TRABAJO DE FIN DE CARRERA

TITULO DEL TFC: Caracterización de un equipo de usuario mediante técnicas de fingerprinting

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: José Manuel García Carrasco

DIRECTOR: Luís Casals Ibáñez

FECHA: 29 de junio de 2005

Titulo: Caracterización de un equipo de usuario mediante técnicas de fingerprinting

Autor: José Manuel García Carrasco

Director: Luís Casals Ibáñez

Data: 29 de junio de 2005

Resumen

Este trabajo final de carrera pretende ser un estudio sobre fingerprinting. Para ello, se ha recopilado información sobre los tipos de fingerprinting y las características de las pruebas que hacen las aplicaciones que implementan este tipo de función. También se ha realizado un estudio del RFC 1122, requerimientos para un host de Internet, donde se explican como deben estar configurados los diferentes protocolos de las capas de enlace, internet y transporte. Aplicando técnicas de fingerprinting se detectarán las diferencias de implementación de una pila de protocolos TCP/IP en relación con el RFC 1122. También se analiza la influencia de un firewall sobre las pruebas que realiza una aplicación de fingerprinting.

Se han analizado dos aplicaciones de fingerprinting que son ampliamente utilizadas por los hackers: nmap, la más conocida, y Winfingerprinting. De estas aplicaciones se ha estudiado como realizan las pruebas y que tipos de resultados dan.

Por último se describe como se ha realizado una aplicación de fingerpriting. Esta aplicación puede enviar paquetes ICMP y TCP. Se puede variar cada uno de los campos de los que están formados los paquetes para así poder realizar todo tipo de pruebas de tipo fingerprinting.

Title: Characterization of user equipment by fingerprinting techniques

Author: José Manuel García Carrasco

Director: Luís Casals Ibáñez

Date: June, 29th 2005

Overview

This final career work pretends to be a study on fingerprinting. Here, the reader will find information about the different types of fingerprinting that exist, and the characteristics of the test done by applications implementing this function.

There is also a study on the RFC 1122, requirements of an Internet Host, which explains how must the different protocols be configured (link, internet and transport layers). Applying fingerprinting techniques, the differences in the implementation of an TCP/IP protocol stack will be found, comparing with the RFC 1122. The influence of a firewall on the tests done by a fingerprinting application is also analyzed.

Two fingerprinting applications, widely used by hackers, have been analyzed: Nmap and Winfingerprinting. The way the tests are done and the kind of results they give are studied.

Finally, the reader will find a description on how a fingerprinting application has been implemented. This application is capable of sending ICMP and TCP frames. Each of the fields forming the frames can be changed, so that a great variety of fingerprinting tests can be done.

DEDICATORIA:

En este momento quiero acordarme de todas las personas que me han ayudado a ser como soy. Quiero acordarme de toda mi familia, sobre todo a mis padres, hermana y cuñado que tanto apoyo me han dado cuando lo necesitaba. También quiero acordarme de cierto personajillo que acaba de cumplir un año y que cada vez que la veo se me alegra el corazón.

A todos los amigos que he dejado atrás, amigos y compañeros, con los que no se podría acabar una carrera si ellos no estuviesen. Hablo de los colegas de la universidad, Javi, Gabi, Nacho, Mirandita, Alex, bala, etc, a todos y a cada uno de ellos, gracias por ser como sois, los más grandes. También acordarme de la peña de la FP, Javi, Dario, y todos. ¡Mari, te hecho de menos!

También a los amigos que practican conmigo Aikido, sin ellos no me podría haber evadido cada vez que lo necesitaba. Gracias a Dani, David, Josep etc.

Por último tampoco quiero olvidarme de los profesores, de los que ha habido buenos y no tan buenos, hay algunos a los que nunca olvidaré.

Gracias a todos los que me he olvidado, y siento haberlos olvidado.

ÍNDICE

INTRODUCCIÓN

Son varios los usos que los hackers dan a las aplicaciones que pueden hacer fingerprinting. El uso principal de este tipo de aplicaciones es averiguar el sistema operativo que utiliza una posible victima para así tener una base por donde empezar un ataque. Dependiendo del sistema operativo que implementa la victima tendrá unos agujeros u otros. En este proyecto se hace un estudio sobre el fingerprinting, empezando por las bases teóricas sobre las que está construido y acabando en una aplicación que pueda servir para este fin. También se hace un estudio de otras aplicaciones ya desarrolladas anteriormente y que son ampliamente utilizadas.

Muy a menudo, el uso de fingerprinting tiene asociadas connotaciones negativas por el hecho de estar relacionado al mundo de los hackers. Pero, también se pueden encontrar aplicaciones más constructivas para este tipo de técnicas, como la que se describe en el esquema de la figura I.1.

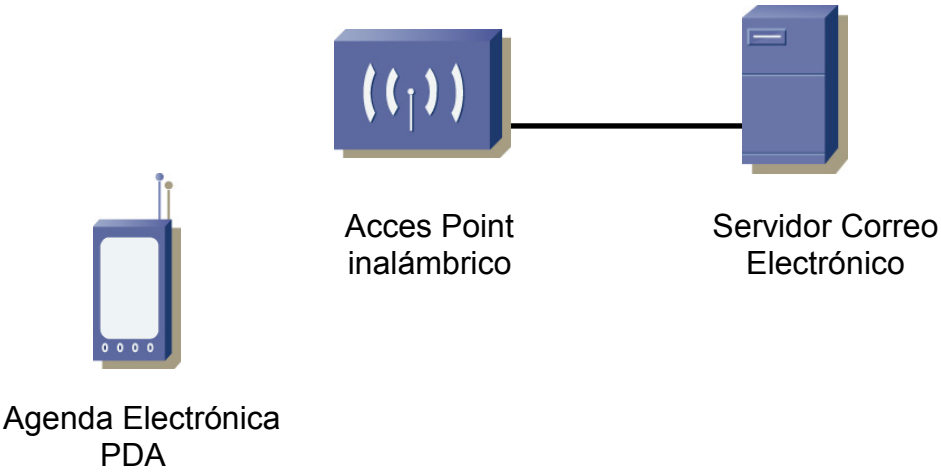


Fig. I.1 Ejemplo conexión inalámbrica

La figura I.1 representa la conexión que se realiza si se quiere leer un correo electrónico desde una PDA conectada a Internet por medio de un punto de acceso inalámbrico. El protocolo TCP sobre el medio radio no funciona correctamente debido a los mecanismos de control de flujo. Estos mecanismos fueron realizados hace mucho tiempo, hay que recordar que TCP se utiliza desde principios de los 70, y fueron diseñados para una comunicación por cable, con baja probabilidad de pérdida de información debida a los pocos errores que hay en la transmisión en este tipo de medio. Esto no pasa cuando se mantiene una comunicación con, por ejemplo, una conexión con GPRS o 802.11. También hay que decir que se están investigando varias soluciones

para el problema anteriormente mencionado como por ejemplo utilizar versiones diferentes del estándar TCP y que intentan adaptarse a este medio, o también poner una especie de Proxy donde acaba la parte cableada y que controle la comunicación. Con este Proxy se intenta "partir" la comunicación en dos, en la parte cableada se utiliza TCP clásico y la otra parte es controlada por el Proxy. Si se necesitan retransmisiones, la información la entrega el Proxy para así poder evitar retransmisiones innecesarias al servidor.

Pero volviendo a la figura I.1 que pasaría si antes de establecer una conexión uno de los dos ordenadores interrogase al otro para poder conocer de alguna manera como tiene implementada su pila de protocolos. Después de tener esta información, se podrían conectar de manera que las opciones que utilicen al mantener la comunicación sean las mejores posibles y así intentar mejorar la comunicación. En el caso práctico expuesto en la figura I.1, si implementásemos en el punto de acceso una pequeña aplicación para realizar fingerprinting, que antes de conectarse le hiciese las pruebas a la PDA, podría estudiarse el comportamiento y estudiar si esto mejora la conexión.

En este trabajo final de carrera se pretende establecer las bases para una aplicación que permita descubrir el tipo de sistema y las características de implementación de la pila TCP/IP para aplicarla en escenarios similares al descrito en párrafos anteriores. Para ello, se han propuesto los tres objetivos principales siguientes:

- Realizar un estudio de las características de las técnicas de fingerprinting, es decir, intentar investigar todo lo posible que es y como se hace este tipo de aplicaciones.
- Estudiar las características de los protocolos TCP e IP. Esto significa que se quiere analizar cuales son la funcionalidades que los protocolos TCP e IP deben incorporar en sus implementaciones, para así poder relacionarlos con las prácticas de fingerprinting que veremos más adelante.
- Especificar un entorno para caracterizar equipos de usuario y comunicación. Para la caracterización se utilizaran aplicaciones de fingerprinting, a partir de los resultados y conclusiones los dos objetivos anteriores.

El contenido de la presente memoria está estructurado como se describe a continuación. En el primer capítulo se explica las bases teóricas del fingerprinting. Se exponen los diferentes tipos que hay y también los test que se realizan. También hay una explicación del RFC 1122, Requerimientos para un host de Internet, donde se exponen las características que los protocolos de un host determinado tienen que implementar de una manera obligatoria, opcional o que no debe utilizar nunca.

En el capítulo 2 hay un estudio de dos aplicaciones que pueden hacer fingerprinting. El estudio consiste en un desarrollo teórico de cómo hacen el fingerprinting, pruebas de funcionamientos realizadas a diferentes ordenadores y por último un análisis de su utilización para nuestros intereses.

En el tercer capítulo se explica el desarrollo que se ha seguido para realizar una aplicación propia que pueda hacer fingerprinting.

El siguiente capítulo contiene las conclusiones a las que se han llegado después de realizar todo el trabajo.

Por último se han adjuntado dos anexos. El en primero se encuentran las tablas de las características mínimas que tienen que tener todos los protocolos de las diferentes capas de un host extraídas del RFC 1122. En el segundo anexo hay un ejemplo de las bases de datos que utiliza NMAP para hacer sus predicciones. Esto puede servir para ver como son las características de los principales sistemas operativos como Windows, Linux, BSD, etc. y también para ver algunas de sus diferencias.

CAPITULO 1. FUNDAMENTOS TEORICOS

1.1. Fingerprinting

En este capitulo se va a explicar las bases teóricas del fingerprinting. Primero veremos su definición y luego se explicarán los diferentes tipos y sus peculiaridades.

1.1.1. ¿Qué es el fingerprinting?

El fingerprinting (“huellas dactilares”) es utilizado principalmente por los hackers para descubrir el sistema operativo que utiliza una posible víctima. Todos los sistemas operativos tienen diferentes implementaciones en sus pilas de protocolos. Estas diferencias son debidas a que cada empresa realiza su producto con personas que obviamente no son las mismas y con una filosofía diferente para alejarse de sus competidores, por lo tanto hacen sistemas operativos cuyas pilas de protocolos presentan detalles de implementación que permiten identificarlas frente al resto. Aquí es donde reside la idea de huella dactilar, cada sistema operativo tiene sus características propias que lo diferencia de los demás.

La metodología que se sigue para poder predecir el sistema operativo de un ordenador es la siguiente. En primer lugar se tienen que hacer una serie de preguntas a los diferentes sistemas operativos, estas pruebas se definirán en los siguientes apartados. Por medio de las contestaciones de los sistemas operativos se pueden ver las diferencias de implementación de los protocolos, y por lo tanto de los sistemas operativos. Estas contestaciones se introducen en una base de datos donde estarán las respuestas y a que sistema operativo pertenecen. Por último, después de haber hecho lo anterior, al realizar las mismas pruebas a un host cualquiera, las respuestas obtenidas se compararán con la base de datos y se intenta correlacionar con el sistema operativo al que pertenezca según la base de datos. Hay que prestar principal atención a los diferentes protocolos, el que más información nos da es TCP, aunque también se utilizan otros como ICMP o ARP. En el protocolo TCP se observa, sobre todo, como se establece la conexión (*three Way Handshake*), y los diferentes campos que forman estos paquetes, que nos da mucha información. Hay dos tipos de fingerprinting, el activo y el pasivo, que se explicarán a continuación.

1.1.2. Fingerprinting Pasivo

En la figura 1.1 se puede ver un esquema de cómo se realiza el fingerprinting pasivo. En el fingerprinting pasivo la aplicación que lo realiza no tiene que enviar ningún paquete, sólo los captura, por medio de unos programas que se

llaman *SNOOF* o *SNIFFERS*. Estos programas tienen que ser muy discretos para que los ordenadores que están manteniendo la comunicación no se den cuenta de lo que está haciendo. Un ejemplo de este tipo de programas es el SNORT.

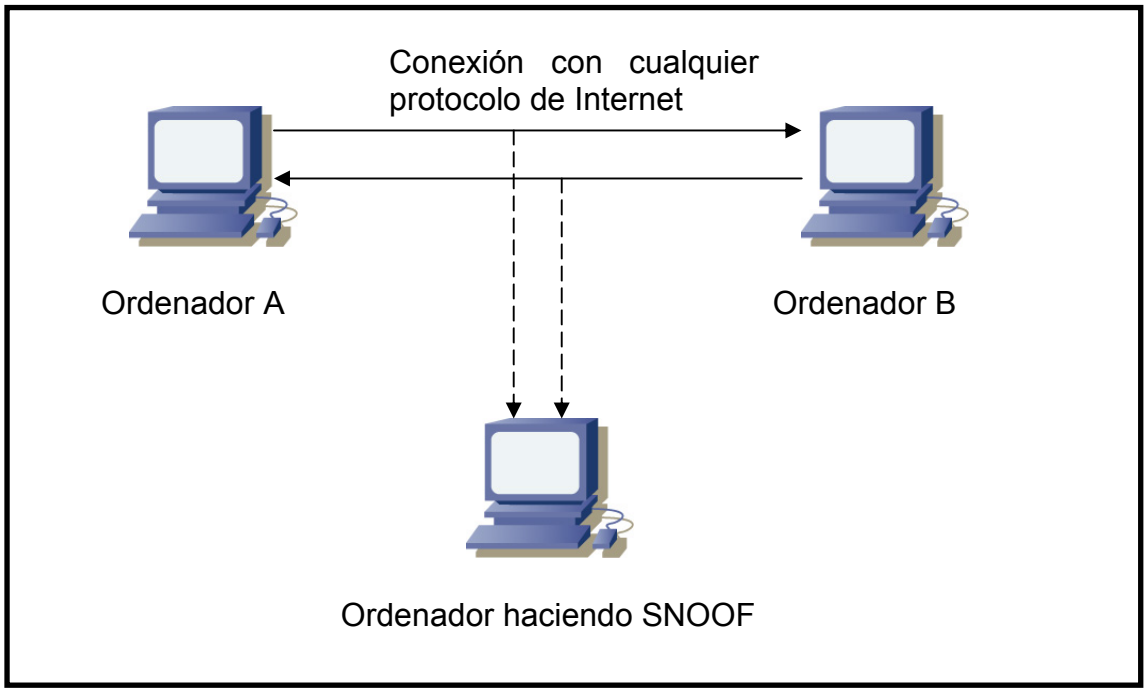


Fig. 1.1 Fingerprinting pasivo

A partir de estas capturas, se puede obtener mucha información. Podemos averiguar la dirección IP de los diferentes posibles host a atacar y deducir los sistemas operativos que llevan los ordenadores debido a las diferencias entre los protocolos, como hemos explicado anteriormente. De los paquetes capturados se observan sus cabeceras, prestando principal atención a las del protocolo TCP. Del protocolo TCP se observan los siguientes parámetros:

- TTL, aunque pertenece a la cabecera IP, es uno de los parámetros a observar.
- Tamaño de la ventana.
- Bit DF (no fragmentación).
- Bit ToS (tipo de servicio).

A partir de estos 4 parámetros se puede decidir las pautas que sigue el sistema operativo y clasificarlo. También se observan otras pautas como el número de secuencia inicial, el número de identificación IP, las opciones TCP o IP y el MMS (tamaño máximo de segmento).

Otros protocolos como el ICMP también son utilizados. En el ICMP se busca el payload del paquete que es diferente en cada sistema operativo.

Para hacer el fingerprinting se utiliza el método explicado en el apartado 1.1.1. Se observan las pautas que siguen estos protocolos por medio de los paquetes capturados, se compara con la base de datos donde estarán definidas estas diferencias y por último se predice el sistema operativo según la base de datos.

1.1.3. Fingerprinting Activo

En la figura 1.2 se puede observar el esquema de la realización del fingerprinting activo. En el fingerprinting activo la aplicación que lo realiza se encarga de enviar paquetes y capturar la respuesta de la posible victima, a diferencia del pasivo, en el que sólo se capturaban paquetes de una manera pasiva.

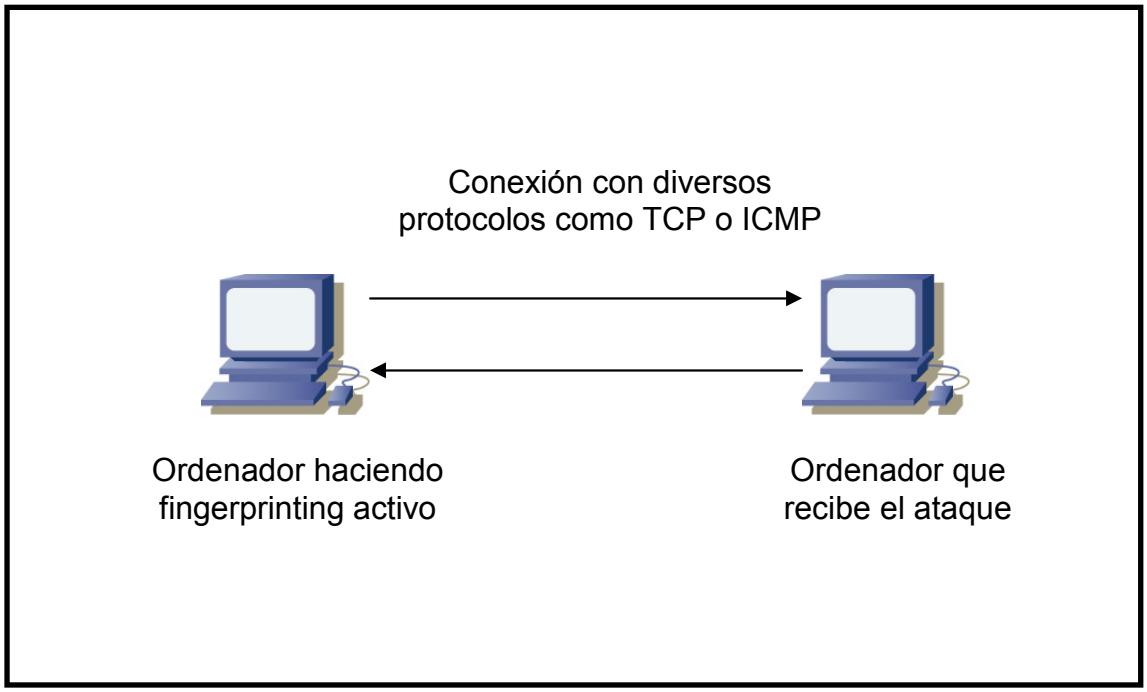


Fig. 1.2 Fingerprinting activo

La ventaja que tiene sobre el pasivo es que al enviar paquetes se tienen más variedad en los test. Por lo tanto, se tienen más pautas donde elegir el sistema operativo y, por lo general, será más exacta la respuesta. Como inconveniente tiene el problema de que es más fácil que sean interceptados. Esto es debido a que se envían paquetes y las diferentes medidas de seguridad, como por ejemplo los firewalls, son más eficientes en estos casos.

Una de las aplicaciones más conocidas para hacer este tipo de fingerprinting es el nmap, que aparte de hacer fingerprinting también tiene otras funcionalidades muy útiles para los hackers. Hoy otros programas como el Winfingerprinting, que es un programa con interfaz gráfica para Windows, o el Queso.

En apartados anteriores se ha explicado la metodología que se realiza para averiguar el sistema operativo que implementa una aplicación que realiza

fingerprinting. El primer paso era hacer una serie de preguntas para ver las pautas que seguían los protocolos. Seguidamente se pasará a explicar las diferentes preguntas o pruebas que se pueden realizar dentro del fingerprinting activo:

- Prueba Fin: Se envía un paquete con bit FIN (o cualquier paquete que no tenga activados los bits de SYN o ACK) a un puerto abierto y se espera la respuesta. Según el RFC 793, especificación del protocolo TCP, no se debería responder pero algunas implementaciones contestan un con reset.

- Prueba de los bits falsos: Se activa uno de los bits que no están definidos en la cabecera TCP (bits 7 o 8 empezando por la derecha) en un paquete SYN y se espera la respuesta. Algunos sistemas operativos mantienen este bit activado en la respuesta, otros resetean la conexión. Actualmente en bit 8 y 9 se utilizan para los campos ECN (notificación de congestión explicita) de control de congestión de TCP.

- Prueba del ISN (numero de secuencia inicial) de TCP: Se trata de buscar pautas en el número ISN de la cabecera TCP que escogen las diferentes implementaciones al contestar a un establecimiento de conexión. Se pueden catalogar en diferentes tipos como por ejemplo 64K, incrementos aleatorios y aleatorios verdaderos, hay otros modelos como los que dependen del tiempo incluso algunos que son constantes, es decir que siempre dan el mismo.

- Prueba IPID (Identificación IP): Muchos sistemas operativos incrementan el valor IPID en cada paquete que envían. Otros utilizan un valor aleatorio y otros utilizan el valor 0 cuando el bit de no fragmentación no está activado.

- TCP *Timestamp*: Los valores de las opciones de *timestamp* pueden ser otro número cuya secuencia puede ser utilizada para la detección de sistemas operativos. Algunos sistemas no soportan estas opciones, otros incrementan el valor a diferentes frecuencias y otros devuelven 0.

- Bit de no fragmentación: algunos sistemas operativos activan el bit de no fragmentación en algunos paquetes que envían. No todos los sistemas operativos hacen esto y otros lo hacen en diferentes casos, prestando atención a esto podemos tener mucha información sobre un posible objetivo.

- Ventana inicial de TCP: Hay que observar el tamaño de la ventana de los paquetes devueltos. Tener conocimiento exacto del tamaño de la ventana nos permite conocer el tipo de sistema operativo Este test nos da mucha información, como algunos sistemas operativos que sólo pueden utilizar un tamaño de ventana.

- Valor del ACK: Las diferentes implementaciones difieren en que valor utilizan para este campo en algunos casos. Por ejemplo, cuando se envía un paquete FIN | PSH | URG a un puerto TCP cerrado, algunas

implementaciones ponen el mismo valor de ACK que el numero de secuencia inicial, otros el numero de secuencia más uno, etc.

- Mensaje error de extinción ICMP: Algunos sistemas operativos siguen lo que sugiere el RFC 1812, requerimientos para un router con IP versión 4, al limitar el número al que varios mensajes de error son enviados. Por ejemplo en algunas implementaciones de Linux se envían a una tasa de 80 en 4 segundos. Una manera de estudiar esto es enviar varios paquetes a un puerto alto UDP al azar y observar el número de puerto inalcanzable recibidos. Este método no es muy utilizado por los diferentes programas de fingerprinting.

- Cuotas de mensajes ICMP: Para un mensaje de puerto inalcanzable casi todas las implementaciones envían de vuelta sólo la cabecera IP más ocho bits. Sin embargo Solaris envía otro bit más y Linux dos más, esto nos permite identificar estos sistemas operativos.

- Integridad en los mensajes de *echo* en ICMP: Como se dice antes algunas máquinas devuelven parte del mensaje original en un mensaje de error de puerto inalcanzable. Dependiendo de cómo lo haga podremos distinguir diferentes sistemas operativos.

- Tipo de servicio: En mensajes de puerto inalcanzable se observa el valor del tipo de servicio (ToS) del paquete que se devuelve. Casi todas las implementaciones utilizan el valor 0, Linux utiliza 0xC0.

- Manejo de la fragmentación: En esta prueba se utiliza el hecho de que las diferentes implementaciones manejan la construcción de los paquetes fragmentados de una manera diferente, algunos escriben las partes nuevas sobre las viejas, otros al revés, etc.

- Opciones TCP: Esta prueba es una de las más útiles. Las características de estas opciones son las siguientes:

    - Al ser opcionales, no todos los host las utilizan.

    - Sabes si el host las implementa simplemente enviándole un mensaje preguntándole las opciones.

  Se pueden enviar varias opciones en un paquete para probarlas de una sola vez.

  Cuando obtienes la respuesta se observa que opciones tiene y cuales soporta, algunas implementaciones como FreeBSD soportan todas, y otras, como algunas de Linux sólo soportan unas pocas. Si algunos sistemas operativos soportan el mismo juego de opciones se puede distinguir por el valor de las mismas. También se pueden distinguir los diferentes sistemas operativos por el orden en el que se reciben y como se aplica el *padding*. Por ejemplo, un Solaris te puede devolver “NNTNWME” que significa:

<no op><no op><timestamp><no op><window scale><echoed MSS>

Mientras que un linux te devuelve “MENNTNW” son las mismas opciones pero las envían de manera diferente

- Resistencia a la inundación de paquetes SYN: Algunos sistemas operativos pararán de aceptar conexiones nuevas si se envían paquetes SYN falsos. Los paquetes falsos provocan problemas con los kernels que resetean la conexión. Algunos sistemas operativos sólo pueden manejar 8 paquetes. Algunos Linux permiten varios métodos para prevenir estos problemas.

No todas las aplicaciones implementan todas las opciones anteriormente descritas, cada una incluye la que más le conviene. Como se ha explicado con anterioridad, al seguir la metodología para averiguar el sistema operativo que se implementa, se captura la respuesta de las diferentes pruebas que realice cada aplicación. Después se compara con las bases de datos y luego se obtiene la solución.

1.2. RFC 1122: REQUERIMIENTOS PARA UN HOST DE INTERNET

1.2.1. Introducción

Como se ha visto en el primer apartado, el fingerprinting se basa en las diferencias de implementación de los diferentes sistemas operativos. Es importante conocer que es lo que los RFC fijan sobre las diferencias entre protocolos. También se tiene que conocer que se tiene que hacer obligatoriamente ante una situación concreta, que no se puede hacer y que es opcional. A partir de todas estas reglas se pueden obtener las diferencias para poder realizar fingerprinting de una manera efectiva.

El RFC 1122 discute los requerimientos de software para un host de Internet, cubriendo las capas de enlace, IP y transporte. El RFC 1123 (*Requeriments for Interenet host – aplication and support*) se encarga de la capa de aplicación.

A continuación se van a definir unos conceptos clave:

- El host: El host es el último consumidor de servicios de comunicación. Es el que ejecuta las aplicaciones en nombre del usuario, empleando redes o servicios de comunicación de Internet como soporte para esta ejecución.

- Sistema de comunicación: Un sistema de comunicación de Internet consiste en redes de paquetes interconectados que soporta la comunicación entre los hosts utilizando protocolos de Internet.

Para que haya una comunicación usando el sistema de Internet, un host tiene que implementar el juego de protocolos por capas comprendido en el conjunto de protocolos de Internet. Un host normalmente tiene que implementar como mínimo un protocolo para cada capa.

Los protocolos por capas que hay en la arquitectura de Internet son los siguientes:

- Capa de aplicación: se distinguen dos categorías:
    - Los que proveen servicio directamente al usuario. Los más usuales son Telnet, Login, SMTP.
    - Protocolos de soporte que proveen funciones de sistema comunes. Los principales son Booting, SNMP, RARP y DNS.
- Capa de transporte: Provee comunicación extremo a extremo. Los dos principales son TCP y UDP.
- Capa internet: Los protocolos de transporte utilizan IP para trasportar datos del host destino al origen. Estos paquetes IP pueden llegar dañados, duplicados, fuera de orden, etc. y las capas superiores son las encargadas de la fiabilidad de este transporte. El protocolo ICMP es un protocolo de control que es considerado una parte integral de IP aunque arquitectónicamente está por encima de IP. ICMP provee de mensajes de error, mensajes de congestión y redirección del primer salto del getaway. También está en esta capa el protocolo IGMP que se usa para el establecimiento de grupos dinámicos de host para multicast.
- Capa de enlace: Esta capa se utiliza para las redes directamente conectadas. También se le llama acceso al medio y hay una gran cantidad de protocolos para esta capa.

Para el correcto entendimiento de este RFC hay dos cuestiones que hay que tener en cuenta. Uno es el enorme crecimiento que tiene Internet, eso provoca que continuamente esté evolucionando y también problemas de control y escala. El otro tema a tener en cuenta es la robusteza que tiene que tener todo sistema. Por eso es importante añadir la máxima información de diagnosis posible al encontrar errores.

1.2.2. Capa de Enlace

La capa de enlace está compuesta por los siguientes protocolos:

- Protocolo de negociación trailer: Sólo se tiene que utilizar cuando los dos dispositivos implementan trailers. El protocolo trailer es una encapsulación de esta capa que reorganiza el contenido de los paquetes enviados en la red física. También se puede encargar de optimizar *throughput* de capas superiores reduciendo la cantidad de datos del

sistema operativo. Sólo los paquetes con unos atributos específicos se pueden encapsular usando trailers y suelen ser pequeñas fracciones de paquetes. Se utiliza simultáneamente con ARP, y puede ser implementado como otro paquete de este protocolo.

- ARP (*Address Resolution Protocol*): la implementación de este protocolo tiene que proveer de un mecanismo para declarar obsoletas las entradas de caché antiguas. Si este mecanismo implica tener *timeout*, también tiene que incluir la manera de controlar su valor. También debería de ser incluido un mecanismo para prevenir el "ARP *flooding*", que es un envío repetitivo de ARP *Request* de una misma IP a una gran velocidad. Hay 4 posibles implementaciones, que a veces se utilizan en combinación, para declarar obsoletas las entradas de caché antiguas:

    - *Timeout*: Periódicamente las entradas de caché se declaran obsoletas, incluso si están siendo utilizadas. Los *timeouts* tienen que ser restaurados cuando se vuelven a obtener las entradas. Los *timeouts* tienen que ser del orden de un minuto.

    - Encuesta Unicast: Se interrogan activamente al host remoto enviando periódicamente un paquete ARP *Request* punto a punto, si el interrogado no responde con un ARP *Replay* después de un número determinado de preguntas se borra la entrada. El tiempo entre preguntas debe de ser cercano al minuto y las el numero al que se borra la entrada suele ser 2.

    - Lo decide la capa de enlace: Si los *drivers* de la capa de enlace detectan algún problema de entrega, entonces se borran las entradas de caché ARP correspondientes.

    - Lo deciden las capas superiores: se provee a las capas de enlace o internet de una llamada que indique un problema de entrega. Cuando se recibe esta llamada, se invalida la entrada de caché.

  La capa de enlace tiene que guardar por lo menos un paquete de cada conjunto destinado a una misma dirección IP no resuelta, y transmitir este paquete cuando la dirección se haya resuelto.

- Ethernet y encapsulación IEEE 802: Las especificaciones para Ethernet se encuentran en el RFC 894, que es el estándar para la transmisión de paquetes IP sobre ethernet, y las de IEEE 802 se encuentran en el RFC 1042, que es el estándar para la transmisión de paquetes IP sobre IEEE 802. Todos los host conectados a un cable de 10 Mbps de Ethernet tienen que:

    - Ser capaces de enviar y recibir paquetes usando la encapsulación del RFC 894.

    - Deberían de ser capaces de recibir paquetes del RFC 1042 mezclados con paquetes del RFC 894.

- Podrían ser capaces de enviar paquetes usando la encapsulación del RFC 1042.

Si pueden enviar de los dos tipos anteriores, ethernet y IEEE 802, tienes que tener un mecanismo *Switch* para elegir cual envía. El RFC 894 tiene que ser el que esté por defecto. La MTU (tamaño máximo del datagrama) para los paquetes Ethernet es de 1500 y para el 802.3 es de 1492.

Entre la capa de enlace y la de Internet tiene que haber una interfaz que tiene las siguientes características. El paquete recibido por esta interfaz tiene una bandera que indica si el paquete entrante fue direccionado a una capa de enlace con dirección broadcast. El paquete enviado por la interfaz entre las capas IP y enlace tiene que incluir 5 bits del campo ToS (tipo de servicio). Por último, la capa de enlace no tiene que enviar reporte de dirección inalcanzable porque no hay entrada destino ARP.

1.2.3.- Capa de Internet

Una de las características más importantes de estos protocolos es la robustez. Los protocolos estándares de esta capa son los siguientes:

- RFC 791 (IP:1): Define el protocolo IP y da una introducción a la arquitectura de Internet.
- RFC 792 (IP:2): Define ICMP, que provee de direccionamiento, diagnosis y funcionalidades de error para IP. Los mensajes ICMP son encapsulados con datagramas IP e ICMP se considera que es parte de la capa IP.
- RFC 950 (IP:3): Define la extensión de subred obligatoria de la arquitectura de direccionamiento.
- RFC 1112 (IP:4): Define IGMP (*Intenet Group Manegement Protocol*) como parte de una extensión recomendada de los host y de los host-gateway para soportar multitcast a nivel IP.

La capa de Internet de un host tiene que implementar IP e ICMP. La capa IP del host tiene 2 funciones básicas, escoger el siguiente salto del host a gateway para datagramas IP de salida y reensamblar los datagramas de entrada. También puede implementar fragmentaciones intencionadas de los datagramas de salida y por ultimo proveer de diagnosis y funcionalidades de error.

Los datagramas entrantes siguen una seria de pasos para averiguar si es correcto o no. Estos pasos son los siguientes: primero se verifica que el datagrama está correctamente formado, luego se verifica que el destino es el host local, seguidamente se procesan las opciones, se reensambla el datagrama si es necesario y por ultimo, se entrega el mensaje encapsulado a la

capa de transporte apropiada. Para los datagramas salientes, el proceso es el siguiente: primero se ponen los campos no rellenados por las capas de transporte, luego se selecciona el primer salto en la red (*routing*), después se fragmenta el datagrama si es necesario y por ultimo se entrega el paquete a la capa de enlace apropiada. En este proceso hay que tener mucho cuidado con el *multihoming*. El *multihoming* se produce cuando un host tiene varias direcciones IP.

A continuación se hará una descripción en profundidad sobre los diferentes protocolos de esta capa. Va a ser analizado lo que tiene que hacer los diferentes protocolos con los datagramas entrantes dependiendo de los campos del paquete:

- IP (*Internet Protocol*, RFC 791):

  Cuando un datagrama llega al protocolo IP y es aceptado, se tienen que analizar los diferentes campos, y dependiendo de los mismos, se hará lo siguiente: Los datagramas de una versión menor a 4 tienen que ser descartados. El host tiene que comprobar el checksum en cada datagrama recibido y descartar los que no sean correctos. En cuanto al direccionamiento se puede comentar una serie de cosas: Hay 6 clases de direcciones, de la A a la E y la D (multicast). Están compuestas por:

  (Numero de red, Numero de host)

  (Numero de red, Numero de subred, Numero de host)

  Dependiendo de las direcciones se tienen una serie de limitaciones en el envío de mensajes. Un host tiene que soportar las extensiones de subred, y también tiene que aceptar las mascaras de red. No se pueden enviar datagramas con una dirección broadcast o multicast como dirección fuente. Un host tiene que descartar los datagramas que no estén destinados para él. El siguiente campo es el de fragmentación. Todos los host tienen que soportar reensamblaje. Cuando enviamos una copia idéntica de un datagrama, un host puede, opcionalmente, retener el mismo número en el campo de identificación en la copia. Un host no puede enviar un paquete con un TTL=0 y tampoco puede descartarlo cuando haya recibido el paquete y su TTL sea menor a 2.

  Uno de los campos más importantes dentro de IP es el campo opciones. Tiene que haber un medio en la capa de transporte para especificar las opciones que estén incluidas en los datagramas IP transmitidos. Todas las opciones tienen que ser enviadas a la capa de transporte o a la capa ICMP, si es un mensaje ICMP, tienen que ser interpretadas las que el sistema entienda y las que no serán descartadas. Seguidamente se pasará a explicar todos los requerimientos para las opciones IP específicas:

  - Opciones de seguridad: En algunos ambientes se requiere que las opciones de seguridad estén en cada datagrama. Hay que

tener en cuenta que algunas de estas opciones descritas en el RFC 791 y RFC 1038 están obsoletas.

- Opciones de identificación de flujo: Estas opciones están obsoletas y no pueden ser enviados paquetes con estas opciones y si se reciben tienen que ser descartados.

- Opciones inicio de ruta: Un host debe poder originar un inicio de una ruta y ser capaz de actuar como punto final de una ruta.

- Opciones para guardar rutas: La implementación de originar y procesar rutas es opcional.

- Opciones *Timestamp*: La implementación y el procesado de esta opción es opcional. Si está implementada se aplican las siguientes reglas:

  - El host origen debe guardar el *timestamp* en la opción cuyos campos de dirección no hayan sido preespecificados o cuyas direcciones preespecificadas sean las interfaces de las direcciones de los host.

  - El host destino debe, si es posible, añadir el suyo a la opción antes de mandar la misma a la capa de transporte o al protocolo ICMP.

  - El valor tiene que seguir las reglas dadas para un mensaje ICMP *Timestmap*.

- ICMP (Internet Control Message Protocol, RFC 792):

  Los mensajes ICMP se agrupan en dos clases, los mensajes de error ICMP (*Destination Unrecheable, Redirect, Souerce Quench, Time Exceded y Parameter Problem*), y los mensajes preguntas de ICMP (*Echo, Information, Timestamp y Address Mask*). Si se recibe un mensaje ICMP desconocido tiene que ser descartado. Los mensajes de error tienen que incluir la cabecera de Internet y, como mínimo, los primeros 8 octetos de datos del datagrama que provocó el error. Un mensaje de error ICMP no será enviado como resultado de recibir un mensaje de error ICMP, un datagrama destinado a una dirección IP broadcast o multicast, si el datagrama enviado es un broadcast de la capa de enlace, un fragmento no inicial o un datagrama donde la fuente no defina un host sencillo.

  Seguidamente se pasará a describir el comportamiento de cada uno de los mensajes ICMP. Se empezará con el ICMP *Destination Unrecheable*. Un host tiene que generar uno de estos mensajes con código 2 cuando el protocolo de transporte no está soportado, y uno con código 3 cuando el protocolo de transporte no puede demultiplexar el datagrama pero no tiene mecanismos de protocolo para informar al que lo envía. Estos mensajes tienen que ser reportados a la capa de transporte. Para los

mensajes *Redirect* un host no puede enviar estos mensajes, sólo son enviados por gateways. Cuando un host lo recibe tiene que actualizar su ruta de acuerdo con su mensaje. Cuando un host recibe un mensaje de este tipo tiene que actualizar su información de ruta. En cuanto a los mensajes *Source Quench*, un host puede enviar este tipo de mensaje cuando prevé que se van a descartar datagramas debido a motivos de buffer, etc. El siguiente tipo de mensajes es el de *Time Exceeded* que siempre que se reciba uno tiene que ser entregado a la capa de trasporte. Continúa el mensaje tipo *Parameter Problem* de los que un host no está obligado a generarlos. Cuando se recibe uno debe ser pasado a la capa de transporte, y, si se desea, puede ser reportado al usuario.

El siguiente tipo de mensajes ICMP es uno de los más usados, es el ICMP *Echo Request/reply*. Un host debería implementar un servidor ICMP Echo que reciba *Echo Request* y enviase su correspondiente respuesta. Y también puede implementar una interfaz de la capa de aplicación para que el manejo de paquetes *Echo Request*, y sus respuestas, sean utilizados como método de diagnosis. Los paquetes *Echo Request* destinados a una IP broadcast o multicast deben ser descartados. Los datos recibidos en un *Echo Request* deben ser enteramente incluidos en el *Echo Reply* resultante. El tipo de mensajes *Information Request/Reply* no debe ser implementado por un host. Los mensajes *Timestamp y Timestamp Reply* pueden ser implementados por un host. Si el host lo implementa tiene que tener en cuenta que el servidor ICMP *Timestamp* retorna un *Reply* por cada *Timestamp* recibido. Algunos casos de este tipo de mensajes tiene que ser manejado con las mismas reglas con lo hace un mensaje *ICMP Echo*. La forma preferida para los *Timestamp* es la que está en milisegundos desde la hora universal de medianoche, pero esto puede ser difícil, por eso puede llevar otro tipo de unidad horaria.

El último tipo de mensajes es el de *Address Mask Request/Reply*. Un host tiene que soportar el primer tipo, y puede implementar todos. Para averiguar la mascara utiliza tres métodos:

1- Se mira la información de la configuración estática.

   Obteniendo la información dinámicamente como un efecto del proceso de inicialización del sistema.

2- Obteniendo la información dinámicamente como un efecto del proceso de inicialización del sistema.

3- Enviando un ICMP *Address Mask Request* y recibiendo el *Reply*.

- IGMP (*Internet Group Management Protocol*):

Es un protocolo usado entre host y gateways en una red sencilla para establecer grupos, en particular grupos multicast. La implementación de IGMP es opcional.

La capa de internet tiene un conjunto de cuestiones específicas que se van a exponer seguidamente. La capa IP tiene que escoger el siguiente salto correcto para cada datagrama que envía. Si la destinación esta en una red conectada, entonces el datagrama se envía directamente al host destino, sino es así tiene que ser enrutado a un gateway de una red conectada. Para decidir si el host destino está en una red conectada se usan unos algoritmos. Para que haya un enrutado eficiente de un grupo de datagramas a una misma destinación el host fuente tiene que mantener una tabla de rutas en las que estén mapeados los gateways de siguiente salto. Esto se hace con una serie de algoritmos. Las entradas de la tabla tienen que tener los siguientes campos: IP local, IP destino, Tipo de servicio y siguiente salto. También tiene que ser capaz de detectar un fallo del gateway siguiente salto que esté listado en sus tablas y escoger uno alternativo. Hay muchas técnicas para hacer esto, como por ejemplo hacer pings para comprobar si el gateway está activo. En la inicialización de la conexión de un host se tiene que configurar una seria de parámetros, que son la dirección IP, la máscara de red y una lista de gateways con sus respectivas prioridades.

La capa IP tiene que implementar el reensamblaje de los datagramas IP. El tamaño máximo con el que se puede reensamblar es el del EMTU_R (MTU efectivo para recibir), que también es llamado tamaño de buffer de reensamblaje. Este valor tiene que ser mayor o igual a 576, y debería ser configurable o indefinido.

Opcionalmente la capa IP puede implementar mecanismos para fragmentar intencionadamente los datagramas salientes. Para esto se designa el EMTU_S (MTU efectiva para enviar) que es el máximo tamaño del datagrama IP que puede ser enviado, para una particular combinación de direcciones IP origen y destino y, a lo mejor, calidad de servicio. El tamaño de este número tiene que ser menor o igual que el tamaño del MTU del interfaz de la red correspondiente a la dirección del datagrama. Si no se implementa este mecanismo se tiene que asegurar que las capas superiores no envían paquetes demasiado grandes.

En cualquier caso práctico, un host tiene que devolver un mensaje ICMP error cuando detecte un error, excepto en los casos en los que devolver un mensaje ICMP error esté específicamente prohibido.

La interfaz entre las capas IP y la de transporte tiene que proveer de un acceso completo a todos los mecanismos de la capa IP incluyendo opciones, tipo de servicio, y tiempo de vida. La capa de transporte tiene que tener también mecanismos para el control de estos parámetros, proveer de un camino para pasarlos de una aplicación, o ambas cosas. Esta interfaz tiene que tener una serie de llamadas a procedimientos como por ejemplo enviar datagrama, recibir datagrama, etc. En estas llamadas se tienen que pasar los parámetros adecuados, estos parámetros son, por ejemplo, las direcciones origen y destino, protocolo, tipo de servicio, etc. La capa IP tiene que pasar los

mensajes ICMP ciertos hacia arriba a la apropiada rutina de la capa de transporte. Para un mensaje de error ICMP los datos que se han pasado para arriba tienen que incluir la cabecera de Internet original más todos los bytes del mensaje original que están incluidos en el mensaje ICMP. Estos datos serán utilizados por la capa de transporte para localizar la información de estado de la conexión.

1.2.4. Capa de Transporte

A continuación se van a analizar los protocolos de la capa de transporte:

- UDP (*User Datagram Protocol*):

  UDP ofrece un servicio de transporte mínimo y da a la aplicación acceso directo al servicio de datagramas de la capa IP. Es utilizado por aplicaciones que no necesitan el nivel de servio de TCP o quieren usar servicios de comunicaciones no disponibles desde TCP. UDP es casi un protocolo nulo, sólo provee de servicio de corrección de errores en los datos (checksum) y multiplexado de puertos. Por lo tanto, una aplicación que funciones sobre UDP debe dar directamente los problemas de una comunicación punto a punto que los protocolos orientados a conexión ya tienen solucionados, como los retrasmisiones, control de flujo, etc.

  Seguidamente abordaremos las cuestiones específicas de este protocolo empezando por los puertos. En UDP se utilizan las mismas reglas con los puertos que en TCP. Si un datagrama llega direccionado a un puerto en donde no se esté escuchando, se debería enviar un mensaje ICMP *Port Unreachable*. Con las opciones IP, UDP tiene que pasar la opción que recibe de la capa IP a la aplicación transparentemente. Una aplicación tiene que ser capaz de especificar las opciones IP para ser enviadas en el datagrama UDP y este tiene que pasar esta opción a la capa IP. Lo siguiente que se va a tratar son los mensajes ICMP. UDP tiene que pasar a la aplicación todos los mensajes ICMP error que recibe de la capa IP, esto se cumple con un "*upcall*" de la rutina Error_report. A continuación se hablará del checksum, un host tiene que implementar la facultad de generar y validar el Checksum UDP. Una aplicación puede ser capaz de controlarlos opcionalmente. Si se recibe un paquete con checksum que no es cero y es incorrecto se desecha el datagrama. En cuanto al UDP *multihoming*, cuando un datagrama es recibido, su dirección específica tiene que ser pasada a la capa de aplicación. Por último se hablará de las direcciones inválidas. Un datagrama UDP recibido con una IP incorrecta tiene que ser descartada por UDP o por la capa IP. Cuando un host envía un datagrama UDP, la dirección fuente tiene que ser la dirección IP del host.

  UDP tiene que tener una interfaz entre el protocolo UDP y la capa de aplicación. Esta interfaz de aplicación a UDP tiene que proveer de todos

los servicios del interfaz IP/Transporte del que se ha hablado en la sección 1.2.3 de este documento.

- TCP (*Transmission Control Protocol*):

  TCP es el principal protocolo de transporte de circuito virtual de todos los del conjunto de Internet. TCP es usado por aquellas aplicaciones que necesitan un servicio de transporte orientado a conexión.

  Seguidamente se va a comentar una seria de características que en el RFC llaman paseo por el protocolo. El primer tema que se va a tratar es el de los puertos, en TCP los puertos con rango de 0 a 255 (puertos "*well-Known*") son usados para el acceso de los servicios que están estandarizados en Internet. El resto puede ser utilizado libremente. En cuanto a la utilización del *push*, cuando una aplicación envía una serie de llamadas *SEND* sin activar el bit *PUSH*, TCP tiene que agregar los datos internamente sin enviarlos. En el caso de recibirlos, internamente se tienen que encolar los datos sin pasarlos a la aplicación. El tamaño de la ventana tiene que ser tratado como un número sin signo, o si no, los tamaños grandes aparecerán como números negativos y TCP no funcionará. En cuanto al puntero urgente, TCP tiene que soportar una secuencia de datos urgentes de cualquier tamaño. TCP tiene que ser capaz de recibir una opción TCP en cualquier segmento. Tiene que ignorar sin error cualquier opción TCP si no la implementa. También tiene que estar preparado para manejar tamaños de opciones ilegales sin romperse.

  A continuación se hablará de las opciones de tamaño máximo de segmento. TCP tiene que implementar las opciones de enviar y recibir. TCP debería enviar una opción MSS (Tamaño máximo de segmento) en cada SYN cuando él reciba un MSS que difiera del que está por defecto (536), y puede enviarlos siempre. Respecto al cheksum, y a diferencia del checksum UDP, el TCP nunca es opcional. En el que se envía en el paquete debe ser generado y el que lo recibe debe mirar si es correcto. El número de secuencia inicial debe usar el especificado por la selección conducida por reloj. TCP debe soportar el intento de apertura simultánea. En cuanto al cierre de la conexión TCP lo puede hacer de dos maneras, de la normal (*FIN handshake*) o con un *reset* (bit RST activado). Para calcular el *Timeout* de retransmisión se tiene que utilizar un algoritmo que combine "*slow start*" y "*congestion avoidance*". El receptor TCP no debe encoger la ventana de transmisión. Sin embargo, un emisor TCP debe ser robusto contra el encogimiento de ventana que puede provocar que la venta útil se convierta en negativa. El valor TTL usado al enviar segmentos TCP debe ser configurable. Aunque no es estrictamente necesario, un segmento TCP debe ser capaz de encolar los segmentos que estén fuera de orden.

  Seguidamente se tratará las características específicas de TCP. Primero hay que empezar por el cálculo del *Timeout* de la retransmisión. Un host TCP tiene que utilizar los algoritmos de Karn (algoritmo para elegir cual es la medida correcta de un RTT y así no corromperla), y el algoritmo

Jacobson (algoritmo para hacer las medidas del RTT). Esta implementación debe incluir un "*backoff*" exponencial para sucesivos valores RTO de un mismo segmento. Otro tema a comentar es cuando se tiene que enviar un segmento ACK. Se puede utilizar el sistema conocido por ACK's retardados, consistente en esperar a más de un segmento recibido para enviar el ACK y así reconocer varios segmentos con un mismo ACK. TCP puede implementar ACK's retardados, pero el retardo no puede ser excesivo, de hecho el máximo retardo debe ser menor a 0.5 segundos, y se tiene que reconocer cada dos segmentos como máximo. TCP debería implementar el algoritmo de Nagle, este algoritmo se encarga de agrupar los segmentos pequeños para enviarlos en un paquete. De todas formas, también debe de haber una menara para que la aplicación pueda desactivar este algoritmo en una conexión individual. Excesivas retransmisiones de un mismo segmento indica que ha habido algún fallo en el host remoto o en el camino, que puede ser de larga o corta duración. Hay que utilizar unos procedimientos para manejar la excesiva retrasmisión de segmentos de datos. Los implementadores pueden incluir "keep-alive" en sus sistemas, aunque esta práctica no esta universalmente aceptada.

La interfaz entre las capas TCP y aplicación tiene que tener una serie de características que se van a explicar en las siguientes líneas. En esta interfaz tiene que haber un mecanismo para reportar los errores que son muy grabes a la aplicación. Generalmente se asume que una rutina se encargará de esta entrega a la capa de transporte de una manera asíncrona. La capa de aplicación también tiene que ser capaz de especificar el tipo de servicio para los segmentos que son enviados en una conexión. También la aplicación debería ser capaz de cambiar esta calidad de servicio durante la vida de la conexión, aunque esto no lo piden explícitamente. TCP debería poder pasar el valor ToS sin tener que cambiar la capa IP. El ToS puede ser especificado independientemente de cada sentido de la conexión. Por último, algunas implementaciones tienen incluidas ciertas llamadas que sirven para poder vaciar las colas de datos que se tienen que enviar pero que todavía están pendientes de su correspondiente ACK en la ventana de paquetes enviados. Esto es útil para algunas funciones de Telnet.

1.3. Firewalls

Hasta este apartado hemos visto que es y como se hace el fingerprinting de una manera teórica. También se ha visto como están hechos los diferentes protocolos para ver sus puntos débiles y así poder atacarlos. En los siguientes puntos se va a analizar como los firewalls actúan en contra del fingerprinting. En el primer apartado explicará desde una visión de seguridad como los firewalls intentan evitar que se produzca. En el segundo apartado se estudiarán los firewalls comerciales para ver si afectan a una aplicación que quiera realizar fingerprinting.

1.3.1. Firewalls & Fingerprinting

En este apartado se va a analizar desde un punto de vista de seguridad cuales son las repercusiones que tiene un firewall a la hora de hacer fingerprinting a un ordenador cualquiera. Se van a comentar las contramedidas, es decir las acciones que deben hacer los firewalls, para que no se pueda saber el sistema operativo del ordenador al que se le hace la prueba con diferentes ataques relacionados con el fingerprinting.

El primer ataque que vamos a analizar es el de la exploración de los puertos. Este ataque consiste en intentar averiguar cuales de los puertos de un host están abiertos para poder entrar en sus sistema por uno de ellos. Para hacer esto se pueden utilizar varios métodos. Uno de estos métodos consiste en intentar conectarse a los puertos por medio de la aplicación Telnet. Si el host acepta la conexión vía Telnet por un puerto es que lo tiene abierto, si no, es que lo tiene cerrado. Hay otras maneras, como la que utiliza nmap y que veremos en siguientes apartados y que consiste en enviar mensajes TCP o UDP a los puertos y, dependiendo de la respuesta saber si está abierto o cerrado.

Hay varias contramedidas para intentar parar este ataque. La más utilizada consiste en detectar cuando se está produciendo una exploración de puertos mediante programas como Real Secure o Snort. Este tipo de programas son de la clase *SNOOF* o *SNIFFER* como los descritos en el fingerprinting pasivo, pero utilizados para detectar ataques como el explicado anteriormente. La mayoría de firewalls puede detectar este tipo de ataques ya que es muy sencillo. Para prevenir este tipo de ataques es aconsejable desactivar servicios innecesarios para que el firewall tenga bien acotado cuales puertos tienen que ser utilizados por algún servicio y cuales no.

El siguiente ataque que se va a tratar es el fingerprinting activo. Para su detección se utilizan programas SNOOF. Al ir estudiando el tráfico, se puede ver cuando se produce un número extraño de paquetes y a un puerto no habitual, entonces sabremos que se está produciendo este tipo de fingerprinting. Para la prevención, algunos expertos nos dicen que en algunos Sistemas Operativos se puede cambiar algún parámetro del mismo, pero que esto puede provocar incompatibilidades con alguno de los RFC que controlan los protocolos, y puede que alguna aplicación no funcione correctamente por este motivo. Que la seguridad por oscuridad no funciona es ampliamente aceptado por los expertos. La seguridad por oscuridad es un tipo de seguridad que se utiliza mucho y en la que se pretende ocultar la mayor cantidad de información posible de la red, servidor, o de lo que se esté intentando proteger. Se pretende que el atacante no tenga datos útiles para poder atacar pero esto no es efectivo porque nunca se puede ocultar totalmente una red o sistema y cuando te encuentra un hacker malicioso entonces ya no hay solución. Es muy importante pasar lo más desapercibido posible, y esto no se consigue con este tipo de seguridad.

Con el fingerprinting pasivo tenemos las mismas contramedidas que el activo y también las mismas recomendaciones. En este tipo de ataques no se tiene que

observar los paquetes que circulan porque son los normales de una comunicación entre ordenadores. Se tiene que poner especial énfasis en intentar descubrir el programa SNIFFER que está capturando paquetes y esto no resulta sencillo.

El último ataque que se quiere frenar es el de consultas ICMP. Este tipo de paquetes nos puede proporcionar mucha información como mascaras de red, información sobre el sistema operativo, etc., como hemos visto anteriormente. Para la prevención de este tipo de ataques se tiene que intentar bloquear los tipos de paquetes ICMP que dejen pasar información, así como restringir la entrada de *Timestamp* y *Addres Mask*. También es muy útil utilizar programas *SNUFF*.

1.3.2. Firewalls Comerciales

En este apartado se han seleccionado algunos firewalls comerciales y se han estudiado sus características para ver cual es su posible interacción con el fingerprinting. El primer firewall analizado es de la marca cisco.

- Cisco PIX Security Appliance Software Version 7.0:

  Este software nos proporciona:

  - Inspección profunda en servicios HTTP, FTP, ESMTP y otros.
  - Bloqueo de mensajería instantánea y redes P2P.
  - Policía basada en flujo.
  - Servicios de firewall virtual.
  - Servicios de seguridad par Wireless 3 G.

  En cuanto al protocolo ICMP, permite el uso seguro de ICMP, rastrea el estado las conexiones de los servicios y también provee de control adicional para los mensajes ICMP error.

  Este firewall se monta en los switch o en un kit de seguridad donde se hagan VPN (*Virtual Private Network*), también tiene AAA (*Authentication Authorization Accounting* para *VPN*), y varias cosas más.

- 3com:

  La empresa 3com ofrece varios productos que van desde la protección perimetral hasta firewalls por VPN, routers, routers wireless con su firewall, etc. No explican nada sobre lo que le permite hacer al protocolo ICMP o no. Los firewalls que implementa son de filtrado de paquetes de última generación. Estos firewalls están monitorizando continuamente la comunicación, controlan los protocolos, los parámetros IP, los datos, etc.

> Al controlar todos estos datos tiene una visión amplia de la conexión y cuando ve algo que no esté dentro de sus parámetros normales, entonces corta la comunicación.

Se ha buscado información sobre otros firewalls pero lo ofrecido es prácticamente lo que implementa el firewall 3com. Como se puede ver, las tendencias actuales intentan controlar todo el tráfico que entra o sale del terminal a proteger, y dependiendo de las conexiones, del flujo de datos y de todo el contexto, detectan si alguien está intentando “colarse” en el ordenador o no, y cuando lo detectan cortan la comunicación, o ya no la permiten de un principio.

Como se puede observar, los firewalls limitan mucho un posible ataque por medio de técnicas de fingerprinting debido a que un ataque de este tipo es muy conocido y muy llamativo, como por ejemplo establecer una conexión TCP en cada puerto para ver si el puerto está abierto o cerrado. Por lo tanto los resultados serán mucho más correctos si se hace fingerprinting a un ordenador que no tenga ningún firewall actuando, de hecho si lo tuviese muy probablemente no tendríamos respuesta de la aplicación que realiza el fingerprinting. Con NMAP, por ejemplo, primero hacemos un prueba ping para saber que hosts están activos, estar activos significa que alguno de sus puertos está escuchando, y, por lo tanto muy probablemente no hay ningún firewall actuando, para después hacer una prueba de fingerprinting para intentar averiguar su Sistema Operativo.

CAPITULO 2. APLICACIONES FINGERPRINTING

En el capítulo anterior se ha estudiado la base teórica necesaria para poder entender la realización de las técnicas de fingerprinting. A continuación se va a intentar profundizar en las aplicaciones ampliamente utilizadas como están implementadas y si son válidas para los fines necesitados.

2.1. NMAP

La primera aplicación que se va a estudiar es una de las aplicaciones más utilizadas por los hackers, el NMAP. En la figura 2.1 se puede ver las utilidades que tiene este programa, como por ejemplo, escanear puertos de varias maneras diferentes, ver que hosts están activos, y muchas más, aunque este documento se va a centrar en el fingerprinting.

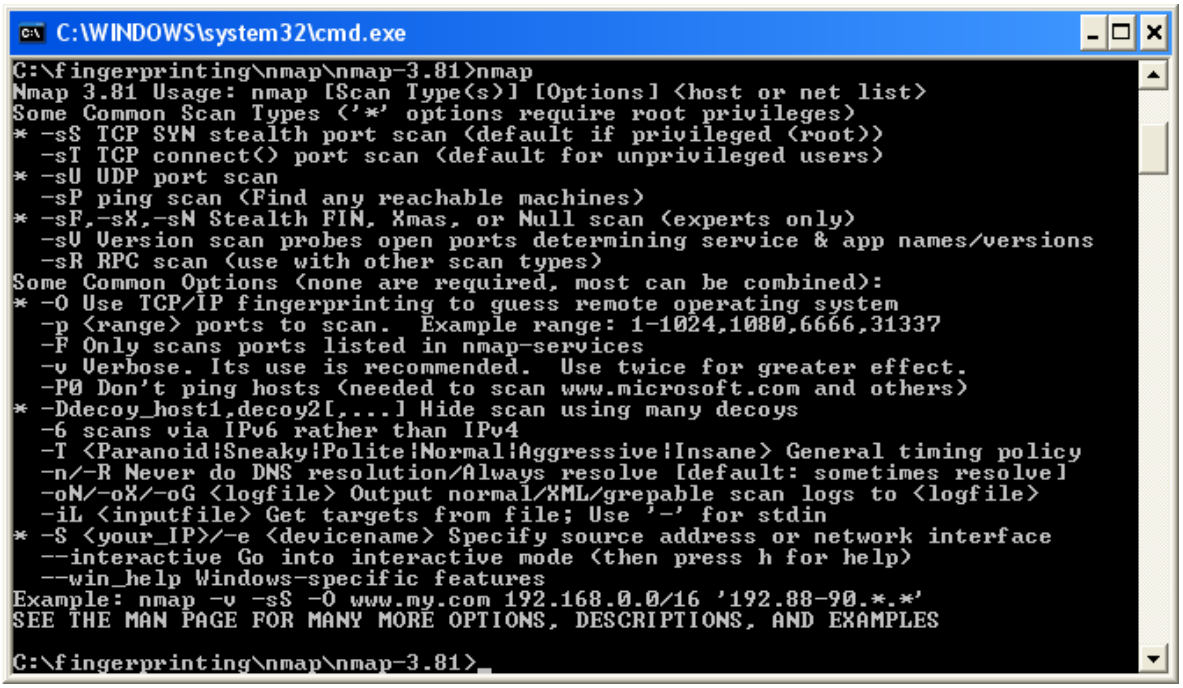


Fig. 2.1- Opciones nmap.

Como se puede observar nmap es un programa realizado para ser utilizado en línea de comandos, y las opciones principales con las que se han echo las pruebas son –v (se encarga de realizar las pruebas varias veces para que los resultados sean más fiables), -sP (que se encarga de hacer un barrido de ping para ver los hosts activos) y -0 (es el comando para hacer fingerprinting). A continuación se va a explicar el funcionamiento del fingerprinting que hace nmap.

2.1.1. Funcionamiento NMAP

En apartado 2.1.2 se explicará la metodología que se ha utilizado para hacer las pruebas de funcionamiento, pero es conveniente ver como el programa hace las funciones que le han sido asignadas. Antes se ha visto que las opciones utilizadas principalmente eran tres de las que una se encarga de repetir varias veces los paquetes para hacer las respuestas más fiables. Quedan dos opciones por explicar, la primera de ellas es la opción –sP que se le llama escaneo ping. Cuando hacemos un escaneo de ping, nmap manda un paquete ICMP Echo y un paquete TCP con el bit de ACK activado al puerto 80 (HTTP). Si el host le responde a uno de estos dos paquetes, entonces está activo. Este escaneo se puede hacer a un host o a varios dentro de una red o rango, esto se especifica al poner los parámetros al arrancar el programa. En la figura 2.2 se puede ver la salida que nos da el programa al hacer una prueba ping dentro de una red.

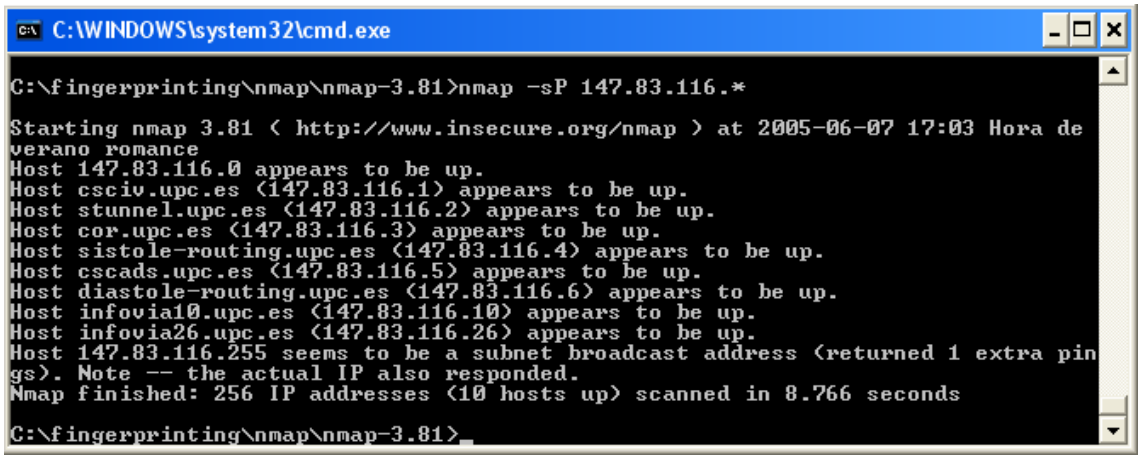


Fig. 2.2 – nmap realizando escaneo de ping

La figura 2.3 es una captura de los paquetes que envía nmap y las respuestas que dan los diferentes ordenadores que reciben estos paquetes. El programa va preguntando uno a uno a todos los ordenadores conectados a esa red con los paquetes TCP e ICMP. Si el ordenador que los recibe no tiene firewall o tiene la opción de recibir paquetes ICMP del firewall activada cuando recibe el paquete ICMP contesta con un ICMP Echo Reply. En cuanto al paquete TCP, si el puerto está abierto, el ordenador contesta con el paquete RESET, si no, no contesta.

A continuación se analizará como nmap realiza el fingerprinting. Nmap necesita un puerto abierto y uno cerrado para hacer los test que utiliza para este fin. Por este motivo, primero hace un escaneo de puertos al host que se pretende atacar. Esto se realiza casi del mismo modo que el escaneo ping. Se envía un paquete TCP con el bit SYN activado a cada uno de los puertos del ordenador a atacar. Si el ordenador tiene el puerto cerrado le contesta con un reset ACK, si el puerto está abierto entonces le contesta son un SYN ACK, esto lo podemos ver en la figura 2.4.

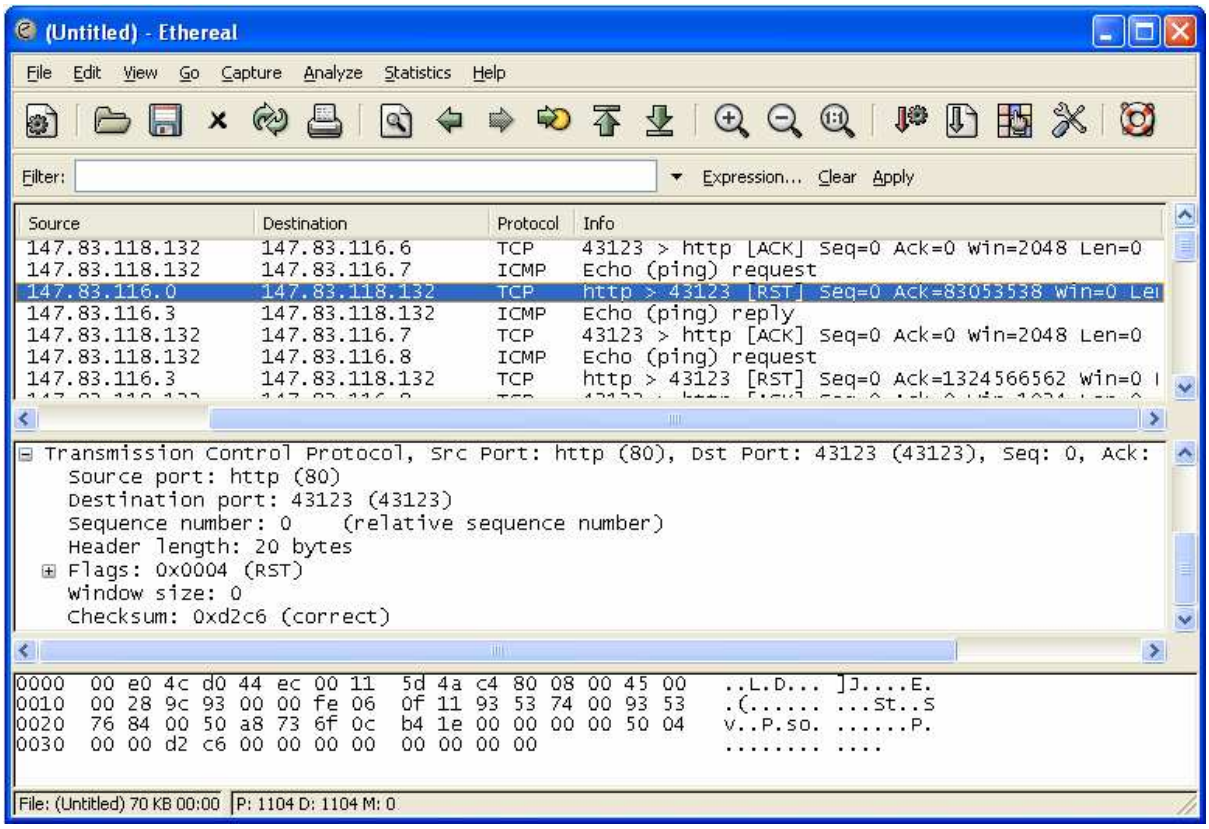


Fig 2.3 – captura de paquetes de escaneo ping

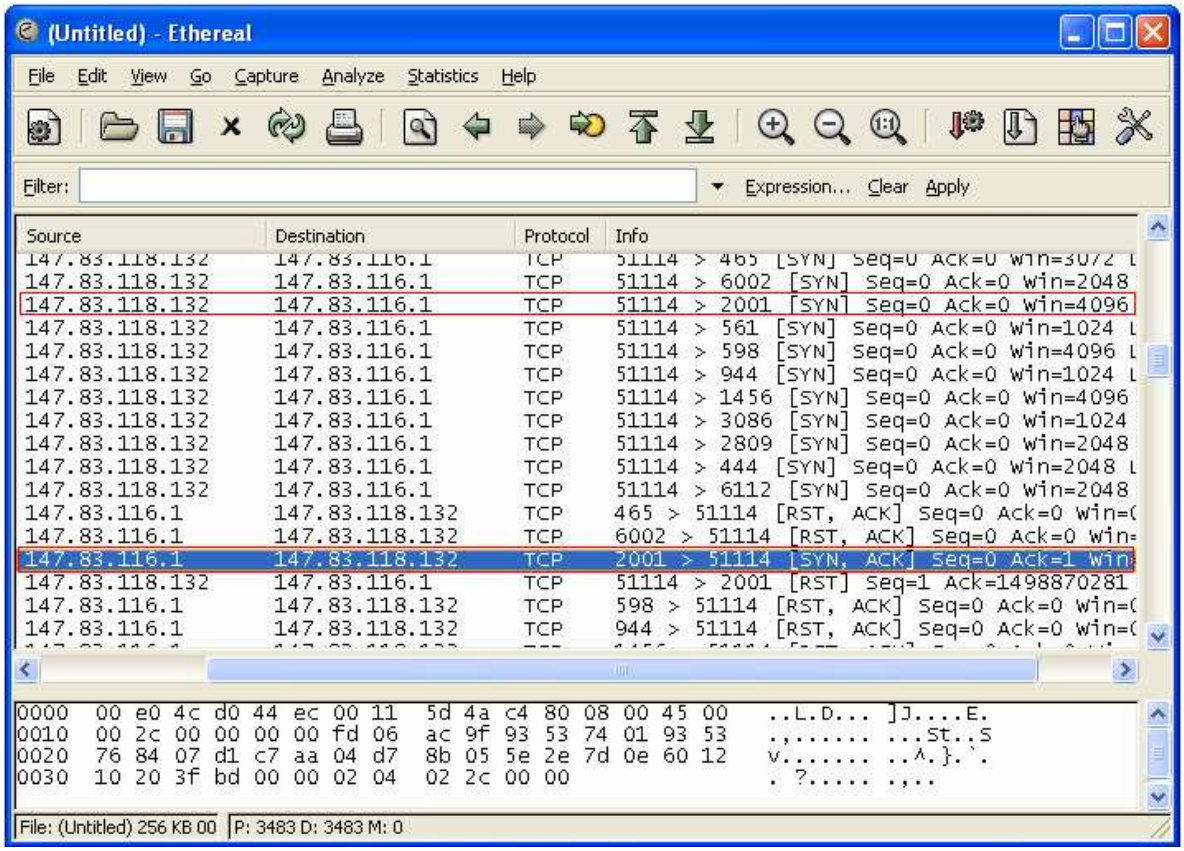


Fig. 2.4 – captura del escaneo de puertos

Una vez se sabe cuales puertos están abiertos y cuales cerrados entonces se hacen los test. Nmap realiza 9 test que son los siguientes:

- Tseq: Es el primer test que se realiza. En él observa el número de secuencia inicial (ISN) del paquete TCP SYN. Cómo varia este número se explicó en el apartado de explicación de los métodos de fingerprinting activo. Un ejemplo de lo que habría en la base de datos para relacionarlo con el sistema operativo es el siguiente:

  TSeq(Class=i800)

  Esto significa que el número de secuencia es un múltiplo de 800 más grande que el anterior.

- T1 (test 1): En este test se envía un paquete SYN con un paquete de opciones TCP a un puerto abierto. En la base de datos se encontraría la siguiente entrada:

  T1(DF=N%W=C000|EF2A%ACK=S++%Flags=AS%Ops=MNWNNT)

  DF significa bit de no fragmentación, y en este caso tendría que estar desactivado, el siguiente término es la ventana del paquete que tiene que seguir la pauta en este campo, en este caso la ventana tiene que ser 0000 o EF2A en hexadecimal. A continuación se encuentra el número de ACK que en este caso tiene que ser el número de la secuencia inicial más 1. A continuación se encuentran los flags que tienen que estar activados, en este caso tienen que estar activados el ACK y el SYN. Para acabar están las opciones que tenemos que recibir, que tienen que estar en el orden que pone en la respuesta, en este caso las opciones son las siguientes:

  <MSS (not echoed)> <NOP> <Window scale> <NOP> <NOP> <Timestamp>

- T2 (test 2): Envía un paquete NULL con las mismas opciones que las del anterior a un puerto abierto. Como hemos visto anteriormente en la base de datos habría la siguiente entrada:

  T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)

  De esta entrada sólo no se ha visto el primer término, que significa que tiene que haber una contestación obligatoriamente. El último término es un poco diferente, quiere decir que puede que no haya opciones en la respuesta.

- T3: Se envía un paquete SYN | FYN | URG | PSH y la opción de escala de la ventana a un puerto abierto. La entrada en la base de datos debería ser la siguiente:

  T3(Resp=Y%DF=N%W=C000|EF2A%ACK=O%Flags=A%Ops=NNT)

  En esta entrada no hay ningún término nuevo.

- T4: El siguiente test envía un ACK a un puerto abierto. Aquí la entrada sería:

  T4(DF=N%W=0%ACK=O%Flags=R%Ops=)

- T5: Se envía un SYN a un Puerto cerrado.

  T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)

- T6: Se envía un ACK a un puerto cerrado.

  T6(DF=N%W=0%ACK=O%Flags=R%Ops=)

- T7: Se envía un FIN | PSH | URG a un puerto cerrado.

  T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)

- PU (prueba de puerto inalcanzable, *Port Unrecheable*):

  PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)

Llegados a este punto hay que remarcar el funcionamiento del programa. Se envía el paquete necesario para recibir una respuesta, y a partir de ahí se analiza la respuesta por medio de unas bases de datos para averiguar el sistema operativo. En figura 2.5 se ve la secuencia de todos los paquetes que se envían para hacer el fingerprinting.

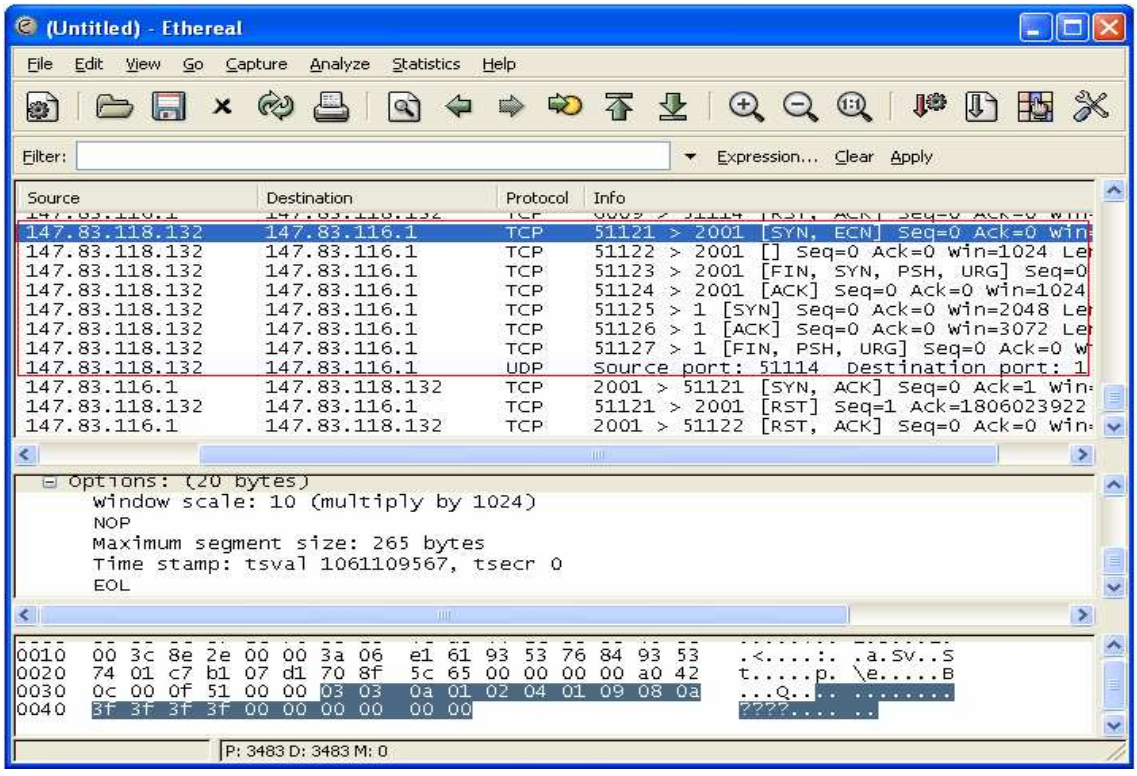


Fig 2.5 Captura de las pruebas de NMAP

Dentro del recuadro rojo se puede ver la secuencia de paquetes, el primer paquete es el test 1 y en el recuadro que está más abajo se pueden ver las opciones que envía, después vienen los test 2, 3, 4, 5, 6, 7 y PU. Los tres paquetes siguientes es la contestación del test 1.

2.1.2. Pruebas de funcionamiento

Después de haber explicado el funcionamiento hay que centrarse en los resultados obtenidos. Se han hecho dos tipos de pruebas, las primeras de una manera indiscriminada para ver su funcionamiento, y después de una menara controlada para ver su efectividad.

De las pruebas indiscriminadas no se va ha hablar porque no son concluyentes debido a que el ordenador al que se le hace la prueba no es conocido, y no podemos saber si los resultados son correctos o no. Sólo se han realizado para ver el funcionamiento del programa.

En las otras pruebas lo que se pretende es ver si las predicciones que realiza el programa nmap son acertadas o no. Para hacer estas pruebas se cogía un host del laboratorio donde se han realizado. Este ordenador tiene que ser conocido, es decir tenemos que conocer el sistema operativo que utiliza y su dirección IP. Al conocer estos datos el siguiente paso es hacer un fingerprinting con nmap a la dirección adecuada.

Después de realizar las operaciones anteriormente descritas hemos tenido los resultados que se expondrán a continuación. En la figura 2.6 se encuentra el primer resultado, que es el de una máquina en la que está instalado un Windows XP profesional y que tiene el firewall por defecto desactivado:

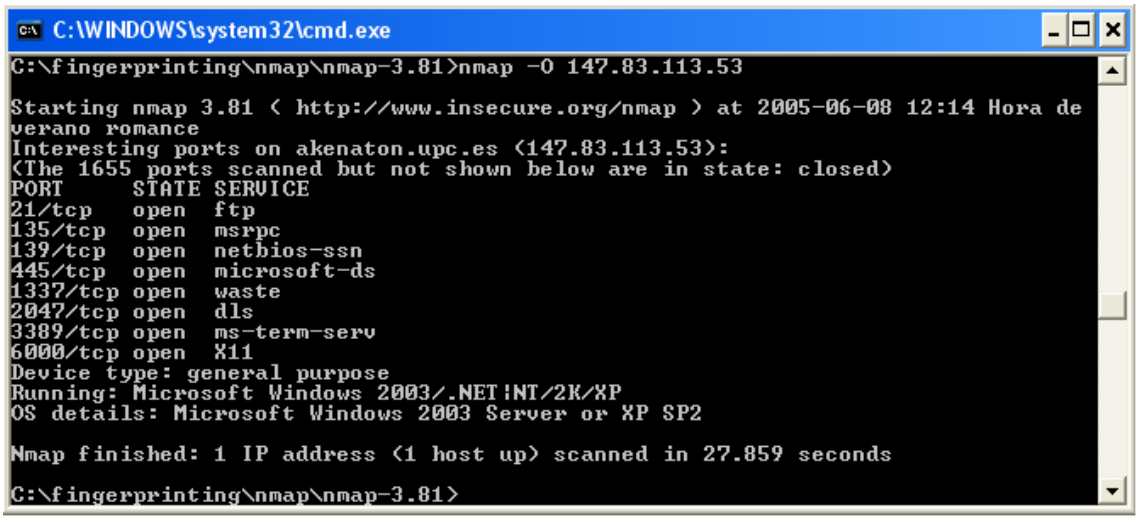


Fig. 2.6 Resultado prueba a un ordenador con Windows XP (con firewall desactivado)

Como se puede ver el resultado está bastante acotado y varía entre el sistema 2003 server y el XP con un Service Pack 2 instalado, seguramente porque

estas dos implementaciones son prácticamente iguales. A continuación se le hizo la misma prueba al mismo ordenador pero esta vez activando el firewall y obtuvimos el resultado es el expuesto en la figura 2.7.

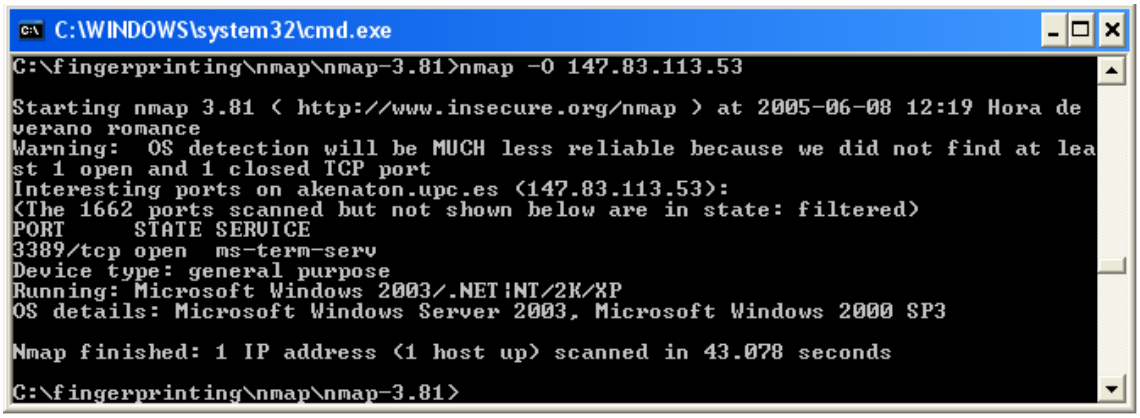


Fig. 2.7 Resultado prueba a un ordenador con Windows XP (con firewall activado)

Al observar este resultado se puede ver como el número de puertos abiertos ha decrecido mucho. De hecho en principio sólo ha encontrado uno abierto, por eso dice que la predicción puede que no sea muy acertada. Al observar con detenimiento los detalles del sistema operativo se tienen dos posibles resultados que ninguno de ellos es correcto, pero son muy cercanos a la realidad. Como la única variación entre los dos casos es el firewall, podemos decir que el firewall afecta mucho los resultados.

La siguiente prueba se hizo a un ordenador con Windows 2003 server, y el resultado se encuentra en la figura 2.8.

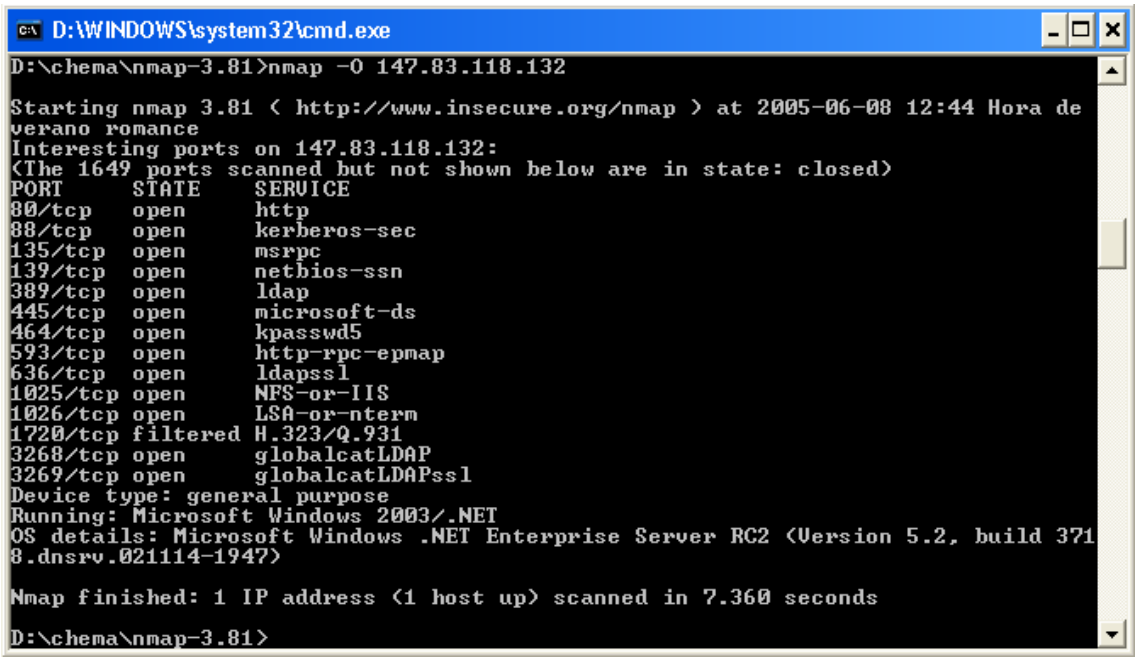


Fig. 2.8 Resultado prueba a un ordenador con Windows 2003

Este ordenador no tiene ningún tipo de firewall instalado. Una posible consecuencia de esto es que ha encontrado muchos puertos abiertos. Hace una única predicción muy detallada y que es acertada, muy seguramente es debido a que este sistema operativo está muy estudiado y se sabe todas sus características.

Para acabar, se le ha hecho la prueba de funcionamiento a un ordenador que tiene instalado un Linux con una implementación Gentoo y que tiene un kernel 2.6.10. Se ha obtenido el siguiente resultado:

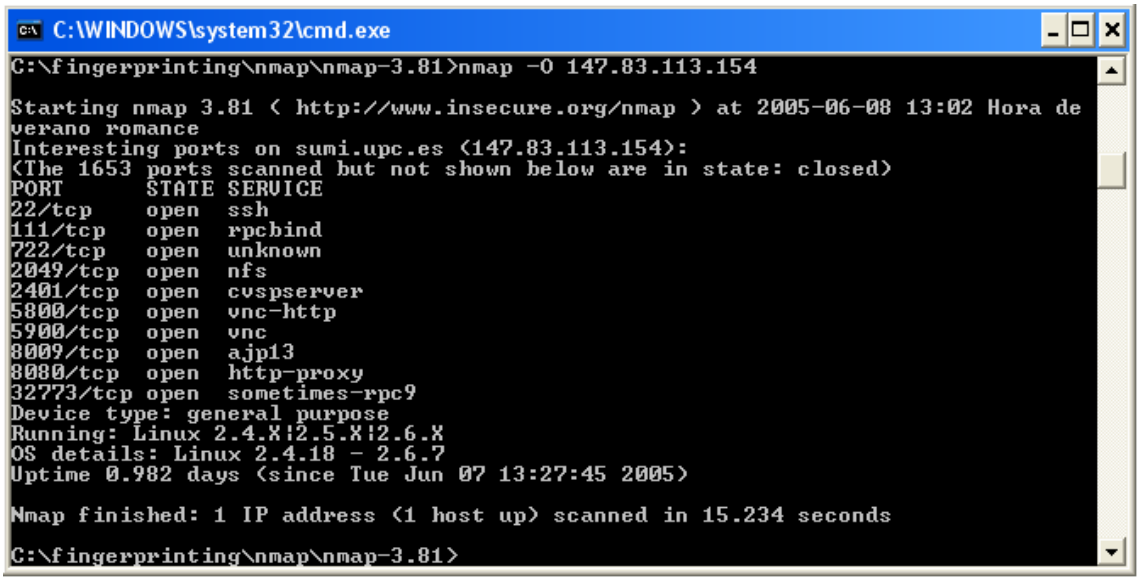


Fig. 2.9 Resultado prueba a un ordenador con Linux

Hay que decir que esta implementación no tenia el firewall activado por eso ha obtenido una gran cantidad de puertos abiertos para hacer la predicción. También podemos ver que el resultado es bastante vago pero casi acertado, aunque hay que tener en cuenta la gran cantidad de kernels e implementaciones de Linux que hay actualmente.

2.1.3. Conclusiones

El nmap es uno de los programas más utilizados para estos menesteres y es obvio que es muy útil gracias a su gran cantidad de test y escaneos. La principal limitación que tiene es la de la vulnerabilidad hacia los firewalls. Cuando el ordenador a atacar tiene instalado uno, o utiliza uno como en el caso de Linux o Windows XP implementado dentro del sistema operativo, los resultados se hacen bastante vagos y es muy probable que no obtenga ningún resultado. Por ese motivo apenas ha habido resultados satisfactorios cuando se intentaba hacer fingerprinting a host con un kernel de Linux, ya que estos suelen tener un firewall implementado por defecto muy efectivo, no como Windows, que incluso se ha podido tener resultados coherentes al aplicar los test a un ordenador con XP profesional con su firewall activado.

En cuanto a los resultados hay que decir dan una idea muy exacta del sistema operativo que está instalado en el ordenador al que se le hacen las pruebas y eso es útil. En otras pruebas realizadas si no encontraba una solución lo suficientemente correcta te da las diferentes opciones con un tanto por ciento que indica la confianza que tiene en ese resultado. Si encuentra algún firewall el programa también avisa de que los resultados puede que no sean correctos. Además da otra información útil como puertos abiertos que el programa cree que son interesantes, como se puede ver en las diferentes capturas que se han añadido a este documento en el apartado anterior.

Por último hay que comentar que esta aplicación está basada totalmente en descubrir características sobre un host para poder atacarlo, y el fingerprinting está basado simplemente en descubrir el sistema operativo del contrario, y esto no es lo que se necesita. Uno de los objetivos de este proyecto es observar que se puede saber sobre las características del protocolo TCP, no sobre el sistema operativo que implementa el otro ordenador. De todas formas estos dos conceptos van ligados, entonces averiguando uno encontraremos el otro. Habría que cambiar alguna cosa al programa nmap, aunque la manera como hace el fingerprintng es totalmente útil para nuestros fines, de hecho la aplicación que se va a realizar y que se explicará en el siguiente capitulo realizará el fingerprinting con los mismos principios que el nmap.

2.2. WINFIGERPRINTING

A continuación se va a analizar otro programa de fingerprinting que es totalmente diferente al nmap. Mientras que el otro era un programa en línea de comandos este tiene interfaz gráfica. El anterior era multiplataforma, winfingerprinting ha sido realizado para Windows y nmap tiene más funciones mientras que winfingerprinting está totalmente hecho para este fin. En la figura 2.10 se puede ver el aspecto del programa.

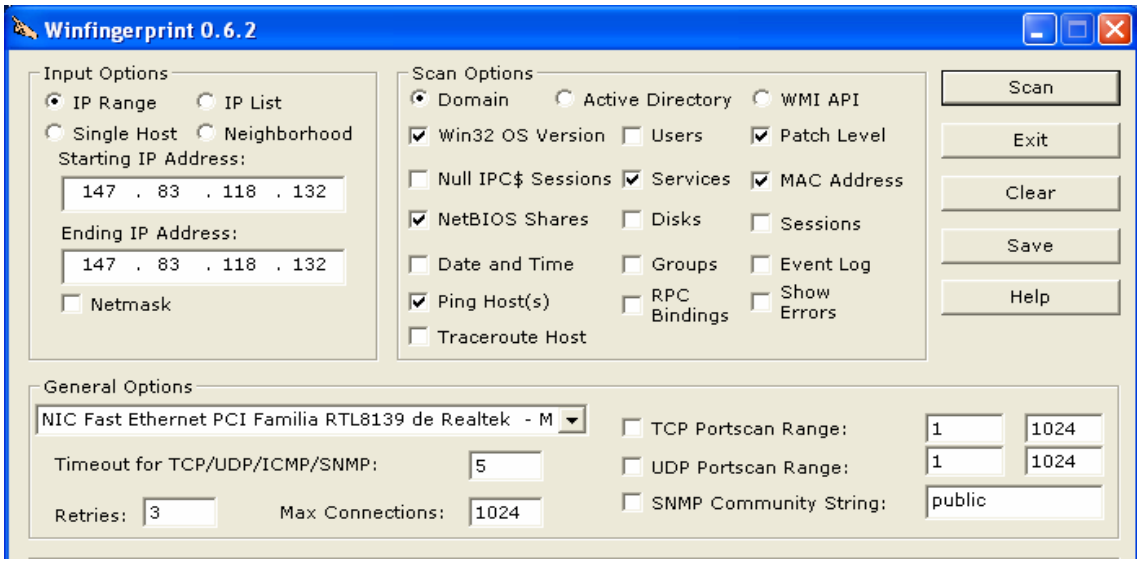


Fig. 2.10 Programa Winfigerprinting

En la figura 2.10 se puede ver todo lo que puede hacer este programa, sus diversas opciones en los test, las posibles variables que se pueden intentar encontrar del host al que se escanea. También se puede ver que es capaz de hacer los test a un grupo de máquinas, a una sola, etc. Y también puede utilizar máscaras de red para intentar analizar todos los ordenadores de una red que utilice este tipo de direcciones.

2.2.1. Funcionamiento Winfingerprinting

A continuación se va a explicar cual es el funcionamiento de este programa. Esta es una aplicación que tiene una concepción muy diferente a nmap. Mientras que con nmap se envían paquetes ICMP, TCP y UDP para interrogar al ordenador del que se quiere saber su Sistema Operativo y por lo tanto se utilizan protocolos de la capa de internet y transporte, con winfirgerprinting se utilizan otros protocolos para hacer estos test que pertenecen a la capa de aplicación. Son protocolos de alto nivel como el protocolo SMB y DEC/RPC y también utiliza otras utilidades de estos protocolos como el SRVSVC, WINREG, SVCCTL y LSA.

El protocolo SMB (Server Message Block) es un protocolo que controla la transferencia de datos en una red, se usa para compartir archivos, dispositivos, pipes (que son conexiones entre programas), etc. Es un protocolo cliente servidor del tipo pregunta-respuesta, es decir, que el cliente le envía una pregunta y el servidor le responde. El servidor hace accesible a los clientes de la red los sistemas de ficheros y los otros recursos. Funciona sobre TCP y se tiene que establecer una conexión. Al establecerse, los clientes pueden enviar comandos al servidor para que le de acceso a diferentes partes, a archivos abiertos, leer y escribir archivos, etc. También se utiliza para proveer de un mecanismo abierto entre plataformas diferentes. Cuando este protocolo va sobre TCP se tienen que utilizar los nombres NETBIOS, estos nombres identifican a un ordenador dentro de una red Microsoft.

El funcionamiento del protocolo es como sigue. Para iniciar una sesión con SMB primero el cliente manda una petición al servidor para una sesión NETBIOS donde se manda su nombre codificado. Entonces el servidor lo recibe y responde con un paquete sesión NETBIOS para validar la sesión. Seguidamente el cliente entra en el establecimiento de sesión donde se negocia el tipo de conexión, para esto el cliente envía las versiones SMB que soporta y el servidor devuelve las características que ambos van a utilizar en la conexión. A partir de aquí el cliente envía una identificación que depende del nivel de seguridad que tenga el protocolo y si el servidor la acepta ya sólo queda que el cliente mande la especificación del nombre de red del recurso al que se quiere acceder. Si todo es correcto el servidor enviará una respuesta indicando que la conexión está aceptada.

Seguidamente se pasará a describir el protocolo DEC/RPC. DCE es un entorno a la computación distribuida que nos proporciona llamadas a procedimientos remotos (RPC) y servicios construidos sobre RPC. Winfirgerprinting utiliza esto

para acceder a diferentes servicios y así obtener la información necesaria para hacer el fingerprinting. Los servicios que utiliza son los nombrados anteriormente, SRVSVC, WINREG, SVCCTL y LSA. El primer servicio es el *Microsoft server service*, y en se pide información sobre el servidor que tenga este ordenador. El servidor contesta con sus características como por ejemplo si es un servidor SQL (base de datos) o no, si controla o no un dominio, si es una estación de trabajo o no, etc. WINREG es, como su nombre indica, Registros de Windows y en el se detallan características propias de Windows, como su tipo, si es NT o no, etc. SVCCTL es el servicio de control de Microsoft, y tiene muchas opciones, En este programa se utiliza la opción OpenSCmanager que se utiliza para conectarse a la base de datos SCM (*Supply Chain Management*). Por ultimo LSA es el servicio de *Local Security Architecture* de Microsoft, donde con unos comandos se observa como está la seguridad de ciertos servicios.

En la figura 2.11 se ve un paquete del protocolo SVCCTL que va sobre DCE/RPC.

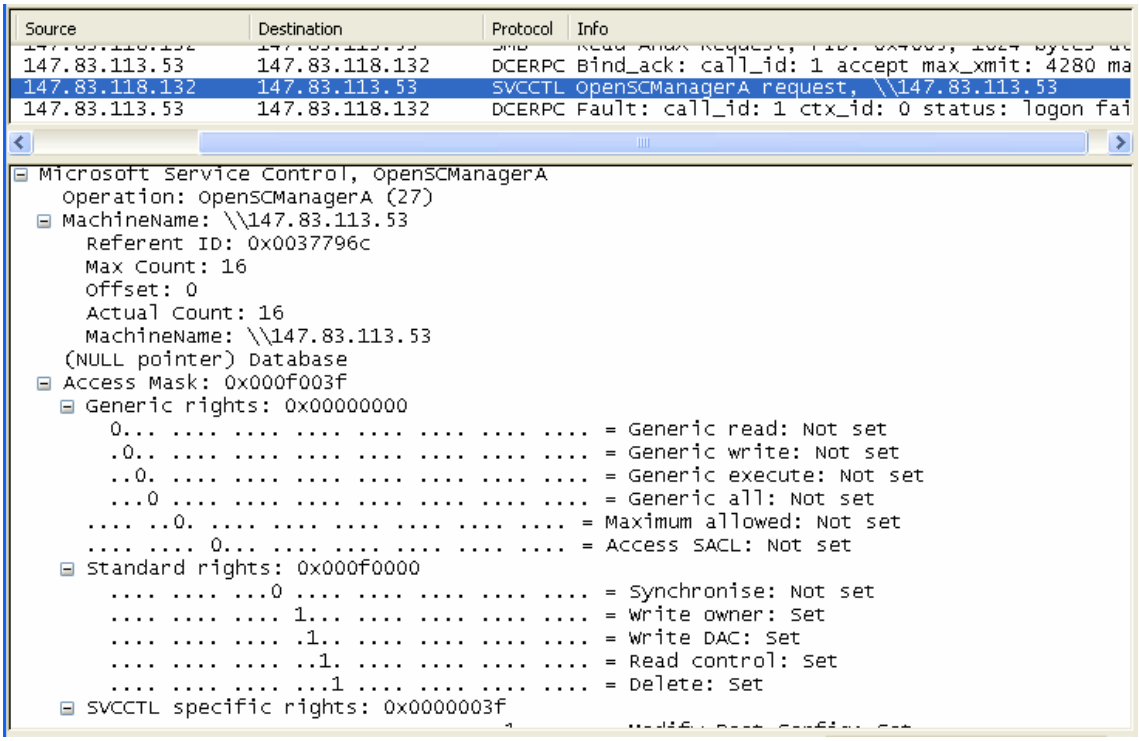
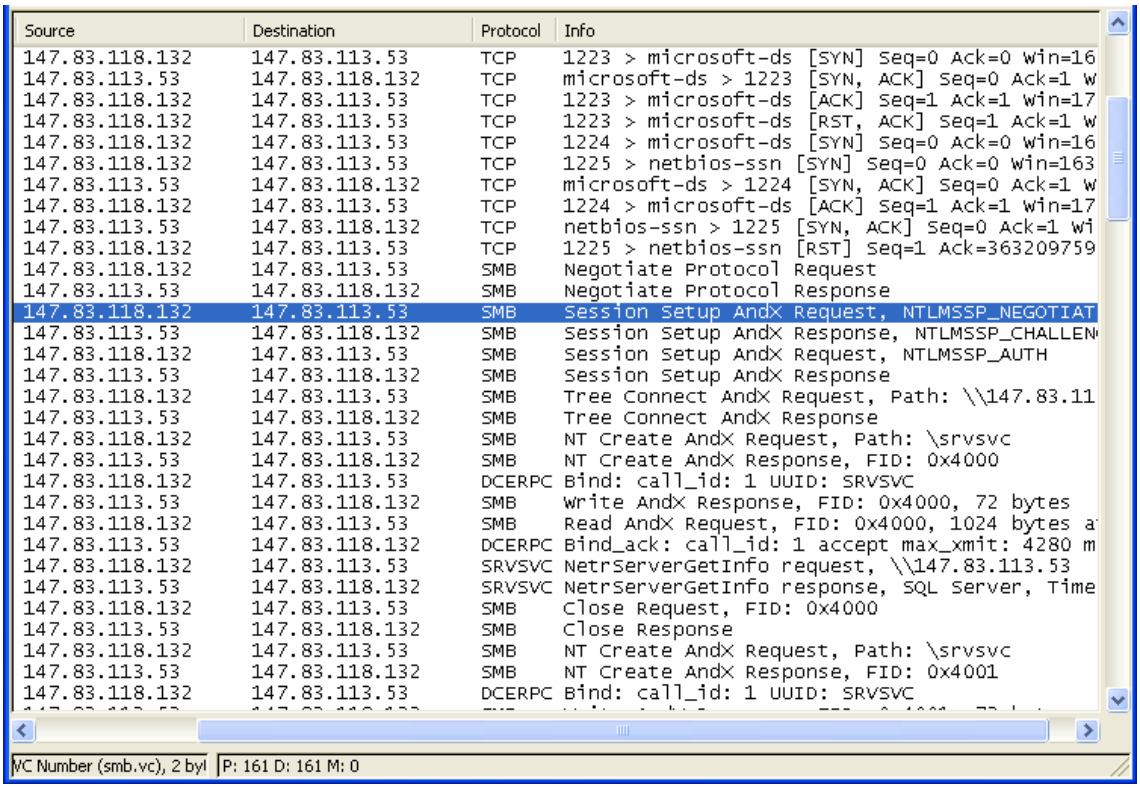


Fig. 2.11Paquete SVCCTL

En esta figura 2.11 se puede ver la información que viaja en este tipo de paquetes.

En la figura 2.12 hay una captura de ethereal de los paquetes que se envían y reciben al hacer el fingerprintig con este programa.



Source	Destination	Protocol	Info
147.83.118.132	147.83.113.53	TCP	1223 > microsoft-ds [SYN] Seq=0 Ack=0 Win=16
147.83.113.53	147.83.118.132	TCP	microsoft-ds > 1223 [SYN, ACK] Seq=0 Ack=1 W
147.83.118.132	147.83.113.53	TCP	1223 > microsoft-ds [ACK] Seq=1 Ack=1 Win=17
147.83.118.132	147.83.113.53	TCP	1223 > microsoft-ds [RST, ACK] Seq=1 Ack=1 W
147.83.118.132	147.83.113.53	TCP	1224 > microsoft-ds [SYN] Seq=0 Ack=0 Win=16
147.83.118.132	147.83.113.53	TCP	1225 > netbios-ssn [SYN] Seq=0 Ack=0 Win=163
147.83.113.53	147.83.118.132	TCP	microsoft-ds > 1224 [SYN, ACK] Seq=0 Ack=1 W
147.83.118.132	147.83.113.53	TCP	1224 > microsoft-ds [ACK] Seq=1 Ack=1 Win=17
147.83.113.53	147.83.118.132	TCP	netbios-ssn > 1225 [SYN, ACK] Seq=0 Ack=1 Wi
147.83.118.132	147.83.113.53	TCP	1225 > netbios-ssn [RST] Seq=1 Ack=363209759
147.83.118.132	147.83.113.53	SMB	Negotiate Protocol Request
147.83.113.53	147.83.118.132	SMB	Negotiate Protocol Response
147.83.118.132	147.83.113.53	SMB	Session Setup AndX Request, NTLMSSP_NEGOTIAT
147.83.113.53	147.83.118.132	SMB	Session Setup AndX Response, NTLMSSP_CHALLEN
147.83.118.132	147.83.113.53	SMB	Session Setup AndX Request, NTLMSSP_AUTH
147.83.113.53	147.83.118.132	SMB	Session Setup AndX Response
147.83.118.132	147.83.113.53	SMB	Tree Connect AndX Request, Path: \\147.83.11
147.83.113.53	147.83.118.132	SMB	Tree Connect AndX Response
147.83.118.132	147.83.113.53	SMB	NT Create AndX Request, Path: \srvsvc
147.83.113.53	147.83.118.132	SMB	NT Create AndX Response, FID: 0x4000
147.83.118.132	147.83.113.53	DCERPC	Bind: call_id: 1 UUID: SRVSVC
147.83.113.53	147.83.118.132	SMB	Write AndX Response, FID: 0x4000, 72 bytes
147.83.118.132	147.83.113.53	SMB	Read AndX Request, FID: 0x4000, 1024 bytes a
147.83.113.53	147.83.118.132	DCERPC	Bind_ack: call_id: 1 accept max_xmit: 4280 m
147.83.118.132	147.83.113.53	SRVSVC	NetrServerGetInfo request, \\147.83.113.53
147.83.113.53	147.83.118.132	SRVSVC	NetrServerGetInfo response, SQL Server, Time
147.83.118.132	147.83.113.53	SMB	Close Request, FID: 0x4000
147.83.113.53	147.83.118.132	SMB	Close Response
147.83.118.132	147.83.113.53	SMB	NT Create AndX Request, Path: \srvsvc
147.83.113.53	147.83.118.132	SMB	NT Create AndX Response, FID: 0x4001
147.83.118.132	147.83.113.53	DCERPC	Bind: call_id: 1 UUID: SRVSVC

Fig 2.12 Captura ethereal winfingerprinting

Esta parte de la captura es del principio y se puede ver como se abre una conexión TCP donde se hace la transacción del nombre NETBIOS y también los otros tipos de paquetes como los DCE/RPC y los paquetes de otros servicios como el SRVSVC.

No se ha podido encontrar cuales son los test exactos que hace el programa para averiguar el Sistema Remoto contrario. Por lo visto aprovecha toda la información que le proporciona los servicios de alto nivel que se han descrito con anterioridad para averiguar todos los detalles que le interesan para sus fines.

2.2.2. Pruebas de funcionamiento

A continuación se va a discutir las diferentes pruebas que se han realizado con esta aplicación. Las pruebas se han realizado exactamente igual que con el programa nmap. Las opciones que estaban activadas a la hora de hacer las pruebas de funcionamiento son las que aparecen en la figura 2.13.

Como se puede ver se han hecho las pruebas a un solo host y se ha intentado averiguar la versión del Sistema Operativo, los servicios que soporta, su dirección MAC, las opciones NETBIOS y por último se hacia una prueba ping para ver si el host está accesible o no

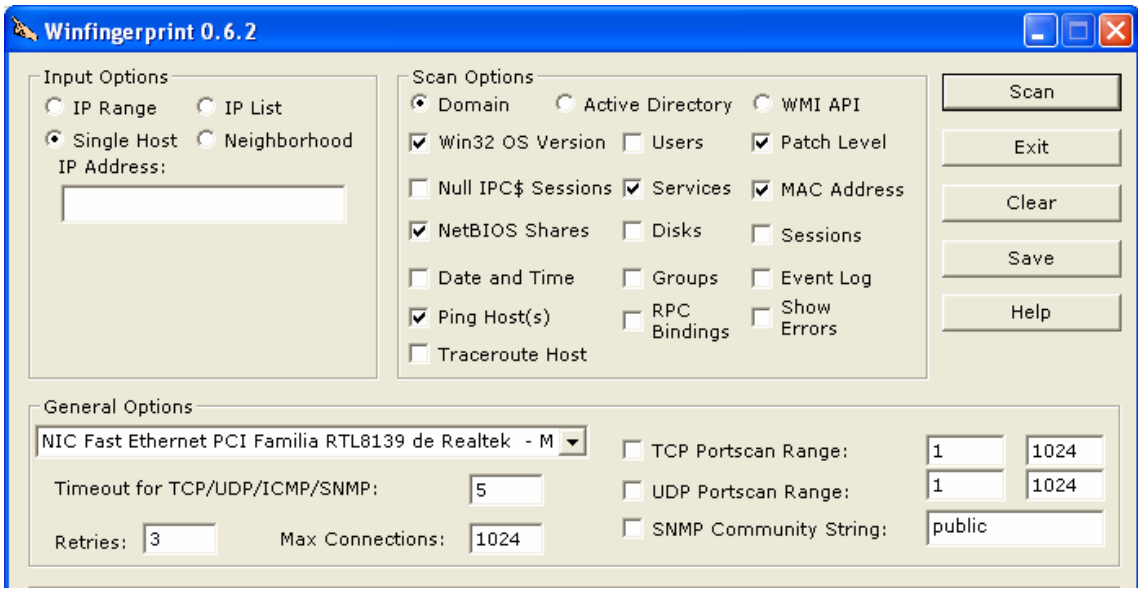


Fig. 2.13 Opciones Winfigerprinting

La figura 2.14 y 2.15 son la respuesta obtenida al realizarla a un ordenador con un Windows XP instalado.

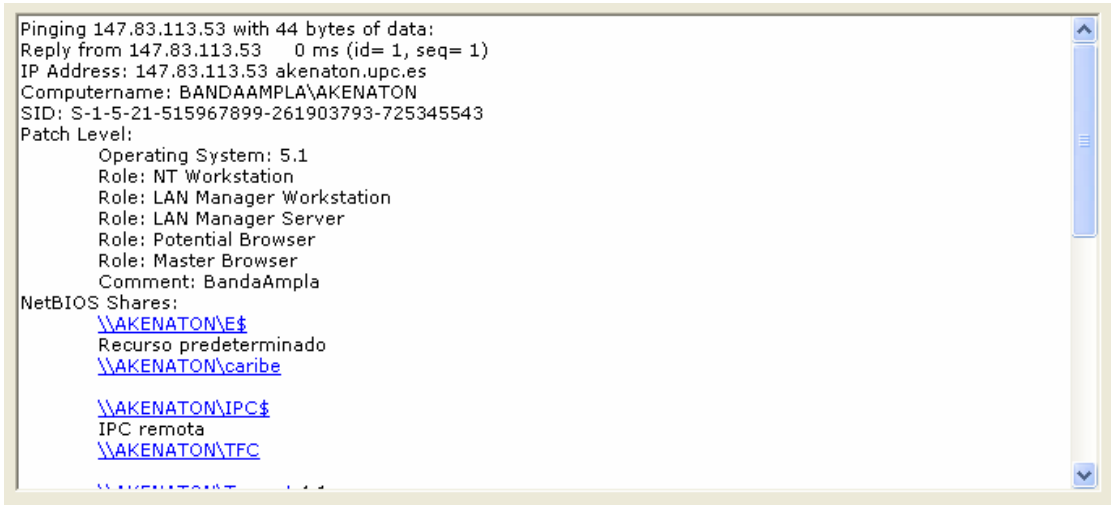


Fig. 2.14 Resultado prueba a un ordenador con XP (I)

El ordenador al que se le realizaba la prueba no tenía firewall activado, y se puede ver que da una información muy completa sobre el mismo. Se puede ver las características de red del host como su IP dirección MAC, mascaras, etc. También se puede ver algunas características del sistema operativo y sus servicios NETBIOS. La información que obtenemos del sistema operativo no es la más adecuada para los requerimientos necesitados y como veremos en los siguientes ejemplos es bastante raro, porque no aparece mucha cantidad de información útil en los siguientes ejemplos.

En la figura 2.16 aparece el resultado obtenido al hacer el fingerprinting a un ordenador con un sistema operativo Windows 2003.

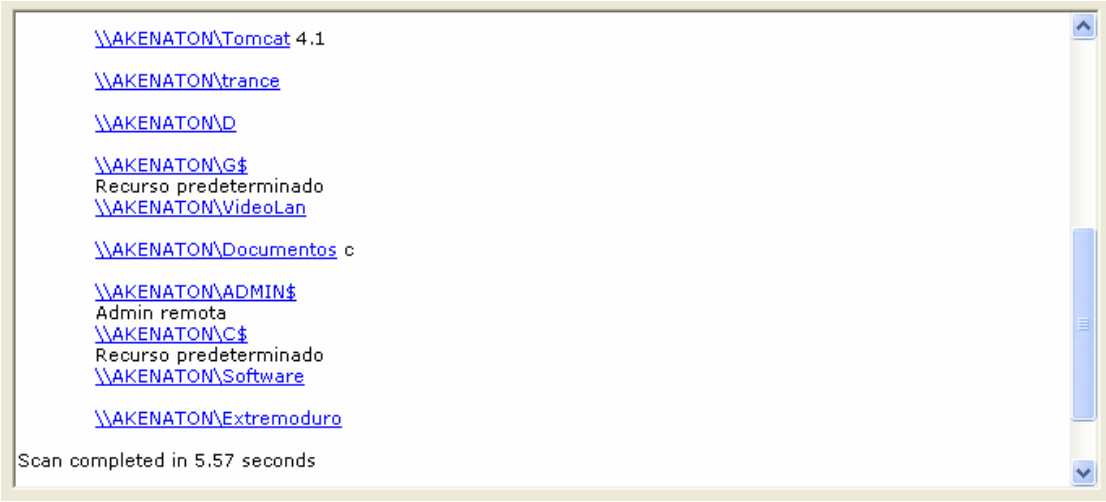


Fig. 2.15 Resultado prueba a un ordenador con XP (II)

Fig. 2.16 Resultado prueba a un ordenador con Windows 2003

En esta figura se puede ver como el resultado es muy escaso. Apenas tenemos información sobre el sistema operativo, sólo aparecen datos de la red, es decir, IP, máscaras, nombre de la máquina. Esto no es suficiente para poder hacer nada con estos resultados.

Por último se le realizó una prueba a un aparato con Linux en su Sistema Operativo. En la figura 2.17 está la capturada la respuesta.

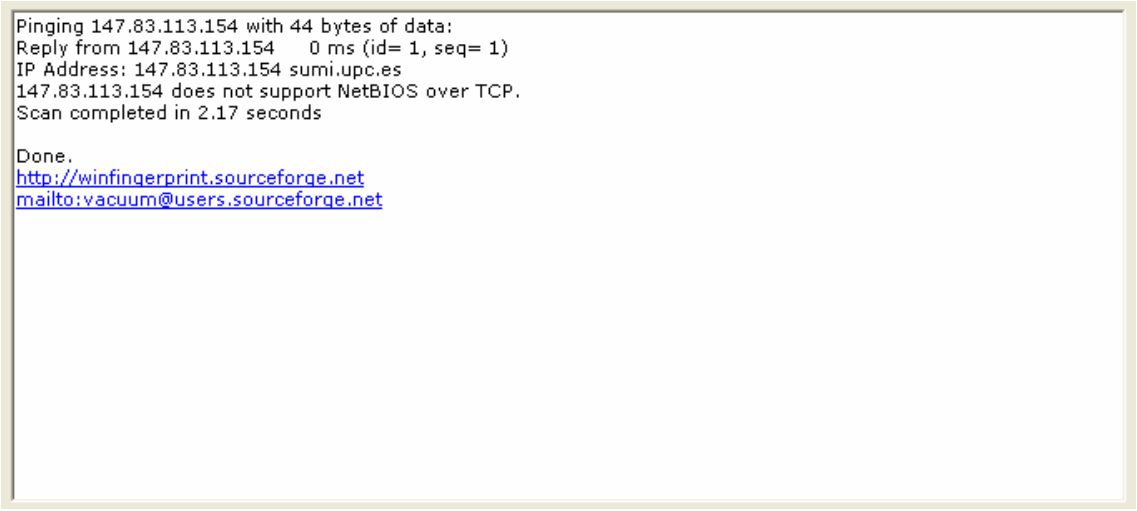


Fig. 2.17 Resultado prueba a un Linux

Como pasa en la figura 2.16, apenas tenemos información, pero antes no se sabía la causa mientras que en la figura 2.17 se nos indica que no ha habido resultados satisfactorios debido a que el host al que se le hace la prueba no soporta los nombres NETBIOS.

2.2.3. Conclusiones

La concepción que tiene este programa para hacer fingerprinting es muy diferente a la que se ha estudiado en el capitulo 1 de este trabajo final de carrera. La base es la misma, hacer preguntas por medio de paquetes y ver las respuestas para asociarlas a unos resultados. La diferencia principal estriba en que utiliza unos protocolos de alto nivel y que está obligando al host al que va ha hacer las pruebas a que tenga implementados estos protocolos, que por lo que se ha visto están muy ligados a Microsoft. Esto limita los ordenadores a los que, al realizarle la prueba, den resultados satisfactorios debido a que actualmente hay muchos ordenadores que tienen instalados otros sistemas operativos que no son Microsoft y que puede que no implementen el protocolo SMB.

Después de analizar los resultados obtenidos en las diferentes pruebas se puede decir que este programa no es valido para los fines requeridos. Esto es debido a que la manera en la que hace el fingerprinting no es correcta para nuestra utilidad. También se debe a la limitación que provoca que el sistema operativo que se quiere averiguar tenga que implementar los protocolos de aplicación no es adecuado para la aplicación a desarrollar. Como hemos visto se descartan muchas implementaciones de sistemas operativos y está muy enfocado a Windows aunque hay muchas implementaciones de Linux que implementan Samba, que utilizan el protocolo SMB.

CAPITULO 3. DESARROLLO DE UNA APLICACIÓN FINGERPRINTING

Uno de los objetivos de este trabajo final de carrera era desarrollar una aplicación que se ajustase a unas ciertas necesidades. Como ha quedado expuesto anteriormente, las aplicaciones que ya han sido implementadas y que ya son ampliamente utilizadas, tienen el objetivo de averiguar el sistema operativo de un ordenador y eso no es exactamente lo que se necesita. Se requiere conocer las características de implementación de la pila de protocolos TCP/IP del sistema para luego poder ajustar la conexión a estas características. También se ha dicho anteriormente que el protocolo TCP y el sistema operativo están ligados en el sentido que averiguando el sistema operativo se puede asociar una determinada implementación del TCP, por lo tanto se puede seguir las mismas pautas de análisis que usan las aplicaciones de fingerprinting descritas anteriormente.

3.1. CARACTERISTICAS DE DISEÑO

Para el diseño de nuestra aplicación, a la cual llamaremos TCPFingerprinting, se ha tomado como referencia las posibilidades de nmap. En esencia, Nmap envía paquetes y recibe la respuesta. Puede enviar varios tipos de mensajes, mensajes ICMP, generalmente *Echo*, mensajes TCP, mensajes UDP, etc.

La aplicación que se ha desarrollado es más sencilla que nmap. TCPFingerprinting puede enviar mensajes ICMP *Echo*, ICMP *TRACEROUTE* y mensajes TCP. De todos estos paquetes puede variarse cualquier campo de sus cabeceras. Con TCPFingerptinting podemos comprobar si el ordenador al que queremos realizar las pruebas está activo mediante los mensajes ICMP *Echo*, y también podemos hacer los diferentes test que hace nmap utilizando un paquete TCP y ajustando los campos de su cabecera.

Para el desarrollo de nuestra aplicación primero se consideró la utilización del lenguaje JAVA, debido a que es un lenguaje multiplataforma y, por lo tanto, funcionaría en todos los ordenadores independientemente de su sistema operativo. Se utilizó la librería JPCAP, que proporciona una interfaz de programación de comunicaciones sobre IP. Esta librería permite ajustar los campos de las cabeceras de los paquetes a enviar. Después de realizar diversas pruebas se optó por cambiar el entorno debido a que los resultados no eran todo lo satisfactorios que se requería.

Finalmente el lenguaje de programación utilizado ha sido C y el conjunto de librerías asociadas a comunicaciones sobre IP. En este lenguaje está muy desarrollado el uso de socket, pero para poder hacer esta aplicación se tuvo profundizar sobre cierta clase de sockets particulares. Todas las funcionalidades de la aplicación desarrollada están basadas en los sockets de Windows “Winsockets”. En este entorno hay tres tipos de sockets, los “*socket_dgram*” que son sockets de datagramas para paquetes UDP, los “*sockets_stream*” que son sockets de flujo de datos para conexiones TCP y por

último “socket_raw” que son un tipo de sockets que nos permiten enviar cualquier tipo de paquete sin que la definición del socket esté ligada a un protocolo concreto, y que también nos permita ajustar las cabeceras de los paquetes que se envían. Este último tipo de sockets es el que utilizamos para el desarrollo de esta aplicación.

3.2. DIAGRAMAS DE LA APLICACIÓN

En la figura 3.1 está representado el diagrama de bloques de la aplicación que se ha realizado. En este esquema se puede observar cuales son las diferentes funciones de la aplicación y cuales son sus interacciones entre ellas.

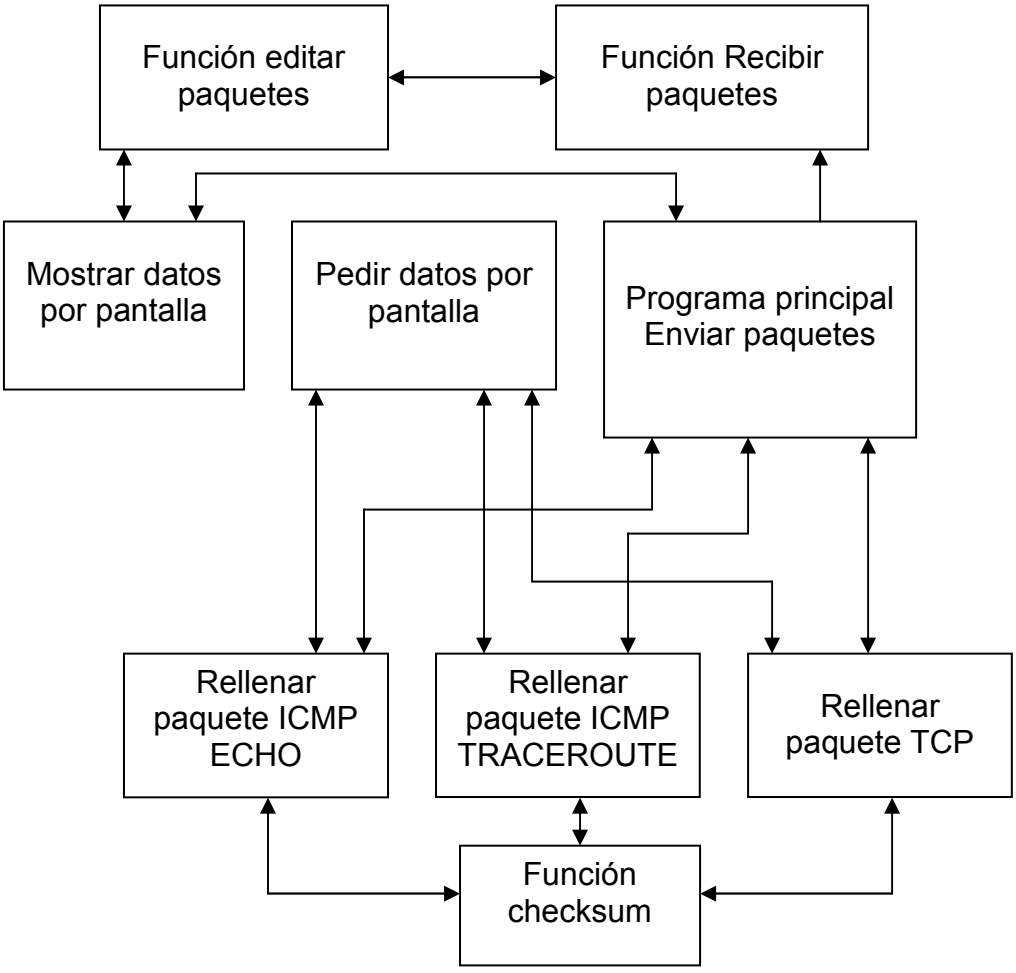


Fig. 3.1 Diagrama de bloques de la aplicación

En la figura 3.1 se puede ver todas las funciones de las que está compuesta esta aplicación. El bloque “programa principal” se corresponde con la función principal de código y es la que se encarga de enviar paquetes y de ella parten todas las secundarias. Las funciones secundarias se encargan de todo lo

demás. Unas realizan las funciones de rellenar los diferentes tipos de paquetes, que necesitan que el usuario introduzca los diferentes términos de los campos de sus cabeceras. Otra de las funciones calcula el cheksum de los paquetes a enviar. Las dos funciones que quedan no son utilizadas por el programa principal. La función “recibir paquetes” se encarga, como su nombre indica, de recibir todos los paquetes que se envían o reciben a la aplicación y por último la función “editar paquetes” se encarga de seleccionar los paquetes recibidos por la función anterior para mostrarlos por pantalla una vez se hayan extraído los valores más interesantes de los campos de las cabeceras.

En la figura 3.2 se puede ver el diagrama de flujo que se ha seguido para hacer la aplicación. Básicamente, en la aplicación tenemos dos hilos de ejecución en el mismo programa. Un hilo que se dedica a la captura de paquetes y otro que se dedica a enviarlos.

En la parte que captura paquetes se abre un socket para recibir paquetes. Este socket tiene que poder capturar cualquier paquete y para conseguir esto se utilizan ciertos parámetros que se introducen cuando se abre el socket. Después se asocia este socket abierto a una dirección IP, que tiene que ser la dirección IP local, por medio de un bind. A continuación se tienen que fijar las características del socket para así ajustarlas a nuestras necesidades, en nuestro caso el socket está en modo en modo promiscuo, es decir, que captura todo tipo de paquetes. Por último se capturan los paquetes y se filtran según convenga.

La parte de enviar paquetes es más complicada. Lo primero que se hace es averiguar la dirección local para, seguidamente, elegir la dirección a la que se va a enviar el paquete. A continuación se elige el tipo de paquete que se quiere enviar debido a que cada paquete de un protocolo determinado necesita un socket diferente. Una vez elegido el paquete que se quiere enviar se sigue el mismo proceso para los tres tipos de paquetes que se pueden enviar:

1. Primero se abre el socket. Como se ha dicho antes, para cada tipo de socket esta inicialización es diferente.

2. Luego se rellena el paquete con los datos que introduce el usuario.

3. Para el protocolo TCP hay un paso más que consiste en ajustar los campos de la cabecera IP y también de la cabecera TCP

4. Por último se envía el paquete una vez rellenado.

Una vez acabado este proceso se vuelve a la parte del programa donde se pide al usuario que introduzca el tipo de paquete que quiere enviar. Este proceso es realizado hasta que el usuario decide salir del programa. Es entonces cuando se elimina el hilo secundario y por último se cierra el entorno que se ha creado al principio del programa para que funcionen los sockets. Realizando estos dos pasos nos aseguramos de que no se quede ningún socket abierto que quede fuera de control.

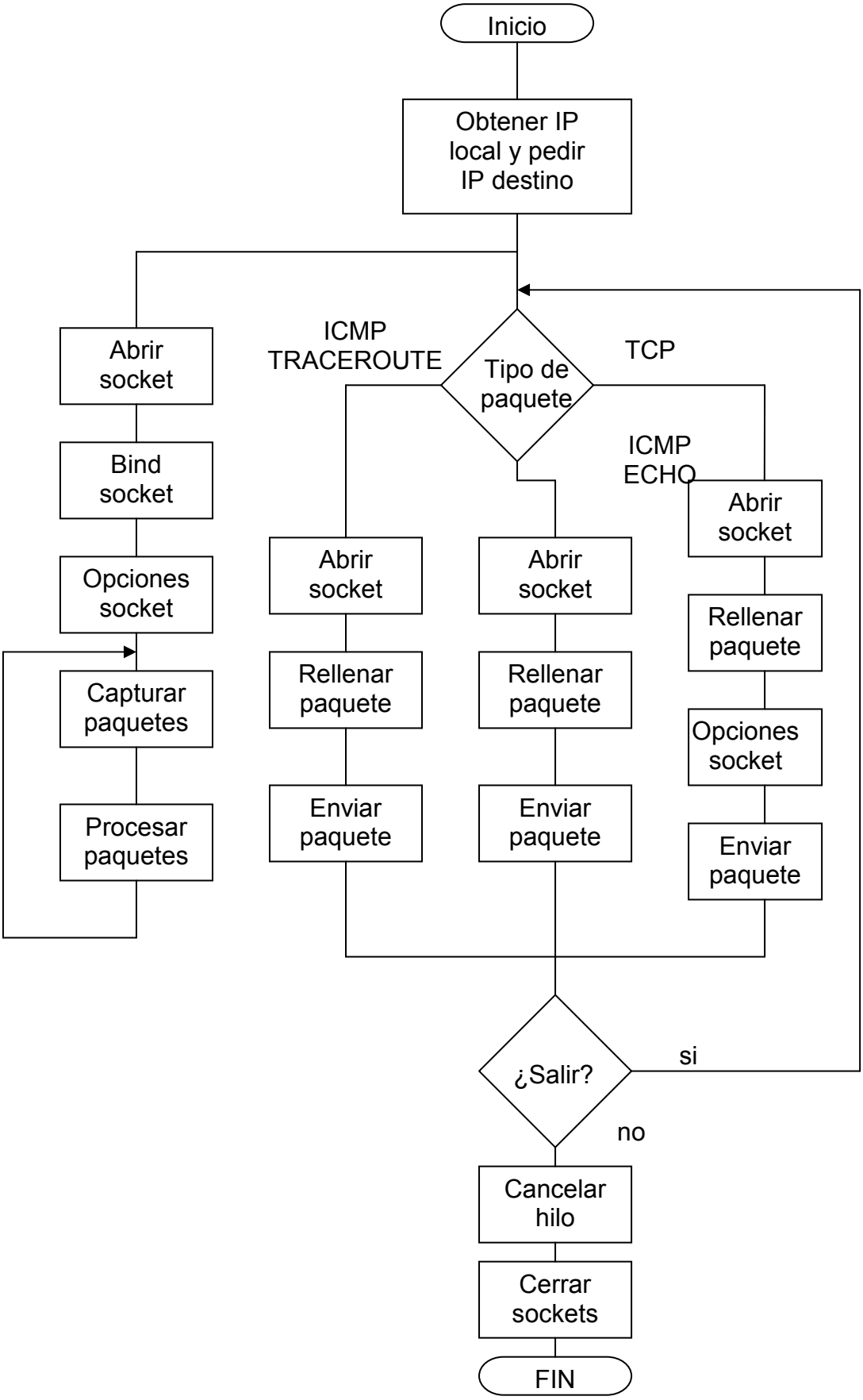


Fig. 3.2 Diagrama de flujo de la aplicación

3.3. DESCRIPCIÓN DE LA APLICACIÓN

En este apartado se va a explicar las partes más interesantes del código fuente del que está compuesta la aplicación. Lo primero que hay que destacar es que el código fuente está compuesto por dos archivos. Uno de los archivos es el "ip.h" donde se encuentran la definición de las estructuras y la declaración de las librerías que se utilizan en el programa principal. El segundo archivo es el programa principal y se llama "TCPFingerprinting.c". En él se encuentra todo el código fuente que nos permiten implementar todas las funcionalidades que se han descrito con anterioridad.

En el archivo ip.h se encuentran unas de las piezas más importantes de esta aplicación. Estas piezas son las estructuras de las cabeceras y con ellas se puede construir el paquete correctamente y también se podrá leer correctamente una vez capturados. Tienen que ajustarse a la cabecera del paquete original y para eso se utilizan los diferentes tipos de variable. En la figura 3.2 se puede ver la cabecera TCP obtenida de su RFC 793. En ella se puede ver todos los campos de las que está compuesta. Como se puede apreciar cada campo necesita un tamaño determinado que varia de unos a otros. La estructura que se debe realizar tiene que imitar la de la figura 3.3 ajustando los tamaños de cada variable a las necesidades que nos obliga la cabecera TCP.

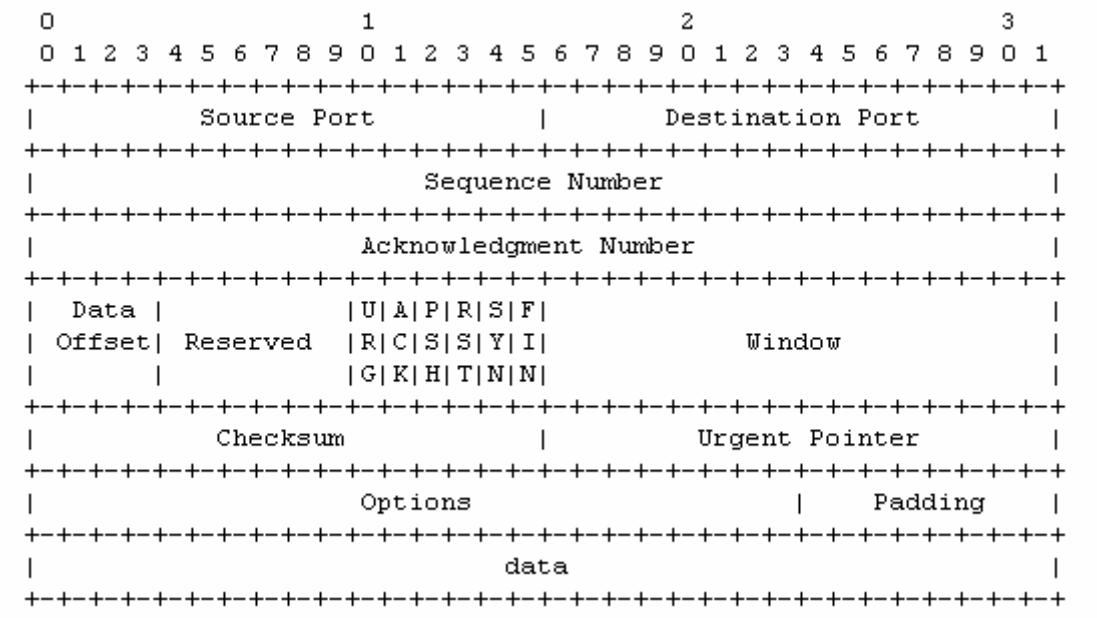


Fig. 3.3 Cabecera TCP

De acuerdo con la cabecera de la figura 3.3, la estructura que se ajusta es la siguiente:

```
struct tcpheader {
      unsigned short int th_sport;
      unsigned short int th_dport;
      unsigned int th_seq;
      unsigned int th_ack;
```

```
    unsigned char th_x2:4, th_off:4;
    unsigned char th_flags;
    unsigned short int th_win;
    unsigned short int th_sum;
    unsigned short int th_urp;
};
```

Las variables que se utilizan son del tipo "*unsigned*" debido a que no hay campos negativos dentro de las cabeceras. Las variables int ocupa 4 Bytes, las variables short ocupa 2 Bytes, mientras que las variables char ocupa 1 Byte. Así podemos ver como se va ajustando los tamaños de la variable con los de su respectivo campo de la cabecera TCP. Esta cabecera tiene la peculiaridad de tener unos campos que no se pueden ajustar a su tamaño de variables. Estos son el campo *Reserved* y el campo de las banderas. La estructura en C se ha realizado de la única manera posible, acortando el campo reservado y aumentando el tamaño del campo de las banderas, debido a que no existen variables que tengan 6 bits.

Seguidamente se va a comentar el código del archivo principal, TCPFingerprinting.c. Como se ha ido observando en la figura 3.2 del diagrama de flujo, este programa tiene dos hilos de ejecución. Para hacer esto se ha utilizado una función de C que se encarga de crear el segundo hilo de ejecución. Esta función es `CreateThread`.

```
a_thread = CreateThread(NULL, 0, recibir, NULL,0, &a_threadId);
```

Dentro del paréntesis se pone en nombre de la función con la que el programa empezará a ejecutar el segundo hilo de ejecución, en este caso la función se llama `recibir`. También se le puede pasar algún tipo de parámetro que iría detrás del nombre de la función antes mencionada. En este caso no se le pasan parámetros y por eso detrás del nombre de la función hay un `NULL`.

Siguiendo con la descripción de cómo se crea un thread, queremos destacar que la definición de la función `recibir` debe definirse de una manera determinada. La definición es el siguiente:

```
DWORD WINAPI recibir (PVOID param)

{

//cuerpo de la función

}
```

Otro de los comandos utilizados y que está relacionado con los threads es el que se utiliza para eliminar el hilo de ejecución. Este comando es `TerminateThread`.

```
TerminateThread(a_thread, 0);
```

De esta función hay que mencionar que el primero de los parámetros tiene que ser el mismo que el de la función `CreateThread`. Este parámetro es utilizado

para guardar información del thread. El siguiente parámetro es un código de salida para cerrar el hilo de ejecución.

A continuación se van a explicar todas las funciones de la librería winsocket de C utilizados. Cuando se van a utilizar este tipo de funciones, lo primero que se tiene que hacer es crear un entorno donde sean posibles abrir sockets. Para poder realizar esto se tiene que utilizar el siguiente comando:

```
WSAStartup (MAKEWORD(2,2), &wsd)
```

Cuando se llama a esta función, estamos permitiendo que el programa pueda utilizar la librería Ws2_32.DLL que es la que controla el uso de los sockets.

Una vez creado el entorno se abre el socket. Para abrir el socket en cada uno de los casos se hace de una manera diferente. Para abrir un socket cuando se tiene que enviar el paquete ICMP se hace de la siguiente manera:

```
sock=socket (AF_INET,SOCK_RAW,IPPROTO_ICMP);
```

El comando que se utiliza es `socket` y esto devuelve un parámetro del tipo socket que es donde se guarda información necesaria para que el socket pueda ser controlado. Entre paréntesis hay tres parámetros, el primero es la familia a la que pertenece, en este caso es `AF_INET` y significa que es de la familia de Internet. El siguiente término es el tipo de socket que crea, en este caso es del tipo `SOCK_RAW` que se explicó en el apartado 3.1. El último parámetro es el tipo de protocolo que se envía por el socket en este caso como se ha dicho antes, se enviará un paquete del protocolo ICMP.

Para enviar un paquete TCP se crea el socket de la siguiente manera:

```
sock=socket (AF_INET,SOCK_RAW,IPPROTO_RAW);
```

La única diferencia con la llamada anterior se encuentra en el protocolo `IPPROTO_RAW`. Con esta inicialización se necesita incluir también la cabecera IP debido a que no estamos enviando ningún protocolo determinado.

Por lo tanto para incluir la cabecera IP se utiliza el siguiente comando:

```
Setsockopt(sock,IPPROTO_IP,IP_HDRINCL,(char*)&bOpt,sizeof(bOpt))
```

Con el comando setsockopt se incluyen las opciones que se deseen al socket. En esta llamada utilizamos las palabras clave `IPPROTO_IP` y `IP_HDRINCL`. El primer parámetro del paréntesis es el parámetro en el que se encuentra la información del socket. Con los dos parámetros siguientes se le está diciendo al socket que se van a enviar paquetes del protocolo IP y que tiene que incluir la cabecera IP que se le introduzca.

Ya sólo queda enviar el paquete. Para enviar el paquete se hace de la siguiente manera:

```
sendto(sock,ip_tcp,sizeof(ip_tcp),0,(SOCKADDR*)&sin,sizeof(sin))
```

El comando que se utiliza es `sendto()`.Los datos que son enviados son los que nos indica el segundo término del paréntesis. El siguiente término es el tamaño de los datos que se envía y seguidamente se encuentran las variables que llevan la información de las direcciones IP destino y su tamaño.

A continuación se va a continuar explicando la función que recibe paquetes. En esta función se crea un socket diferente a los que se han visto anteriormente.

```
sockRecv=socket(AF_INET,SOCK_RAW,IPPROTO_IP);
```

Con este comando estamos abriendo un socket del tipo *raw* y que acepte paquetes del protocolo IP. Como tenemos que recibir paquetes, se tiene que asociar este socket con la dirección local, y esto se realiza con el comando `bind` de la siguiente manera:

```
bind (sockRecv, (PSOCKADDR)&sin, sizeof(sin))
```

Dentro de la variable `sin`, que es una estructura del tipo `sockaddr_in`, viaja la dirección IP local, el puerto y la familia del protocolo. Después de haber hecho el `bind` se tienen que introducir en el socket que recibe paquetes unas ciertas características. Para eso se utiliza el siguiente comando.

```
WSAIoctl(sockRecv,SIO_RCVALL,&opt,sizeof(opt),NULL,0,&bytes,NUL
L,NULL))
```

En el comando `WSAloctl` estamos poniendo el socket en modo promiscuo, es decir, que capture todo tipo de paquetes, esto se hace con el valor `SIO_RCVALL`, que traducido sería algo así como recibir todo. Y para acabar el proceso de recibir paquetes ya sólo falta el comando utilizado para recibir paquetes `recv`:

```
bytes=recv (sockRecv, paqueteRecv, sizeof(paqueteRecv), 0))
```

La primera variable es el tamaño del paquete que ha recibido. Dentro del paréntesis se encuentra el primer parámetro que contiene la información del socket. El segundo parámetro es el nombre del buffer donde guarda la información que recibe y el tercero es el tamaño de este buffer.

El último comando relacionado con sockets que se va a explicar es `WSACleanup`. Una vez se haya realizado todo lo necesario se tiene que cerrar el entorno que se ha creado al principio del programa. Para esto se utiliza la función `WSACleanup`.

```
WSACleanup();
```

Por último, sólo queda comentar cómo se realiza el rellenado de paquetes. Como hemos visto en el comando `sendto` se envía un buffer de datos. Por lo tanto, se ha creado un buffer para cada uno de los paquetes que se pueden enviar. Estos buffers son como el siguiente:

```
Char ip_tcp[sizeof(struct ipheader)+sizeof(struct tcpheader)]
```

Como se puede observar, creamos un array de variables char del tamaño de las cabeceras. Como se va a formar un paquete TCP, tenemos que incluir la cabecera IP y la cabecera TCP. Una vez realizado esto solo falta rellenar este buffer. Para rellenarlo se hace coincidir un puntero que pertenezca a una estructura de la cabecera adecuada al sitio del buffer correcto. Esto se realiza en las funciones que rellenan los paquetes de la siguiente manera:

```
struct ipheader *iph;

iph= (struct ipheader *) ip_tcp;

struct tcpheader *tcph

tcph=(struct tcpheader*)(ip_tcp+sizeof(struct ipheader))
```

Simplemente estamos apuntando las estructuras específicas al sitio que le corresponde dentro del array para que se pueda completar las cabeceras como si estuviésemos rellenando una estructura normal. Por ejemplo, si se quiere mandar el paquete TCP al puerto 80, se hace lo siguiente:

```
tcph->th_dport = 80;
```

En este comando se puede observar que tcph es el puntero de la esctructura y th_dport es el campo del puerto destino de la estructura TCP que se ha explicado al principio de este apartado. Al rellenar los campos hay que tener mucho cuidado en como se guardan los datos. Dependiendo de la máquina guarda el bit más significativo a un lado o a otro y es posible que cuando se introduzcan los datos se guarden de una manera que al enviar el paquete no se entiendan porque están al revés. Para solucionar esto existen los comandos htons y htonl dependiendo del tamaño de la variable.

Para el caso contrario se hace exactamente lo mismo pero al revés. En el paquete ya está guardada la información, por lo tanto se tiene que leer. Para ello se sigue el método explicado anteriormente, se apunta la estructura al buffer de la siguiente manera:

```
struct ipheader *ip;

ip = (struct ipheader *)paquete;
```

En este caso, *paquete* es donde se ha guardado previamente el paquete recibido. Una vez realizado estas instrucciones, sólo se tiene que leer el valor de la siguiente manera:

```
Protocolo = ip->ip_p;
```

Protocolo es la variable donde se guarda el valor que hay en el campo protocolo de la cabecera IP.

Todos los comandos que no se han explicado en este apartado son los clásicos que se utilizan con normalidad en cada programa.

CAPITULO 4. CONCLUSIONES

En este proyecto se ha realizado un estudio amplio del fingerprinting. Se han estudiado tanto las bases teóricas como las prácticas, se han estudiado programas ampliamente utilizados y se ha creado una aplicación que es la base para poder realizar fingerprinting.

En el apartado introducción se han explicado los objetivos que se tenían al empezar a realizar este trabajo. Uno de estos objetivos era hacer un estudio de las características de las técnicas de fingerprinting. Esto se ha realizado en el apartado 1.1 donde se explica pormenorizadamente de que se trata cada una de las posibles pruebas de las que consiste hacer fingerprinting ya sea pasivo o activo. Otro de los objetivos era hacer un estudio de los protocolos TCP e IP para poder relacionarlo con las técnicas de fingerprinting. En el apartado 1.2 se ha resumido el RFC 1122, que especifica los requerimientos que tiene que tener un host para conectarse a Internet. Este RFC es el que regula todo lo relacionado con los que los protocolos están obligados ha realizar, también explica que no están obligados, que cosas son opcionales y que no se puede realizar bajo ningún concepto. Todas estas reglas son aplicables para los protocolos de las capas de enlace, internet y transporte. Además, para completar esta información, el apartado 1.3 hace un estudio de cómo afecta los firewalls al fingerprinting viéndolo desde el ángulo de la seguridad con las contramedidas que utilizan los firewalls para evitar que le realicen fingerprinting y como afecta esto a las pruebas. Se ha observado que cuando actúa un firewall los resultados de las pruebas fingerprinting se ven seriamente afectados y mientras no actúa los resultados son muy acertados.

El tercer objetivo es el que más tiempo ha ocupado. Se ha estudiado con detalle dos aplicaciones muy utilizadas en el capítulo 2. Al estudiar estas aplicaciones se ha visto de que manera estos programas averiguaban el sistema operativo remoto, como guardan los datos y como salen representados los diferentes dispositivos en sus bases de datos. Se ha observado que el programa nmap es muy útil y se puede utilizar como base para la aplicación explicada en el capítulo 3. También se ha observado que el nmap, tal y como está implementado, no es exactamente lo que se necesita. También hay que decir que el winfingerprinting no es válido para los fines necesitados en este proyecto debido a la manera con la que está implementado y a los protocolos que utiliza.

Finalmente se han diseñado las bases para una aplicación de fingerprinting adaptada a las necesidades del proyecto. Está desarrollada en el capitulo 3. Esta aplicación es la base que se puede utilizar para empezar a caracterizar estos dispositivos de una manera que se ajustase más a las necesidades de este proyecto. El porqué tenemos otras necesidades es debido a que los programas dedicados a fingerprinting que se han encontrado están basados en hallar el sistema operativo remoto y en este trabajo no se quiere eso, se quiere conocer las características de su pila de protocolos.

Por lo tanto se puede decir que se han cumplido los tres objetivos principales de una manera amplia.

Ahora bien, en cuanto a la aplicación que se ha realizado hay que decir que es muy básica, pero establece las partes esenciales que se necesitaban.

Para un futuro esta aplicación se tendría que ampliar o mejorar en diferentes aspectos. Una de las mejoras tendría que ser dotar a la aplicación de funcionalidades para realizar los test adecuados y concretos para que se pudiese especular sobre las características de la pila del sistema remoto. Estas funcionalidades más complejas serían las que tienen por ejemplo el nmap, como la función que realiza el fingerprinting o la función que hace un escaneo de puertos o con mensajes ICMP, adaptados a los objetivos de la aplicación de este proyecto.

Otra parte importante a ampliar consiste en confeccionar una base de datos específica de respuestas a las pruebas a un conjunto variado de dispositivos con implementaciones de pila TCP/IP diferente. Una vez acabado todos estos pasos, seria posible realizar una aplicación autónoma que pudiera analizar cualquier dispositivo y dar resultados coherentes.

BIBLIOGRAFÍA

[1] McClure, S, Scambray, J , Kurtz, G, "*Hackers 3, Secretos y soluciones para la seguridad de redes*". Editado por McGraw-Hill en el 2002

[2] Know Your Enemy: Passive Fingerprinting.
En este articulo se encuetra la descripción del fingerprinting pasivo. Fue encontrado en la página Web del projecto Honeynet (http://project.honeynet.org). Consultado en marzo del 2005: http://project.honeynet.org/papers/finger/

[3] Remote OS detection via TCP/IP Stack FingerPrinting
En este articulo se encuetra la descripción del fingerprinting activo. Fue encontrado en la página Web del nmap (www.insecure.org).Consultado en marzo del 2005:
http://project.honeynet.org/papers/finger/

[4] Bramen, R, "*RFC1122, Requerimientos para un host de internet*" Editado en 1989.

[5] Cisco PIX Security Appliance Software Version 7.0
Se puede encontrar en la siguiente página Web: http://www.cisco.com/en/US/products/hw/vpndevc/ps2030/products_data_sheet0900aecd80225ae1.html

[6 3com SuperStack Firewall.
Se puede encontrar en la siguiente página Web: http://www.3com.com/prod/es_ES_EMEA/prodlist.jsp?tab=cat&cat=134482&subcat=134490

[7] Aplicación nmap:
La versión utilizada es la 3.81 que se encontraba vigente en marzo del 2005. Se puede encontrar en la siguiente página Web: http://www.insecure.org/nmap/nmap_download.html

[8] Aplicación Winfinerprinting:
La versión utilizada es la 1.75 que se encontraba vigente en marzo del 2005. Se puede encontrar en la siguiente página Web: http://cvs.sourceforge.net/viewcvs.py/winfingerprint/winfingerprint/winfingerprint.plg

[9] Microsoft msdn ("*Microsoft Developer Network*"):
Página Web de consulta cuando surgen problemas con el código de programación C. Se puede encontrar en la siguiente página Web: http://msdn.microsoft.com/

Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TITULO DEL TFC: Caracterización de un equipo de usuario mediante técnicas de fingerprinting

TITULACIÓN: Enginiería Técnica de Telecomunicaciones, especialidad en Telemàtica

AUTOR: José Manuel García Carrasco

DIRECTOR: Luís Casals Ibáñez

FECHA: 29 de junio de 2005

Anexo 1. Resumen de los requerimientos de un host de Internet

A continuación se expone unos cuadros resumen del RFC 1122 donde se puede encontrar todas las características de los protocolos TCP/IP. Con este cuadro resumen se puede tener toda la información de una forma detallada y compacta para poder consultarla en caso de necesidad.

LINK LAYER REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT
Trailer encapsulation	2.3.1			x		
Send Trailers by default without negotiation	2.3.1					x
ARP	2.3.2					
Flush out-of-date ARP cache entries	2.3.2.1	x				
Prevent ARP floods	2.3.2.1	x				
Cache timeout configurable	2.3.2.1		x			
Save at least one (latest) unresolved pkt	2.3.2.2		x			
Ethernet and IEEE 802 Encapsulation	2.3.3					
Host able to:	2.3.3					
Send & receive RFC-894 encapsulation	2.3.3	x				
Receive RFC-1042 encapsulation	2.3.3		x			
Send RFC-1042 encapsulation	2.3.3			x		
Then config. sw. to select, RFC-894 dflt	2.3.3	x				
Send K1=6 encapsulation	2.3.3					x
Use ARP on Ethernet and IEEE 802 nets	2.3.3	x				
Link layer report b'casts to IP layer	2.4		x			
IP layer pass TOS to link layer	2.4		x			
No ARP cache entry treated as Dest. Unreach.	2.4					x

INTERNET LAYER REQUIREMENTS SUMMARY

FEATURE	SECTION	ST	LD	AY	NOT	NOT	tte
Implement IP and ICMP	3.1	x					
Handle remote multihoming in application layer	3.1	x					
Support local multihoming	3.1			x			
Meet gateway specs if forward datagrams	3.1	x					
Configuration switch for embedded gateway	3.1	x					1
Config switch default to non-gateway	3.1	x					1
Auto-config based on number of interfaces	3.1					x	1
Able to log discarded datagrams	3.1		x				
Record in counter	3.1		x				
Silently discard Version != 4	3.2.1.1	x					
Verify IP checksum, silently discard bad dgram	3.2.1.2	x					
Addressing:							
Subnet addressing (RFC-950)	3.2.1.3	x					
Src address must be host's own IP address	3.2.1.3	x					
Silently discard datagram with bad dest addr	3.2.1.3	x					
Silently discard datagram with bad src addr	3.2.1.3	x					
Support reassembly	3.2.1.4	x					
Retain same Id field in identical datagram	3.2.1.5			x			
TOS:							
Allow transport layer to set TOS	3.2.1.6	x					
Pass received TOS up to transport layer	3.2.1.6		x				
Use RFC-795 link-layer mappings for TOS	3.2.1.6				x		
TTL:							
Send packet with TTL of 0	3.2.1.7					x	
Discard received packets with TTL < 2	3.2.1.7					x	
Allow transport layer to set TTL	3.2.1.7	x					
Fixed TTL is configurable	3.2.1.7	x					
IP Options:							
Allow transport layer to send IP options	3.2.1.8	x					
Pass all IP options rcvd to higher layer	3.2.1.8	x					
IP layer silently ignore unknown options	3.2.1.8	x					
Security option	3.2.1.8a			x			
Send Stream Identifier option	3.2.1.8b				x		
Silently ignore Stream Identifer option	3.2.1.8b	x					
Record Route option	3.2.1.8d			x			
Timestamp option	3.2.1.8e			x			
Source Route Option:							
Originate & terminate Source Route options	3.2.1.8c	x					
Datagram with completed SR passed up to TL	3.2.1.8c	x					
Build correct (non-redundant) return route	3.2.1.8c	x					
Send multiple SR options in one header	3.2.1.8c					x	
ICMP:							
Silently discard ICMP msg with unknown type	3.2.2	x					
Include more than 8 octets of orig datagram	3.2.2			x			
Included octets same as received	3.2.2	x					
Demux ICMP Error to transport protocol	3.2.2	x					
Send ICMP error message with TOS=0	3.2.2		x				
Send ICMP error message for:							
- ICMP error msg	3.2.2					x	
- IP b'cast or IP m'cast	3.2.2					x	
- Link-layer b'cast	3.2.2					x	
- Non-initial fragment	3.2.2					x	

- Datagram with non-unique src address	3.2.2						x
Return ICMP error msgs (when not prohibited)	3.3.8	x					
Dest Unreachable:							
Generate Dest Unreachable (code 2/3)	3.2.2.1		x				
Pass ICMP Dest Unreachable to higher layer	3.2.2.1	x					
Higher layer act on Dest Unreach	3.2.2.1		x				
Interpret Dest Unreach as only hint	3.2.2.1	x					
Redirect:							
Host send Redirect	3.2.2.2					x	
Update route cache when recv Redirect	3.2.2.2	x					
Handle both Host and Net Redirects	3.2.2.2	x					
Discard illegal Redirect	3.2.2.2		x				
Source Quench:							
Send Source Quench if buffering exceeded	3.2.2.3			x			
Pass Source Quench to higher layer	3.2.2.3	x					
Higher layer act on Source Quench	3.2.2.3		x				
Time Exceeded: pass to higher layer	3.2.2.4	x					
Parameter Problem:							
Send Parameter Problem messages	3.2.2.5		x				
Pass Parameter Problem to higher layer	3.2.2.5	x					
Report Parameter Problem to user	3.2.2.5			x			
ICMP Echo Request or Reply:							
Echo server and Echo client	3.2.2.6	x					
Echo client	3.2.2.6		x				
Discard Echo Request to broadcast address	3.2.2.6			x			
Discard Echo Request to multicast address	3.2.2.6			x			
Use specific-dest addr as Echo Reply src	3.2.2.6	x					
Send same data in Echo Reply	3.2.2.6	x					
Pass Echo Reply to higher layer	3.2.2.6	x					
Reflect Record Route, Time Stamp options	3.2.2.6		x				
Reverse and reflect Source Route option	3.2.2.6	x					
ICMP Information Request or Reply:	3.2.2.7					x	
ICMP Timestamp and Timestamp Reply:	3.2.2.8			x			
Minimize delay variability	3.2.2.8		x				1
Silently discard b'cast Timestamp	3.2.2.8			x			1
Silently discard m'cast Timestamp	3.2.2.8			x			1
Use specific-dest addr as TS Reply src	3.2.2.8	x					1
Reflect Record Route, Time Stamp options	3.2.2.6		x				1
Reverse and reflect Source Route option	3.2.2.8	x					1
Pass Timestamp Reply to higher layer	3.2.2.8	x					1
Obey rules for "standard value"	3.2.2.8	x					1
ICMP Address Mask Request and Reply:							
Addr Mask source configurable	3.2.2.9	x					
Support static configuration of addr mask	3.2.2.9	x					
Get addr mask dynamically during booting	3.2.2.9			x			
Get addr via ICMP Addr Mask Request/Reply	3.2.2.9			x			
Retransmit Addr Mask Req if no Reply	3.2.2.9	x					3
Assume default mask if no Reply	3.2.2.9		x				3
Update address mask from first Reply only	3.2.2.9	x					3
Reasonableness check on Addr Mask	3.2.2.9		x				
Send unauthorized Addr Mask Reply msgs	3.2.2.9					x	
Explicitly configured to be agent	3.2.2.9	x					
Static config=> Addr-Mask-Authoritative flag	3.2.2.9		x				
Broadcast Addr Mask Reply when init.	3.2.2.9	x					3
ROUTING OUTBOUND DATAGRAMS:							
Use address mask in local/remote decision	3.3.1.1	x					

Operate with no gateways on conn network	3.3.1.1	x				
Maintain "route cache" of next-hop gateways	3.3.1.2	x				
Treat Host and Net Redirect the same	3.3.1.2		x			
If no cache entry, use default gateway	3.3.1.2	x				
Support multiple default gateways	3.3.1.2	x				
Provide table of static routes	3.3.1.2			x		
Flag: route overridable by Redirects	3.3.1.2			x		
Key route cache on host, not net address	3.3.1.3			x		
Include TOS in route cache	3.3.1.3		x			
Able to detect failure of next-hop gateway	3.3.1.4	x				
Assume route is good forever	3.3.1.4				x	
Ping gateways continuously	3.3.1.4					x
Ping only when traffic being sent	3.3.1.4	x				
Ping only when no positive indication	3.3.1.4	x				
Higher and lower layers give advice	3.3.1.4		x			
Switch from failed default g'way to another	3.3.1.5	x				
Manual method of entering config info	3.3.1.6	x				
REASSEMBLY and FRAGMENTATION:						
Able to reassemble incoming datagrams	3.3.2	x				
At least 576 byte datagrams	3.3.2	x				
EMTU_R configurable or indefinite	3.3.2		x			
Transport layer able to learn MMS_R	3.3.2	x				
Send ICMP Time Exceeded on reassembly timeout	3.3.2	x				
Fixed reassembly timeout value	3.3.2		x			
Pass MMS_S to higher layers	3.3.3	x				
Local fragmentation of outgoing packets	3.3.3			x		
Else don't send bigger than MMS_S	3.3.3	x				
Send max 576 to off-net destination	3.3.3		x			
All-Subnets-MTU configuration flag	3.3.3			x		
MULTIHOMING:						
Reply with same addr as spec-dest addr	3.3.4.2		x			
Allow application to choose local IP addr	3.3.4.2	x				
Silently discard d'gram in "wrong" interface	3.3.4.2			x		
Only send d'gram through "right" interface	3.3.4.2			x		4
SOURCE-ROUTE FORWARDING:						
Forward datagram with Source Route option	3.3.5			x		1
Obey corresponding gateway rules	3.3.5	x				1
Update TTL by gateway rules	3.3.5	x				1
Able to generate ICMP err code 4, 5	3.3.5	x				1
IP src addr not local host	3.3.5			x		1
Update Timestamp, Record Route options	3.3.5	x				1
Configurable switch for non-local SRing	3.3.5	x				1
Defaults to OFF	3.3.5	x				1
Satisfy gwy access rules for non-local SRing	3.3.5	x				1
If not forward, send Dest Unreach (cd 5)	3.3.5		x			2
BROADCAST:						
Broadcast addr as IP source addr	3.2.1.3				x	
Receive 0 or -1 broadcast formats OK	3.3.6		x			
Config'ble option to send 0 or -1 b'cast	3.3.6			x		
Default to -1 broadcast	3.3.6		x			
Recognize all broadcast address formats	3.3.6	x				
Use IP b'cast/m'cast addr in link-layer b'cast	3.3.6	x				
Silently discard link-layer-only b'cast dg's	3.3.6		x			
Use Limited Broadcast addr for connected net	3.3.6		x			

MULTICAST:						
Support local IP multicasting (RFC-1112)	3.3.7		x			
Support IGMP (RFC-1112)	3.3.7			x		
Join all-hosts group at startup	3.3.7		x			
Higher layers learn i'face m'cast capability	3.3.7		x			
INTERFACE:						
Allow transport layer to use all IP mechanisms	3.4	x				
Pass interface ident up to transport layer	3.4	x				
Pass all IP options up to transport layer	3.4	x				
Transport layer can send certain ICMP messages	3.4	x				
Pass spec'd ICMP messages up to transp. layer	3.4	x				
Include IP hdr+8 octets or more from orig.	3.4	x				
Able to leap tall buildings at a single bound	3.5		x			

UDP REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHLD	MUST NOT	SHLD NOT	FOOTNOTE
UDP						
UDP send Port Unreachable	4.1.3.1		x			
IP Options in UDP						
- Pass rcv'd IP options to applic layer	4.1.3.2	x				
- Applic layer can specify IP options in Send	4.1.3.2	x				
- UDP passes IP options down to IP layer	4.1.3.2	x				
Pass ICMP msgs up to applic layer	4.1.3.3	x				
UDP checksums:						
- Able to generate/check checksum	4.1.3.4	x				
- Silently discard bad checksum	4.1.3.4	x				
- Sender Option to not generate checksum	4.1.3.4			x		
- Default is to checksum	4.1.3.4	x				
- Receiver Option to require checksum	4.1.3.4			x		
UDP Multihoming						
- Pass spec-dest addr to application	4.1.3.5	x				
- Applic layer can specify Local IP addr	4.1.3.5	x				
- Applic layer specify wild Local IP addr	4.1.3.5	x				
- Applic layer notified of Local IP addr used	4.1.3.5		x			
Bad IP src addr silently discarded by UDP/IP	4.1.3.6	x				
Only send valid IP source address	4.1.3.6	x				
UDP Application Interface Services						
Full IP interface of 3.4 for application	4.1.4	x				
- Able to spec TTL, TOS, IP opts when send dg	4.1.4	x				

- Pass received TOS up to applic layer	4.1.4			x		

TCP REQUIREMENT SUMMARY

FEATURE	SECTION	MUST	SHLD	MUST NOT	SHLD NOT	Footnote
Push flag						
Aggregate or queue un-pushed data	4.2.2.2			x		
Sender collapse successive PSH flags	4.2.2.2		x			
SEND call can specify PUSH	4.2.2.2			x		
If cannot: sender buffer indefinitely	4.2.2.2				x	
If cannot: PSH last segment	4.2.2.2	x				
Notify receiving ALP of PSH	4.2.2.2			x		1
Send max size segment when possible	4.2.2.2		x			
Window						
Treat as unsigned number	4.2.2.3	x				
Handle as 32-bit number	4.2.2.3		x			
Shrink window from right	4.2.2.16			x		
Robust against shrinking window	4.2.2.16	x				
Receiver's window closed indefinitely	4.2.2.17			x		
Sender probe zero window	4.2.2.17	x				
First probe after RTO	4.2.2.17		x			
Exponential backoff	4.2.2.17		x			
Allow window stay zero indefinitely	4.2.2.17	x				
Sender timeout OK conn with zero wind	4.2.2.17				x	
Urgent Data						
Pointer points to last octet	4.2.2.4	x				
Arbitrary length urgent data sequence	4.2.2.4	x				
Inform ALP asynchronously of urgent data	4.2.2.4	x				1
ALP can learn if/how much urgent data Q'd	4.2.2.4	x				1
TCP Options						
Receive TCP option in any segment	4.2.2.5	x				
Ignore unsupported options	4.2.2.5	x				
Cope with illegal option length	4.2.2.5	x				
Implement sending & receiving MSS option	4.2.2.6	x				
Send MSS option unless 536	4.2.2.6		x			
Send MSS option always	4.2.2.6			x		
Send-MSS default is 536	4.2.2.6	x				
Calculate effective send seg size	4.2.2.6	x				
TCP Checksums						
Sender compute checksum	4.2.2.7	x				
Receiver check checksum	4.2.2.7	x				
Use clock-driven ISN selection	4.2.2.9	x				

Opening Connections						
Support simultaneous open attempts	4.2.2.10	x				
SYN-RCVD remembers last state	4.2.2.11	x				
Passive Open call interfere with others	4.2.2.18					x
Function: simultan. LISTENs for same port	4.2.2.18	x				
Ask IP for src address for SYN if necc.	4.2.3.7	x				
Otherwise, use local addr of conn.	4.2.3.7	x				
OPEN to broadcast/multicast IP Address	4.2.3.14					x
Silently discard seg to bcast/mcast addr	4.2.3.14	x				
Closing Connections						
RST can contain data	4.2.2.12		x			
Inform application of aborted conn	4.2.2.13	x				
Half-duplex close connections	4.2.2.13			x		
Send RST to indicate data lost	4.2.2.13		x			
In TIME-WAIT state for 2xMSL seconds	4.2.2.13	x				
Accept SYN from TIME-WAIT state	4.2.2.13			x		
Retransmissions						
Jacobson Slow Start algorithm	4.2.2.15	x				
Jacobson Congestion-Avoidance algorithm	4.2.2.15	x				
Retransmit with same IP ident	4.2.2.15			x		
Karn's algorithm	4.2.3.1	x				
Jacobson's RTO estimation alg.	4.2.3.1	x				
Exponential backoff	4.2.3.1	x				
SYN RTO calc same as data	4.2.3.1		x			
Recommended initial values and bounds	4.2.3.1		x			
Generating ACK's:						
Queue out-of-order segments	4.2.2.20		x			
Process all Q'd before send ACK	4.2.2.20	x				
Send ACK for out-of-order segment	4.2.2.21			x		
Delayed ACK's	4.2.3.2		x			
Delay < 0.5 seconds	4.2.3.2	x				
Every 2nd full-sized segment ACK'd	4.2.3.2	x				
Receiver SWS-Avoidance Algorithm	4.2.3.3	x				
Sending data						
Configurable TTL	4.2.2.19	x				
Sender SWS-Avoidance Algorithm	4.2.3.4	x				
Nagle algorithm	4.2.3.4		x			
Application can disable Nagle algorithm	4.2.3.4	x				
Connection Failures:						
Negative advice to IP on R1 retxs	4.2.3.5	x				
Close connection on R2 retxs	4.2.3.5	x				
ALP can set R2	4.2.3.5	x				1
Inform ALP of R1<=retxs<R2	4.2.3.5		x			1
Recommended values for R1, R2	4.2.3.5		x			
Same mechanism for SYNs	4.2.3.5	x				
R2 at least 3 minutes for SYN	4.2.3.5	x				
Send Keep-alive Packets:	4.2.3.6			x		
- Application can request	4.2.3.6	x				
- Default is "off"	4.2.3.6	x				
- Only send if idle for interval	4.2.3.6	x				
- Interval configurable	4.2.3.6	x				
- Default at least 2 hrs.	4.2.3.6	x				
- Tolerant of lost ACK's	4.2.3.6	x				
IP Options						

Ignore options TCP doesn't understand	4.2.3.8	x					
Time Stamp support	4.2.3.8			x			
Record Route support	4.2.3.8			x			
Source Route:							
ALP can specify	4.2.3.8	x					1
Overrides src rt in datagram	4.2.3.8	x					
Build return route from src rt	4.2.3.8	x					
Later src route overrides	4.2.3.8		x				
Receiving ICMP Messages from IP	4.2.3.9	x					
Dest. Unreach (0,1,5) => inform ALP	4.2.3.9		x				
Dest. Unreach (0,1,5) => abort conn	4.2.3.9					x	
Dest. Unreach (2-4) => abort conn	4.2.3.9		x				
Source Quench => slow start	4.2.3.9		x				
Time Exceeded => tell ALP, don't abort	4.2.3.9		x				
Param Problem => tell ALP, don't abort	4.2.3.9		x				
Address Validation							
Reject OPEN call to invalid IP address	4.2.3.10	x					
Reject SYN from invalid IP address	4.2.3.10	x					
Silently discard SYN to bcast/mcast addr	4.2.3.10	x					
TCP/ALP Interface Services							
Error Report mechanism	4.2.4.1	x					
ALP can disable Error Report Routine	4.2.4.1		x				
ALP can specify TOS for sending	4.2.4.2	x					
Passed unchanged to IP	4.2.4.2		x				
ALP can change TOS during connection	4.2.4.2		x				
Pass received TOS up to ALP	4.2.4.2			x			
FLUSH call	4.2.4.3			x			
Optional local IP addr parm. in OPEN	4.2.4.4	x					

Anexo 2. Ejemplos de la base de datos del programa nmap

En este anexo se puede encontrar unos ejemplos de las entradas que hay en la base de datos de la aplicación nmap. Estas entradas se corresponden a las diferentes respuestas de los sistemas operativos a los que previamente se les ha realizado las pruebas comentadas en el apartado 2.1.1. Aquí podemos encontrar los sistemas operativos más usados como BSD, Linux o Windows.

Fingerprint BSDI BSD/OS 4.0-4.0.1

Class BSDI | BSD/OS | 4.X | general purpose
TSeq(Class=RI%gcd=<A%SI=<1974A&>16F)
T1(DF=Y%W=2017%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=2017%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=15C%RID=E%RIPCK=0%UCK=E%UL
EN=134%DAT=E)

Fingerprint BSDI BSD/OS 4.0.1

Class BSDI | BSD/OS | 4.X | general purpose
TSeq(Class=RI%gcd=<6%SI=<F85E8&>27A9)
T1(DF=Y%W=2017%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=2017%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E|F%
ULEN=134%DAT=E)

Fingerprint BSDI BSD/OS 4.2

Class BSDI | BSD/OS | 4.X | general purpose
TSeq(Class=RI%gcd=<6%SI=<F3E94&>1A4F%IPID=I%TS=2HZ)
T1(DF=Y%W=805C%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=805C%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%UL
EN=134%DAT=E)

Fingerprint FreeBSD 5.1-RELEASE (x86)

Class FreeBSD | FreeBSD | 5.X | general purpose
TSeq(Class=TR%gcd=<6%IPID=I%TS=100HZ)
T1(DF=Y%W=8000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=8000%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=0%UL
EN=134%DAT=E)

Fingerprint FreeBSD 5.2-CURRENT (Jan 2004) on x86

Class FreeBSD | FreeBSD | 5.X | general purpose
T1(DF=Y%W=FFFF%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=FFFF%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=0%UL
EN=134%DAT=E)

FreeBSD 5.2-CURRENT (Jun 25, 2004) on x86 running pf as firewall with "scrub in all"

FreeBSD 5.3-Beta2 (x86)
Fingerprint FreeBSD 5.2-CURRENT - 5.3-BETA2 (x86) with pf scrub all
Class FreeBSD | FreeBSD | 5.X | general purpose
TSeq(Class=TR%gcd=<6%IPID=I%TS=100HZ)
T1(DF=Y%W=FFFF%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=N)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=N)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=0%UL
EN=134%DAT=E)

Fingerprint Linux 2.5.5 (Gentoo)

Class Linux | Linux | 2.6.X | general purpose
TSeq(Class=RI%gcd=<6%SI=<306B8C0&>6ABCE%IPID=Z%TS=1000HZ)
T1(DF=Y%W=1680%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=1680%ACK=S++%Flags=AS%Ops=MNNTNW)

T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=F%U
LEN=134%DAT=E)

Linux 2.6.6-rc2-bk3

Fingerprint Linux 2.6.6
Class Linux | Linux | 2.6.X | general purpose
TSeq(Class=RI%gcd=<6%SI=<2E04BFE&>4ADED%IPID=Z%TS=1000HZ)
T1(DF=Y%W=1164%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=1164%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%
ULEN=134%DAT=E)

Debian Sarge Kernel 2.6.8

Fingerprint Linux 2.6.8 (Debian)
Class Linux | Linux | 2.6.X | general purpose
TSeq(Class=RI%gcd=<6%SI=<2C96D4E&>34B7A%IPID=Z%TS=100HZ)
T1(DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=N)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=N)
PU(DF=Y%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%
ULEN=134%DAT=E)

Fingerprint Microsoft Windows 98

Class Microsoft | Windows | 95/98/ME | general purpose
TSeq(Class=RI%gcd=<6%SI=<FFF)
T1(DF=Y%W=BB80%ACK=S++%Flags=AS%Ops=M)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=BB80%ACK=S++%Flags=AS%Ops=M)
T4(DF=N%W=0%ACK=S++%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=S++%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%UL
EN=134%DAT=E)

Microsoft Windows 2003 Server (Version 5.2 build 3790.srv03_rtm.030324-2048)

Fingerprint Microsoft Windows 2003 Server
Class Microsoft | Windows | 2003/.NET | general purpose
TSeq(Class=TR%gcd=<6%IPID=I%TS=U)
T1(DF=N%W=7FFF%ACK=S++%Flags=AS%Ops=MNW)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=N%W=7FFF%ACK=S++%Flags=AS%Ops=MNW)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=B0%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)

Fingerprint Microsoft Windows XP Pro

Class Microsoft | Windows | NT/2K/XP | general purpose
TSeq(Class=RI%gcd=<6%SI=<2EBEE&>45E%IPID=I%TS=U)
T1(DF=Y%W=F424%ACK=S++%Flags=AS%Ops=MNW)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=F424%ACK=S++%Flags=AS%Ops=MNW)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)

Fingerprint OpenBSD 3.5 with pf "scrub in all"

Class OpenBSD | OpenBSD | 3.X | general purpose
TSeq(Class=TR%gcd=<6%IPID=RD%TS=2HZ)
T1(DF=Y%W=4000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=N)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=N)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)