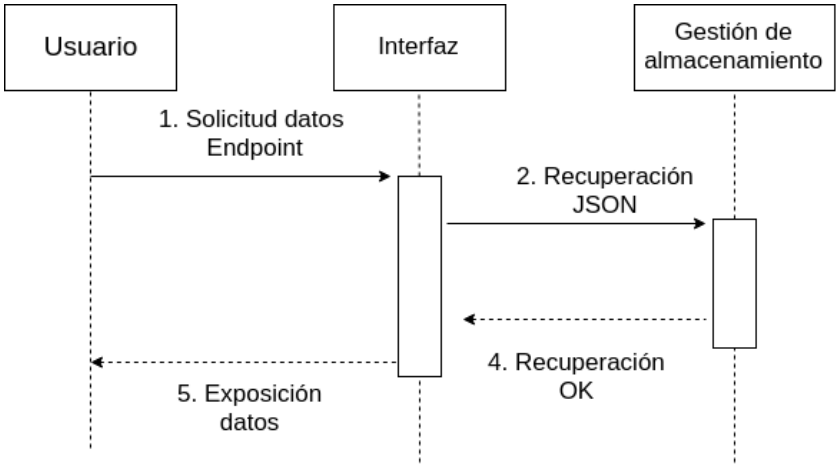

Figura 4.12: Diagrama de secuencia del proceso de exposición utilizando API

En este caso, en la figura 4.12, se representa la solicitud de datos por parte de la aplicación web al API que se encontrará en el dispositivo de captura (colector).

4.3.2 Aplicación web

La aplicación web se encontrará en el agregador (figura 4.2), donde se centralizarán todos los datos obtenidos por la aplicación de captura (instalada en cada recolector), visualizándolos gráficamente y proponiéndonos técnicas para su explotación.

En la figura 4.13 se representa la lógica utilizada por el *framework* Django y la aprovecharemos para mostrar donde se aplicaría cada caso de uso.

Así, vemos como en cuanto un usuario realiza una petición al servidor web, lo primero que actuará en Django, será la vista que renderizará la *template* de *login* para validarlo en la aplicación. Esta validación de usuarios es el único caso en el que utilizaremos una tabla de datos. Como solo necesitaremos una tabla, utilizaremos la base de datos ofrecida por el *framework*, facilitándonos mucho el trabajo de diseño e implementación.

Por cada función definida en la vista, se validarán internamente las credenciales del usuario por medio de *cookies* de sesión. Esta definición de funciones en la vista de Django siguen un flujo bien definido como se observa en el gráfico, pasando de la verificación del usuario a la solicitud de datos por el mismo; posteriormente su representación gráfica y explotación por nuestro algoritmo clasificatorio, capaz de predecir el sistema operativo instalado en los nodos, ofreciendo un mayor nivel de detalle.

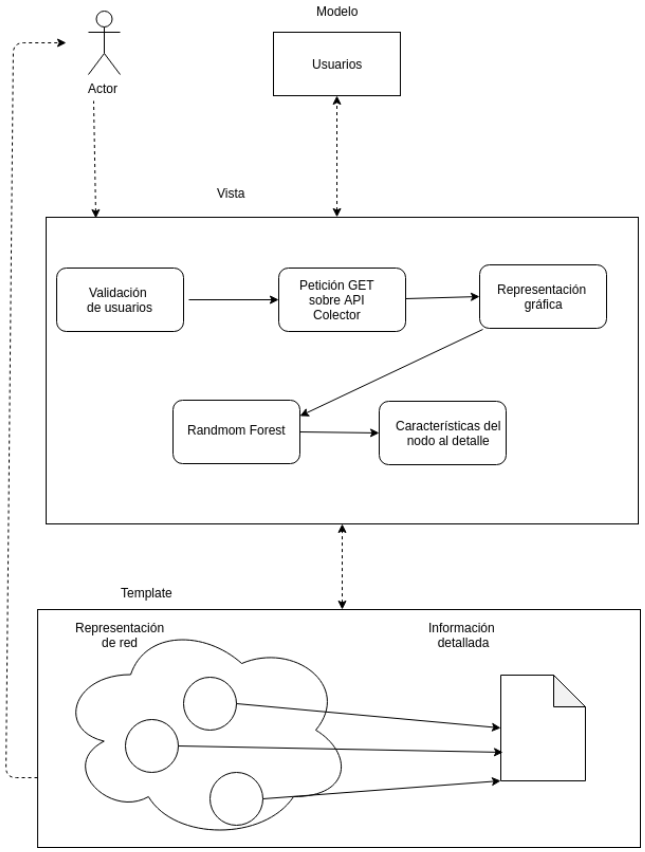

Figura 4.13: Arquitectura utilizada por el *framework*

4.3.2.1 Patrón de diseño de la aplicación web

Como ya hemos comentado, Django redefine el patrón de diseño «Modelo,Vista,Controlador» por «Modelo,Vista,Template». En esta nueva definición se aúna parte de la vista y el controlador en una sola capa capaz de *renderizar* el código para enviarlo a la capa *template* (donde todo se transforma en código estático, dejando lo dinámico a un pséudocodigo definido para las *templates* de Django), interactuar con la capa modelo y establecer la definición de funciones que satisfagan los requerimientos establecidos.

Se muestra en la figura 4.14 como se conecta la base de datos con la capa modelo. Esta capa nos abstrae de la complejidad del diseño, dándonos la capacidad de definir una base de datos en lenguaje menos técnico o con mínimos conocimientos.

4.3.2.2 Diagrama de secuencia de la aplicación web

En el diagrama de secuencia de la aplicación web 4.15, se representa la autenticación del usuario, la solicitud de información para un colector y captura concreta y la información que se logra a nivel de nodo (equipo) tras analizar los datos con distintas

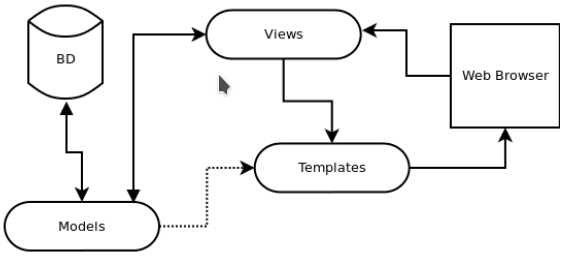

Figura 4.14: Patrón de diseño utilizado por el framework Django

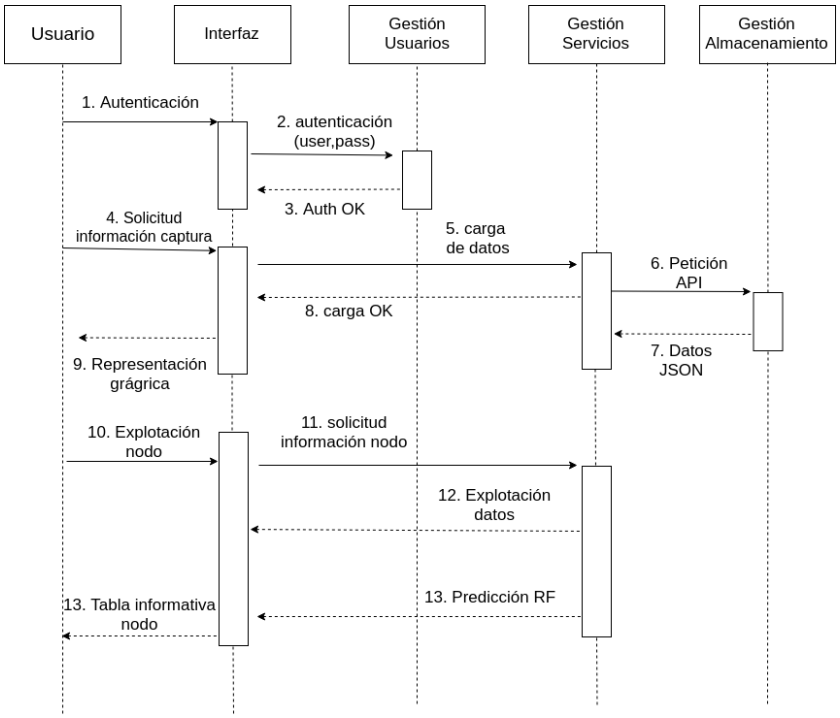

Figura 4.15: Diagrama de secuencia aplicacion web

técnicas de *OS fingerprinting* y realizar la predicción del sistema operativo utilizando el algoritmo clasificatorio Random Forest (RF).

4.3.2.3 Explotación de datos

La explotación de datos la llevará a cabo la aplicación web, aislando completamente el análisis y la captura de datos que realizan los colectores de forma pasiva,de esta fase más agresiva, en la que se tratan estos datos con diferentes técnicas con el fin de obtener toda la información posible a nivel de nodo.

4.3.2.3.1 Estudio de la configuración de la paquetería

De entrada en la representación inicial del mapa de red ya se utilizan los datos capturados para obtener un ”avance” del posible sistema operativo en cada nodo.

Nos basamos en dos parámetros (DF y TTL) de la cabecera IP comentados en el apartado 2.1.1.2.1, y en como se encuentran configurados para aproximar el operativo.

Sistema Operativo	DF	TTL
Windows	False	128
GNU/Linux	True	64
Sistema embebido (Switch,Routers...)	False	TTL != 128
Sistemas moviles (Android..)	True	1 || 255
Otros (no clasificables con estos parámetros)	Distinto de las configuracioines anteriores	

Figura 4.16: Configuración de parámetros DF y TTL según el sistema operativo

En la tabla 4.16 se muestran distintas configuraciones basadas en varias pruebas hechas sobre los datos de diferentes redes escaneadas. Se obtienen estas configuraciones por medio de la observación de la paquetería de red que configura cada sistema operativo para su envío.

Por ejemplo, se observa que en los sistemas Windows (Windows 10,7, XP), el bit de *Don't Fragment* lo envían deshabilitado y el tiempo de vida del paquete se configura siempre con un valor de 128.

En cambio los sistemas Linux configuran el bit de DF como habilitado y tienen un TTL de 64, gracias a como configuran estos valores los operativos, podemos realizar una clasificación inicial.[34] En los casos de routers o switchs se observa que no suelen tener activo el bit DF y tienen TTLs poco comunes (1,4...) y para los dispositivos móviles el bit de DF se encuentra activo y los TTLs oscilan entre los valores 1 o 255.

Todo lo que se sale de estas configuraciones, lo clasificaremos como ”otros dispositivos” e intentaremos obtener su sistema por medio de técnicas de ”Machine Learning”.

4.3.2.3.2 Conociendo el fabricante del dispositivo de red

Se diseñará un método que satisfaga este requisito en la aplicación web en cuanto se solicite mas información sobre un *host* (Equipo conectado a la red local) en concreto. lo primero que nos encontraremos será con su dirección MAC y con el fabricante de la *interfaz* de red.

Por ejemplo, en el caso de que el fabricante fuese ”OKI ELECTRIC INDUSTRY CO., LTD”, es probable que el dispositivo sea una impresora, de esta forma un atacante podría centrarse en las vulnerabilidades conocidas de estos equipos.

Para conocer al fabricante del dispositivo, utilizaremos un API externa llamada ”macvendors”. Esta herramienta nos ofrece una gran facilidad en su uso y se encuentra en constante actualización de su base de datos que cuenta ya con mas de 16.500 registros.

Se utilizará este API para conocer el fabricante de la *interfaz* de red que posee una dirección MAC, que es una dirección física y única para cada dispositivo de red. Su funcionamiento es simple, se basa en buscar en su base de datos los 24 primeros bits de esta dirección que son los encargados de identificar al fabricante.

Figura 4.17: Bloques hexadecimales utilizados por macvendors

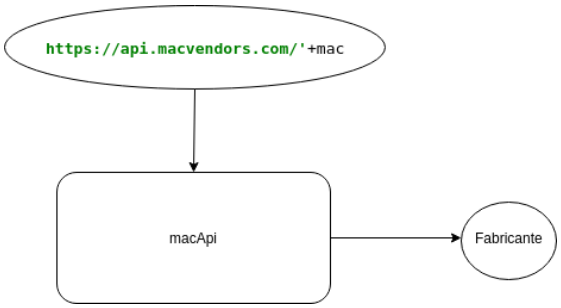

Figura 4.18: Definición gráfica de la función ”macApi”

En la figura 4.18 se muestra de forma gráfica la definición de la función que se implementará para realizar la llamada al API. Nuestra herramienta realizará la petición contra el *access point* enviando como parámetro la dirección MAC del dispositivo analizado, "Macvendors" nos devolverá el nombre del fabricante y lo mostraremos en el detalle del *host*.

4.3.2.3.3 Análisis de puertos

Otro de los datos que se explotarán son los puertos origen y destino de los paquetes capturados. Se compararán estos puertos con los más comunes, registrados por la IANA [5] con el fin de obtener los servicios que pueden estar ejecutándose en la máquina.

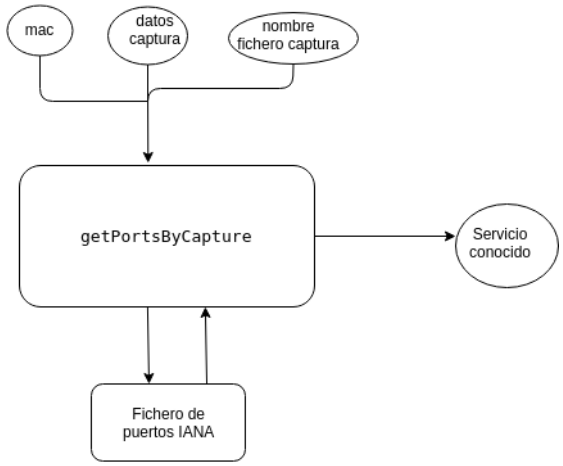

Figura 4.19: Definición gráfica de la función "getPortsByCapture"

En la misma ventana de detalles en la que se muestra el fabricante, mostraremos los puertos, sus posibles servicios y una breve descripción facilitada por la base de datos de la IANA. En la figura 4.19 se representa el diseño con el que cumpliremos este requisito.

4.3.2.4 Utilización del algoritmo clasificatorio Random Forest

Como se comentó en el apartado 2.1.2.1.3 se va a utilizar *machine learning*, en concreto el algoritmo clasificatorio *Random forest*. Este algoritmo nos ofrece la posibilidad de predecir en base a un conocimiento previo, que sistema operativo se esta ejecutando en el equipo analizado.

4.3.2.4.1 Generando conocimiento previo

Para generar el conjunto de datos del que se nutre el algoritmo, se crearán una serie de funciones en Python capaces de categorizar, clasificar y establecer los valores adecuados en campos que no tenemos inicialmente pero que el algoritmo necesita (Sistema operativo).

Se analizaron distintas redes locales con nuestro dispositivo de captura pasiva de datos (colector) generando ficheros JSON en distintos intervalos de tiempo. Definimos inicialmente una función capaz de cargar estos ficheros JSON en memoria y añadir el nuevo campo ”sistema operativo”.

Para establecer valor en este nuevo campo, utilizamos la dirección MAC como medio para identificar los dispositivos de la red y pasamos como parámetro de entrada a la función una relación ”MAC-sistema operativo” asegurando así que se establece este nuevo campo sobre cada paquete de datos capturado.

El siguiente paso será obtener estos datos en un formato concreto (CSV). Generaremos el *dataset* categorizando cada campo con valores enteros (a excepción de la columna resultado, sistema operativo). Se muestra gráficamente este proceso en la figura 4.20.

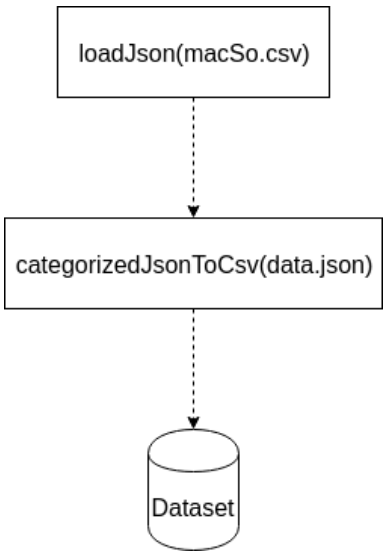

Figura 4.20: Funciones generadoras de Dataset

4.3.2.4.2 Diseño de Random Forest

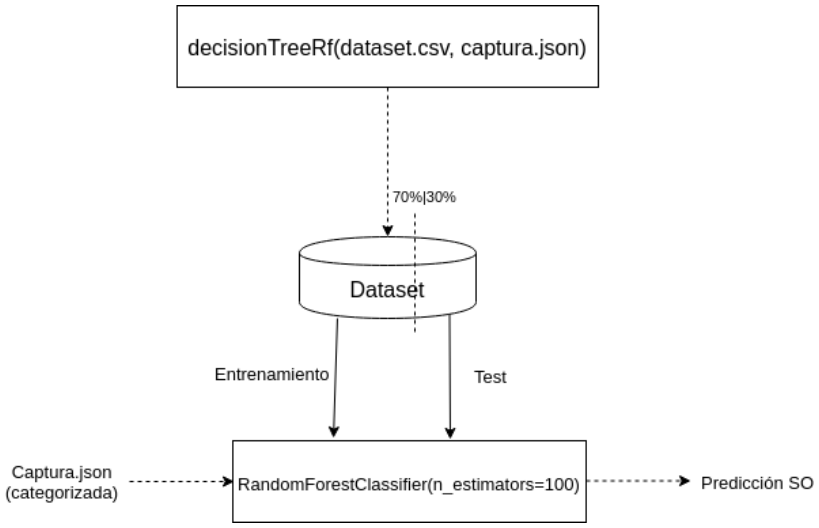

Figura 4.21: Diseño predictivo utilizando Random Forest

En la figura 4.21 se muestra el diseño que se utilizará para la implementación de la predicción del sistema operativo. Se utilizarán 100 estimadores (árboles de decisión binarios) para realizar la clasificación con el algoritmo Random Forest (este, será un parámetro de entrada para el algoritmo).

También podemos observar en la imagen como el algoritmo utilizará el 70% del *dataset* para entrenamiento y el 30% para test. El mismo proceso que se utiliza para el *testeo*, se utilizará para validar las capturas categorizadas que le pasamos al algoritmo, devolviendo el sistema operativo y un porcentaje de acierto (*acuracy*).

4.4 Implementación

Siguiendo las fases clásicas del ciclo de vida en cascada, después de definir los requisitos y el diseño del desarrollo pasaríamos a realizar la implementación de la herramienta. Como se definió en el diseño, la implementación se realizará en dos partes bien definidas. La primera parte se referirá al desarrollo de la aplicación del sistema de análisis y la segunda parte será para el desarrollo de la aplicación que se ejecuta en el servidor web. Como se estructura el código y el funcionamiento de las aplicaciones se detallan en este apartado.

4.4.1 Sistema de análisis

Como se introdujo durante la fase de diseño 4.6, nuestra aplicación de captura pasiva utiliza el patrón de diseño MVC (Modelo-Vista-Controlador), patrón que separa los datos, la lógica y la representación de la aplicación, durante la implementación de la herramienta. El modelo se encontrará en el módulo *modelPF.py*, que incluirá todas la funciones encargadas de realizar operaciones con los ficheros JSON donde se almacenan los datos.

Siguiendo el orden establecido por el patrón de diseño, la vista la tendríamos en el módulo *endPoints.py*, encargado de hacer accesibles los datos almacenados para el usuario (aplicación web).

El controlador de esta aplicación se desarrolla en el módulo *passiveFingerprinting.py*, siendo este el principal, aunque depende de otros dos módulos:

- *config.py*: módulo separado de la aplicación principal que se encarga de abstraer la configuración del código.
- *promisc.py*: módulo encargado de establecer la interfaz de red en ”modo promiscuo”[31], paso previo necesario para la ejecución del análisis.

4.4.1.0.1 Controlador (*passiveFingerprinting.py*)

Este es el módulo principal de la aplicación, encargado de establecer la tarjeta física del dispositivo de escaneo en modo promiscuo llamando al módulo *promisc.py*, realizar la escucha y captura de los datos y llamar al modelo cuando necesite escribir estos datos en disco.

Lo primero que se realizará tras la ejecución de este módulo es comprobar la existencia de la interfaz de red del dispositivo de escaneo, y si se encuentra en estado *promisc*. En el caso de que no se encuentre en este estado, se establecerá.

Una vez la tarjeta se encuentre dispuesta para la captura del tráfico, se ejecutará la clase •Sniffer encargada de crear en un nuevo hilo de ejecución el proceso de

captura y el tratamiento de los paquetes recibidos. Esta clase estará formada por un constructor y por un método *run* que inicia la captura del trafico y lo trata en tiempo de ejecución (captura solo los valores definidos en fase de diseño), enviando cada paquete al método *recolection*. También tendrá un método *join* para indicar mediante un evento que se quiere detener la captura (establece una variable a true), y otro que le indicará al método *run* que detenga la captura , *stopSnifferThread*.

Serán necesarios dos métodos para detener la captura por como funciona el método *sniff* perteneciente a la librería de *Scapy*.

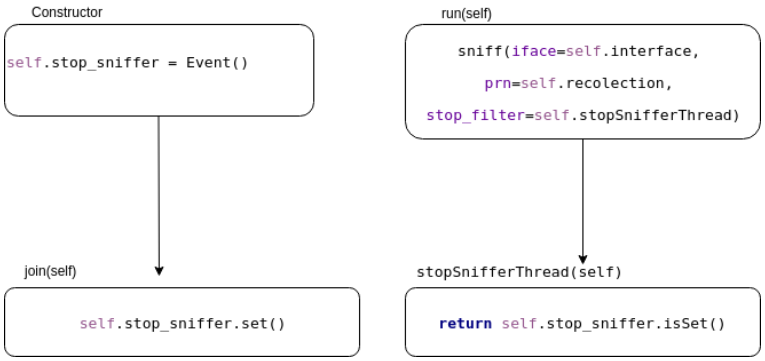

Figura 4.22: Proceso de detención de la captura

En la figura 4.22 se muestran gráficamente los pasos seguidos para detener la captura y destruir el hilo de ejecución. Se utilizará un objeto del tipo *Event()* definido en el constructor, que se establecerá a *True* cuando se quiera realizar la parada. El método encargado de realizar esta acción será *join()*.

Por otro lado tenemos el método *sniff()* perteneciente a la librería de *Scapy* que se encuentra en continua ejecución en un hilo distinto al principal. Uno de los parámetros que recibe este método es *stop_filter* dándole valor a través del método *stopSnifferThread()*, encargado de leer continuamente el estado del objeto *Event(), stop_sniffer*

Tras la ejecución de la captura, se realizarán las llamadas pertinentes a la capa modelo (*modelPF.py*) para la creación del fichero JSON y su nombrado, así como para la creación del fichero *metadata.json* que se muestra en la figura 4.7. Durante toda la captura se controlará los intervalos de tiempo para la división de lo ficheros JSON y su nombrado siguiendo el patrón año-mes-día-hora:minuto.json.

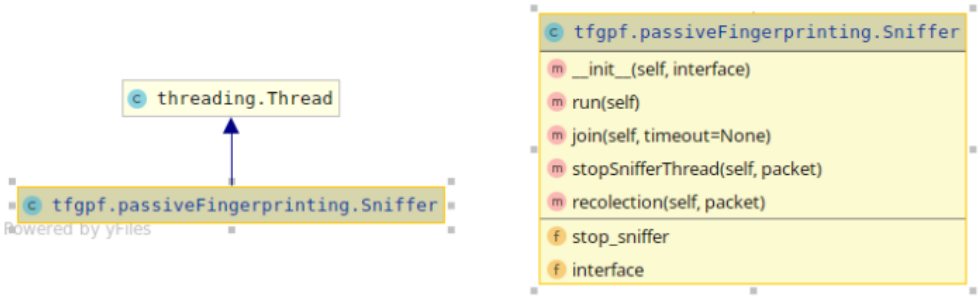

Figura 4.23: UML de la clase Sniffer

4.4.1.0.1.1 Transformación en tiempo de ejecución

El método *sniff* facilitado por la librería de Scapy (2.2.2.6.1), nos permite indicarle como parámetro una función que se ejecutará sobre cada paquete capturado en tiempo real.

Se utilizará este parámetro y se establecerá el método *recolection*, que es un método de la clase *Sniffer*. Este método sigue la jerarquía de capas OSI [35] para el tratamiento de las capturas (física,enlace,red,transporte...), y ademas filtra las capturas que no se han podido tratar y por tanto no han generado un diccionario. También se descartan aquellas de las que no se obtiene la dirección física y se restringe la captura de paquetes a 200 por dispositivo (este valor es configurable).

4.4.1.0.2 Modelo (modelPF.py)

En este módulo se incluirán las funciones utilizadas por el principal para escribir y leer en disco. Tendrá funciones para la creación de los ficheros JSON, para realizar la división de estos ficheros en un intervalo de tiempo, escribir y leer sobre el fichero *metadata.json* y realizar diversas comprobaciones sobre estos.

La estructura diseñada para escribir los datos en ficheros JSON, se crea a partir de listas y diccionarios en Python (imagen 4.26). Almacenaremos de esta forma los campos analizados durante la fundamentación teórica 2.1.1.2, incluyendo el nuevo campo optSort que utilizando una lista en Python almacena las opciones en un orden dado.

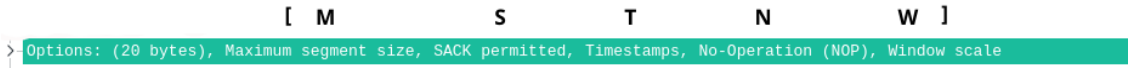

Figura 4.24: Posicionamiento de las opciones en un sistema Linux

Figura 4.25: Posicionamiento de las opciones en un sistema Windows 10

En las imágenes anteriores (4.24 Linux,4.25 Windows), se muestran ejemplos de distintos sistemas operativos y el orden que establecen en sus opciones TCP. También se indica la

- MSS - M
- NOP - N
- Timestamp - T
- Windows Scale - W
- Sack - S

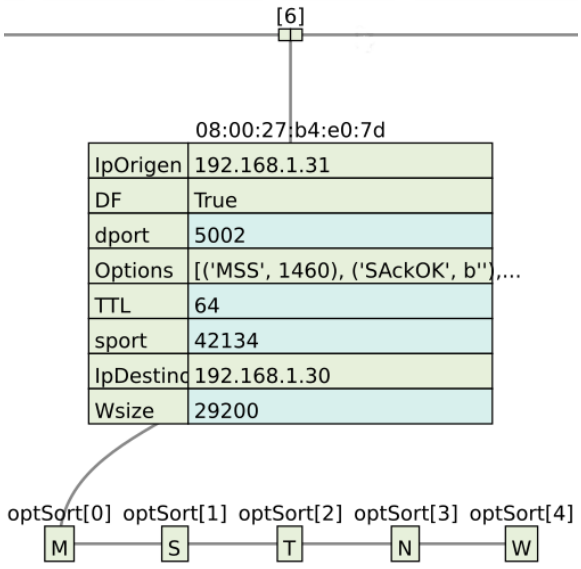

Figura 4.26: Ejemplo de captura con campos TCP realizada por la herramienta

4.4.1.0.3 Vista (endPoints.py)

Como se indicó durante la fase de diseño 4.3.1.2, este módulo será el encargado de exponer los datos capturados a la aplicación web. Para esta tarea se creará un API con dos *endpoints*, que obtienen datos de los ficheros JSON capturados y los hacen accesibles a la aplicación web.

El desarrollo del API REST se simplifica mucho con la utilización del *framework* (conjunto de herramientas diseñadas para desarrollar y organizar un *software* determinado) Flask que con mínimas lineas de código nos permitirá crear una aplicación web funcional.

La metodología REST nos ofrece 4 operaciones básicas:

- GET : consultar o leer recursos
- PUT : editar recursos
- POST : crear recursos
- DELETE : eliminar recursos

La aplicación web solo utilizará la operación GET con la que se obtienen los datos en formato JSON almacenados por los dispositivos de escaneo.

4.4.2 Aplicación web

Para el desarrollo de la aplicación web se utilizará el *framework* Django que tiene su propio patrón MVC 4.3.2.1. El controlador en Django se delega al propio *framework* que a partir del módulo URLconf llama a la función apropiada.

En cuanto se realice una petición sobre la aplicación web Django, este cargará los módulos Python y buscará de forma secuencial sobre la variable ”urlpatterns” del fichero *urls.py*, si existe algún patrón que encaje con la url de la petición. En cuanto se encuentra una coincidencia, Django carga la función asociada en la vista. Se listan los patrones que contiene *urls.py*:

- login : validación de credenciales
- index : página principal
- logout : destruye la sesión actual y vuelve a la página de login
- rasp/<str:ip>/ : carga listado de las capturas realizadas por una Raspberry
- capture/<str:date>/ : Muestra los datos de una captura concreta
- info/<str:mac>/ : Carga información detallada de un nodo en concreto

4.4.2.1 login

La primera petición que se realizará a la aplicación (/) coincidirá en el fichero *urls.py* con el patrón ' ' y producirá la carga en el fichero *views.py* de la función login asociada. En esta función se validará mediante credenciales al usuario para permitirle el acceso a la aplicación.

Django renderizará la template *login.html* y mostrará un formulario con los campos "login" y "password". Estos datos son enviados realizando una petición POST (se evita enviar la contraseña en la propia url), y validados.

Se obtendrá el usuario y la contraseña de la tabla "Users" que nos facilita Django y se compararán con los datos enviados en el formulario.

La contraseña se validará utilizando el método *verify* de la librería *Argon2*, con el que fue cifrada. Si todas la validaciones son correctas, se renderizará la plantilla *index.html* enviando a esta vista los datos del usuario, su imagen que se encuentra almacenada en la tabla usuarios y listado existente de colectores.

4.4.2.2 index

Esta vista solo será cargada si previamente la sesión ha sido validada en la ventana de login, en el caso de que se intente cargar directamente esta vista, se realizará la redirección al formulario de validación. La gestión de las sesiones se realiza utilizando el método *session* que se incluye en la librería *HttpRequest* y facilita la creación, gestión y destrucción de estas.

Antes de que se realice ninguna selección se cargará la estructura básica de la aplicación, donde se mostrarán en un panel lateral las opciones y desplegables, el mapa de red vacío utilizando la librería *Plotly* y los contadores de sistemas a cero.

4.4.2.2.1 Mólulo dataHandler.py

Módulo utilizado para la gestión de los datos tanto para interactuar con el API de la aplicación de análisis como para realizar la representación de la red utilizando la librería *Plotly*.

En este módulo se crearán los métodos encargados de realizar las peticiones GET a los access points del API y obtener los datos de los dispositivos de escaneo. Además se crearán métodos para conocer las conexiones entre nodos utilizando los campos "source" y "target", con la intención de representar estas conexiones mediante aristas.

Este módulo contendrá el método *MapPlotly* encargado de dibujar el mapa de red a partir de una lista de nodos y una lista de aristas (conexiones entre nodos). En este método se tratarán los datos recibidos eliminado los nodos repetidos y uniéndolos entre si con aristas. Se etiquetará cada nodo con su ip que será un enlace a una nueva página con información detallada del nodo.

El enlace generado en cada nodo contendrá información relevante como, la MAC del dispositivo, su IP, el nombre de la captura y el nombre del dispositivo de escaneo, que será

enviada mediante una petición GET. Este método devuelve un (parte de cógigo html) con toda la información de la imagen que será incrustada y cargada en la plantilla *index.html*

4.4.2.3 logout

El objetivo principal de esta vista será cerrar la sesión activa en la aplicación de forma correcta. En cuanto se realice una petición a la url */logout/*, se ejecutará la vista *logout* en el módulo *views.py*, se comprueba si existe una sesión activa y en el caso de encontrarla se destruirá realizando una redirección a la página de login.

4.4.2.4 rasp/<str:ip>/

Si la petición encaja con el patrón *info/<str:mac>/*, se cargará el método *rasp* del módulo *views.py*. Cuando se realice una petición a esta url, se le indicará la MAC de uno de los dispositivos de escaneo (colectores), y este realizará una redirección a la vista index con el nuevo dato. La vista index se recargará por defecto con la última captura realizada por el dispositivo seleccionado, además se mostrará en un nuevo desplegable una lista con las capturas realizadas por el dispositivo.

4.4.2.5 capture/<str:date>/

La aparición del nuevo listado *"Captures"*, nos permitirá la selección de una de las capturas realizadas, enviando una petición GET *capture/<captura seleccionada>* que encajaría con el patrón existente en *urls.py* y que hace que se ejecute en el módulo *views.py* el método *capture*. Como en los métodos anteriores, se valida la sesión actual y una vez validada se hace la redirección a la vista *index* con los nuevos datos seleccionados.

4.4.2.6 info/<str:mac>/

Esta url se utilizará cuando se realice la petición sobre un nodo concreto, con la intención de obtener el detalle del dispositivo. Para cada nodo de la red representado en el mapa, se etiqueta con una url única que contiene información del dispositivo. En ella se almacena la MAC del dispositivo, la dirección IP, el nombre del fichero JSON de captura y el nombre del dispositivo colector.

El método extrae cada uno de los parámetros de la url y los utiliza para obtener la información necesaria. Se realizan llamadas a funciones tales como:

- **macApi**: encargada de obtener el fabricante del dispositivo de red
- **lookupIanaPorts**: se encarga de comparar los puertos obtenidos con el listado publicado por IANA
- **testDecisionTree**: método encargado de la ejecución del algoritmo genético "Random Forest"

4.4.2.7 Módulo de explotación dataMining.py

En este módulo se incluirán los métodos utilizados para extraer información sobre los datos capturados por el dispositivo colector.

- **macApi:** recibirá como parámetro de entrada la dirección MAC del dispositivo, hace una petición REST al API externa "https://api.macvendors.com/<MAC>" y devolverá el fabricante asociado a los 24 primeros bits como se indica en el apartado 4.3.2.3.2
- **getSoByNode:** recibirá como parámetro de entrada un diccionario Python con valores de *don't fragment* (DF) y *time to live* (TTL) y se realizarán comprobaciones de las configuraciones tomadas por estos valores (configuraciones comentadas en el apartado 4.3.2.3.1). Devuelve un diccionario Python con información sobre el sistema operativo inferido.
- **getPortsByCapture y lookupIanaPorts:** recibirá como parámetros de entrada la mac del dispositivo y los datos del fichero json. Se recorre la captura creando dos listas con los puertos origen y destino del dispositivo identificado con la MAC. Se devolverán las dos listas python con los puertos origen y los puertos destino. Estas dos listas se recorren enviando cada puerto a la función *lookupIanaPorts* que realizará una búsqueda en un fichero csv publicado por la IANA y devolverá el servicio y la descripción encontrada por puerto.
- **createDicRawData:** esta función recibirá como entrada los datos obtenidos del fichero json donde se almacena la captura. Se recorrerá esta captura buscando en la paquetería UDP los campos con texto en claro y se generará un nuevo diccionario con la dirección MAC como clave y el texto en claro como valor.

4.4.2.8 Módulo de machine learning mlExplotation.py

En este módulo se crearán los métodos utilizados en la implementación del algoritmo predictivo . Se creará una función *loadCsv*, encargada de realizar la carga en memoria de los datos que se utilizarán tanto par el entrenamiento como para el test del algoritmo (comentados en el apartado 2.1.2.1.2). Se creará otra función *categorizedAllCaptures* que recibirá como entrada todas las capturas de un nodo en concreto y devolverá de forma categorizada todos los campos seleccionados (comentado en el apartado fudamentación teórica 2.1.2.1.2).

Con los datos devueltos por estos dos métodos se ejecuta el método *testDecisionTree*, estableciéndolos como parámetros de entrada. Este método reconfigura el conjunto de datos que se cargó eliminando columnas innecesarias y categoriza los datos recibidos ejecutando la función *dataCategorized*. El método utiliza la librería Pandas para generar un nuevo *dataframe* con los datos de test categorizados.

Antes de continuar con la implementación del método *testDecisionTree* se van a comentar de forma secuencial los pasos seguidos durante el desarrollo del algoritmo predictivo clasificatorio.

1. Carga de datos capturados por nodo (*getAllCaptures()*)
2. Categorizar datos y establecer una estrategia para los datos que faltan (*categorizedAllCaptures()*)
3. Carga de Dataset (*loadCsv()*)
4. Dividir el conjunto en entrenamiento (Train) y validación (Text) (*(train_test_split()*)
5. Crear el objeto Random Forest con 100 arboles de decisión binarios (*RandomForestClassifier(n_estimators=100)*)
6. Construir y ajustar el modelo (fit sobre el objeto creado en el paso previo) que se va a utilizar sobre el conjunto entrenamiento (*(fit()*)
7. Utilizar el modelo creado sobre el conjunto de validación (*predict()*)
8. Validad la precisión (accuracy) que tiene el modelo (*accuracy_score()*)
9. Utilizar el modelo con nuevas entradas de datos para predecir resultados (*predict([entrada no conocida])*)

Continuando a partir del punto cuarto, donde utilizaremos el método facilitado por la librería Sklearn *train_test_split* que se encargará de realizar la división del conjunto de datos en un 70% de los datos para entrenamiento y un 30% de los datos para validación. Indicamos como variable de entrada un porcentaje del 30% para la validación quedándonos con el resto para el entrenamiento. Una relación 70-30 es una práctica común en el aprendizaje máquina.

Construiremos el modelo utilizando la clase incluida en la librería Sklearn *RandomForestClassifier* (el fucionamiento del algorítmo Random Forest se explica en el apartado 2.1.2.1.3). Esta clase recibe como parámetro de entrada la cantidad de árboles de decisión que se van a generar para conformar el bosque (en nuestro caso utilizamos 100 árboles) y nos devolverá un objeto. Este objeto (modelo) se va a ajustar por medio del método *fit* que también forma parte de la librería sklearn utilizando datos de entrenamiento.

Se utiliza el modelo generado sobre el conjunto de validación (test) utilizando el método *predict* del objeto y se valida la precisión obtenida en la predicción utilizando la librería *metrics* de sklearn y el método *accuracy_score*. El valor devuelto se encontrará por debajo de 1 e indicará el nivel de precisión obtenido en la predicción. Este método realiza una comparación de etiquetas (campos categóricos registrados) entre el conjunto de test (30% del dataset) y la predicción obtenida por el método predict del modelo.

Por último, se utiliza este modelo para obtener predicciones con valores que no se encuentran en el conjunto de datos inicial. En el caso de nuestra herramienta los datos de un nodo en concreto se pasan por el modelo de predicción con la finalidad de obtener el sistema operativo que se ejecuta en la máquina.

4.5 Pruebas

Una vez implementada la herramienta vamos a realizar una serie de pruebas en diferentes redes, tamaños y duración de los análisis. La mayor parte de las pruebas realizadas han sido en un entorno de laboratorio controlado, conociendo los operativos de las máquinas en la red desde el principio. Esta red local formada por nueve equipos fue la utilizada para la creación del *dataset* con el que se construyó el modelo predictivo utilizado para entrenar y validar el algoritmo predictivo *"Random Forest"*.

Se ha realizado también un análisis de la red local del laboratorio del departamento TIC de la facultad obteniendo una gran cantidad de datos. Esta prueba resultó útil para probar la robustez de la herramienta ante un análisis intensivo en una red mayor a la utilizada hasta ese momento. El gran inconveniente en este análisis fue el desconocimiento inicial de los sistemas operativos que se ejecutaban en las máquinas haciendo imposible actualizar nuestro conjunto de datos. Por tanto las predicciones en esta red se realizaron a partir del *dataset* inicial que cuenta con unas 1000 capturas clasificadas y etiquetadas con el sistema operativo.

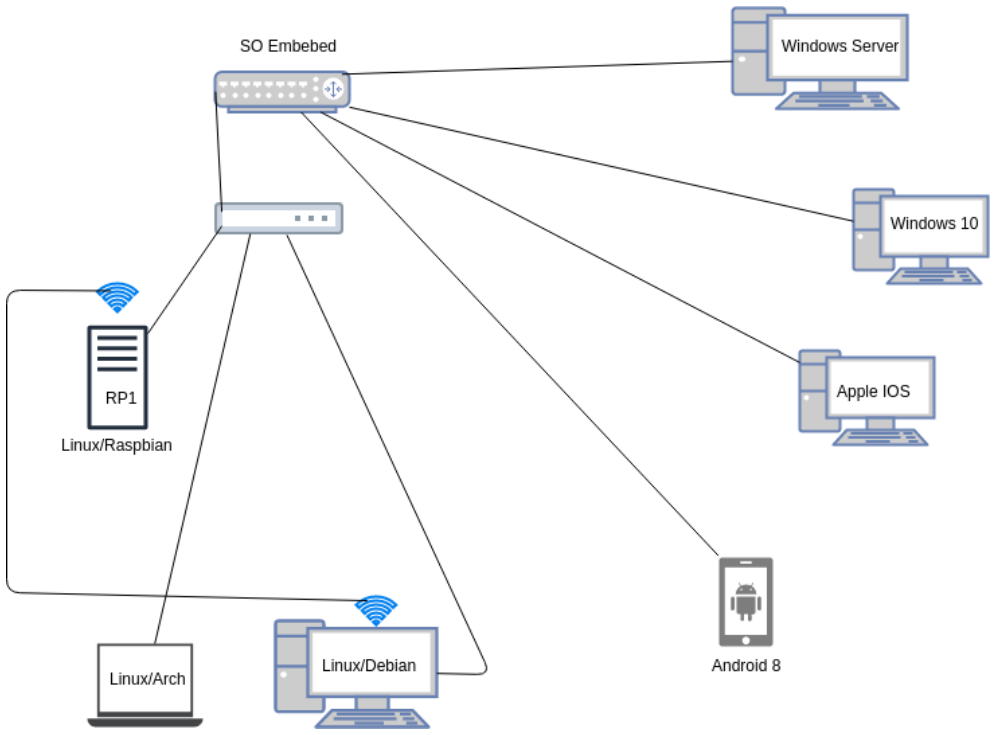

Figura 4.27: Esquema de red, laboratorio controlado

4.5.1 Análisis pasivo de la red de laboratorio controlada

El colector se comunicará con la aplicación web a través de la tarjeta *Wifi*, dejando la tarjeta Ethernet sin configuración IP. Para la ejecución de la captura se utilizan *scripts* escritos en Bash, estos establecerán el tiempo de truncado sobre los ficheros JSON, y configurará el puerto en el que se levantará el servidor web del API. Tras el primer intervalo de tiempo, se puede acceder a los datos capturados desde la aplicación web.

Figura 4.28: Mapa de red generado a partir de los datos capturados con el análisis pasivo

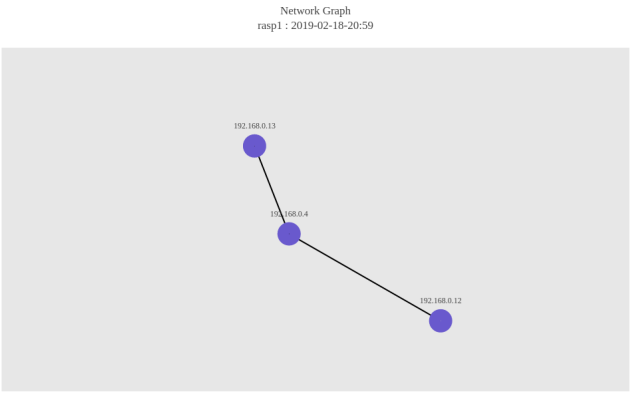

Figura 4.29: Ampliación del mapa de red

En la figura 4.29 se puede observar las conexiones entre nodos, estas serán creadas a partir del campo dirección destino (*target IP*) de la paquetería capturada de forma pasiva. La mayor parte del tráfico de red es de *broadcast* o *multicast*, haciendo imposible conocer el destino real de los datos capturados. En este ejemplo (imagen 4.29) los tres nodos que se unen entre sí son el agregador, el colector y el equipo cliente desde el que se carga la herramienta. Es decir las conexiones se han generado a partir del tráfico capturado durante la utilización de la herramienta.

Figura 4.30: Detalles del nodo con ip 192.168.0.16

En la imagen 4.30 se observa la información detallada del nodo, que tiene la IP 192.168.0.16 asignada. Para acceder a esta información, se pinchará sobre las IPs que se muestran en el mapa de red sobre cada nodo, siendo estos, enlaces a una nueva ventana que amplía su información. En la cabecera de esta nueva ventana se mostrará la dirección MAC del dispositivo y la dirección IP asignada, además tendremos el SO obtenido a partir de los campos capturados TTL y DF. En la fila "*Vendor*" se mostrará el fabricante del dispositivo de red, en la siguiente fila tendremos los puertos obtenidos para los paquetes capturados, tanto los de origen como lo de destino. Se compararán con los puertos publicados por la IANA (puertos conocidos) para mostrar el nombre del servicio y una breve descripción. En la fila "*Other information*" se mostraría la información en texto plano obtenida de la paquetería UDP (este no es el caso).

Por último se mostrará el resultado de la predicción obtenida a partir del modelo predictivo generado por el algoritmo clasificatorio "*Random Forest*" y del dataset dividido en el subconjunto de test y el subconjunto de entrenamiento. Se mostrará también el porcentaje de acierto estimado para la predicción(*Accuracy*, un 90% es un buen porcentaje). En este ejemplo aparecen dos probabilidades, esto sucede por que se evalúa con el modelo de predicción toda la paquetería capturada para este nodo concreto y en este caso el 20% de la paquetería obtuvo una predicción de "SO Embebed", sin embargo el 80% de la paquetería analizada por el modelo de predicción indica que es un "Windows 10".

4.5.2 Prueba simplificada de predicción

Una de las pruebas realizadas para comprobar la eficacia del algoritmo genético fue obtener uno de los árboles de decisión utilizados por el algoritmo "*Random Forest*". Como se explicó en el apartado 4.4.2.8 este algoritmo genera N árboles de decisión a partir de subconjuntos de datos aleatorios que forman parte del *dataset*, y utilizando un sistema de votación escoge la mejor predicción. Simplificando el algoritmo, vamos a probar un árbol de decisión para una sola captura (un host concreto), categorizandolo y comprobando el camino que toma en el árbol.

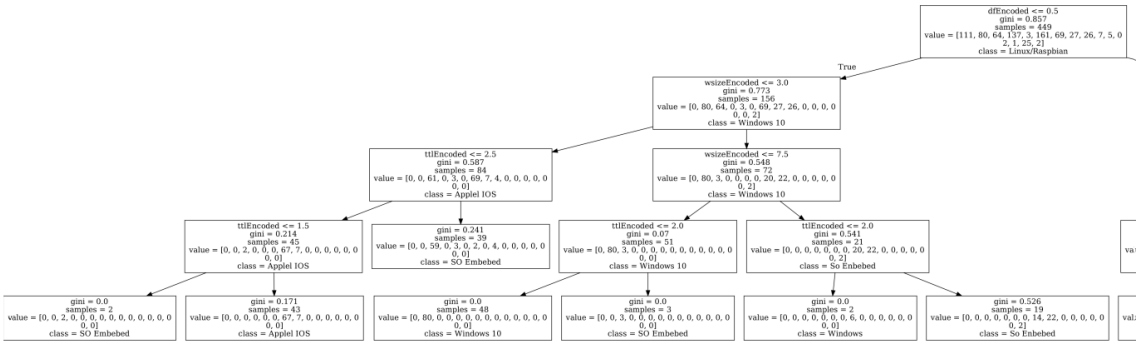

Figura 4.31: Uno de los 100 arboles de decisión generados por Random Forest. Rama izquierda

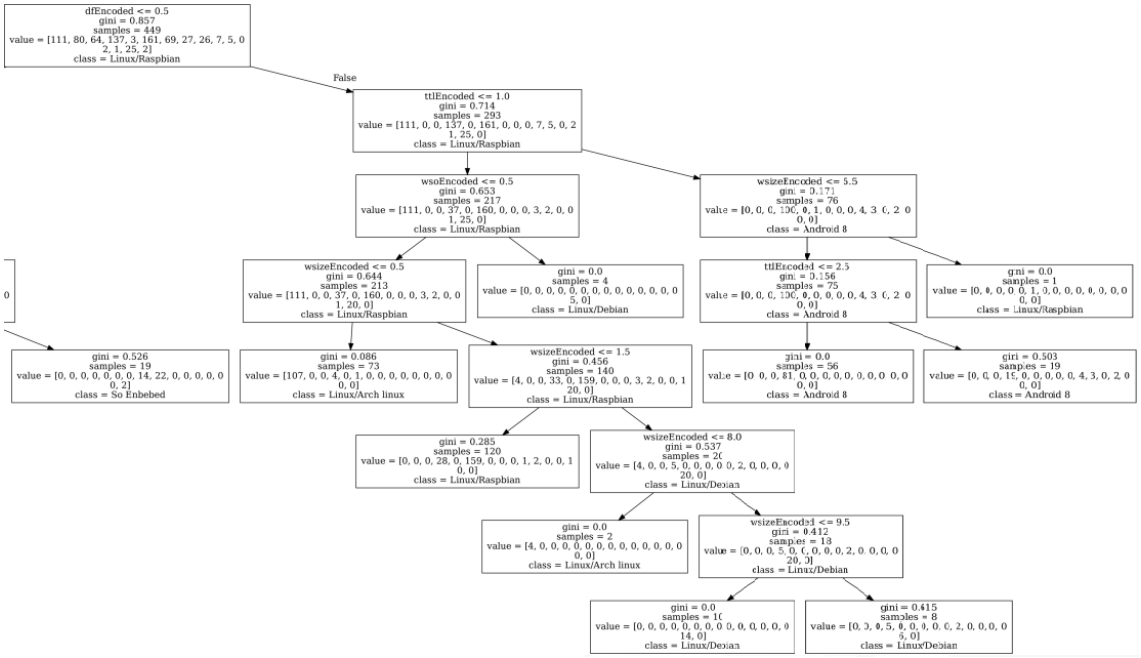

Figura 4.32: Uno de los 100 arboles de decisión generados por Random Forest. Rama derecha

Tomaremos una captura concreta de uno de los fichero JSON, por ejemplo:

```
{"f4:6d:04:97:c4:f5": {"sport": 54915, "Raw": '',
"IpOrigen": "192.168.0.16", "dport": 54915, "TTL": 128,
"IpDestino": "192.168.0.255", "DF": "False"}}
```

Se categorizará la captura obteniendo la siguiente firma:

```
['DF', 'MSS', 'SAKOK', 'Ts', 'NOP', 'Ws. Options', 'TTL', 'Wsize', 'SO']
[ 0,     0,       0,     0,     0,          0,       1,       4]  =  ?
```

Como se puede apreciar los atributos tienen un orden, la firma generada etiqueta estos parámetros posicionalmente. En la figura 4.33 se muestran las decisiones tomadas para la captura del ejemplo terminando en una predicción que será votada entre todos los arboles de decisión generados.

En esta firma se observan varias opciones que no existen. El modelo de predicción generado también tendrá en cuenta estos casos y utilizará estas ausencias para compararlas con otras firmas similares. No es bueno tener siempre los mismos datos para un sistema, esto puede provocar un sobre entrenamiento (overfitting) haciendo que en cuanto la firma cambie lo más mínimo la predicción falle. Tampoco es beneficioso tener para un sistema operativo concreto un gran abanico de posibilidades, esto puede provocar que se obvien otros sistemas distintos por encajar con alguno de estos casos.

En definitiva, lograr un conjunto de datos eficiente es una tarea difícil pero que en cuanto se consigue obtiene buenos resultados.

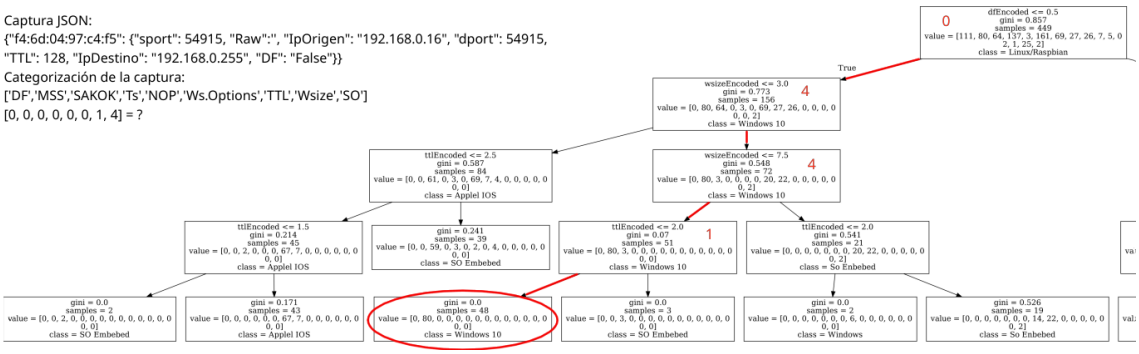

Figura 4.33: Camino seguido por el ejemplo

4.5.3 Análisis pasivo en la red del laboratorio del departamento TIC en la FIC

Para esta captura de datos se dejó el dispositivo colector conectado a la red del laboratorio durante 24 horas. El parámetro de tiempo establecido para la división de los ficheros JSON fue de 1 hora. Finalizada la captura, se acceden a los datos desde la aplicación web, que se encuentra en el agregador y centraliza toda la información.

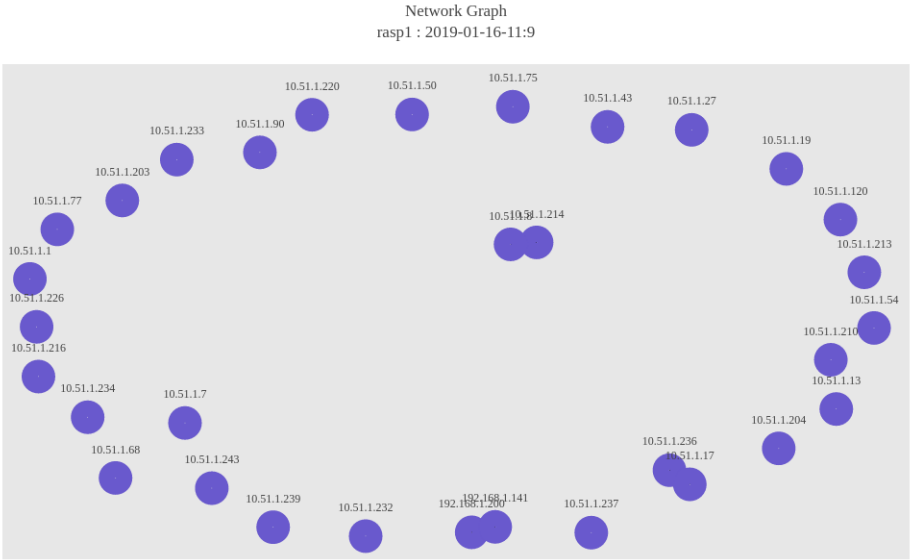

Figura 4.34: Mapa de red del laboratorio TIC

Gracias a los contadores mostrados en la propia aplicación web, sabemos que se han capturado de forma pasiva, datos de 32 equipos, de los cuales (a partir de los parámetros TTL y DF) 7 son Windows, 5 Linux y 20 no han sido posible identificarlos.

Figura 4.35: Contadores de sistemas operativos

Las conexiones establecidas no son muchas, destaca la unión entre el nodo que ejecuta la aplicación web y el dispositivo de escaneo. También se muestran conectados un dispositivo con el "*Vendor*" OKI y un nodo que en el detalle nos indica que el fabricante de la tarjeta de red es Dell y que la aproximación del SO es un sistema embebido, se confirma esta afirmación con la predicción del algoritmo *Random Forest.*

Todos estos datos dan pie a concluir que el dispositivo OKI probablemente sea una impresora de red que se encuentra conectada a un router o switch (DELL) con un sistemas operativo embebido.

4.5.4 Prueba de eficacia del algoritmo Random Forest

Se validará la eficacia del algoritmo predictivo, probándolo sobre una red local cuyos dispositivos conocemos de antemano. De esta forma se podrá validar el resultado ofrecido por el algoritmo *Random Forest* en comparación con el real.

MAC	IP	S.O. Instalado	Predicción Random Forest	Probabilidad
c8:60:00:29:f2:ab	192.168.0.12	Manjaro 4.14.124	Linux/Arch linux	100.0%
b8:27:eb:60:6e:53	192.168.0.4	Raspbian 4.14	Linux/Raspbian	100.0%
d8:c4:6a:a7:7c:51	192.168.0.13	Debian 9	Linux/Arch linux	100.0%
f4:6d:04:97:c4:f5	192.168.0.16	Windows 10	Windows 10	100.0%
c0:ee:fb:d0:82:cc	192.168.0.15	Android 8	Linux/Arch linux	100.0%
00:15:5d:00:10:22	192.168.0.22	Windows Server 2016	SO Embebed	100.0%
54:99:63:70:fd:f7	192.168.0.17	Mac OS 10.5	SO Embebed	10.0%
			Windows 10	90.0%
70:5a:9e:90:68:69	192.168.0.1	SO Embebido (Router)	SO Embebed	98.0%
			Windows 10	2.0%
3c:a0:67:0f:72:4f	192.168.0.19	Smart TV LG WebOS	SO Embebed	100.0%

Tabla 4.1: Resultados del algoritmo Random Forest

En la tabla 4.1 se muestra la predicción y la probabilidad ofrecida por el algoritmo *Random Forest*. En las otras columnas tenemos, listando de izquierda a derecha la dirección MAC del dispositivo, su dirección IP y el sistema operativo que realmente tiene instalado.

Se observa que los resultados son muy precisos para algunos casos (los 4 primeros de la tabla) pero erróneos en otros. En el caso de la máquina con *Windows Server 2016*, el algoritmo nos dice que con toda probabilidad es un sistema embebido (etiqueta que damos a sistemas que utilizan por ejemplo los routers, switches...) siendo esta predicción claramente errónea.

En la siguiente fila nos encontramos con el operativo de *Apple*, que en este caso difiere en la probabilidad ofrecida entre un sistema operativo embebido y un *Windows 10*, ninguna de estas predicciones es acertada.

Para el caso del dispositivo con *Android 8* podríamos considerar válida la predicción por etiquetarlo como Linux (Android esta basado en Linux), sin embargo sabemos que el dataset utilizado contiene datos para este tipo de dispositivos, siendo el posible problema de etiquetado la cantidad de datos capturados para el mismo.

Capítulo 5
Conclusión

El objetivo principal indicado al comienzo de este trabajo era el desarrollo de una herramienta sobre un hardware de bajo coste, escalable y con un uso lo más sencillo posible, que nos permitiese analizar la seguridad de una red utilizando técnicas no intrusivas. Estos objetivos iniciales han sido alcanzados y mejorados logrando unos resultados satisfactorios.

El coste total del proyecto incluirá el presupuesto para el equipo de desarrollo 3.4.1, y el hardware necesario formado íntegramente por el dispositivo colector (Raspberry Pi 3B). Sumando ambos costes obtenemos un total de 22.447€ + 50€ = 22.497€. Teniendo en cuenta que se cuenta con un equipo de 6 personas cualificadas y que el desarrollo se realiza en 3 meses es un precio reducido para un desarrollo software de este tipo.

También se indicó que debía ser un sistema escalable. La aplicación se ha diseñado siguiendo esta máxima y proporciona una solución sencilla aumentando los dispositivos de escaneo. Este dispositivo captura toda la información relevante que fluye por la red sin interactuar con esta y la aplicación web cumple los puntos indicados en el resumen, creando un mapa de red, estableciendo conexiones entre los nodos, obteniendo el número de equipos en la red como sus posibles servicios y deduce los posibles sistemas operativos.

La comunicación entre el dispositivo de escaneo y el servidor de la aplicación web se realiza utilizando la tarjeta wifi incorporada en la Raspberry dejando libre para las capturas, la tarjeta ehternet.

La interfaz de la aplicación web es muy sencilla, presentando gráficamente los datos obtenidos por los dispositivos de captura, cumpliendo de está forma el objetivo de hacer fácil y accesible la utilización de la herramienta al usuario final.

Sobre estos objetivos iniciales se ha propuesto como método de explotación adicional el análisis de los datos utilizando el algoritmo predictivo *Random Forest*. Esta aproximación básica al campo del aprendizaje máquina ha permitido ofrecer información más detallada sobre el sistema operativo que fue uno de los objetivos establecidos. En las pruebas realizadas se muestran carencias durante el uso del algoritmo. Estas son provocadas en gran medida por la cantidad limitada de datos de entrenamiento siendo necesario recopilar un mayor número de datos etiquetados para entrenar el algoritmo y mejorar su precisión.

5.1 Posibles mejoras futuras

Aunque se hayan alcanzado los objetivos especificados inicialmente, caben destacar ciertas funcionalidades que podrían dar valor a la herramienta.

- **Mejorar el tamaño y calidad del dataset:** actualmente el conjunto de datos utilizado para generar el modelo de predicción es muy pequeño (menos de 2000 entradas). Sería conveniente aumentar este conjunto a partir de distintas redes analizadas, buscando mejorar las predicciones realizadas.

- **Generación del conjunto de datos de forma autónoma:** siguiendo con lo comentado en el punto anterior, podría implementarse la funcionalidad de aprendizaje automatizado. Tras el análisis inicial de una nueva red, la aplicación web podría añadir estos nuevo datos capturados al conjunto de datos, permitiendo añadir manualmente el sistema operativo del nodo capturado o que el propio algoritmo haga la predicción y lo registre.

- **Cifrado de las conexiones entre Raspberry y servidor:** aunque esta herramienta obtiene los datos compartidos en una red local, sería conveniente cifrar la conexión entre el dispositivo de escaneo y el servidor web utilizando certificados firmados.

- **Configuración remota del dispositivo de escaneo:** la configuración del dispositivo de escaneo (tiempos de división de capturas, puerto en el que se publican los endpoints...) se hace justo antes de la ejecución del análisis, sería interesante poder realizar esta configuración previa desde la propia aplicación web.

Apéndice A
Apéndices

A.1 Manuales de instalación

En el siguiente procedimiento se va a mostrar como configurar, desplegar y ejecutar las aplicaciones desarrolladas. Estará dividido en dos partes diferenciadas por aplicación desarrollada (aplicación de escaneo y aplicación web).

A.1.1 Dispositivo de escaneo Raspberry Pi

El hardware del dispositivo de escaneo lo forma exclusivamente la placa Raspberry Pi 3B, y a nivel de *software* se instala el sistema operativo Raspbian versión Strech.

A.1.1.1 Instalación de paquetería necesaria

- Paquete *python-pip*: sudo apt-get install python-pip
- Paquete *python3-ipy*: sudo apt-get install python3-ipy
- Paquete *python-flask*: sudo apt-get install python-flask
- Paquete *flask-restful*: pip install flask-restful

A.1.1.2 Configuración del dispositivo de escaneo

Tras la instalación de la paquetería indicada en el apartado anterior, se descarga la aplicación de escaneo que se encuentra subida a un repositorio de Bitbucket:

- git clone https://Juanfreijeiro@bitbucket.org/Juanfreijeiro/tfg.git

Tomando como base la ruta donde se haya descargado la aplicación, nos situamos en el directorio *tfg/tfgpf/deploy*.

Para la configuración de la tarjeta wifi que será el enlace con el servidor web y el exterior, nos valdremos de scripts desarrollados de forma especifica para el despliegue. El primero que hay que ejecutar es 'wifiConfig.sh' que nos solicitará todos los parámetros de red para establecer la conexión wifi. En el paso final nos solicitará un reinicio del dispositivo.

```
pi@fingerPInting:~/tfg/tfgpf/deploy $ ./wifiConfig.sh
Introduce el SSID de la red WIFI: RA13_3D
Introduce la password de la red WIFI:
```

Figura A.1: Salida del script WifiConfig.sh

Tras el reinicio comprobamos el direccionamiento IP.

```
pi@fingerPInting:~/tfg/tfgpf/deploy $ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether b8:27:eb:35:3b:06 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.18/24 brd 192.168.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::d17c:50db:4f94:ca03/64 scope link
       valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether b8:27:eb:60:6e:53 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.4/24 brd 192.168.0.255 scope global wlan0
       valid_lft forever preferred_lft forever
    inet6 fe80::6ea:59df:4a50:9277/64 scope link
       valid_lft forever preferred_lft forever
```

Figura A.2: configuración Ip de la Raspberry

Nuestra tarjeta destinada al escaneo de paquetes es en este caso eth0.

Ejecutamos el script *releaseIp.sh* que se encargará de dejar sin IP a la tarjeta eth0.

```
pi@fingerPInting:~/tfg/tfgpf/deploy $ ./releaseIp.sh
Liberando Ip de la tarjeta de red: eth0
pi@fingerPInting:~/tfg/tfgpf/deploy $ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether b8:27:eb:35:3b:06 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::d17c:50db:4f94:ca03/64 scope link
       valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether b8:27:eb:60:6e:53 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.4/24 brd 192.168.0.255 scope global wlan0
       valid_lft forever preferred_lft forever
    inet6 fe80::6ea:59df:4a50:9277/64 scope link
       valid_lft forever preferred_lft forever
```

Figura A.3: Script releaseIp.sh y resultado tras su ejecución

Para terminar la configuración del dispositivo, ejecutaremos *configAndUp.sh*, *script* encargado de configurar el intervalo de tiempo en el que se dividen los ficheros de captura (la unidad de tiempo son las centésimas de segundo), establecer la tarjeta de red que funcionará de puente con el exterior, puertos donde se publicarán los *endpoints* del API (por defecto es el 5002) y por último lanzar nuestra API en segundo plano.

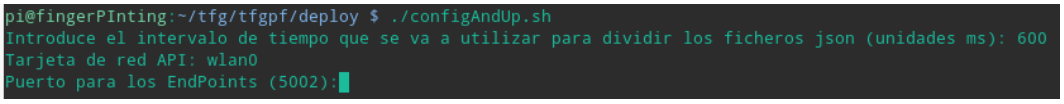

Figura A.4: Script configAndUpScript.sh

A.1.2 Despliegue del servidor web

Tras descargar la máquina virtual del servidor web de Mega: **Servidor web**[1]

Lo descomprimimos y obtendremos una carpeta con la máquina y su disco. La tendremos que añadir en VirtualBox:

```
File -> Import Appliance
```

```
Seleccionamos 'tfg_vagrant_debian9_1507146000873_98856.ova'
```

Una vez este añadida a VirtualBox, para iniciarla nos pedirá los credenciales:

```
user:vagrant
password:abc123.
```

Una vez nos encontremos dentro, tendremos que configurar la red. Nos dirigiremos a */home/vagrant/tfgpf/tfg/server/pfServer/deploy* y ejecutaremos el script *configNetwork.sh*

Figura A.5: Salida del script configNetwork.sh

[1] http://cort.as/-JwhB

Tras todos estos pasos y teniendo en cuenta que en nuestra Raspberry Pi tenemos levantados los *endpoints* que nuestra aplicación va a consultar, solo faltaría ejecutar el servidor web:

```
cd /home/vagrant/tfgpf/tfg/server/pfServer

    python3 manage.py runserver 192.168.0.5:8000
```

Figura A.6: Ejecutando servidor web Django

A.2 Manual de Usuario

A.2.1 Ejecutando un análisis

Para ejecutar una captura pasiva en la red donde se configuró el dispositivo colector, nos situaremos en */tfg/tfgpf* lanzaremos los siguientes comandos:

```
nohup sudo python3 passiveFingerprinting.py [tarjeta de red] &

(ejemplo: nohup sudo python3 passiveFingerprinting.py eth0 &)
```

La captura pasiva de paquetes se lanzará en segundo plano y la salida por pantalla se escribirá en un fichero nohup (para el log de la aplicación se utiliza el fichero output.log).

Para detener el escaneo se utilizará el script:

```
/tfg/tfgpf/stopCapture.sh
```

que detendrá de forma ordenada los trabajos en ejecución.

Si lo volvemos a ejecutar una segunda vez, veremos que el proceso de escaneo ya no se lista. Se debe detener el programa de esta forma para lograr que el fichero de captura escriba el ”cierre” de forma correcta.

A.2.2 Visualizando los datos capturados

Una vez realizada la captura (o pasado el tiempo del primer intervalo configurado en el dispositivo de escaneo), se podrá acceder a la aplicación web y visualizar el mapa de red, el número de equipos encontrados e información sobre cada uno de ellos.

Para acceder a la aplicación web, desde un navegador entramos a la url proporcionada que cargará la página de acceso a la aplicación (figura A.7).

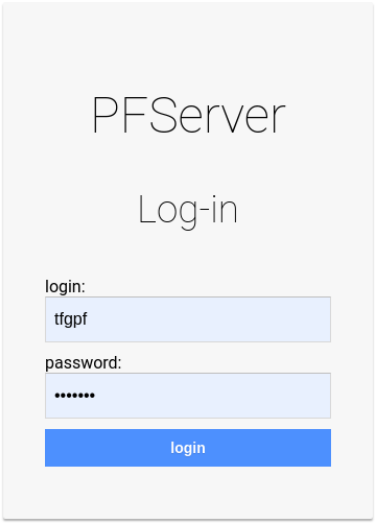

Figura A.7: Acceso a la aplicación web

Después de la validación de los credenciales de usuario, se carga la página principal de la aplicación (figura A.8). En cualquier momento, una vez dentro de la aplicación, podremos cerrar sesión en el boton **log out**.

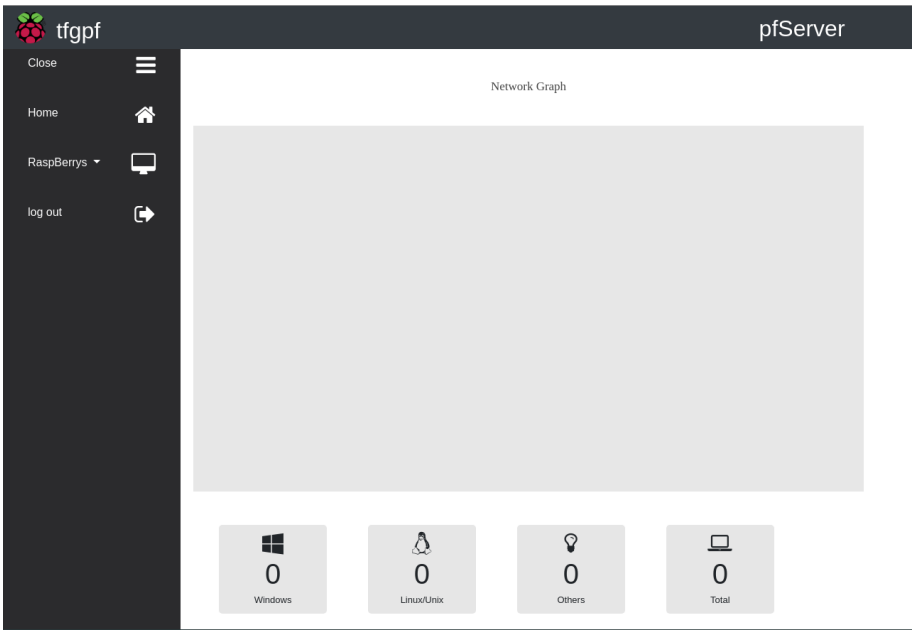

Figura A.8: Página de inicio de la aplicación web

Como se puede ver en la imagen A.8, tenemos un desplegable **RaspBerrys**, en el que se listarán todos los dispositivos de escaneo que se encuentran accesibles.

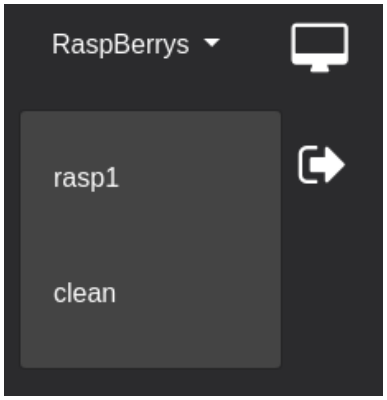

Figura A.9: Listado de dispositivos de escaneo accesibles

En este desplegable (figura A.9) además de las Raspberrys, se mostrará la opción **clean** con la que podremos limpiar los datos que se están mostrando en un momento dado, haciendo que la aplicación vuelva a su estado inicial (figura A.8). Si pulsamos sobre uno de los dispositivos listados, se mostrará un nuevo desplegable **Captures** que muestra un listado con las capturas realizadas por el dispositivo de escaneo. Además en cuanto se pulsa sobre un dispositivo, se carga por defecto la última captura realizada y se dibuja el mapa de red y se muestran los contadores de dispositivos (figura A.10).

Figura A.10: Datos de la última captura realizada por el dispositivo seleccionado

Se muestran diversas opciones para interactuar con el mapa de red, pudiendo descargar como imagen el mapa o hacer ampliación de zonas entre otras.

Al situar el puntero del ratón sobre un nodo, se muestra una información básica indicando la dirección MAC, las ips a las que el nodo ha enviado paquetes y una aproximación del sistema operativo instalado (Figura A.11).

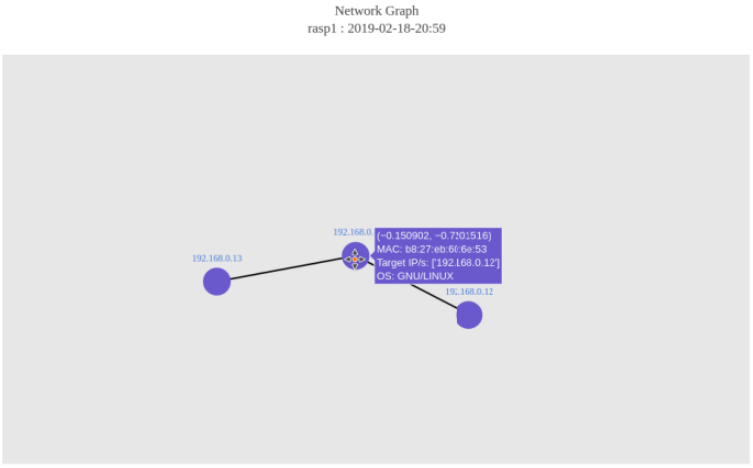

Figura A.11: Zona ampliada del mapa de red. Se muestra información básica del nodo

Sobre cada uno de los nodos presentados se muestra su dirección IP. Esta dirección es un enlace que abre una nueva ventana detallando la información del dispositivo concreto (figura A.12).

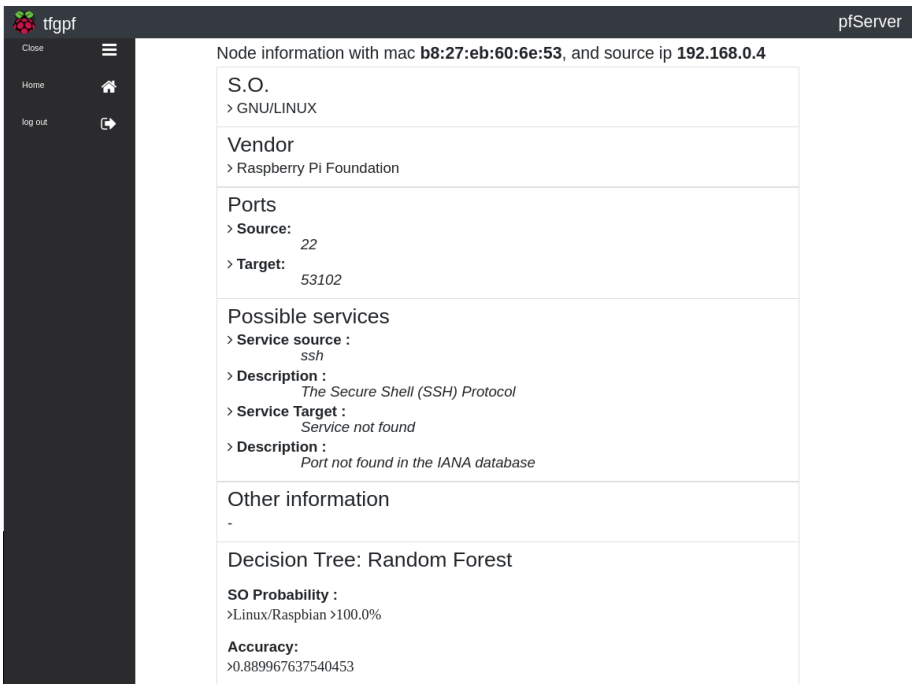

Figura A.12: Información detallada de un nodo

En la imagen A.12 se muestra una tabla con toda la información obtenida de un nodo concreto.

- Dirección MAC e IP: se muestra en la cabecera la dirección física del dispositivo y la dirección de red asignada.
- S.O.: se muestra la primera aproximación al sistema operativo de la máquina. Este dato se obtiene a partir de la comparación de los campos *Don't Fragment* y *Time To Live* de los paquetes capturados para este dispositivo.
- Vendor: se indica el fabricante de la tarjeta de red. Se obtiene a partir de los 24 primeros bits de la dirección MAC y la ayuda de un API externa.
- Ports: se listan los puertos origen y destino capturados en la paquetería.
- Possible services: se comparan los puertos capturados con un listado publicado por la IANA y se muestra el servicio y una pequeña descripción.
- Other information: en el caso de obtener datos sin cifrar en paquetería UDP se mostraría en este campo.
- Decision Tree - Random Forest: datos obtenidos después de tratarlos con el modelo de predicción creado a partir del algoritmo "*Random Forest*" y el conjunto de datos (test y entrenamiento)
  - SO Probability: los paquetes que se capturaron para este nodo se pasan por el modelo predictivo obteniendo un sistema por cada uno.
  - Accuracy: este valor muestra el porcentaje de acierto del modelo para la predicción dada en base al subconjunto del *dataset*, test.

Apéndice B

Glosario de acrónimos

GNU *Acronimo recursivo GNU's Not Unix*

PA *Predictive Algorithm (Algoritmos genéticos).*

IANA *Entidad supervisora de la asignación global de direcciones IP.*

MAC *Dirección física de la interfaz de red, siendo única en cada dispositivo.*

API *Interfaz de programación de aplicaciones.*

REST *Transferencia de estado representacional.*

GET *Operación básica de recuperación en HTTP.*

RAM *Memoria de acceso aleatorio.*

ARP *Protocolo de Resolución de Direcciones.*

Framework *Conjunto de librerías que proveen una funcionalidad.*

Dataset *Conjunto de datos normalmente tabulados.*

IDE *Entorno Integrado de Desarrollo.*

JSON *Formato ligero de trasmisión de datos.*

SSH *Terminal remoto seguro.*

Wifi *Tecnología inalámbrica de red.*

Apéndice C

Glosario de terminos

OS Fingerprinting *Técnica basada en la recopilación de información con el objetivo de identificar el sistema operativo.*

GNU *Licencia Pública General, es una licencia que nos proporciona la libertad de usar, estudiar, compartir (copiar) y modificar el software.*

Machine Learning *conjunto de técnicas que permiten a una máquina realizar tareas propias de un humano. Se basan en el paradigma "Aprender de la experiencia".*

Passive analysis *Termino inglés referido a la técnica que trata de analizar el tráfico de red sin interactuar con ella.*

Algorítmos Predictivos *Nos centraremos en machine learning supervisado pues nuestro objetivo es clasificar sistemas en función de los datos obtenidos.*

IANA *Entidad que supervisa la asignación global de direcciones IP, sistemas autónomos, servidores raíz de nombres de dominio DNS y otros recursos relativos a los protocolos de Internet.*

Dataset *Colección de datos habitualmente tabulada por columnas.*

MAC *Dirección física de la interfaz de red, siendo única en cada dispositivo.*

Intranet *Red no pública (normalmente empresarial) que se basa en las tecnologías utilizadas por los proveedores de servicios de internet.*

Datagrama *Paquete de datos que constituye el mínimo bloque de información en una red conmutada.*

API REST *Biblioteca de funciones que abstrae al cliente de las complejidades de esta y utiliza una metodología de desarrollo basada en HTTP.*

Endpoint *URL que ofrece una funcionalidad del API.*

GET *Una de las cuatro operaciones básicas del protocolo REST utilizada para leer u obtener datos.*

Argon2 *Algoritmo seguro que se basa en la utilización de funciones hash para cifrar contraseñas.*

Host *Equipo conectado a una red.*

Atributo categórico *Atributos clasificables en categorías, por ejemplo, hombre/mujer.*

Selección aleatoria con reemplazo *Se selecciona una muestra de datos al azar, se extrae del conjunto y tras operar con ella la volvemos a dejar.*

Sobreajuste (Overfitting) *Problema en machine learning que provoca que la máquina falle la predicción por no poseer estrictamente los mismos valores que las muestras de entrenamiento.*

Modelo-Vista-Controlador *Patrón que separa los datos, la lógica y la representación de la aplicación.*

Modo promiscuo *Modo en el que se fuerza a la interfaz a aceptar todo el tráfico de red compartido.*

Framework *Estructura establecida para desarrollar y organizar un software determinado.*

Bibliografía

[1] Netresec, "Passive os fingerprinting," *https://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting*, 2011.

[2] I. nacional de tecnologías de la comunicación, "Guía de gestión de fugas de la información," tech. rep., INTECO, 2015.

[3] ESET, "Eset security report 2018," tech. rep., ESET, 2018.

[4] INCIBE, "Decálogo de ciberseguridad para empresas," tech. rep., INCIBE, 2017.

[5] Wikipedia, "Internet assigned numbers authority," *https://es.wikipedia.org/wiki/Internet_Assigned_Numbers_Authority*, 2019.

[6] D. internet program, "Rfc: 791 internet protocol," *https://www.rfc-es.org/rfc/rfc0791-es.txt*, 1981.

[7] D. internet program, "Rfc: 793 transmision control protocol," *https://tools.ietf.org/html/rfc793*, 1981.

[8] A. security, "Network traffic analysis," *https://awakesecurity.com/glossary/network-traffic-analysis/*, 2017.

[9] J. M. Allen, "Os and application fingerprinting techniques," tech. rep., SANS, 2007.

[10] D. C. Plummer, "Ethernet protocol," *https://tools.ietf.org/html/rfc826*, 1982.

[11] M. Zalewski, "p0f v3: passive fingerprinter," *https://github.com/p0f/p0f/blob/master/p0f.fp*, 2014.

[12] M. Zalawski, "Silence on the wire," tech. rep., William Pollock, 2005.

[13] uscybersecurity, "Cybersecurity magazine," *https://www.uscybersecurity.net/risks-2019/*, 2018.

[14] P. R. Ruiz, "Machine learning con python," *https://www.kaggle.com/pablorr10/algoritmos-de-machine-learning-con-python-spa*, 2018.

[15] A. Navlani, "Understanding random forest classifiers in python," *https://www.datacamp.com/community/tutorials/random-forests-classifier-python*, 2018.

[16] W. Koehrsen, "How to visualize a decision tree from a random forest in python using scikit-learn," *https://towardsdatascience.com/how-to-visualize-a-decision-tree-from-a-random-forest-in-python-using-scikit-learn-38ad2d75f21c*, 2018.

[17] J. Fred L. Drake, "Python 3.7.4rc1 documentation," *https://docs.python.org/3/whatsnew/3.7.html*, 2018.

[18] Wikipedia, "Control de versiones," *https://es.wikipedia.org/wiki/Control_de_versiones*, 2019.

[19] P. Biondi, "Welcome to scapy's documentation," *https://scapy.readthedocs.io/en/latest/*, 2019.

[20] macvendors.com, "Api para el descubrimiento del fabricante de la interfaz de red," *https://macvendors.com*, 2018.

[21] D. S. Foundation, "Oficial django documentation," *https://docs.djangoproject.com/en/2.2/*, 2019.

[22] P. Company, "Getting started with plotly for python," *https://plot.ly/python/getting-started/*, 2019.

[23] A. D. Montigny-Leboeuf, "A multi-packet signature approach to passive operating system detection," tech. rep., Defence Research and Development Canada, 2005.

[24] Wikipedia, "Scrum (rugby)," *https://es.wikipedia.org/wiki/Scrum_(rugby)*, 2019.

[25] B. Anderson and D. McGrew, "Os fingerprinting: New techniques and a study of information gain and obfuscation," tech. rep., Cisco Systems, Inc., 2017.

[26] A. Aksoy and M. H. Gunes, "Operating system classification performance of tcp/ip protocol headers," tech. rep., IEEE 41st Conference on Local Computer Networks Workshops, 2016.

[27] K. P. Richard Lippmann, David Fried and W. Streilein, "Passive operating system identification from tcp/ip packet headers," tech. rep., MIT Lincoln Laboratory, 2003.

[28] G. de España, "Boletín oficial del estado," *https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf*, 2018.

[29] Wikipedia, "Desarrollo en cascada," *https://es.wikipedia.org/wiki/Desarrollo_en_cascada*, 2019.

[30] H. Schlawack, "Argon2 documentation," *https://buildmedia.readthedocs.org/media/pdf/argon2-cffi/stable/argon2-cffi.pdf*, 2019.

[31] Wikipedia, "Modo promiscuo," *https://es.wikipedia.org/wiki/Modo_promiscuo*, 2019.

[32] Wikipedia, "Patrón de diseño modelo,vista,controlador," *https://es.wikipedia.org/wiki/Modelovistacontrolador*, 2019.

[33] Wikipedia, "Api rest," *https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional*, 2019.

[34] M. Zalewski, "p0f v3: passive fingerprinter," tech. rep., SANS, 2007.

[35] Wikipedia, "Modelo osi," *https://es.wikipedia.org/wiki/Modelo_OSI*, 2019.