INSTITUTO TECNOLÓGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY

CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN TECNOLOGÍAS DE INFORMACIÓN Y ELECTRÓNICA

FLF4DoS:
MITIGACIÓN DE DDOS MEDIANTE UNA TÉCNICA DE FILTRADO BASADA EN EL CAMPO TTL USANDO AMBIENTES VIRTUALES EN LINUX Y REGLAS DE LÓGICA DIFUSA

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA
OBTENER EL GRADO ACADÉMICO DE:

MAESTRO EN CIENCIAS EN TECNOLOGÍA INFORMÁTICA

POR:

JULIO CÉSAR COVARRUBIAS RODRÍGUEZ

MONTERREY, N. L. DICIEMBRE 2005

Capítulo 1

Introducción

La información es un activo mas para las organizaciones el cual tiene valor y necesita ser protegido contra cualquier tipo de ataques. La seguridad de información protege a ésta de una amplia variedad de amenazas para asegurar la continuidad del negocio y minimizar los daños en la organización. La información puede existir en diferentes formas. Impresa en papel, almacenada electrónicamente, transmitida por correo tradicional o electrónico, de manera hablada, entre otras. Cualquiera que sea la forma en la que exista, debe ser protegida.
La información tiene 3 características importantes según Pfleeger en [26]:

- Confidencialidad: Significa que los activos de un sistema de cómputo estén accesibles solo por las partes autorizadas. El tipo de acceso es de lectura pudiendo ser: lectura, vista, impresa o aún sólo conociendo la existencia de ella. La confidencialidad algunas veces es llamada secrecía o privacidad.
- Integridad: Significa que los activos pueden ser modificados sólo por las partes autorizadas o sólo de forma autorizada. En este contexto, las modificaciones incluyen: escritura, cambios, cambios de estado, borrado y creación.
- Disponibilidad: Significa que los activos están accesibles a las partes autorizadas. A una parte autorizada no debe impedírsele el acceso a objetos a los cuales se tiene acceso legítimo.

La confidencialidad, integridad y disponibilidad de la información son esenciales para que una organización mantenga un nivel competitivo, buen flujo de capital y buena imagen.

Internet conecta a un gran número de computadoras a través de todo el mundo, corriendo en múltiples sistemas de software y hardware. Le da servicio a una cantidad enorme de personas y a las necesidades profesionales de la gente y de las empresas. Al ser una red a la que cualquier persona con una computadora con un modem, una línea telefónica y un acceso a la red proporcionado por algún Internet Service Provider (ISP), hace posible el acceso a la red y a los servicios de las organizaciones que ofrecen servicios en línea tales como bancos, búsquedas de información, compras en línea, pago de servicios, reservaciones, etc.

1.1. Definición del problema

La negación de servicios, DoS (Denial of Service), es un problema que consume el ancho de banda de una red o los recursos de un servidor, degradándolos a tal punto que impide que los usuarios legítimos tengan acceso a los recursos de red o del servidor atacado[1]. En algunas ocasiones, la degradación de los servicios es tal, que se llegan a interrumpir completamente por algún tiempo, provocando que los sistemas tengan que ser reiniciados y reconfigurados. Afectando principalmente la característica de disponibilidad mencionada enteriormente.

En un ataque DoS, principalmente interviene un atacante usando solo un origen de ataque, enviando miles de solicitudes de algún tipo, con el propósito de consumir el ancho de banda de la víctima o degradar los recursos de un servidor.

El problema aumenta cuando en el ataque intervienen más de un origen, este tipo de ataques es llamado Distributed Denial of Services (DDoS) [1,2], en el cual el atacante previamente ha comprometido otras máquinas conectadas a Internet instalándoles un programa conocido como “zombie”, el cual espera una señal del atacante para activarse y comenzar el ataque contra un blanco específico.

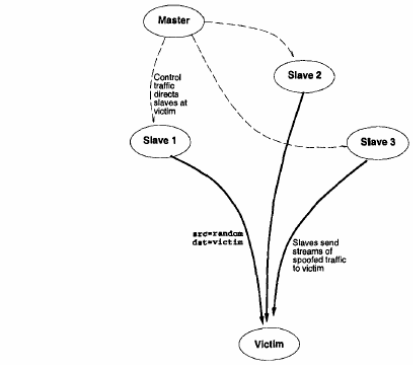

Figura 1.1: DoS distribuido con la ayuda de agentes o esclavos

Existe otra forma de realizar estos ataques, usando reflectores [2], en el cual el atacante compromete una cantidad enorme de máquinas conectadas a Internet en los cuales ya ha instalado un zombie para tener control sobre cada una de éstas. Al momento de que se inicia un ataque, las máquinas comprometidas o esclavas, reciben una señal para comenzar el ataque y envían miles de paquetes con solicitudes de algún tipo a los reflectores, los cuales son máquinas que no han sido comprometidas por el atacante, simplemente son usadas para responder a una falsa solicitud debido a que los reflectores reciben los paquetes con la dirección de origen de la víctima y por lo tanto enviaran la respuesta a la misma, provocándole un DDoS.

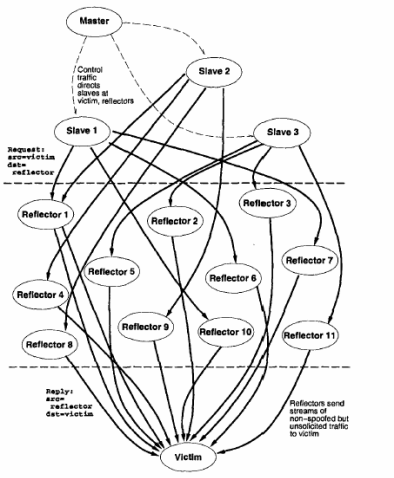

Figura 1.2: DoS Distribuido y usando reflectores

1.2. Justificación

Actualmente, las organizaciones y sus sistemas de información y redes se están enfrentando con amenazas de seguridad desde un amplio rango de fuentes, incluyendo fraude por computadora, espionaje, sabotaje, etc. Amenazas como virus de computadoras, hackers y ataques de negación de servicios se han convertido en los ataques más comunes, más ambiciosos y más sofisticados. La dependencia en los sistemas de información y servicios en línea hace que las organizaciones sean más vulnerables a las amenazas de seguridad. La interconectividad de redes privadas y públicas y la compartición de información incrementan el grado de dificultad de lograr un adecuado control de acceso. Las tendencias a distribuir la información y el cómputo móvil, ha debilitado la efectividad de lo centralizado, especialmente el control de acceso y seguridad. Muchos sistemas de información no han sido diseñados para ser seguros.

Todos estos servicios ofrecidos por diferentes tipos de organizaciones, al estar interconectadas en red pueden sufrir ataques, uno de ellos es la negación de servicios, el cual se ha sofisticado con el paso del tiempo y ha causado pérdidas económicas muy grandes. Este ataque afecta a la disponibilidad de la información, una característica importante, ya que puede interrumpir el funcionamiento de los servicios de las empresas como facturación, reservaciones, servicios de acceso remoto a librerías digitales. Se puede decir que cualquier red de computadoras es propensa a sufrir algún tipo de ataque de DoS.

En Febrero del 2000, se llevaron a cabo ataques de negación de servicios, contra algunos sitios conocidos como son CNN.com, eBay, Amazon [21] y Yahoo! [21,22]. Estos ataques interrumpieron el funcionamiento de los servicios de estas organizaciones, impidiendo el acceso a usuarios legítimos.

Los ataques de este tipo cada vez son más fáciles de llevar a cabo debido a que existen muchas herramientas [5], que permiten realizarlos y que se encuentran disponibles en Internet. Estas herramientas únicamente necesitan de una conexión a Internet o a alguna red y la víctima a la que se desea atacar, haciendo posible que no se necesiten conocimientos profundos acerca del DoS.

Los ataques de DoS han producido pérdidas económicas a las empresas, el CSI (Computer Security Institute)[24] del FBI (Federal Bureau of Investigations) en su reporte anual Computer Crime and Security Survey, en la edición 2003 muestran los resultados de las perdidas económicas de las empresas de acuerdo al tipo de incidente. Mostrando en primer lugar al robo de información propietaria y al DoS en segundo lugar. La figura 1.3 muestra las perdidas económicas de las principales amenazas de seguridad.

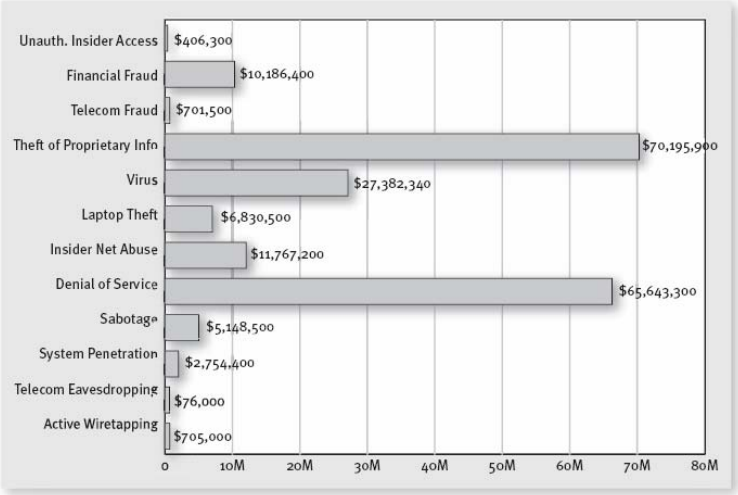

Figura 1.3: Pérdidas económicas, Computer Crime and Security Survey 2003,[24]

En la edición 2004 del mismo reporte, la negación de servicios sigue siendo el segundo caso de pérdidas económicas en las empresas según el FBI. La figura 1.4 muestra que hay cambios considerables con respecto al reporte del año 2003.

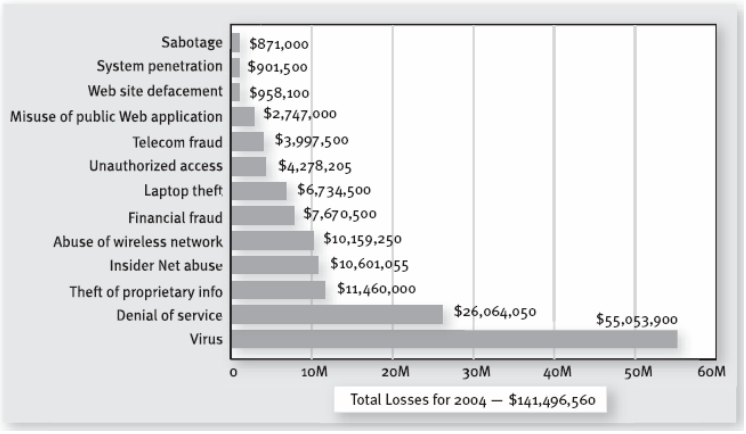

Figura 1.4: Pérdidas económicas, Computer Crime and Security Survey 2004,[24]

Las gráficas muestran una diferencia significativa de las pérdidas causadas por DoS entre los años 2003 y 2004, bajando a menos de la mitad con respecto al año 2003, la documentación no explica el porque, posiblemente debido al aumento en el interés por tratar de mitigar este problema que en ambas gráficas sigue siendo grande.

1.3. Time To Live

El campo TTL (Time To Live, por sus siglas en inglés) es un campo del encabezado IP que sirve para controlar el tiempo de vida de los paquetes IP en la red, para evitar que estos queden dando vueltas indefinidamente. Cada router por el que pasa un paquete IP decrementa en una unidad este campo y, al llegar a cero, el paquete es desechado por el ruteador, evitando ciclos indefinidos.

Mediante el campo TTL podemos identificar el número de saltos que el paquete ha dado desde su origen hasta el final de su destino, lo cual será la base del algoritmo utilizado para realizar el filtrado del tráfico atacante.

1.4. Lógica Difusa

La lógica difusa es un concepto el cual trata de aproximarse a la forma de pensar de las personas y en lo que para una persona representa una idea y lo que representa esa misma idea para otra persona. Es decir, para una persona el término “alto” podría representar una altura de 190 centímetros, pero para otra persona el mismo término “alto” podría ser a partir de 170 centímetros. El término “calor” para una persona podría ser una temperatura de 35^{o} C mientras que para otra persona “calor” significaría una temperatura de 25^{o} C.

La lógica difusa introduce el término de “función de pertenencia” en la cual se definen conjuntos difusos, y con la ayuda de la función de pertenencia podremos identificar en que porcentaje la temperatura de 30^{o} C cae en el conjunto difuso “caliente” y en que porcentaje la misma temperatura cae en el conjunto difuso “frío”.

Con esto tenemos un rango de respuestas más amplio que un simple cierto/falso de la lógica binaria. El cual es representado por porcentajes entre 0 y 100%.

1.5. Ambientes Virtuales

Un ambiente virtual es la simulación de un escenario con ayuda de una herramienta para poder realizar pruebas de algún software o alguna configuración especial de un sistema operativo. Estos ambientes nos sirven entre otras cosas, para poder crear

computadoras virtuales con características similares a las físicas, ahorrándonos el costo de las máquinas físicas, así como de los componentes para su interconexión.

Para probar el funcionamiento de FLF4DoS, se hizo uso de ambientes virtuales con Linux y la herramienta User Mode Linux[11] para crear una red virtual dentro de una máquina física y poder generar las simulaciones de ataques desde una máquina virtual hacia otra, así como de la mitigación de dichos ataques usando FLF4DoS.

1.6. Monitores de Red

Un monitor de red es una herramienta que nos permite checar el estado de la red en cuanto al tráfico que circula por la misma. Puede checar el ancho de banda en general así como de los tipos de paquetes que circulan por la red.

Estos monitores de red nos permiten identificar problemas diversos en la red, como son congestiones de tráfico, interrupciones de servicio, etc. Ethereal [7], es un monitor de red que se usara para obtener las gráficas del tráfico en las simulaciones de envío de paquetes atacantes en la red virtual construida. tcpdump es una herramienta de Linux para monitorear el estado de la red. Esta herramienta la usaremos para obtener los campos del encabezado de los paquete IP que pasan por la red, dirección IP origen y TTL, para el análisis de los mismos por parte de FLF4DoS.

1.7. Problemas en DDoS

Las víctimas de un ataque de vulnerabilidades, generalmente se bloquean, se reinician o algunos de sus servicios quedan inhabilitados. Para realizar estos ataques solo se necesitan de algunos paquetes que pueden ser enviados por únicamente algunos agentes o zombies.

Por el contrario, en los ataques de inundación o flooding, los recursos de la víctima son inhabilitados en la medida en que los paquetes atacantes lleguen a su destino. Para que este tipo de ataque sea efectivo, se necesita un flujo constante de paquetes atacantes hacia la víctima.

Debido a que los ataques de inundación no necesitan de ningún paquete en específico, los atacantes generan una mezcla variada de tráfico que se une al tráfico legítimo de los clientes. Además, los atacantes usan IP Spoofing para crear una gran variedad de direcciones IP de origen a los paquetes y ocultar la identidad y ubicación de los agentes o zombies. Cuando sucede esto, la víctima percibe un repentino incremento en las peticiones de servicio por parte de los clientes e intenta servir o responder a todas estas peticiones, y como resultado, la víctima comienza a tirar algunos paquetes que no puede manejar.

En [17] se mencionan algunas características que hacen más efectivos los ataques y más difíciles las técnicas de defensa:

- Simplicidad: Existen muchas herramientas de ataque de DDoS que pueden ser fácilmente descargadas de la red y ponerlas en acción. Estas herramientas hacen el reclutamiento de agentes y la activación automáticos, y pueden ser usados por usuarios sin experiencia. Además, estas herramientas son excesivamente simples y pueden generar ataques muy efectivos con muy poco esfuerzo.
- Variedad de tráfico: La similaridad del tráfico de ataque al del tráfico legítimo hacen de la separación y el filtrado, tareas difíciles de llevar a cabo. A diferencia de otras amenazas de seguridad que necesitan paquetes específicamente falsificados (intrusiones, gusanos, virus), los ataques de inundación necesitan un alto volumen de tráfico y se puede variar el contenido de los paquetes así como de los valores del encabezado.
- IP Spoofing: Esta técnica de ataque hace posible que el tráfico atacante parezca provenir de numerosos clientes legítimos. IP Spoofing supera a aquellos métodos de defensa basados en la compartición de recursos que identifican a los clientes por medio de su dirección IP. La presencia de esta técnica en un ataque, hace que la víctima vea un número elevado de peticiones de inicio de servicio por un gran número de clientes aparentemente legítimos. Al momento del ataque, la víctima puede saber cuales son las direcciones IP que han ido llegando en el tráfico de ataque, y puede hacer la distinción de ellas con las direcciones de los clientes que actualmente estaba atendiendo, pero no podrá diferenciar ni atender las nuevas peticiones de clientes realmente legítimos, provocando el DoS.
- Alto volumen de tráfico: El alto volumen del tráfico de ataque sobre la víctima, no sólo satura sus recursos, también hace difícil la caracterización del tráfico para poder clasificarlo. En altos índices de tráfico, los mecanismos de defensa sólo pueden realizar procesamiento paquete por paquete. El principal reto de los mecanismos de defensa de DoS es diferenciar el tráfico legítimo del tráfico de ataque a velocidades altas en el envío de los paquetes.
- Numerosas máquinas zombies o agentes: La fortaleza de un ataque DDoS depende en gran medida del número de zombies distribuidos por toda la red Internet. Con un gran número de agentes, un atacante puede lanzar un ataque variando la frecuencia en la que cada zombie envía los paquetes, incluso, puede ser que cada máquina zombie envíe un número pequeño de paquetes atacantes. Variando las estrategias de ataque, diversos mecanismos de defensa de DoS o DDoS que intentan rastrear el origen del ataque no funcionarían adecuadamente. Aún en los casos en los que el traceback sea posible, debido al número de máquinas atacantes se hace poco viable el traceback.

La seriedad del problema de DDoS, el incremento en la frecuencia, la sofisticación y fuerza de los ataques han propiciado la llegada de numerosos mecanismos de defensa.

Mirkovic, et al. [17] define dos tipos de retos a los que se enfrentan los desarrolladores de algún mecanismo de defensa contra los ataques de DDoS: Retos técnicos y retos sociales.

Los retos técnicos involucran problemas asociados con los actuales protocolos de Internet y a las características de los ataques de DDoS. Los retos sociales están relacionados con la manera en que los usuarios adoptan las soluciones propuestas para defenderse de un DDoS.

1.7.1. Retos Técnicos

La naturaleza distribuida de los ataques DDoS, que son similares a los paquetes del tráfico de los clientes legítimos y que usando el IP Spoofing representan el principal reto técnico en el diseño de mecanismos de defensa contra ataques de DDoS.

- Necesidad de una respuesta distribuida en diversos puntos de Internet: Existen muchos tipos de ataques de DDoS y pocos de ellos pueden ser manejados únicamente por la víctima, debido a esto es necesario contar con un sistema de respuestas distribuido y coordinado.
- Falta de información detallada de los ataques: Es ampliamente conocido que reportar las ocurrencias de ataques daña la reputación de las empresas que son víctimas de ataques. Debido a esto existe información limitada sobre algunos ataques y algunos incidentes sólo han sido reportados a organizaciones de gobierno con la condición de mantenerlas en secreto. La información de los ataques debería incluir el tipo de ataque, tiempo y duración, número de zombies involucrados, defensa utilizada y que efectividad tuvo y los daños sufridos por el ataque.
- Falta de un sistema de defensa estandarizado: Actualmente no existe un método estandarizado de pruebas para probar los sistemas de defensa de DDoS que ayudarían a la comparación y caracterización. Debido a esto, los investigadores proponen soluciones que se acercan a los sistemas donde realizan las pruebas de sus propuestas, además de que no pueden comparar sus soluciones con otras soluciones existentes.
- Dificultad en las pruebas a gran escala: Los mecanismos de defensa deberían ser probados en ambientes reales. Esto es muy difícil debido a que no existe una forma segura de probar un ataque real en Internet o herramientas de simulación realistas que puedan soportar varios cientos de nodos.

1.7.2. Retos sociales:

- Buen desempeño: Un producto debe conocer las necesidades de los clientes. Los requerimientos de desempeño no son rígidos, y cualquier producto que mejore el estado actual es aceptable.
- Buen modelo económico: Cada cliente debe obtener beneficios económicos directos o al menos reducir el riesgo de pérdidas económicas al implementar el producto. Además, el cliente puede implementar otras soluciones con el fin de mejorar sus servicios durante un ataque de DDoS.
- Beneficio creciente: A medida que el grado de despliegue de mecanismos de defensa se incremente, los clientes deben experimentar un incremento en los beneficios. Sin embargo, un producto debe ofrecer beneficios suficientes aunque el despliegue de diversos mecanismos de defensa sea escaso.

1.7.3. IP Spoofing

Una de las tácticas usadas en los ataques distribuidos de negación de servicios es el "IP Spoofing" o falsificación de IP. El encabezado del Protocolo de Internet o IP contiene la dirección IP del emisor y la dirección IP del receptor o destinatario. El IP Spoofing ocurre cuando un programa malicioso crea sus propios paquetes y no coloca la dirección IP de origen correcta en el encabezado del paquete. Es fácil falsificar paquetes individuales controlando completamente el encabezado IP y enviarlos por la red si se cuentan con los privilegios suficientes dentro del sistema operativo usado.

Existen diversos niveles de IP Spoofing

- Dirección IP completamente aleatoria: El sistema genera direcciones IP que son tomadas aleatoriamente del espacio completo de direcciones IPv4, 0.0.0.0-255.255.255.255. Esta estrategia algunas veces generará direcciones IP inválidas, como direcciones en el rango 192.168.0.0, que son reservadas para redes privadas además de generar direcciones IP no-ruteables, direcciones de broadcast, o direcciones inválidas como 0.2.43.7. Algunas de estas direcciones IP causan problemas significantes a los routers de la red, provocando que se bloqueen o tengan que ser reiniciados. Sin embargo, la mayoría de estas direcciones IP generadas aleatoriamente serán válidas y ruteables. Si el atacante es cuidadoso, y verifica si una dirección generada es ruteable, podrá enviar grandes cantidades de paquetes falsificados aleatoriamente y esperar que la mayoría de ellos sean entregados correctamente.
- Falsificación de Subred: Si una máquina pertenece, digamos a la red 192.168.1.0/24, es relativamente fácil falsificar paquetes de una máquina perteneciente a la misma red. Por ejemplo, la máquina 192.168.1.2 puede fácilmente falsificar cualquier otra dirección dentro del rango 192.168.1.0/24 a menos que el administrador de la red haya establecido medidas que impidan que las tarjetas de red Ethernet sean asociadas con diferentes direcciones IP, lo cual es una medida difícil y cara desde el punto de vista de la administración de la red.
  Si la máquina pertenece a una red más grande, una red /16 que contiene 65,536 direcciones disponibles, se puede falsificar un número mayor de direcciones pertenecientes a mismo dominio, suponiendo que las medidas preventivas por parte del administrador de la red no han sido establecidas.
  Los paquetes falsificados a nivel de subred, pueden ser filtrados mediante técnicas de filtrado de ingreso/egreso al mismo nivel de subred. Aún usando este tipo de filtrado el IP Spoofing puede ser difícil de detectar a la máquina que está participando en un ataque de inundación.
- Falsificar las direcciones de las víctimas: Este es el escenario típico de un ataque de DoS usando reflectores. El atacante falsifica la dirección origen en paquetes de petición de servicios, por ejemplos paquetes SYN en conexiones TCP, para comenzar con una inundación de respuestas enviadas hacia la víctima. Si no existe un mecanismo de filtrado de paquetes en el router principal de la red, estos paquetes serán parte de un ataque de DoS utilizando reflexión.

Las razones por las que un atacante puede usar IP Spoofing en un ataque se muestran a continuación, sin embargo, algunos ataques no lo utilizan dado que el atacante puede saturar los recursos de la víctima inundándola con un número suficientemente grande de paquetes.

- Ocultar la ubicación de las máquinas agente: En ataques de DoS usando sólo una máquina, el IP Spoofing es usado para ocultar la ubicación de la máquina atacante. En este tipo de ataques, es difícil para los administradores de red bloquear el origen de los ataques y remover la máquina atacante de la red. En ataques distribuidos de DoS, los agentes son la ruta hacia los manejadores, los cuales proporcionan una capa adicional de indirección hacia el atacante. Ocultar a los agentes, significa ocultar la rápida localización del atacante.
- Uso de reflectores: Los ataques distribuidos de DoS usando reflectores requieren IP Spoofing para ser realizados. Los agentes o zombies deben ser capaces de falsificar la dirección de la víctima en peticiones de servicio dirigidas a legítimos servidores, para invocar respuestas que inunden a la víctima.
- Evadir los sistemas de defensa de DDoS: Algunos mecanismos de defensa de DDoS construyen listas de clientes legítimos que accedan a la red. Durante el ataque, a las direcciones IP de los clientes que se encuentren en estas listas, se les da un trato especial. Otros sistemas de defensa intentan compartir recursos equitativamente entre todos los clientes actuales. En ambos escenarios, el IP Spoofing vence a la separación de clientes basada en su dirección IP y le da la posibilidad a los paquetes de ataque de asumir la dirección IP de los clientes.

Capitulo 2

Antecedentes y Trabajos Relacionados

2.1. Como funciona un ataque

Hay dos tipos generales de métodos para negar un servicio[1,17]: explotando una vulnerabilidad presente en una víctima o enviando un gran número de mensajes aparentemente legítimos. El primer tipo de ataque es llamado “ataque de vulnerabilidades” y el segundo “ataque de inundación” (flooding attack).

Los ataques de vulnerabilidad funcionan enviando algunos paquetes especialmente arreglados a la aplicación de la víctima que posee la vulnerabilidad. Esta vulnerabilidad es usualmente un error en el software al momento de la implementación o un error en la configuración de un determinado servicio. Los mensajes maliciosos de un atacante representan una entrada inesperada a la aplicación que el programador no previó. Esos mensajes causan que la aplicación caiga en ciclos infinitos, severamente degradar la velocidad de servicio, congelarse o reiniciar la computadora donde es ejecutada, o consumir una gran cantidad de memoria y negar los servicios a los usuarios legítimos.
En algunos casos, vulnerabilidades de este tipo pueden ser explotadas en el sistema operativo, en algún middleware o en algún protocolo de red, así como también en programas de aplicación. Mientras que es imposible detectar todas las vulnerabilidades, también puede ser muy difícil encontrar formas de explotarlas. Esto significa que cada vulnerabilidad que es detectada y parchada es un gran logro y un paso seguro para seguir con los mecanismos de defensa.

Los ataques de inundación o flooding funcionan enviando un gran número de mensajes cuyo procesamiento consume recursos importantes en la víctima. Una vez que el recurso es bloqueado por el atacante, los usuarios legítimos no pueden recibir el servicio. La característica importante de este ataque es que su fortaleza esta basada en la cantidad en lugar del contenido. Esto tiene dos implicaciones principales:

1. El atacante puede enviar una gran variedad de paquetes. El tráfico de ataque puede ser hecho arbitrariamente en forma similar al tráfico normal, lo cual grandemente dificulta la defensa.

2. El flujo de tráfico debe ser tan grande para consumir los recursos de la víctima. El atacante usualmente tiene que emplear más de una máquina para enviar el tráfico de ataque. Los ataques de inundación son por lo tanto comúnmente ataques de negación de servicios distribuidos, DDoS (Distributed Denial of Service).

La forma más simple de un ataque DDoS es enviar una cantidad muy grande de mensajes, divididos en paquetes a un servicio en una máquina víctima. A menos que algo o alguien entre la máquina atacante y la víctima descarten esos paquetes, la víctima perderá recursos tratando de recibir y manejar correctamente los paquetes. Si hay suficientes paquetes de este tipo, todos los recursos de la máquina serán desperdiciados tratando de manejar esos paquetes que no tienen ningún valor.

Otra forma de ataque de DDoS es atacar a la interfaz de red de la víctima. Si la tarjeta de red de la máquina víctima puede transmitir sólo 10 Mbps de tráfico, el atacante necesita generar solamente 10 o más Mbps de cualquier paquete IP y enviarlos a la víctima. Una vez más, suponiendo que ninguna otra entidad descarte los paquetes antes de que lleguen a la tarjeta de red de la víctima, fácilmente saturarán sus recursos de red y también crearán una congestión grande en la ruta hacia la víctima. Si hay algunos paquetes legítimos además del tráfico de inundación de paquetes de ataque, es poco probable que reciban el servicio.

El atacante puede también inundar la red en la cual se encuentra la víctima, en el caso anterior. Si el atacante sabe que dicha red posee un enlace de 1 Gbps puede generar y enviar suficientes paquetes a la víctima o a otra máquina dentro del segmento de red para saturarla.

Los ataques mencionados anteriormente están basados en grandes volúmenes de tráfico. El atacante algunas veces puede perpetrar un efectivo ataque de inundación con volúmenes más pequeños de tráfico. Si la víctima tiene algún servicio corriendo que requiere de mayor tiempo de procesamiento de peticiones remotas del que toma generar estas peticiones, o que bloquean un escaso recurso en el servidor, el atacante puede hacer uso de esta desproporción. Aún pequeños o poco frecuentes envíos de tráfico malicioso bloquearán efectivamente recursos críticos de la víctima. Un ejemplo común es el ataque de inundación TCP-SYN. El atacante inunda a la víctima con paquetes TCP-SYN, los cuales son usualmente usados para iniciar nuevas conexiones de comunicación. La víctima reserva un poco de memoria en un buffer de tamaño limitado para cada nueva petición de comunicación, mientras que el atacante puede enviar esas peticiones sin ningún desperdicio de memoria. Esta desproporción ayuda al atacante a desactivar cualquier nueva petición de comunicación durante el ataque, enviando pocas peticiones de paquetes TCP-SYN

2.2. Instalando zombies y controlando máquinas atacantes.

Un ataque DDoS requiere el agrupamiento de de múltiples máquinas, las cuales enviarán el tráfico de ataque a la víctima. Estas máquinas que no pertenecen al atacante, son máquinas pobremente aseguradas en universidades, compañías y casas, aún en instituciones de gobierno. El atacante irrumpe en ellas, toma el control completo de las máquinas y las utiliza para el ataque. Estas máquinas comúnmente son llamadas zombies, daemons, slaves o agentes.

Estos zombies son sistemas pobremente asegurados, a los cuales no se les han instalado los últimos parches de seguridad o actualizaciones de software, no están protegidos por firewalls o algún otro sistema de seguridad o sus usuarios tienen contraseñas fácilmente adivinables.

El atacante toma ventaja de vulnerabilidades bien conocidas en ciertos sistemas que no son muy seguros, explota las vulnerabilidades y obtiene control total de la máquina de la que se intenta apoderar, incluso como si tuviera cuenta de usuario administrador.

Una vez que el atacante ha obtenido el control de las máquinas, instala un agente de ataque de DDoS y se asegura que todas las huellas de su intrusión fueron eliminadas u ocultadas y de que el código correrá aún si la máquina es reiniciada.

Los ataques de DDoS frecuentemente involucran cientos o miles de agentes, sería tedioso para el atacante realizar la intrusión además de que implicaría perder mucho tiempo si tuviera que entrar en cada una de estas máquinas, en lugar de esto, existen herramientas automatizadas que descubren máquinas potencialmente vulnerables para poder instalar los agentes y hacerlas parte de un ataque. Además de encontrar máquinas para formar el grupo de ataque, las herramientas mencionadas ofrecen también el control de esta red de máquinas brindando una forma fácil de enviar comandos a todas ellas al mismo tiempo. El atacante únicamente tiene que teclear el comando para que todos los agentes inicien la inundación contra la máquina víctima.

El atacante adicionalmente oculta su identidad desplegando varias capas de indirección entre su máquina y los agentes o zombies[17]. Para realizar esto, utiliza una o más máquinas que entregan los comandos a los agentes. Estas máquinas son llamadas *handlers* o *masters*. Otra capa de indirección consiste en que el atacante acceda a diversas máquinas en secuencia antes de enviar los comandos a los handlers. Estas máquinas son llamadas *stepping stones*.

Ambas formas de indirección son usadas para dificultar los intentos de investigación. Si las autoridades localizan y examinan a uno de los agentes, toda su comunicación apuntará a uno de los handlers. Adicionalmente, si se examina a los handlers, estos apuntarán a unas de las máquinas stepping stone y esta, a su vez, a otra stepping stone. Si éstas últimas se encuentran ubicadas en diferentes países y continentes, será muy difícil seguir el rastro hacia la máquina del atacante y descubrir su identidad.

Otra forma de ocultar el ataque es a través del IP-Spoofing. Cada paquete que es transmitido en Internet lleva información de control precediendo a los datos, en el encabezado IP. Un campo en el encabezado IP especifica la dirección IP de la máquina que envía el paquete, el campo IP source. Esta información es llenada por la máquina que

envía la información al momento de transmitir los paquetes, y este campo es usado por el destinatario o por los ruteadores en la ruta hacia el destinatario, para enviar respuestas al emisor. Los atacantes comúnmente falsifican este campo para lograr una mayor efectividad en el ataque y dificultar el descubrimiento de los agentes. El IP-Spoofing también dificulta enormemente algunos mecanismos de defensa de DDoS que se basan en la dirección de origen para diferenciar entre las máquinas legítimas y las atacantes. Con el IP-Spoofing, el atacante fácilmente puede tomar la identidad de un cliente legítimo o aún más, la de varios de ellos.

El IP-Spoofing crea una gran oportunidad de engañar a máquinas no comprometidas y perfectamente aseguradas para participar en un ataque de DDoS. El atacante selecciona un servicio disponible públicamente o un protocolo, tales como un DNS (Domain Name System), web o ping, y envía peticiones de servicios a muchos de esos servidores, falsificando la dirección IP de la víctima. Los servidores responderán las peticiones a la víctima y así inundarán con respuestas a la víctima produciéndole una negación de servicios. Este tipo de ataque es conocido como ataque de reflección, los servidores participantes son llamados reflectores. Son de especial interés para los atacantes los servicios que puedan generar prolongadas o numerosas respuestas a peticiones cortas. Esto es llamado *amplificación* y le da la posibilidad al atacante de de crear un efecto DoS con muy pocos paquetes.

La negación de servicios es posible sin usar técnicas distribuidas. Pero presenta un reto para el atacante. Por ejemplo, si se quisiera realizar un ataque de DoS basado en inundación, originado desde una máquina con un enlace de 10 Mbps y es dirigido hacia una víctima que posee un enlace de 100 Mbps. En un intento por saturar el enlace de la víctima, el atacante saturará su propia red y se producirá un DoS a sí mismo.

Sin embargo, que pasaría si el mismo ataque se realiza en forma distribuida, por un ciento de máquinas. Cada una de éstas, envía 1 Mbps hacia la víctima. Suponiendo que cada máquina posee un enlace de 10 Mbps, ninguna de ellas genera suficiente tráfico para producir algún daño en su propio enlace, y la red Internet enviará todo el tráfico de ataque hacia la víctima, saturando su enlace y negando los servicios mientras que el atacante esta completamente libre de DoS.
La distribución trae ciertos beneficios para el atacante[17]:

- Un servidor típico posee más poder de cómputo, memoria y recursos de ancho de banda que una típica máquina cliente. Un atacante con el control de una única máquina cliente tendría dificultades para poder saturar los recursos del servidor sin antes saturarse a sí mismo. Usando técnicas distribuidas, el atacante puede multiplicar los recursos en el lado atacante, permitiéndole negar los servicios a máquinas más poderosas en el lado de la víctima.
- Para detener un ataque de DoS de un único agente, el defensor necesita identificar al agente y tomar alguna acción en su contra que le impida seguir enviando grandes volúmenes de tráfico. En muchos casos, el ataque de una simple máquina puede ser detenido mediante alguna acción llevada a cabo por el administrador de la red. Pero si en el ataque intervienen 100 o 1000 agentes, detener el ataque de uno o varios agentes no produciría grandes beneficios para la víctima. Sólo deteniendo a la mayoría de ellos o a todos, el efecto DoS podría ser eliminado.

- Si el atacante selecciona agentes dispersos ampliamente a través de Internet, los intentos por detener el ataque serán mucho más difíciles ya que el único punto en el cual todo el tráfico converge es muy cerca de la víctima. Este punto es llamado *punto de agregación*. Otros nodos de la red, pueden no darse cuenta de las señales de ataque y pueden no distinguir el tráfico de ataque del legítimo. Así, estos nodos no pueden ayudar a defender a la víctima en un ataque.
- En un ataque de DoS ejecutado desde un solo agente, la víctima puede ser capaz de recuperarse obteniendo más recursos. Por ejemplo, un servidor Web puede ser capaz de ayudarse de otros servidores para poder manejar la carga extra. Sin importar el poder del agente, el defensor puede agregar mayor capacidad hasta superar la capacidad del agente para generar peticiones. El método de agregar más capacidades a la víctima es menos efectivo en ataques de DDoS. Si el defensor duplica sus capacidades para atender el doble de peticiones, el atacante necesita duplicar el número de agentes, siendo esta una tarea sumamente sencilla.

2.3. Formas de ataques

Para lanzar un ataque de DoS, un usuario malicioso primero compromete un cierto número de computadoras en Internet explotando alguna deficiencia en la seguridad de las mismas. Subsecuentemente, el usuario malicioso instala herramientas de ataque (conocidas como “zombies”) en las máquinas comprometidas dándoles la capacidad de ser parte de un ataque contra una víctima al recibir un comando.

Teniendo el control de los zombies, el usuario malicioso puede construir un ataque formado por paquetes de varios tipos, con paquetes con checksum’s incorrectos, valores incorrectos en los encabezados o una combinación inválida de las banderas.

Alefiya Hussain, et al en [1] dan una clasifican general a los ataques de DoS: software exploits y ataques de inundación o flooding.

En los ataques de software exploits, el atacante envía pocos paquetes con la finalidad de explotar alguna falla dentro del sistema operativo o aplicación atacados. En los ataques de inundación o flooding, uno o más atacantes envían una cantidad constante de paquetes cuya meta es la de saturar el ancho de banda o los recursos de cómputo de la víctima.

También en [1] dan una clasificación de los ataques en función al análisis de encabezados, al incremento en el tráfico de la red y mediante un análisis espectral.

En [18], Mirkovic da la siguiente clasificación a los ataques:

- Como deniegan el servicio
  - Por medio de una vulnerabilidad
  - Envío de grandes cantidades de paquetes a la víctima

- Vulnerabilidad explotada
  - Sistema Operativo
  - Aplicación
  - Protocolo
- Tipo de tráfico enviado a la víctima
  - Tipo específico de paquetes
  - Mezcla aleatoria de paquetes
- Quien envía los paquetes
  - Máquinas zombies, cuantas.
  - Reflector
  - La misma máquina
- Quien es la víctima
  - Aplicación
  - Máquina
  - Ancho de banda de la red
- Efecto causado
  - Interrumpir
  - Degradar

2.4. Soluciones al problema de DDoS

En [3], Chang describe 3 líneas de defensa contra un ataque de DoS.
1. Prevención del ataque (antes del ataque).
2. Detección del ataque y filtrado (durante el ataque).
3. Identificación y trazado a la fuente de ataque (durante y después del ataque).

Prevención del ataque

La primera línea de defensa es la prevención del ataque. Los hosts deben estar seguramente protegidos de implantes de zombies. Otra medida es monitorear el tráfico de la red para conocer los mensajes de ataque enviados entre el atacante y los agentes. Una medida preventiva también es el uso de cyber-informantes o cyber-espías para interceptar planes de ataque.

Detección del ataque y filtrado

Este método consiste en dos fases: la detección del ataque DDoS y filtrado de los paquetes de ataque. La parte de detección es responsable de identificar ataques de DDoS o paquetes de ataque. Después de identificar flujos de paquetes de ataque o paquetes atacantes, la parte de filtrado debe clasificarlos y después descartarlos.

Identificación y rastreo (traceback) de la fuente de ataque

Esta es una medida de respuesta ante un ataque de DDoS. El rastreo se refiere al problema de identificar la fuente de ataque de cualquier paquete enviado a través de la red sin confiar en la información que contiene el paquete. Existen generalmente dos

métodos para el problema de trazado de direcciones IP. El primero es que los routers registren la información de los paquetes que van pasando por ellos y otro es que los routers envíen información adicional acerca de los paquetes que pasan a la dirección de destino del paquete.

Este método no es útil para detener un ataque de DoS, únicamente se utiliza para conocer el origen del ataque y para un filtrado posterior de las direcciones involucradas en el ataque.

2.5. Temas relacionados

Han sido muchas las investigaciones para intentar solucionar pequeños problemas relacionados con el DoS y DDoS. Debido a las características del tráfico de una red de datos y a la complejidad del DoS, es difícil desarrollar un método eficaz para resolver completamente este problema.

Muchos prototipos se han probado en laboratorios sin tomar en cuenta un tráfico real. Algunos suponen que el tráfico es mayormente falsificado, otros suponen un cierto conocimiento en la topología de la red. Algunos otros requieren cambios significativos en la infraestructura de Internet lo que la haría incompatible con los protocolos existentes y con las aplicaciones cliente-servidor que existen actualmente.

Algunos trabajos relacionados con el tema son:

2.5.1.Pushback

Este método es propuesto por Mahajan, et al. en [14], la idea principal es que el operador de la red intente eliminar el tráfico de ataque hacia la víctima ya sea, desconectando completamente el cable de red de la víctima desde el router y observar si el tráfico de ataque se detiene además de monitorear el tráfico en equipos de monitoreo. El índice de velocidad limita la propagación hacia fuera de la víctima y así aliviando la presión sobre ésta, permitiéndole intercambiar tráfico y efectivamente sobrevivir por unos momentos mientras las fuentes de ataque son detenidas o removidas. Este método supone que el tráfico atacante no es uniformemente distribuido a través de todos los posibles puntos de ingreso.

Aquí hay dos técnicas importantes: el Control de Congestión Agregado local (ACC de Aggregate Congestion Control) y Pushback. El Control de Congestión Agregado local, detecta la congestión al nivel del router y genera una “firma de ataque” o “firma de congestión”, la cual puede ser traducida en una regla de filtrado de paquetes. Esta firma define un subconjunto de tráfico y el ACC local determina una taza apropiada de transferencia para ese subconjunto.

Pushback, la otra técnica, propaga este límite para los elementos que participan en el Control de Congestion Agregado para filtrar el tráfico en base a la firma de congestión.

Este mecanismo funciona mejor contra ataques de DoS de inundación que compartan ciertas características e intenten combatir este problema desde el punto de vista de control de congestión.

2.5.2.Traceback

Las primeras propuestas de los mecanismos de defensa de ataques de DoS y DoS distribuidos incluyen medios para rastrear a los agentes de la red de ataque de DoS. La suposición está basada en que algunas herramientas de ataque falsificaban las direcciones IP de origen de los ataques así como el uso de relativamente pocos agentes, entre 100 y 2,500, suponiendo además, de que era improbable que redes mas grandes de 100,000 computadoras de ataques existirían en un futuro.

Algunas propuestas usaban técnicas llamadas Probabilistic Packet Marking (PPM). Para una cantidad de 20,000 paquetes dirigidos a un destino final, un router marcaría un paquete con una referencia a si mismo. Analizando diversos paquetes marcados de una fuente de ataque dada, la víctima del ataque podría intentar construir una ruta hacia el atacante o al menos a un punto cercano a la víctima o al router de marcado más cercano al atacante. La propuesta inicial de Savage, et al. [15] no contaba con ningún medio de autentificación de dichas marcas, pero en trabajos posteriores se agregó junto con medios de chequeo de integridad.

La técnica Hash-based traceback [16] requiere que los routers participantes recuerden cada uno de los paquetes que ven, pero sólo por un tiempo limitado. Esto permite rastrear cada uno de los paquetes, pero sólo si el rastreo es llevado a cabo en forma rápida.

De acuerdo con [17] PPM y otros esquemas de traceback hacen las siguientes suposiciones:

- Los atacantes son capaces de enviar cualquier tipo de paquete.
- Múltiples atacantes pueden actuar juntos.
- Los atacantes están concientes de las técnicas de traceback.
- Los atacantes pueden enviar miles de paquetes.
- Los routers entre las máquinas son estables, pero los paquetes pueden ser reordenados o perdidos.
- Los routers no pueden realizar muchos procesos por cada paquete.
- Los routers no están comprometidos por los atacantes, pero no todos los routers tienen que participar.

Existen dos problemas serios para las soluciones de traceback[17]:

1. Estas soluciones frecuentemente son inaceptablemente complejas y caras cuando en el ataque participa un gran número de fuentes o cuando las fuentes están bien distribuidas.
2. Los métodos de traceback por si mismos, no hacen nada por detener el ataque. Algunos únicamente pueden caracterizar un ataque lo suficiente para poder distinguir entre los paquetes de ataque y los paquetes legítimos.

Estos mecanismos de rastreo de rutas no pueden actuar en contra de un ataque, únicamente nos sirven para detectar el origen de los ataques y actuar contra ellos una vez que el ataque ha sido realizado.

2.5.3.D-WARD

Desarrollado por Mirkovic [18], es un sistema que ayuda a detectar ataques. Es un sistema en línea, transparente a los usuarios de la red, que recolecta tráfico de entrada y salida en el router principal en la red atacada, para calcular algunas estadísticas y compararlas con modelos de tráfico construidos sobre ciertas especificaciones de protocolos de transporte y aplicaciones, reflejando ciertas características y comportamientos en modelos de tráfico de ataque que pueden ser: normal, sospecha de ataque y ataque. Los experimentos realizados con D-WARD mostraron que tienen la capacidad de detectar rápidamente aquellos ataques de inundación. Además, puede controlar el tráfico normal y el tráfico de ataque de forma efectiva con un bajo daño colateral y niveles bajos de falsos positivos.

2.5.4.NetBouncer

Este método propuesto por Thomas, et al.[19], financiado por el programa FTN (Fault Tolerant Network) de DARPA (Defense Advanced Research Projects Agency), es un mecanismo basado en la legitimidad de los clientes, ubicada en redes potencialmente atacadas. Idealmente es colocado en el punto de entrada del flujo de paquetes o tráfico hacia la red y tiene como finalidad el permitir paquetes únicamente de los clientes o usuarios legítimos. Se realizan diversas pruebas para comprobar la legitimidad de los clientes, por ejemplo mediante pings para ver si existe un cliente actual detrás del paquete que acaba de ser recibido.

Una vez que el cliente ha probado ser efectivamente legítimo, es agregado al conjunto de clientes legítimos y le es dado un trato preferencial sobre los clientes no legítimos o que aún no han sido autentificados.

NetBouncer supone que los clientes puedan contestar los pings, lo cual no todos los clientes soportan, como aquellos que se encuentran detrás de firewalls.

Este método proporciona un buen servicio a los clientes legítimos, y no requiere cooperación de otros sistemas NetBouncer colocados en distintas redes. Por otro lado, los atacantes pueden falsificar los paquetes usando las identidades de los clientes legítimos y realizar un ataque sin que NetBouncer detecte dicho ataque.

2.5.5.Pi

Este método de defensa basado en la víctima, desarrollado por Yaar, et al.[20], construido sobre técnicas previas de marcado de paquetes, inserta identificadores de rutas en campos no usados del encabezado de los paquetes IP. La idea principal es que estos identificadores o huellas sean insertados por los routers a lo largo de la ruta de red. La

víctima rechazaría los paquetes cuyo identificador o huella corresponda con la de aquellos paquetes que han sido claramente identificados como parte de un ataque.

El Filtrado mediante Pi puede llevarse a cabo una vez que el esquema de marcado haya sido instalado en la infraestructura. Este esquema supone que la víctima sabe identificar el tráfico de ataque del tráfico legítimo. Los filtros entonces pueden descartar todo el tráfico con la marca o el identificador dado. Algunos paquetes legítimos podrían ser descartados debido a que comparten la misma ruta que los que pertenecen al tráfico de ataque.

2.6. Hop-Counter Filtering (HCF)

Este método es propuesto en [6] y se basa principalmente en el análisis del campo TTL (Time To Live) del encabezado IP de los paquetes que llegan a alguna interfaz de red en un ataque de DoS.

El método tiene dos estados de ejecución, “acción” y “alerta”, que se usan para inspeccionar la dirección IP de cada paquete. En condiciones normales, el método se mantiene en un estado de alerta, en el cual se encuentra inspeccionando y verificando alguna variación importante en el campo TTL de los paquetes, y se mantiene actualizando la tabla llamada IP2HC (IP to Hop-Counter), sin descartar ningún paquete. El estado de “acción” entrara cuando en el estado de “alerta” se detecta un ataque, y en ese momento se comenzarán a descartar los paquetes que pertenecen al ataque que se está registrando en base a la tabla IP2HC.

De acuerdo con los autores de este método, se puede detectar hasta el 90% de los paquetes falsificados, además de reducir significativamente el porcentaje de falsos positivos así como también un daño colateral muy pequeño al descartar los paquetes falsos.

2.6.1. Detalles del funcionamiento del método

Debido a que la información del Hop-Count no esta directamente almacenada en algún campo del encabezado de los paquetes, se tiene que calcular en base al campo TTL, que es un campo de 8 bits en el que cada router, por el que pasan los paquetes, decrementa en una unidad ese campo antes de enviarlo al siguiente router.

Así, el TTL que llega con el paquete es conocido como TTL final. El problema aquí, es el cálculo del Hop-Count, ya que como se mencionó anteriormente, la víctima sólo ve el TTL final, otro problema es que los diversos sistemas operativos no usan el mismo valor para el TTL inicial.

No se puede asumir un mismo TTL inicial para el cálculo del Hop-Count para todos los paquetes IP, ya que las direcciones pueden cambiar con el paso del tiempo. Sin embargo, de acuerdo con [8] la mayoría de los sistemas operativos modernos solo utilizan alguno de los siguientes valores para el TTL inicial: 30, 32, 60, 64, 128 y 255. Este conjunto de TTL iniciales cubre la mayoría de los sistemas operativos populares como

son Microsoft Windows, Linux, variantes de BSD, y varias versiones comerciales de Unix.

Para el cálculo del Hop-Count se realiza lo siguiente: se extrae el TTL del paquete, y pasa a ser el TTL final. Se calcula el TTL inicial en base al conjunto previo, por ejemplo, si se lee un TTL final de 112, el valor del TTL inicial será el entero inmediato superior del conjunto de valores iniciales anterior. Para el ejemplo el TTL inicial es 128, por último se calcula el Hop-Count restando el TTL final del TTL inicial, en el ejemplo, el Hop-Count es 16.

También en el estudio realizado en [8], mencionan que existen algunos sistemas operativos que usan ciertos valores iniciales raros para el TTL inicial de los paquetes IP, pero se tratan de sistemas operativos viejos y se espera que ese grupo de sistemas operativos sea un grupo realmente pequeño.

El algoritmo usado para identificar paquetes falsificados extrae el TTL final de cada uno de los paquetes que llegan así como de la dirección IP de origen, posteriormente se infiere el valor del TTL inicial y se calcula el Hop-Count restando el TTL final del TTL inicial. La dirección IP origen sirve como índice en la tabla para poder extraer el Hop-Count de cada dirección. Si el Hop-Count calculado corresponde a uno previamente calculado, el paquete es autentificado, de otra forma, el paquete es probablemente falsificado y se descartará. Cuando una dirección IP legítima tiene el mismo Hop-Count que un zombie, el método HCF no es capaz de identificar los paquetes falsificados, pero de acuerdo a [6], aún en estos casos, el método es altamente efectivo en la identificación de los paquetes falsificados.

De acuerdo a un estudio de ruteo punto a punto, realizado en [9], las rutas en Internet prevalecieron con pocos cambios, cerca de dos terceras partes de las rutas estudiadas tuvieron rutas persistentes por días e incluso semanas. Otro estudio en [10] realizado durante un periodo de 5 meses y observando un total de 10,814 rutas diferentes, mostró que la mayoría de estas rutas experimentaron muy pocos cambios en el cálculo del Hop-Count: 95% de las rutas tuvieron menos de 5 cambios notables por día.

Debido a que HCF no puede reconocer paquetes falsificados cuya dirección IP tenga el mismo valor de Hop-Count que el de los zombies, es crítica una distribución de los Hop-Counts para un filtrado efectivo. Es necesario examinar la distribución de los Hop-Counts en varias ubicaciones en Internet para asegurarse que el Hop-Count no esta concentrado en un solo valor. Si el 90% de las direcciones IP de los clientes están mas allá de 10 saltos del servidor, o la víctima, no se podrá distinguir un gran número de paquetes falsificados de los legítimos usando HCF.

Un atacante puede aprender la forma en que funciona HCF y tratar de generar paquetes falsificados que puedan evadir las inspecciones del Hop-Count y evitar el HCF. Sin embargo, esos intentos requerirán una gran cantidad de recursos o tiempo y una planeación muy elaborada. Es muy improbable que los atacantes casuales sean capaces de evadir HCF.

La clave para que un atacante pueda evadir el HCF esta en la habilidad que tenga para generar paquetes IP con campos TTL adecuados. Esto lo podría hacer cada uno de

los zombies enviando peticiones legítimas a la víctima y observando los valores del campo TTL de los paquetes que la víctima responda, o simplemente utilizando un "traceroute". Así, el zombie podría generar el tráfico aleatoriamente y colocando el valor del TTL adecuado. Pero de acuerdo a [4] y [5], muchas herramientas existentes no alteran el campo TTL. Si el zombie realiza peticiones legítimas o utiliza traceroute para ver el número de saltos hacia la víctima, podría dejar rastro y se podría bloquear su dirección IP una vez realizado un análisis de los paquetes recibidos por la víctima.

En un ataque DoS distribuido utilizando reflectores en los que se genere aleatoriamente la dirección IP del reflector, será prácticamente imposible adivinar o encontrar un valor TTL adecuado para poder evitar el HCF, debido a que en los reflectores no se encuentran comprometidos por parte del atacante, únicamente son usados para esconder el origen del ataque y ellos simplemente enviarán paquetes de respuesta a la víctima con su dirección IP y con el valor en el campo TTL de acuerdo al sistema operativo usado.

2.6.2.Efectividad de HCF

La evaluación y las mediciones hechas en [6], por parte de los autores de HCF, hace las siguientes suposiciones:

- La víctima conoce todas las direcciones IP de sus clientes, así como de los respectivos Hop-Counts.
- Que el atacante falsifica las direcciones IP de origen tomando todo el espacio posible de direcciones IP, 2^{32}.
- El atacante escoge los Hop-Counts o TTL inicial de los paquetes generados desde el zombie.
- También suponen que el atacante dividirá el tráfico uniformemente entre las fuentes del ataque de inundación.

Ataques simples

Para evaluar el desempeño de HCF, se evalúan dos escenarios posibles: ataques con un origen y ataques con múltiples orígenes.

En un ataque de un solo origen, los paquetes IP son generados aleatoriamente por un zombie. Debido a esto, todos los paquetes tendrán el mismo Hop-Count ya que son enviados por la misma fuente.

La figura 2.1 muestra la distribución de los paquetes en base al Hop-Count, en la cual se aprecia claramente el aumento en el Hop-Count de los paquetes que pertenecen al ataque de inundación.

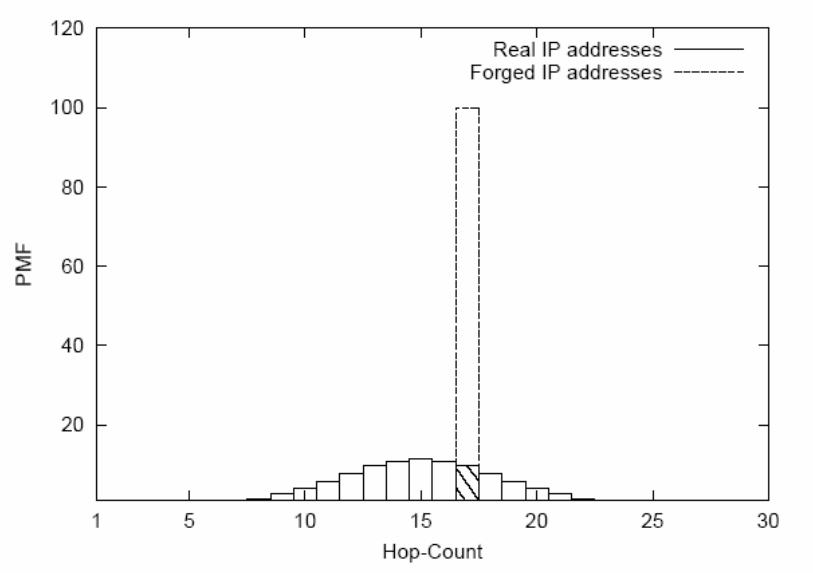

Figura 2.1: Distribución del Hop-Count de las direcciones IP con una simple fuente de inundación

Ataques de múltiples fuentes

En los ataques de múltiples fuentes, ataques de DoS distribuidos, normalmente involucran a más de un zombie, para esto es necesario analizar el caso en el que participen múltiples fuentes de ataque, suponiendo que cada fuente genera una cantidad de tráfico igual a las demás fuentes, es decir cada fuente generará un total de tráfico dividido entre *n* número de zombies. En la figura 2.2 se muestra un análisis realizado en base al Hop-Count en un ataque realizado desde dos fuentes.

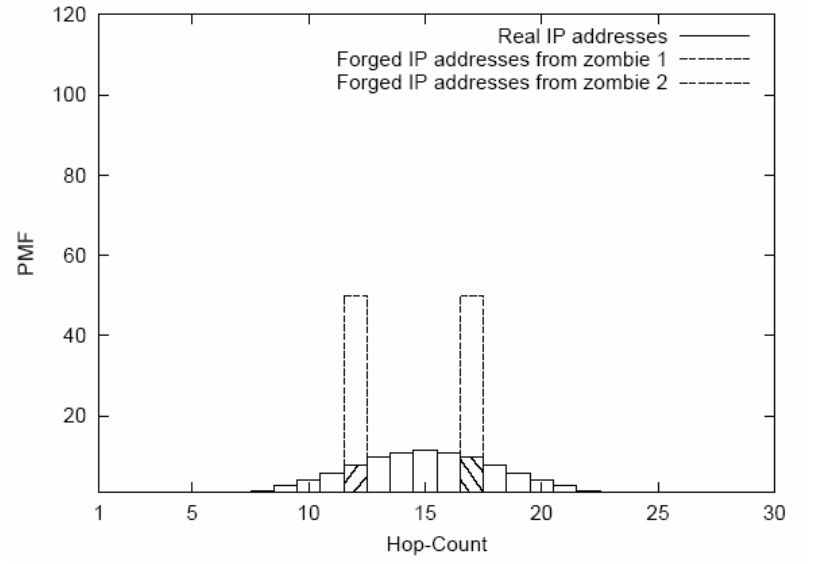

Figura 2.2: Distribución del Hop-Count de las direcciones IP con dos fuentes de inundación

De acuerdo con [6], agregar más fuentes de inundación al ataque no debilita la capacidad de HCF de identificar los paquetes IP falsificados, sino por el contrario, debido a que la distribución del Hop-Count sigue una distribución Gausiana, la existencia de fuentes de ataque menos efectivas habilita al HCF a identificar y descartar mas paquetes IP falsificados que en el caso de ataques con una sola fuente de inundación.

Ataques sofisticados

La mayoría de los atacantes eventualmente reconocerán que no es suficiente únicamente falsificar la dirección IP origen. En lugar de usar el mismo valor inicial para el campo TTL, el atacante fácilmente puede generar aleatoriamente esos valores para cada uno de los paquetes. A pesar de que el Hop-Count de los paquetes de una simple fuente a la víctima son estables, aleatorizando el valor inicial del campo TTL creará una ilusión de que los paquetes tienen distintos Hop-Counts a la víctima. En los paquetes donde se falsifique uniformemente los valores para el campo TTL de cada uno de los paquetes, se esconden pocos paquetes al HCF.

Para los valores del campo TTL generados uniformemente, cada dirección IP origen falsificada tiene la probabilidad de 1/H de tener el mismo valor en el campo TTL, donde H es el número de Hop-Counts posibles.

De acuerdo a los análisis presentados en [6], ninguno de los ataques "sofisticados", en los cuales se falsifica el campo TTL, es mucho más efectivo que los ataques de una o múltiples fuentes descritos previamente.

2.6.3.Construcción de la tabla

Es altamente improbable que un servidor en Internet reciba peticiones legítimas de todas las direcciones activas en Internet, además de que todo el espacio de direcciones disponibles no es completamente utilizado, por ejemplo las direcciones clase D.

Para la construcción de la tabla IP2HP en [6] se asume que el atacante conoce las direcciones IP de los clientes de cada víctima y generará paquetes IP seleccionando aleatoriamente entre las direcciones IP de las víctimas. Además, para generar el mejor escenario para el atacante y quizás el peor para el HCF, se asume que el atacante conoce la distribución general del Hop-Count y la usa para generar un ataque más efectivo.

Los falsos positivos se definen como aquellas direcciones IP de clientes legítimos que incorrectamente son identificadas como direcciones IP falsas. Los falsos negativos se definen como aquellas direcciones IP falsas que HCF no puede ser capaz de detectar.

En [6] evalúan el rendimiento del método usando 3 tipos de filtrado:

- Filtrado estricto

- Filtrado + 1
- Filtrado + 2

Estos tipos de filtrado funcionan de forma similar, la diferencia radica en que en el filtrado estricto, aquellos paquetes que no correspondan al Hop-Count almacenado en la tabla serán descartados. Los del filtrado + 1 y + 2, descartarán los paquetes en los cuales su Hop-Count sea más grande en 1 o 2 unidades de aquellos almacenados en la tabla IP2HC.

Para realizar las pruebas, en [6] inicializaron la tabla mediante la recolección de las direcciones IP de los clientes de un servidor así como de su correspondiente TTL. El periodo durante el cual recolectaron estos datos fue proporcional a la cantidad de tráfico que recibe un servidor. Para un servidor con una carga de tráfico pesada un periodo de recolección de unos días sería suficiente, mientras que para servidores con carga ligera, el periodo recomendable es de un par de semanas, para que se puedan recolectar las direcciones IP de todos los clientes.

El Hop-Count o la distancia de un cliente a un servidor pueden cambiar como resultado de una reubicación de la red, inestabilidad en los ruteadores o sistema de ruteo, o fallas temporales en la red.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

DIVISIÓN DE TECNOLOGÍAS DE INFORMACIÓN Y ELECTRÓNICA
PROGRAMA DE GRADUADOS EN TECNOLOGÍAS DE INFORMACIÓN Y ELECTRÓNICA

Los miembros del comité de tesis recomendamos que la presente tesis del Ing. Julio César Covarrubias Rodríguez sea aceptada como requisito parcial para obtener el grado académica de Maestro en Ciencias en Tecnología Informática

Comité de Tesis:

_____
M. en C. Alejandro Parra Briones
Asesor de tesis

_____
Ph.D. Jorge Carlos Mex Perera
Sinodal

_____
Ph.D. Juan Arturo Nolazco Flores
Sinodal

_____
Ph.D. David A. Garza Salazar
Director del Programa de Graduados en
Tecnologías de Información y Electrónica

Diciembre 2005

Capítulo 3

Propuesta y modificación del HCF

En el presente trabajo, tomando como base el HCF de [6], se reprodujo el algoritmo para calcular el Hop-Count en una simulación de un ataque en una red virtual utilizando la herramienta User Mode Linux [11] (UML) de Linux y se realizó la simulación de una red en condiciones de tráfico normal, bajo ataques de DoS con una única fuente de inundación y también con una única fuente de inundación generando aleatoriamente las direcciones IP de los paquetes enviados a la víctima. Utilizando las herramientas de generación de tráfico [12] y [13] se pudieron realizar estas simulaciones.

User Mode Linux es una herramienta segura que nos permite crear ambientes virtuales, nos permite crear máquinas virtuales que pueden tener mas recursos virtuales de hardware y software que la máquina física donde se ejecute UML. El almacenamiento en disco de la máquina virtual esta contenido completamente en un simple archivo dentro de la máquina física. Se le puede asignar a la máquina virtual los permisos de acceso que se deseen o se necesiten y con el acceso debidamente asignado, nada que se haga en la máquina virtual podrá cambiar o dañar a la máquina física o a su software. También cuenta con los medios necesarios para poder comunicarse en red con la máquina física o con otras máquinas virtuales, más aún, con la red física a la cual se encuentre conectada la máquina donde se ejecuta UML. En el anexo A se dan los detalles para la construcción de los ambientes virtuales desde la compilación de un Kernel UML hasta la configuración de en red con otras máquinas virtuales UML.

El algoritmo de [6] se implemento de la misma forma en el presente trabajo, con la diferencia de que la inicialización de la tabla se realizará al momento de que el programa entre en funcionamiento y se actualizará dinámicamente en base a los paquetes que sean recibidos por parte de la víctima aplicando reglas de lógica difusa en el filtrado.

A diferencia de la simulación y análisis realizados en el método original, aquí se reprodujo de manera virtual, utilizando UML [11] y los generadores de tráfico D-ITG [12] y Syn Flooder [13] para reproducir escenarios de tráfico normal y tráfico de inundación entre las máquinas virtuales y la máquina física, que es en donde se instaló la herramienta desarrollada.

En la tabla 3.1 se muestra las características de cada método visto de acuerdo a las líneas de defensa de Chang [3], si son independientes es decir, orientados a host o de forma distribuida en colaboración con sistemas del mismo tipo instalados en hosts o routers cercanos y una característica principal es la de combatir el IP Spoofing para identificar correctamente el tráfico atacante con niveles de falsos positivos bajos.

	Independiente	Tipo de defensa	Identifica IP Spoofing
Traceback	No	Rastreo	No
Pushback	No	Reactivo	No
D-Ward	Si	Reactivo	Si, previo análisis estadístico
Netbouncer	No	Reactivo	Si, usando pings
Pi	No	Rastreo	No
HCF	Si	Reactivo	Si, previa autentificación de clientes mediante TTL
FLF4DoS	Si	Reactivo	Si, autentificación mediante TTL

Tabla 3.1. Características principales de los mecanismos de defensa de DDoS

La principal mejora de FLF4DoS es que disminuye enormemente el tiempo de respuesta sobre HCF, éste último requiere un análisis previo de direcciones IP de clientes que puede ir de una semana hasta un mes, dependiendo la carga de la red, FLF4DoS identifica el tráfico actual en la red, y si observa incrementos muy grandes en el trafico de la red, aplica reglas de lógica difusa para impedir el paso de esos paquetes. Esto no requiere de semanas de análisis de tráfico, únicamente parametrizar FLF4DoS con tiempos y porcentajes en los que se aplicarán las reglas de lógica difusa.

El procedimiento que se siguió para realizar las pruebas de las modificaciones al algoritmo HCF es el siguiente:

- Definir el escenario de ataque.
- Crear la red virtual.
- Generar los diferentes tipos de tráfico, normal y de ataque.
- Extraer los encabezados de los paquetes IP entrantes a la interfaz de red.
- Analizar los encabezados de los paquetes en base al campo TTL.
- Mediante reglas de lógica difusa, realizar el filtrado de los paquetes usando el Kernel de Linux.

3.1.Definiendo el escenario de ataque

De acuerdo con Dietrich[4] y Dittrich[5] la mayoría de las herramientas de ataque por inundación o flooding, no alteran el valor inicial del campo TTL de los paquetes, así, el valor del campo TTL final de un paquete falsificado dará como resultado el valor Hop

Count, que es en el que se basa el método de filtrado, entre el origen del ataque y la víctima.

Algunas herramientas para análisis de tráfico como Syn Flooder de Zakath [13] a la que se tuvo acceso al código fuente, se le puede modificar el código para generar valores TTL aleatoriamente.

Los tipos de ataque que puede mitigar hasta en un 90% el HCF [6] son aquellos en los que se hace uso del Spoofing de direcciones IP ya sea ataques distribuidos de una sola fuente o de múltiples fuentes así como también aquellos que utilicen reflectores.

De acuerdo con [6], los ataques haciendo uso de la generación aleatoria de valores TTL son menos efectivos que aquellos donde simplemente se falsifica la dirección IP de origen.

El tipo de ataque que se pretende mitigar con el presente trabajo es el DoS Distribuido(DDoS) generando aleatoriamente las direcciones IPs de los paquetes, para darle al ataque un efecto de uso de reflectores, el cual consiste en el uso de máquinas que simplemente contestaran los mensajes hacia la víctima.

A diferencia del HCF original de [6], se pretende realizar la modificación en la construcción de la tabla de HCF para que se vaya creando al momento de que se registra un ataque y verificar las direcciones IP que se generan aleatoriamente así como de su campo TTL en cada paquete recibido.

Además, se aplicarán algunas reglas de lógica difusa en el control del filtrado de los paquetes que son candidatos a descartarse en base al incremento del tráfico en cada uno de estos basado en el campo TTL con el que vallan llegando a la máquina atacada.

3.2.Creando la red virtual con User Mode Linux

Para probar FLF4DoS, se utilizó la herramienta UML [11] de Linux, con la cual se configuró una pequeña red que consiste en dos máquinas virtuales y una máquina física sobre la cual se crearon dichas máquinas virtuales.

En la máquina física se instaló FLF4DoS, la cual filtrará el tráfico que sea identificada como tráfico de ataque. Una de las máquinas virtuales, será el atacante, la cual generará tráfico con D-ITG[12] y Syn Flooder[13] hacía la segunda máquina virtual, esta última sólo será receptora del tráfico y tendrá instalada una parte de la herramienta D-ITG para la recepción del tráfico proveniente de la máquina atacante.

Además, en la máquina física se utilizará la herramienta de monitoreo de redes y analizadora de protocolos Ethereal[7], la cual estará monitoreando la interfaz tap1, que es el medio por el cual se comunicará la máquina física con la máquina virtual que será la víctima de los ataques.

La figura 3.1 muestra la configuración de la red empleada para la prueba de FLF4DoS, el ruteo de una máquina virtual a otra se realiza mediante el kernel de Linux de la máquina física, así como el filtrado del tráfico que se identifique como atacante.

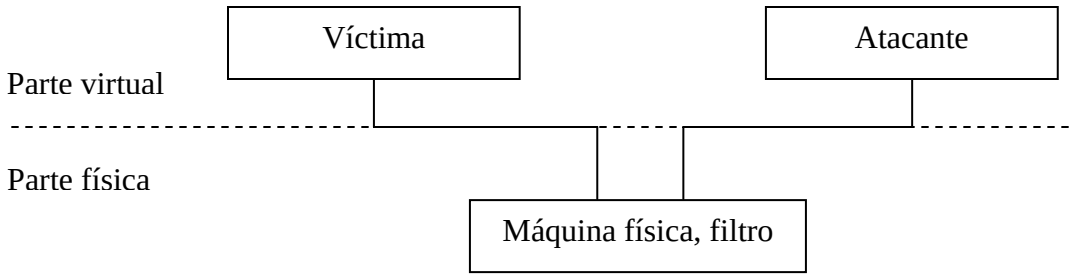

Figura 3.1: Configuración de red usada en la prueba de los ataques usando UML

En el anexo A se muestran los detalles de cómo se levantan las máquinas virtuales, desde la aplicación del parche a la máquina física hasta la configuración en red de cada una de éstas máquinas con el host o máquina física.

3.3.Generando el tráfico de una máquina virtual a otra

Una vez que la red virtual ha sido configurada, y que existe comunicación entre ellas, ver anexo A, el tráfico que podemos generar desde una máquina a otra va desde un simple ping hasta el uso de herramientas como ssh, telnet o para el caso de esta prueba, se usaron las herramientas D-ITG[12] y Syn Flooder [13], con las cuales se envió tráfico normal y tráfico de ataque inundando a la víctima.

D-ITG

Este conjunto de herramienta nos permite realizar una simulación del envío de tráfico de una máquina a otra, consta de los siguientes programas: DITGSend, DITGRec, DITGLog, DITGDec, DITGPlot y DITGApi. Los dos primeros programas fueron los usados en las pruebas de FLF4DoS. A continuación se describen las herramientas usadas dentro del experimento realizado.

DITGSend

Esta herramienta es la parte emisora de la plataforma DITG. Puede enviar un solo flujo de tráfico o muchos y de diferentes tipos usando un archivo de texto especificando línea en cada línea las características de cada flujo deseado.

Nos permite especificar la dirección IP del destinatario, puerto de origen y destino, valor del campo TTL, la duración del envío del tráfico, el tiempo entre los paquetes enviados, registrar en bitácoras el tráfico enviado y recibido, tiempos de salida entre paquetes con distribuciones constante, uniformemente distribuida, exponencial, pareto, cauchy, normal, poisson y gamma; tamaños de los paquetes con las mismas distribuciones, entre otras. Además, simula trafico con características de protocolos en la capa de aplicación: DNS, Telnet y VoIP.

DITGRec

Esta herramienta es la parte receptora del tráfico enviado por DITGSend. Simultáneamente puede recibir flujos de tráfico de diferentes emisores, en diferentes puertos, y con las características especificadas por cada DITGSend usado. Simplemente se debe correr el comando sin ningún parámetro, ya que estará monitoreando el tráfico proveniente de la herramienta anterior.

Una de las formas usadas en el comando ITGSend para enviar tráfico es la siguiente:

$ ./ITGSend –a 10.0.0.2 –sp 9400 –rp 9500 –C 100 –c 500 –t 2000

El resultado de la ejecución de este commando es el envío de paquetes UDP (default) a la dirección 10.0.0.2 enviado por el puerto 9400 y recibido en el puerto 9500 a una velocidad constante de 100 paquetes por segundo y de un tamaño de 500 bytes. La duración del envío de paquetes es de 20 segundos.

Syn Flooder

Es una herramienta para generar tráfico de sincronización al momento de establecer conexiones con el protocolo TCP. Los ataques SYN-FLOOD aprovechan la desproporción de tiempos mencionada en [17] en la cual la máquina atacada reserva memoria un periodo corto de tiempo hasta que se establece la conexión. Esta herramienta simplemente genera paquetes SYN hacia la víctima. La forma de usar esta herramienta es la siguiente:

./synk4 ip_origen ip_destino puerto_inf puerto_sup

- ip_origen especifica la dirección IP que se colocará en el campo Source IP del encabezado del paquete. Si la dirección IP es sustituida por un 0, se usará una dirección IP aleatoria en cada paquete.
- ip_destino especifica la dirección IP de la víctima. No es necesario tener una parte recetora en la víctima para simular el envío de tráfico, ya que los paquetes realmente se envían al destinatario.
- puerto_inf especifica el límite inferior de los puertos que usará Synk4 para el envío de paquetes.
- puerto_sup especifica el límite superior de los puertos que usará Synk4 para el envío de paquetes.

3.4.Extrayendo los encabezados de los paquetes

Para poder extraer los encabezados de los paquetes, se utilizó la herramienta tcpdump de Linux, que es una herramienta que permite monitorear el tráfico de la red en tiempo real.

Algunos usos comunes de tcpdump son:

- Debugeo de programas que utilicen comunicaciones de red.

- Debugeo de la configuración de red de la máquina, determinar si el ruteo está operando o no apropiadamente permitiendo al usuario verificar con certeza puntos de fallo.
- Interceptar y desplegar comunicaciones entre otros usuarios o máquinas. Algunos protocolos como el telnet y http envían su información en texto plano y es fácil de capturar.

Un usuario en control de un gateway o router puede utilizar tcpdump para visualizar cuentas de usuarios, contraseñas, direcciones de Internet y contenidos de sitios que están siendo vistos al momento de ejecutar tcpdump.

FLF4DoS se desarrolló en C, recibe en la línea de comandos ciertos parámetros los cuales nos permiten controlar la aplicación en cuanto al tiempo que estará monitoreando la red, la interfaz que se monitoreará y el nombre del archivo donde se almacenarán los encabezados de los paquetes.

FLF4DoS, ejecuta tcpdump usando la siguiente sintaxis para obtener los encabezados de cada uno de los paquetes que se reciben:

$ tcpdump –i tap0 –qtvn dst host host_ip > archivo &

Las opciones usadas dentro del comando tcpdump se describen a continuación:

- -i tap0: Con esta opción especificamos la interfaz donde se estará monitoreando el tráfico donde se esté corriendo FLF4DoS; tap0 es el nombre de dicha interfaz, la cual es la interfaz virtual con la que se comunica el host con la máquina virtual.
- -q: Le indica a tcpdump que imprima menos información acerca del encabezado de los paquetes que recibe.
- -t: Le indica a tcpdump que no muestre información acerca de la hora en la cual se van recibiendo los paquetes.
- -v: Con esta opción, tcpdump muestra información más detalla, específicamente el campo TTL, el cual es la base del funcionamiento del HCF.
- -n: Con esta opción evitamos que tcpdump convierta las direcciones IP a nombres de dominio.
- dst host host_ip: Aquí se le indica a tcpdump que sólo nos muestre el tráfico de entrada hacia la máquina que está siendo atacada y en la cual se ejecuta FLF4DoS.
- > archivo &: Con la opción > archivo, tcpdump envía la salida de la ejecución del comando hacia el archivo especificado en “archivo”. El último parámetro de tcpdump, &, hace que el comando se ejecute en background, para que el flujo de control de FLF4DoS pueda continuar sin bloquearse hasta que el comando tcpdump termine su ejecución.

Existe la opción –w dentro de tcpdump la cual envía la salida a un archivo, por razones mostradas en el anexo B, se prefirió escribir el archivo con la opción antes mencionada.

La figura 3.2, muestra el contenido del archivo de texto donde se guarda la salida de la ejecución del comando tcpdump.

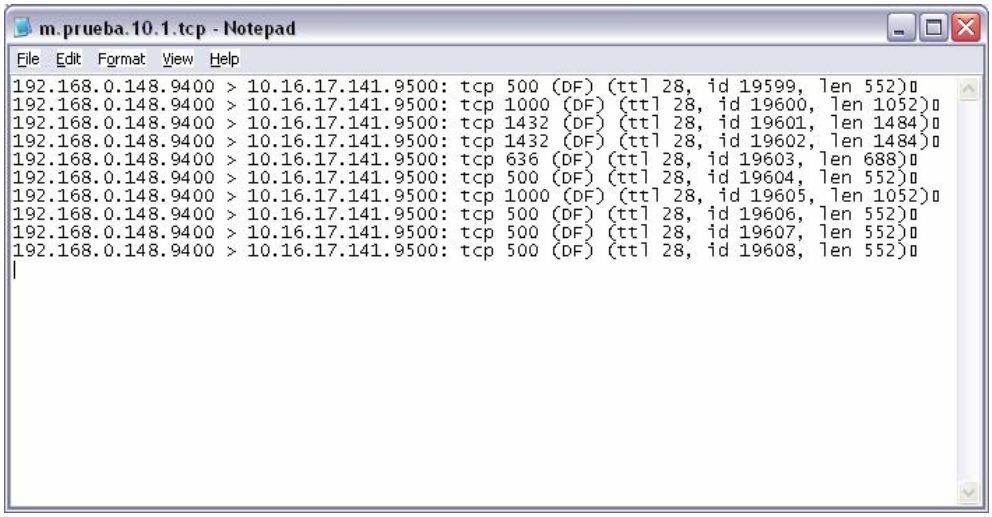

Figura 3.2: Salida de la ejecución del comando tcpdump por FLF4DoS

3.5.Analizando los encabezados de los paquetes en base al campo TTL

Para realizar el análisis de los paquetes que han entrado a la interfaz de red de la máquina atacada, los cuales previamente han sido extraídos con la herramienta de Linux tcpdump, se utilizan árboles y listas ligadas para la ordenación de las direcciones IP de cada uno de los paquetes que se reciben y almacenan en el archivo de texto.

El programa comienza leyendo el archivo que contiene los encabezados de los paquetes que previamente ha almacenado tcpdump, en el que en cada línea del archivo se encuentra la información del encabezado de cada uno de los paquetes recibidos. Posteriormente el programa separa las direcciones IP con su respectivo campo TTL, con la cual se procede a la construcción del árbol y la lista.

El árbol almacena las direcciones IP sin repetirlas además del campo TTL, la lista almacena la dirección IP de cada paquete además de su correspondiente Hop-Count, que es el cálculo de restar el valor del campo TTL final del valor TTL inicial inferido de acuerdo a [8], así como también se almacena la frecuencia .

La dirección IP y el Hop-Count servirán como índices para poder analizar las diferentes direcciones IP con sus respectivos Hop-Counts y poder aplicar las reglas de

ruteo para poder descartar los paquetes que se identifiquen como parte de un ataque de DoS.

FLF4DoS no agrupa las direcciones IP como lo hace el método original, con esto tenemos todas las direcciones que podrían ser utilizadas dentro del ataque, haciendo posible otra forma de mitigar un ataque de DoS o DDoS de acuerdo a la dispersión de las direcciones IP participantes en el ataque, en la sección de trabajos futuro se menciona más a detalle esta posible forma de mitigación.

3.6.Filtrando paquetes atacantes

Una vez que se han insertado las direcciones IP, con sus respectivos Hop-Counts y frecuencias por cada ocurrencia, se procede al filtrado de los paquetes que se han identificado como falsificados para que no sigan pasando hacia la víctima.
Este filtrado se realiza en dos etapas:

- Usar lógica difusa para el filtrado de los paquetes.
- Usando el comando de Linux iptables, aplicar las reglas de ruteo correspondientes.

La lógica difusa usa el concepto de conjuntos difusos y pertenencia, lo que significa que los datos son vistos como pertenecientes a un conjunto o no, y lo más importante, el grado de pertenencia o que tanto pertenece a cada conjunto[25].

En lugar de responder exclusivamente con valores de SI/NO, se puede expresar como: pertenece 65% o 32% a cierto conjunto. En esencia, la lógica difusa nos permite tener un rango de respuestas mas variables entre SI y NO, expresado en porcentajes de pertenencia a cada conjunto. Esto nos permite tener mucho más control sobre las variaciones en los datos de entrada, pudiendo ver exactamente en donde caen, en que conjunto, los datos de entrada de alguna aplicación.

La figura 3.3 muestra los niveles en los cuales se trata de un tráfico normal, un tráfico que incrementa y pertenece a una alerta de DoS y el momento en el que se puede convertir o sospechar de un ataque de DoS, en el cual se procederá al filtrado de los paquetes atacantes.

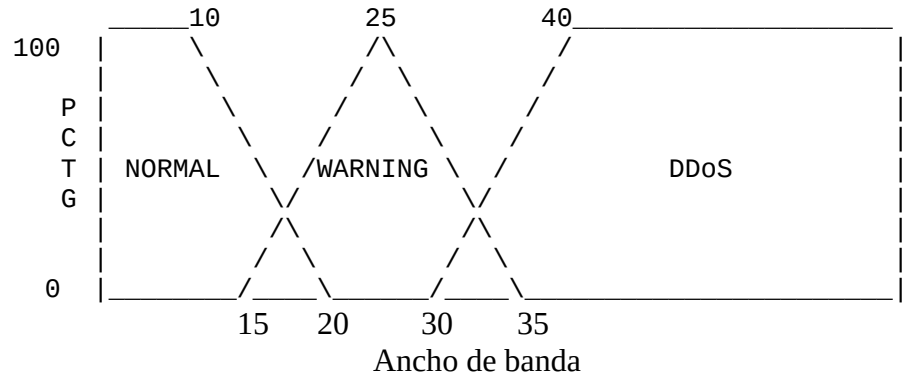

Figura 3.3: Niveles en los cuales se considera un tráfico normal o un DoS por IP

iptables es un sistema de firewall vinculado al kernel de linux que se ha extendido enormemente a partir del kernel 2.4 de este sistema operativo. iptables no es como un servidor que se tenga que iniciar o detener, o que se pueda caer por un error de programación. iptables esta integrado con el kernel, es parte del sistema operativo. Lo que se hace es aplicar reglas. Para ellos se ejecuta el comando iptables, con el que se añaden, borran, o crean reglas de filtrado. Por ello un firewall de iptables no es sino un simple script de shell en el que se van ejecutando las reglas de firewall.

FLF4DoS ejecuta iptables usando la siguiente sintaxis para filtrar cada uno de los paquetes que se reciben cuando correspondan a una expresión dada, en este caso al valor del campo TTL que se le especifique o a la dirección IP de origen espeficifada:

$ iptables -A FORWARD –i tap0 –s dirección_ip -j DROP

Las opciones usadas dentro del comando iptables se describen a continuación:

- -A FORWARD: agrega una regla de ruteo que será aplicada por el kernel de Linux, a una de las tres tablas existentes, INPUT, FORWARD, OUTPUT. La primera descarta los paquetes que son enviados a la misma máquina que aplique las reglas de ruteo, la segunda, FORWARD, es utilizada para el reenvío de paquetes a través de esa máquina a una tercera máquina y la última opción, es para filtrar paquetes que son originados desde la máquina que aplica el filtrado. La opción usada, FORWARD, se utiliza para indicar que las reglas de ruteo se aplicarán a los paquetes que intentan pasar por esa máquina hacía la víctima, descartando los paquetes cuando coincidan con la dirección indicada en la opción –s.
- -i tap0: Aquí le especificamos la interfaz por donde FLF4DoS estará monitoreando el tráfico, es decir el nombre de la interfaz por donde ser recibirán los paquetes.
- -s dirección_ip: Con esta opción (source) especificamos la dirección IP que queremos bloquear especificándola con dirección_ip. Más adelante se muestra como se aplica el comando iptables para bloquear las direcciones que correspondan a un valor TTL dado.
- -j DROP: Aquí se establece que es lo que hay que hacer con cada uno de los paquetes que corresponden con la expresión dada. Las posibles opciones son: ACCEPT, REJECT, DROP, REDIRECT, LOG (existen más, pero estas son las básicas). ACCEPT aceptará el paquete, REJECT o DROP lo desecharán, la diferencia entre ellos reside en que DROP descartará el paquete silenciosamente y REJECT emitirá un paquete ICMP Port Unreachable. REDIRECT redirigirá el paquete a donde se indique en el criterio del comando y por último LOG registrará el paquete en bitácoras para su posterior análisis.

$ iptables -A FORWARD –i tap0 –s dirección_ip –ttl – –ttl-eq ttl_calculado –j DROP

- -ttl: Carga el módulo de iptables que analiza el campo TTL del encabezado de los paquetes IP.
- --ttl-eq ttl_calculado: Especifica con ttl_calculado los paquetes que serán descartados cuando coincidan con el valor del campo TTL especificado. Puede usarse con la

opción –s para un filtrado más preciso o sin ella, descartando todas las direcciones que correspondan al valor especificado.

FLF4DoS:
MITIGACIÓN DE DDOS MEDIANTE UNA TÉCNICA DE FILTRADO BASADA EN EL CAMPO TTL USANDO AMBIENTES VIRTUALES EN LINUX Y REGLAS DE LÓGICA DIFUSA

POR:

JULIO CÉSAR COVARRUBIAS RODRÍGUEZ

TESIS

Presentada al programa de graduados de la división de tecnologías de información y electrónica

Este trabajo es requisito parcial para obtener el grado de Maestro en Ciencias en Tecnología Informática

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

DICIEMBRE 2005

Capítulo 4

Pruebas y resultados

En la red virtual mostrada en la figura 3.1, se realizaron pruebas simulando diversos tipos de tráfico enviado desde una máquina virtual (atacante) a otra (víctima), pasando por un filtro en donde se instaló FLF4DoS para el presente trabajo.

En la figura 4.1, se muestra el envío de tráfico TCP usando la herramienta Syn Flooder [13] junto con comandos ping de la máquina del atacante hacía la máquina víctima, en estas pruebas, no se empleó FLF4DoS. En la figura 4.2 se utiliza la herramienta D-ITG[12] para generar tráfico con características de un servidor DNS.

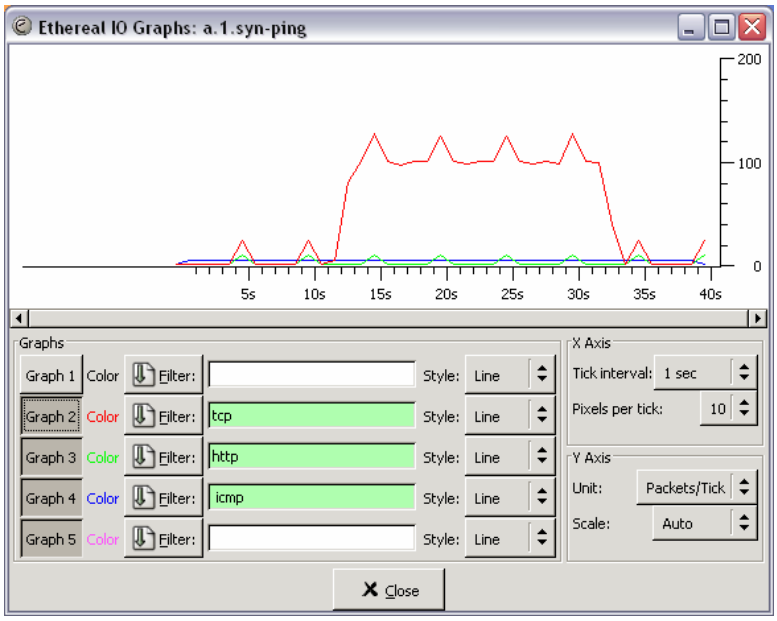

Figura 4.1: Tráfico TCP, HTTP e ICMP

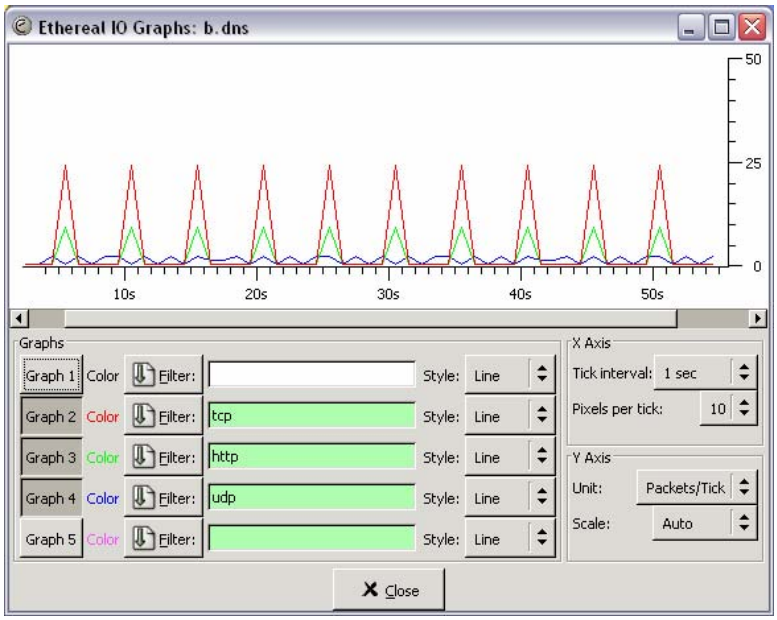

Figura 4.2: Trafico DNS: TCP, HTTP y UDP

En la gráfica 4.3 se muestra un tráfico con características del protocolo Telnet

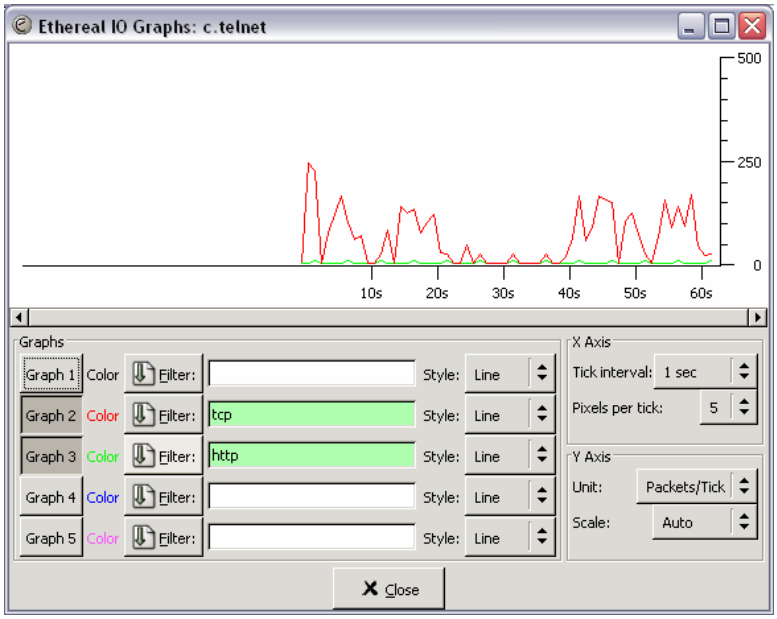

Figura 4.3: Tráfico simulando las características del protocolo Telnet

En la figura 4.4 se muestra el tráfico generado con la combinación de tráfico Telnet y DNS hacia la máquina víctima.

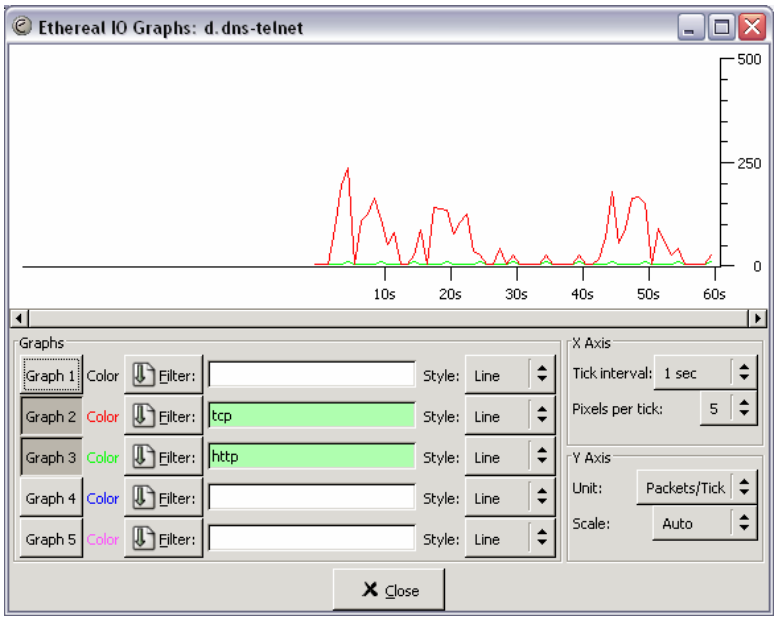

Figura 4.4: Tráficos con características Telnet y DNS

La figura 4.5 muestra la combinación de tráfico Telnet, DNS y el uso de la herramienta Syn Flooder sin IP Spoofing en la dirección IP de origen, en esta se aprecia el incremento del número de paquetes en determinados tiempos.

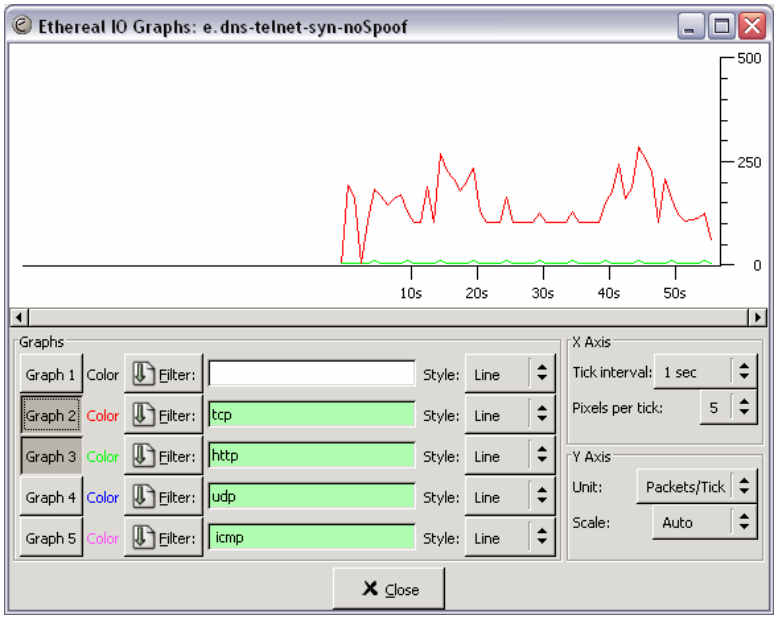

Figura 4.5: Tres tipos de tráfico: Telnet, DNS y la herramienta Syn Flooder

La figura 4.6 muestra el envío del tráfico anterior, con la diferencia de que se uso el IP Spoofing con la herramienta Syn Flooder, el comportamiento del tráfico es notablemente diferente al anterior.

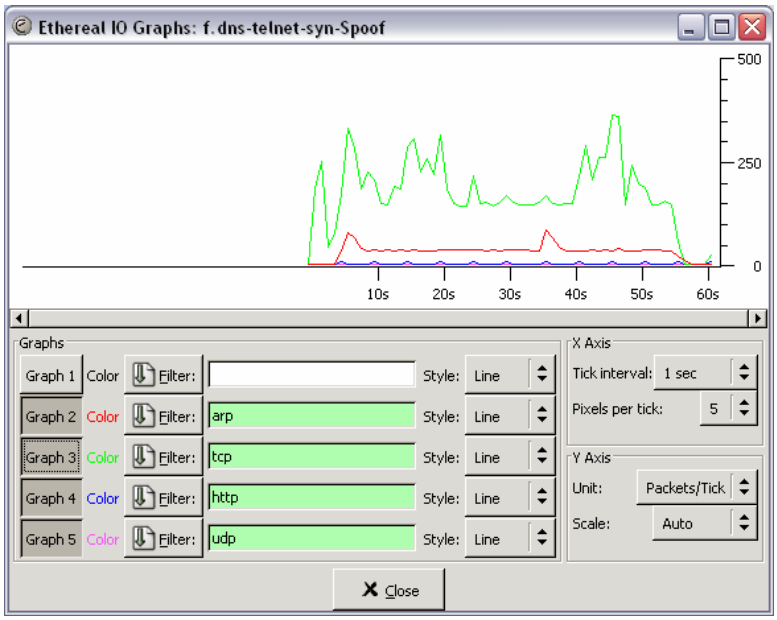

Figura 4.6: Tres tipos de tráfico: Telnet, DNS y Syn Flooder usando IP Spoofing

En las figuras siguientes, se realiza en envío de tráfico con diferentes características, pero haciendo uso de FLF4DoS. En la figura 4.7, usando las herramientas Syn Flooder y D-ITG, se envía tráfico hacía la víctima pasando por la máquina donde se ejecuta el filtro. El tráfico disminuye en el segundo 25 cuando FLF4DoS comienza a aplicar las reglas de ruteo, lo mismo hace en el segundo 55.

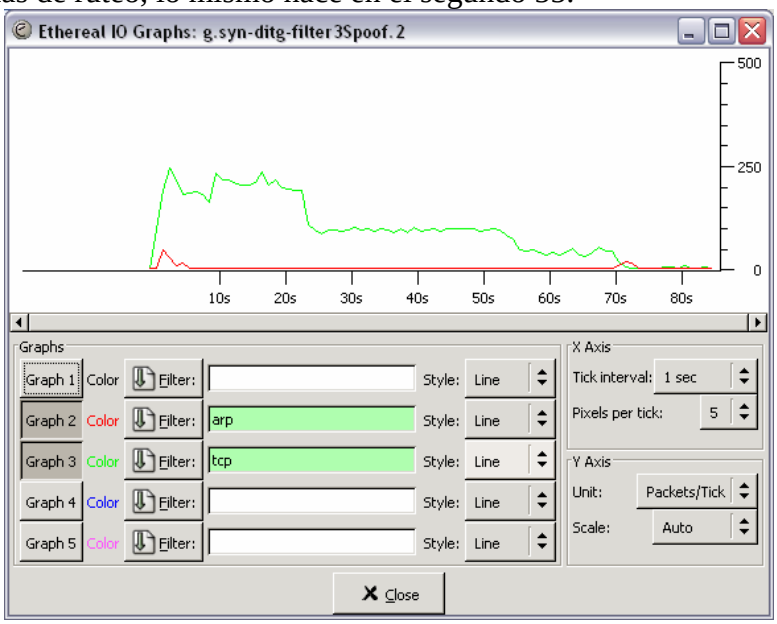

Figura 4.7: Uso de FLF4DoS con tráficos Syn Flooder y D-ITG

La figura 4.8 muestra otra ejecución de FLF4DoS entre tráficos enviados por Syn Flooder y D-ITG. En los tiempos 50 y 55 se nota el decremento del tráfico como consecuencia de la aplicación de las reglas de ruteo.

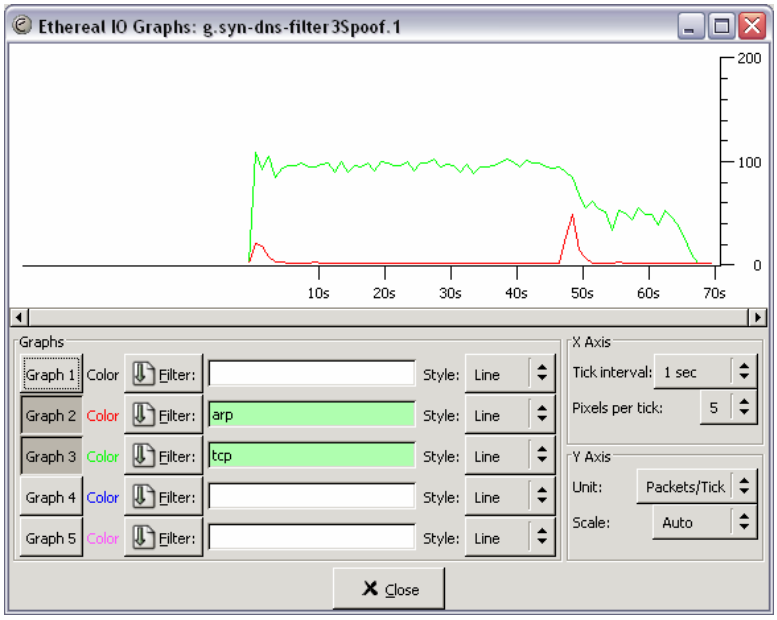

Figura 4.8: Uso de FLF4DoS con tráficos Syn Flooder y D-ITG

Para comprobar los resultados de FLF4DoS, se realizaron pruebas haciendo uso del comando ping, para mostrar que no se producen daños colaterales muy grandes. La figura 4.9 muestra únicamente la ejecución de FLF4DoS entre el tráfico generado por Syn Flooder falsificando la dirección IP de origen.

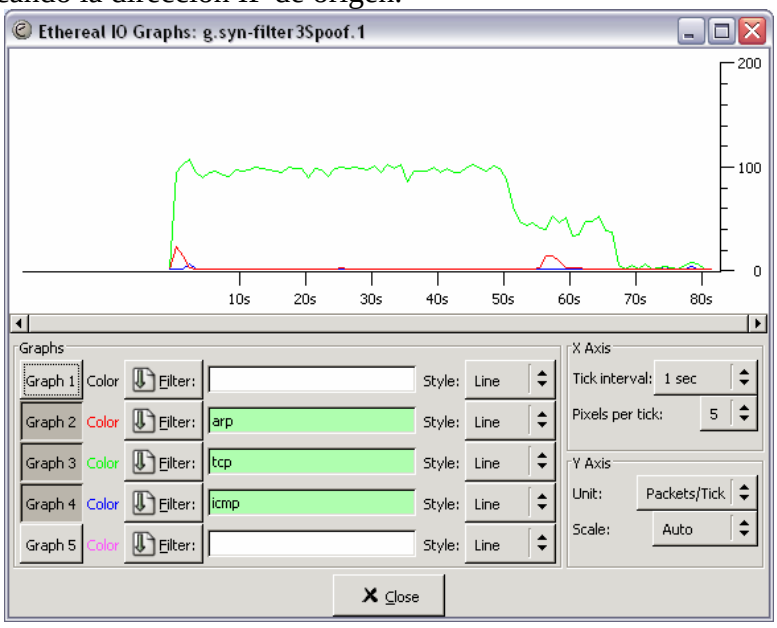

Figura 4.9: Tráfico Syn Flooder mitigado con la FLF4DoS

La figura 4.10 muestra el tráfico generado con Syn Flooder y el comando ping, el cual no se ve afectado cuando se mitiga el DoS, mostrando con esto que el daño colateral es prácticamente nulo.

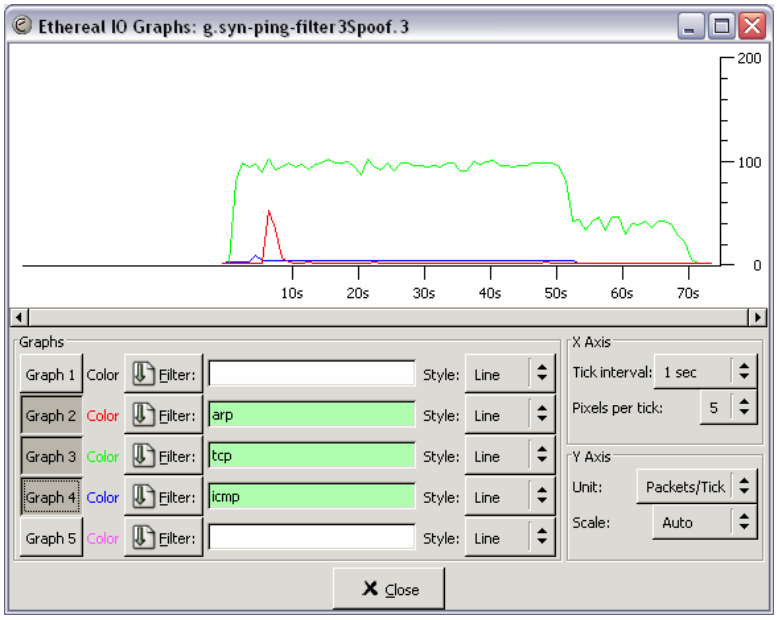

Figura 4.10: Mitigación de DoS sin producir daño colateral en tráfico legítimo

Se realizaron simulaciones de diferentes tipos con diversos patrones de tráfico y con dos máquinas atacando a una víctima. El experimento no se pudo escalar a más máquinas virtuales debido a las limitaciones de espacio de disco duro y memoria de la máquina donde se realizaron los experimentos, sin embargo, se pudieron realizar pruebas desde la red física escolar, desde varias máquinas físicas ubicadas en diferentes edificios donde se corrió el programa Syn Flooder y lanzando el ataque contra la máquina virtual dentro de un host físico. Los resultados fueron muy similares, pero la identificación del ataque fue más rápida debido a que para la simulación se tomó un rango de direcciones aleatorias desde 10.0.0.1 hasta 10.0.0.50, esto debido a que desde varios puntos de la red llegaban paquetes con el mismo rango de direcciones, esto reducía grandemente el tiempo de identificación de un ataque y consecuentemente el filtrado de estos paquetes.

Una prueba muy interesante que se realizó fue la simulación del ataque en medio de una red con tráfico de VoIP. En una red real, la carga de tráfico con VoIP es muy variable, esto es de acuerdo a la empresa y a las necesidades que la misma tenga.

Para la realización del experimento, se usó DITG, el cual nos da la opción de simular tráfico de éste tipo, aunque con pocas opciones de configuración.

Las opciones de VoIP de DITG son algunos codecs (compresores) de voz y el tipo de protocolo, Real Time Protocol y Real Time Protocol con compresión de encabezados. Al realizar las primeras pruebas, la dirección IP de la máquina que simulaba el envío del tráfico de VoIP era bloqueada rápidamente, lo que impedía la transmisión del mensaje. Estos paquetes enviados por DITG son mediante el protocolo UDP, y llegaban en grandes

cantidades a la víctima, lo que hacía actuar a FLF4DoS y bloqueara la dirección IP del emisor.

Una solución aplicada a la simulación, fue la de excluir a las direcciones IP de la máquina que emitía el flujo de VoIP. Esto básicamente dio como resultado una transmisión completa y sin errores del tráfico de VoIP. La figura 4.11 muestra la simulación del experimento con tres máquinas atacantes, una enviando tráfico de VoIP mediante paquetes UDP, otra enviando el ataque usando Syn Flooder y una tercer máquina virtual recibiendo todo este tráfico. En el host físico, se corrió FLF4DoS, el cual arrojó los resultados mostrados en las figuras 4.11, 4.12, 4.13 y 4.14.

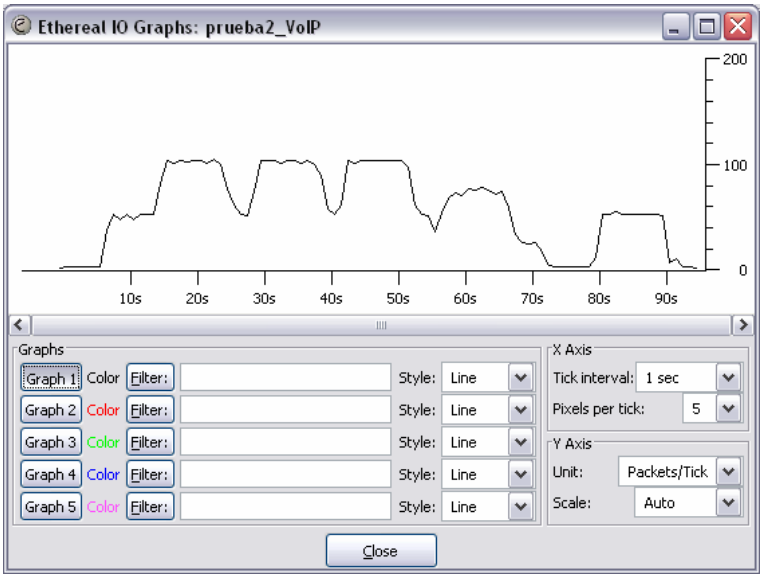

Figura 4.11: Tráfico generado con Syn Flooder y VoIP con DITG

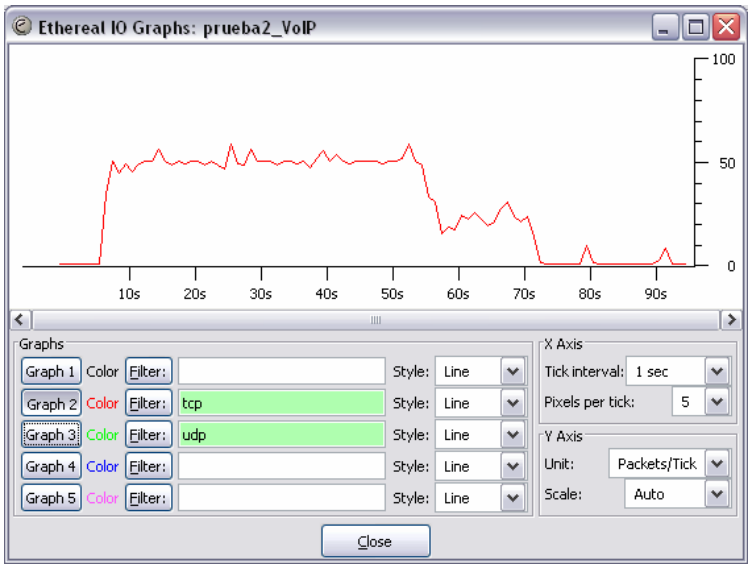

Figura 4.12: Tráfico de ataque filtrado al segundo 55 aproximadamente

En la figura 4.13, observamos un tráfico de VoIP, debido a las opciones con las que cuenta DITG para el envío de éste tipo de tráfico, se ejecutó en varias ocasiones la misma línea de comandos para que se enviara el tráfico de VoIP mientras se identificaba el ataque de Syn Flooder.

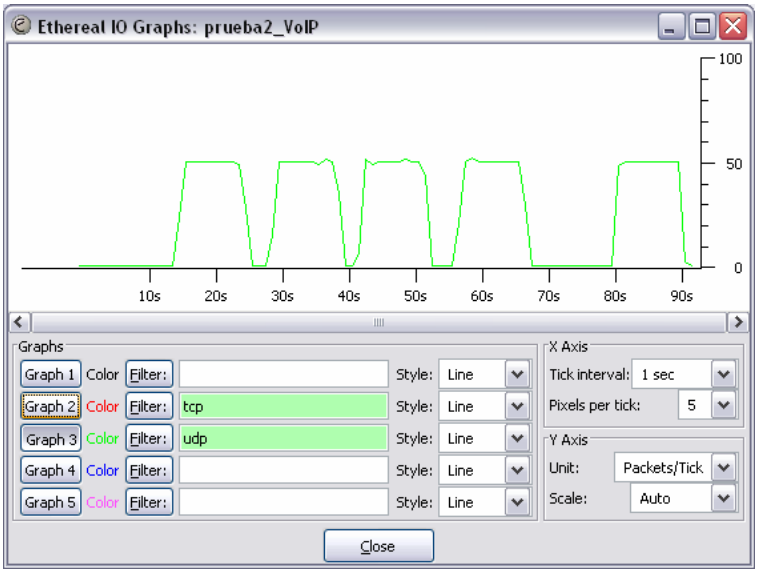

Figura 4.13: Tráfico de VoIP usando DITG mediante paquetes UDP

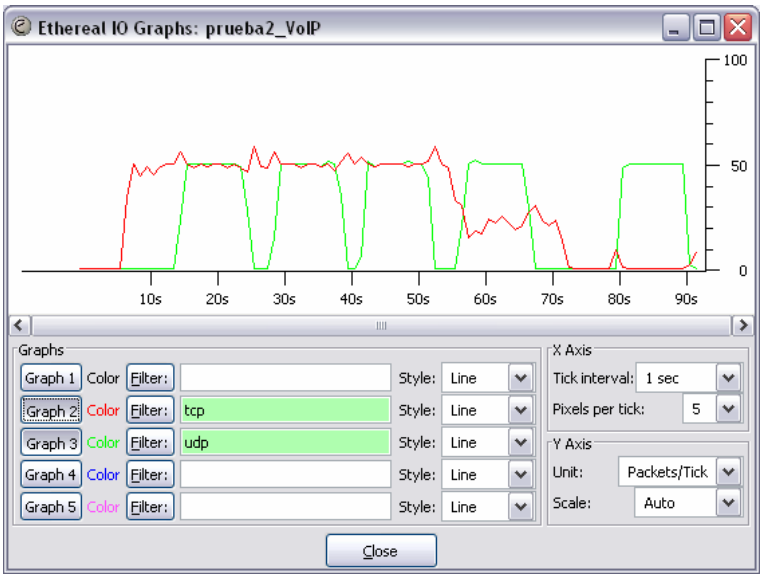

Figura 4.14: Tráficos de ataque y VoIP juntos, aquí se aprecia que el tráfico de VoIP no es afectado.

Dedicatoria

A la familia Covarrubias Rodríguez

Anexo A

Construyendo y Configurando UML

Contenido

1. Distribución usada
2. Requerimientos previos del kernel
3. Compilación del kernel UML
4. Booteo de un kernel UML
5. Configuración de la máquina virtual UML
6. Prueba de comunicación con el host kernel
7. Prueba de comunicación con la red física

Distribución usada.

La distribución usada en la prueba y configuración de la red UML es Red Hat 9.0 con una versión del kernel 2.4.20-8 la cual soporta la mayoría de las características necesarias para las pruebas con UML.
El file system usado en la prueba es de la distribución de Red Hat 9.0, pero puede ser sustituido por alguno de los muchos que existen en la red, que van desde los 14 hasta los 600 MB o más.

Requerimientos previos del kernel

Antes de comenzar a configurar UML, se necesita bajar el parche UML para el kernel; las utilerías de UML, que son utilizadas para configurar y administrar la red, así como también bajar el file system imagen que va a ser utilizado como file system de root.

Hay numerosos parches UML para una gran variedad de kernels, en este caso, vamos a utilizar el parche user_mode_linux-2.4.19.5um-0.i386.rpm de [11] para linux 2.4.24-8. Es recomendable usar la última versión estable de estos parches para el host kernel. El kernel UML está completamente en el userspace, de esta forma podemos construirlo totalmente en el directorio /home, que es mejor que colocarlo en el /usr/src. Sería recomendable crear un subdirectorio en el directorio home llamado “uml”, por ejemplo, y almacenar el kernel, el file system de root y todas las utilerías en ese subdirectorio.

Aplicamos el parche anterior al host kernel de la siguiente manera:

```
# rpm -ivh user_mode_linux-2.4.19.5um-0.i386.rpm
```

Una vez aplicado el parche, podremos continuar con la construcción de nuestro kernel UML.

El file system usado en esta prueba es root_fs.rh-9-full.pristine.20030724.bz2 el cual se puede obtener de [11].

Compilación del kernel UML

Para realizar la compilación de un kernel UML, necesitamos tener los códigos fuente de Linux de alguna versión en especial, así como del parche UML correspondiente a esa versión. Para revisar las versiones de parches disponibles checar en [11] y los códigos fuente del kernel de Linux los puedes obtener de [23]. En esta prueba, usaremos las fuentes del kernel 2.4.24.

Para descomprimir las fuentes de Linux, utilizaremos el comando siguiente:

```
# tar -jxvf linux-2.4.24.tar.bz2
```

Una vez que descomprimimos las fuentes de Linux, hay que aplicarle el parche uml-patch-2.4.24-1.bz2 utilizando los siguientes comandos:

```
# cd uml/linux
# bzcat ../uml-patch-2.4.24-1.bz2 | patch –p1
```

Esto aplicará el parche UML a las fuentes de Linux, de esta manera podemos comenzar a configurar y a construir el kernel UML. Configurar y construir el kernel es exactamente lo mismo a construir un kernel regular, únicamente con un parámetro extra al momento de la configuración y compilación. Vamos a configurar el kernel UML de la siguiente manera:

```
# cd uml/linux
# make xconfig ARCH=um
```

La configuración por default debería ser suficiente para obtener un sistema básico levantado y funcionando, sin embargo, si se desea agregar o modificar algo mas complejo, es necesario saber que es lo que se necesita y modificar las opciones adecuadas al momento de la configuración. Construir el kernel es ligeramente diferente a la instalación normal pero es mucho más sencillo ya que solo queremos construir el binario del kernel. Para hacerlo usamos:

```
# make linux ARCH=um
```

Después de un momento, se obtendrá un archivo binario llamado “linux” dentro del directorio “uml/linux”. Este archivo, se puede ejecutar desde la línea de comandos:

```
$ ./linux
```

Con este comando, podemos levantar nuestra primer máquina virtual UML, pero nos mandará un mensaje de error si no encuentra el file system o no le especificamos uno como parámetro en el comando anterior. Más adelante se muestra la forma en la que se realiza esto con el parámetro ubd0, o si se desea omitir este parámetro, se debe renombrar el file system seleccionado a “root_fs” que es el nombre que UML buscará por default en la carpeta donde se ejecute el comando ./linux.

Así como el kernel, también se necesitarán algunas herramientas UML, principalmente para obtener las funcionalidades de red, que también incluye algunas herramientas de administración que son muy útiles. Estas utilerías vienen incluidas en la mayoría de los parches para el kernel UML. Si se intentara utilizar ciertas capacidades de red dentro de UML, el único requerimiento necesario es el “TAP/TUN driver” ya sea como módulo o dentro del kernel en el sistema del host, debido a que UML lo usa para comunicarse con el host para acceder la red. Naturalmente, si se intenta utilizar el host para enrutar paquetes IP, particularmente si va a hacer uso del masquerading (enmascaramiento) u otras características de firewalling, se requerirá el soporte de netfilter en el kernel del host ya sea con ipchains o con iptables, el que se prefiera.

Booteo de un kernel UML

Bootear el kernel UML es muy sencillo, solo requerimos avisarle al kernel donde está el filesystem de root, y cuanta memoria se va a usar. Por default, la instalación UML tendrá 16 MB de memoria “real”, aunque esta es obtenida de la memoria del host, puede tener parte del swapped en el disco. Por supuesto, podemos darle al sistema UML una gran cantidad de RAM. Si no le especificamos la ruta donde tenemos nuestro file system, por default lo buscará en la carpeta donde se encuentre el binario linux y con el nombre root_fs.

Para bootear nuestro sistema UML con 32MB de RAM “real”, necesitamos hacer lo siguiente:

```
# cd uml/linux
# ./linux ubd0=../root_fs ubd1=../swap_fs mem=32M
```

Esto bootearía el kernel UML de linux y montaría el filesystem apropiado. Ahora tenemos una máquina virtual con características propias e independientes a las de la máquina física, host kernel, ahora sólo nos resta realizar el booteo de esta máquina virtual para establecer la comunicación con el host kernel.

Configuración de la máquina virtual UML

Un sistema UML no es muy útil sin acceso a la red, y como no hay una interfaz física que lo conecte al mundo exterior, se necesita configurarlo para que enrute paquetes a través del host. Sin embargo, antes de que se tenga que complicar con asuntos más profundos de red, necesitamos configurarlo para que cree una pequeña red con la máquina host.

Actualmente hay cinco tipos de transporte disponible en UML para el intercambio de paquetes con otros hosts[11]:

- Ethertap
- TUN/TAP
- Multicast
- Un demonio de switch
- Slip
- Sliprp
- Pcap

Los transportes TUN/TAP, ethertap, slip y sliprp permiten a las máquinas UML intercambiar paquetes con el host. Estos paquetes pueden ser dirigidos directamente al host o que el host actúe como un router para proporcionar acceso a otras máquinas virtuales o físicas.

El transporte pcap es una interface sintética de solo lectura que se utiliza con el binario libpcap para recolectar paquetes de las interfaces en el host y filtrarlas. Es útil para construir monitores de tráfico o sniffers.

Los transportes multicast y demonio de switcheo generan una red completamente virtual a otras máquinas virtuales. Estas redes están completamente desconectadas de la red física a menos que una de las máquinas virtuales actúe como gateway al exterior.

Si tenemos nuestra red física, 10.1.1.0 con máscara 255.255.255.0, podemos hacer que el sistema UML aparezca como host de esa red o podemos tener una subred separada que será enrutada por las otras máquinas. La primera opción es simple para una sola instalación UML, pero si comenzamos a tratar con múltiples host UML en más de una máquina física, se requiere una configuración de enrutamiento mas estructurado, simplemente para estar al tanto de lo que está sucediendo en la red.

La forma más sencilla es crear una red /30 que consiste de una dirección de subred, una dirección de broadcast y dos direcciones Ips: una para el host y una para UML. Esto ayudará a enrutar fácilmente los paquetes entre las dos máquinas (host y UML), para que posteriormente los paquetes sean enrutados a la LAN física. Para el ejemplo de la red 10.1.1.0/24, vamos a usar la subred 10.1.1.252/30, la dirección IP del host sería la 10.1.1.253 y la dirección IP del sistema UML sería la 10.1.1.254.

Configurar la dirección IP del lado del host se hace a través de la línea de comandos, el host UML se configura una vez que esté arriba y funcionando utilizando el comando de sistema “ifconfig”.

Para bootear nuestro sistema UML, dándole al lado del host la dirección IP 10.1.1.253 se ejecuta desde la línea de comandos lo siguiente:

```
# ./linux mem=32M eth0=ethertap,tap0,,10.1.1.253
```

Una vez que el sistema UML ha booteado, se configura la interfaz eth0 de la máquina UML con los siguientes comandos:

```
# ifconfig eth0 10.1.1.254 netmask 255.255.255.252 up
# route add default gw 10.1.1.253
```

Prueba de comunicación con el host kernel

Una vez levantada y configurada la máquina virtual, ésta puede enviar y recibir paquetes al host y también a la red física a la que se encuentre conectada la máquina host. Para realizar esta prueba, se utilizará el comando ping, el cual nos dirá si existe comunicación entre las máquinas virtual y física.

La figura A.1 muestra un ejemplo de la salida del comando ifconfig en una terminal UML, así como el uso del comando ping hacia el host kernel con dirección ip 10.1.1.253.

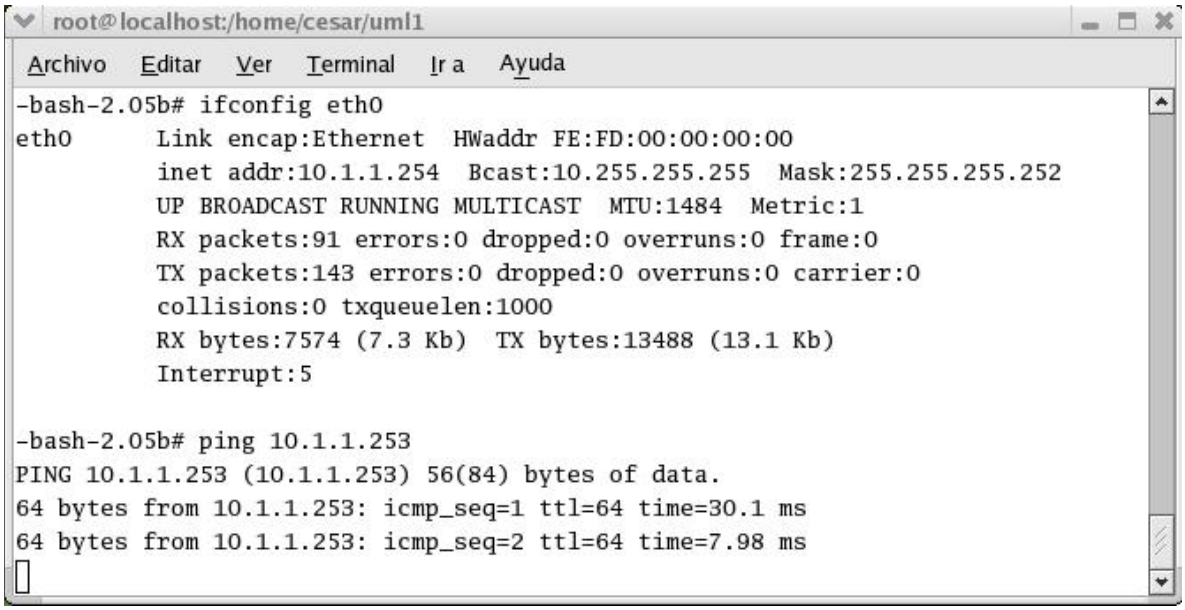

Figura A.1: Salida de los comando ifconfig y ping en una de las máquinas virtuales

La siguiente prueba se hizo con dos máquinas virtuales conectadas al host kernel mediante el uso de las interfaces tap0 y tap1. El kernel es el que actúa como ruteador y da la salida a la red física a la que está conectado.

La conexión de estas máquinas al host kernel podría describirse como se muestra en la figura A.2:

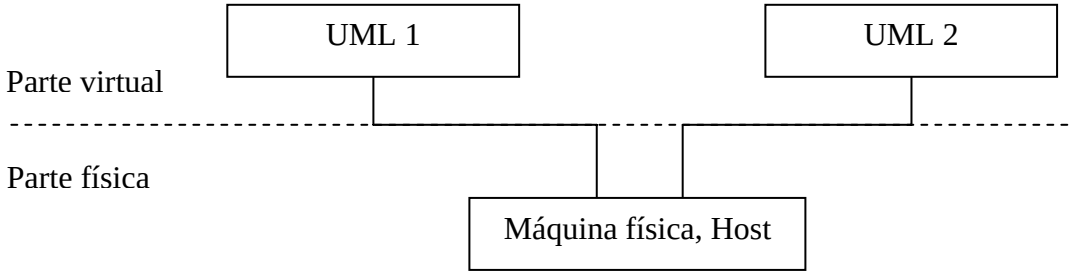

Figura A.2: Configuración lógica de la red virtual usada en la prueba de conexión

Las siguientes imágenes son terminales que muestran la comunicación entre las máquinas virtuales y el host kernel. La primera, A.3, muestra al host kernel realizando un ping a la máquina virtual UML1, la segunda, A.4, es un ping de la máquina UML1 a la máquina UML2 y la tercera, A.5, muestra a la máquina virtual UML2 realizando un ping al host kernel.

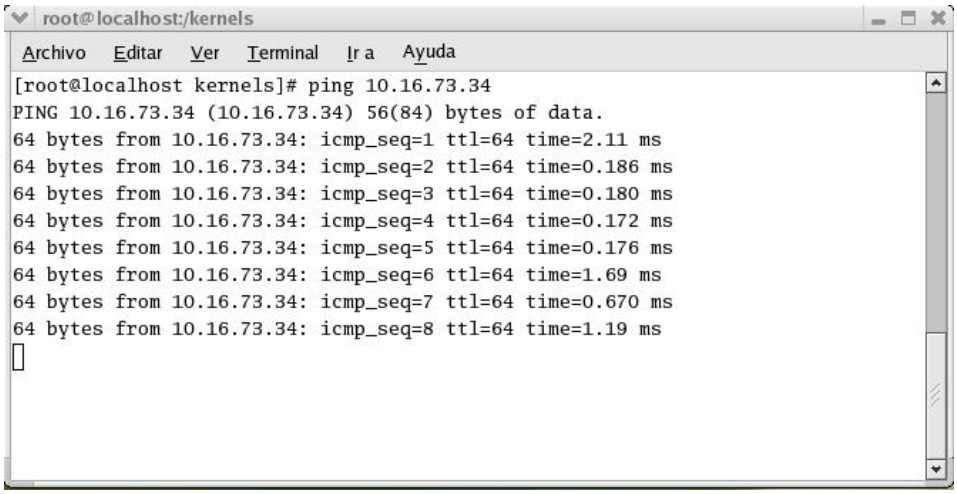

Figura A.3: Uso del comando ping del host a la máquina UML 1

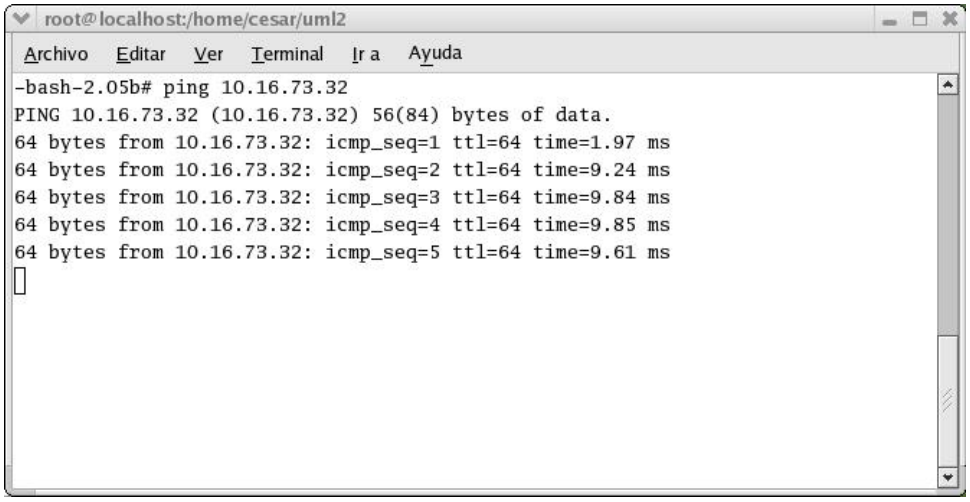

Figura A.4: Uso del comando ping de la máquina UML1 a la máquina UML 2

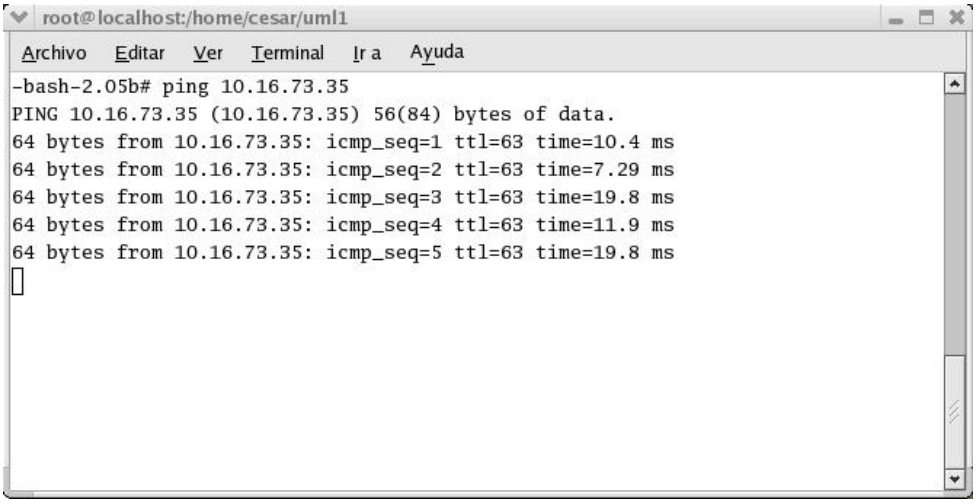

Figura A.5: Uso del comando ping de la máquina UML 2 al host

Capítulo 5

Conclusiones y Trabajo Futuro

En este trabajo se presentó una modificación al algoritmo Hop Counter Filtering presentado por Jin en [6], el cual identifica el tráfico de ataque falsificado por medio del campo TTL del encabezado de los paquetes.

Las modificaciones propuestas en este trabajo fueron en la inicialización de la tabla IP2HC, la cual sirve para el filtrado de los paquetes atacantes. Esta tabla se inicializa de forma manual con base a un análisis previo del tráfico recolectado en un periodo de tiempo definido en base a la carga del servidor que se va a defender. La modificación consiste en que esta tabla se inicializa al momento que el programa empieza a correr en la máquina defendida, calculando el Hop-Count de cada paquete y almacenándolo en una lista con su correspondiente dirección IP y la frecuencia con la que llegan los paquetes con la misma dirección IP y Hop-Count.

Otra modificación realizada es en la forma de aplicar el filtrado. Originalmente el algoritmo HCF descarta los paquetes que no coincidan con el Hop-Count registrado en la tabla IP2HC, o los que difiera en una o dos unidades.

Usando reglas de lógica difusa podemos aplicar un filtrado más preciso. Si el tráfico atacante coincide con el tráfico de un cliente legítimo, el algoritmo original bloqueará todos los paquetes con el mismo Hop-Count, produciendo daño colateral. Con las reglas de lógica difusa se puede definir un rango, en porcentajes, de paquetes recibidos por cada IP, y si estos exceden el rango predefinido, se descartarán dichos paquetes. Si el cliente tiene un comportamiento normal, no sufrirá ningún daño colateral causado por el algoritmo HCF.

Se realizaron simulaciones usando únicamente máquinas virtuales, una víctima y un atacante, pero también se realizó una prueba usando la red escolar, en la cual, desde algunas máquinas físicas ubicadas en diferentes edificios, se instaló la herramienta Syn Flooder para lanzar un ataque hacia la máquina virtual de un host ubicado en otro nodo de la misma red. Los resultados fueron similares a los realizados únicamente con máquinas virtuales, con la diferencia que el ataque se detectaba en forma más rápida y se bloqueaban las direcciones IP de estos ataques.

Una solución a este problema de tráfico VoIP, fue excluir las direcciones IP de los dispositivos que están enviando ese tráfico, como se mostró en la figura 4.14.

Otra solución, en base a la anterior, es separar lógicamente la red de datos de la red de voz, mediante el uso de vlans por medio de switches y de esta forma tendremos excluidas todas las direcciones IP de los dispositivos que puedan estar enviando VoIP sobre la red.

Durante la realización de las pruebas, se utilizaron direcciones IP generadas aleatoriamente usando todo el espacio disponible en el rango 2^{32}, y al momento del envío de estos paquetes, se notó el incremento en los paquetes producidos por el protocolo ARP (Address Resolution Protocol), lo cual nos daría un indicador de la presencia de un ataque de DoS.

También, con base a lo mencionado anteriormente, tendríamos una cantidad enorme de direcciones IP en nuestra tabla IP2HC, realizando un análisis en cuanto al número de paquetes de cada una de ellas, podríamos encontrar alguna forma de identificar la presencia de un DoS y tratar de mitigarlo.

Anexo B

Detalles de tiempos entre las opciones –w de tcpdump y > de Linux

La opción –w de tcpdump graba en un archivo especificado el contenido de los paquetes analizados en base a las expresiones dadas en el comando cuando corresponden a éstas. El formato en el que se guarda esta información, es un formato específico de tcpdump, el cual no pudo ser leído por FLF4DoS. Con la opción de direccionamiento de Linux, >, se guardará en el archivo especificado, el resultado que el comando normalmente muestra en la salida estándar, con un formato de texto plano y separando cada contenido de un encabezado en cada línea de texto en el archivo.

También, de acuerdo a pruebas realizadas, se observó que el tamaño de los archivos guardados con la opción –w de tcpdump, es mayor que aquellos guardados con la opción de direccionamiento de Linux, >.

Las pruebas se realizaron con ayuda de la herramienta D-ITG[12] y usando UML[11] de Linux de la siguiente forma:

- Se envió un número constante de paquetes por segundo de una máquina UML a otra.
- Se probó el comando tcpdump con la opción –w archivo y "> archivo" tomando tiempos con el comando time de Linux.
- Se tomaron muestras de tiempos en pruebas diferentes variando la velocidad de paquetes por segundo y en el número de paquetes que se recibían en una de las máquinas virtuales.
- Las velocidades de paquetes por segundo que se emplearon en la prueba fueron 10, 100, 1 000, 10 000, 100 000.

Los resultados de los tiempos obtenidos así como del tamaño de los archivos producidos con tcpdump con las dos opciones se muestran en las tablas B.1 y B.2:

	velocidad de 100 paquetes por segundo								
	10 pqts	**archivo: 2k**		100 pqts	**archivo: 11k**		1,000 pqts	**archivo: 110k**	
	prueba 1	prueba 2	prueba 3	prueba 1	prueba 2	prueba 3	prueba 1	prueba 2	prueba 3
real	**.161 s**	**.156 s**	**.151s**	**1.05 s**	**1.046 s**	**1.021 s**	**10.374 s**	**10.134 s**	**10.315 s**
user	.010 s	--	--	--	--	--	--	--	--
system	--	--	.010 s	.010 s	--	.010 s	--	--	--

	velocidad de 1,000 paquetes por segundo			velocidad de 10,000 paquetes por segundo		
	10,000 pqts	**archivo: 1.094Mb**		100,000 pqts	**archivo: 10.913 Mb**	
	prueba 1	prueba 2	prueba 3	prueba 1	prueba 2	prueba 3
real	**24.783 s**	**24.791 s**	**24.440 s**	**28.073 s**	**28.889 s**	**29.850 s**
user	--	.010 s	.010 s	.020 s	.050 s	.020 s
system	.010 s	.020 s	--	.030 s	.010 s	.020 s

Tabla B.1: Pruebas de tiempos con la opción -w de tcpdump; formato del archivo: formato especial de tcpdump.

	velocidad de 100 paquetes por segundo								
	10 pqts	**archivo: 1k**		100 pqts	**archivo: 8k**		1,000 pqts	**archivo: 80k**	
	prueba 1	prueba 2	prueba 3	prueba 1	prueba 2	prueba 3	prueba 1	prueba 2	prueba 3
real	**.166 s**	**.155 s**	**.201 s**	**1.772 s**	**1.080 s**	**1.022 s**	**10.050 s**	**10.041 s**	**10.037 s**
user	--	--	--	--	--	--	--	--	--
system	--	--	--	.040 s	--	--	--	.010 s	--

	velocidad de 1,000 paquetes por segundo			velocidad de 10,000 paquetes por segundo		
	10,000 pqts	**archivo: 806k**		100,000 pqts	**archivo: 8.074Mb**	
	prueba 1	prueba 2	prueba 3	prueba 1	prueba 2	prueba 3
real	**24.998 s**	**24.833 s**	**24.923 s**	**29.724 s**	**30.615 s**	**28.840 s**
user	.010 s	.010 s	.020 s	.050 s	.040 s	.040 s
system	.030 s	--	.020 s	.030 s	.030 s	--

Tabla B.2: Pruebas de tiempos con la opción de direccionamiento de Linux >; Formato del archivo: texto plano.

De acuerdo a estas pruebas, se comprobó que los tiempos en los que cada una de las opciones con las que se cuenta para poder almacenar la salida del comando tcpdump, no es muy variante. El tamaño de los archivos producidos en cada opción, crece un poco más con la opción –w de tcpdump, pero esto tampoco es un punto a discutir.

La principal diferencia entre una opción y otra, es el formato en el que se guarda el archivo producido, ya que para el funcionamiento de FLF4DoS, se necesita de un archivo de texto plano en el cual se guarden las direcciones IP de cada paquete así como del valor del campo TTL. La opción que se ajustó a este requerimiento fue la de direccionamiento de salida de Linux, >, la cual nos produce el formato requerido, éste se muestra en la figura 6.2.

Anexo C

Código fuente de FLF4DoS

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

struct nodo{
        unsigned long int ip;
        int     ttl, count;
        struct nodo *nextPtr, *prevPtr;
        struct lista *listaPtr;
};

struct lista{
        unsigned long int ip;
        int hc;
        int frec;
        int frec1;
        int warning;
        float porcentaje;
        struct lista *nextPtr;
};

typedef struct nodo Nodo;
typedef Nodo *NodoPtr;

typedef struct lista Lista;
typedef Lista *ListaPtr;

void insertarNodo(NodoPtr *,unsigned long int,int,int);
void inOrder(NodoPtr);

void insertarLista(ListaPtr *,long unsigned int,int);
void imprimeLista(ListaPtr);
void agregaFrec(ListaPtr curPtr,unsigned long int,int);
int buscaNodo(ListaPtr curPtr,unsigned long int,int);
```

```
int getOcteto(unsigned long int,int);
unsigned long int setIp(int,int,int,int);

void leerCadena(char []);
void getIp(char []);
int getTtl(int);
long int leeInterfaz(char *);

unsigned long int ip;
char protocolo;
int ttl;
ListaPtr inicio = NULL;
NodoPtr raizPtr = NULL;
long int pqtsAntes=0,pqtsDespues=0,difPaq=0;
char *interface, *host, *archivo;

int main(int argc, char *argv[])
{
        int ip1,ip2,ip3,ip4,ttl,frec,a;
        FILE* f_read;
        char leer[85];
        char cadena[85];
        char *token;
        char *comando="kill -2  ";

        pqtsAntes=0;
        pqtsDespues=0;

        if(argc!=4)
        {
          printf("\nusar: ./filter3 interface ip-host archivo_destino\n");
          return 0;
        }
        else
        {
          interface=argv[1];
          host=argv[2];
          archivo=argv[3];
        }
        a=0;
        while (a<=3)
        {
                strcpy(cadena,"tcpdump -i ");
                strcat(cadena,interface);
```

```
                strcat(cadena," -qtvn dst host ");
                strcat(cadena,host);
                strcat(cadena," > ");
                strcat(cadena,archivo);
                strcat(cadena," &");
                printf("%s",cadena);
                pqtsAntes=leeInterfaz(interface);
                system(cadena);
                sleep(15);
                system("ps -A | grep \"tcpdump\" > pidtcpdump.txt");
                pqtsDespues=leeInterfaz(interface);
                difPaq=pqtsDespues-pqtsAntes;
                f_read = fopen("pidtcpdump.txt","r");
                if(!f_read)
                {
                        perror("Error al abrir el archivo tcpdump.txt");
                        exit(1);
                }
                fgets(leer,85,f_read);
                token=strtok(leer," ");
                strcpy(cadena,comando);
                strcat(cadena,token);

                system(cadena);

                //f_read = fopen("tcpdump.txt","r");
                f_read = fopen(archivo,"r");
                if(!f_read){
                        perror("error al abrir el archivo");
                        exit(1);
                }
                //printf("\n1");
                while(!feof(f_read))
                {
                        fgets(leer,85,f_read);
                        //printf("%i: %s",i++,leer);
                        leerCadena(leer);
                }

                inOrder(raizPtr);

                //printf("\n");
                imprimeLista(inicio);
                a++;
        }       //fin del while...
```

```
	return 0;
}


long int leeInterfaz(char *inter)
{
	FILE* f_read;
	char leer[35];
	char *token;

	strcpy(leer,"ifconfig ");
	strcat(leer,inter);
	strcat(leer," | grep \"RX packets\" > temp.txt");

	printf("%s\n",leer);
	system(leer);

	f_read=fopen("temp.txt","r");
	if(!f_read)
	{
	  perror("error al abrir el archivo");
	  exit(1);
	}
	fgets(leer,35,f_read);

	token=strtok(leer," ");
	token=strtok(NULL,":");
	token=strtok(NULL," ");
	return atoi(token);
}


void insertarNodo(NodoPtr *nodoPtr,unsigned long int ip,int i1,int i2)
{
	if(*nodoPtr==NULL)
	{
		*nodoPtr=malloc(sizeof(Nodo));
		if(*nodoPtr!=NULL)
		{
			(*nodoPtr)->ip=ip;

			(*nodoPtr)->ttl=i1;
			(*nodoPtr)->count=i2;

			(*nodoPtr)->listaPtr=NULL;
			(*nodoPtr)->nextPtr=NULL;
```

```
                        (*nodoPtr)->prevPtr=NULL;

                        insertarLista(&inicio,ip,i1);

                }
                else
                        printf("%lu no insertado, no hay memoria disponible\n",ip);
        }
        else
                if(ip<(*nodoPtr)->ip)
                {
                        insertarNodo(&((*nodoPtr)->prevPtr),ip,i1,i2);
                }
                else if(ip>(*nodoPtr)->ip)
                {
                        insertarNodo(&((*nodoPtr)->nextPtr),ip,i1,i2);
                }
                else
                {
                        if(buscaNodo(inicio,ip,i1)==1)
                                agregaFrec(inicio,ip,i1);
                        else
                                insertarLista(&inicio,ip,i1);
                }
}


void inOrder(NodoPtr nodoPtr)
{
        if(nodoPtr !=NULL)
        {
                inOrder(nodoPtr->prevPtr);
                printf("%d.%d.%d.%d -->",getOcteto(nodoPtr->ip,4),
                                                        getOcteto(nodoPtr->ip,3),
                                                        getOcteto(nodoPtr->ip,2),
                                                        getOcteto(nodoPtr->ip,1));
                inOrder(nodoPtr->nextPtr);
        }
}


void insertarLista(ListaPtr *lPtr,long unsigned int ip,int i1)
{
        ListaPtr newPtr, prevPtr, curPtr;
        newPtr=malloc(sizeof(Lista));
```

```
	if(newPtr!=NULL)
	{
		newPtr->ip=ip;
		newPtr->hc=i1;
		newPtr->frec=1;
		newPtr->nextPtr=NULL;
		newPtr->frec1=35;
		prevPtr=NULL;
		curPtr=*lPtr;

		while(curPtr!=NULL && ip > curPtr->ip){
			prevPtr=curPtr;
			curPtr=curPtr->nextPtr;
		}

		if(prevPtr==NULL)
		{
			newPtr->nextPtr= *lPtr;
			*lPtr=newPtr;
		}
		else
		{
			prevPtr->nextPtr=newPtr;
			newPtr->nextPtr=curPtr;
		}
	}
	else
		printf("memoria no disponible");
}



void agregaFrec(ListaPtr curPtr,unsigned long int ip,int ttl)
{
	int diferencia;
	if(curPtr==NULL)
		printf("\nLista vac¡a");
	else
	{
		//printf("\nla lista es: \n");
		while (curPtr!=NULL)
		{
			if(curPtr->ip==ip&&curPtr->hc==ttl)
			{
				curPtr->frec++;
```

```
                                /*diferencia=(curPtr->frec)*100/difPaq;
                                if(curPtr->porcentaje>diferencia)
                                {
                                  curPtr->warning=1;
                                }
                                else
                                {
                                  curPtr->warning=2;
                                }
                                curPtr->porcentaje=(curPtr->frec*100)/difPaq;
                                */
                                return;
                        }
                        curPtr=curPtr->nextPtr;
                }
                //printf("NULL\n\n");
        }
}


int buscaNodo(ListaPtr curPtr,unsigned long int ip,int ttl)
{
        if(curPtr==NULL)
                printf("\nLista vac¡a");
        else
        {
                while (curPtr!=NULL)
                {
                        if(curPtr->ip==ip&&curPtr->hc==ttl)
                                return 1;
                        curPtr=curPtr->nextPtr;
                }
        }
}



void imprimeLista(ListaPtr curPtr)
{
        char comando[85];
        int paquetes;
        char *temp;
        if(curPtr==NULL)
                printf("\nLista vac¡a");
        else
        {
```

```
                printf("\nla lista es: \n");
                while (curPtr!=NULL)
                {
                        paquetes=curPtr->frec * 0.20;

                        printf("%d.%d.%d.%d,%d-%d -->%f ",getOcteto(curPtr->ip,4),
                                                          getOcteto(curPtr->ip,3),
                                                          getOcteto(curPtr->ip,2),
                                                          getOcteto(curPtr->ip,1),
                                                          curPtr->hc,curPtr->frec,
                                                          curPtr->porcentaje);

                        if(curPtr->frec>=(curPtr->frec1+paquetes))
                        {
                          strcpy(comando,"iptables -A FORWARD -i ");
                          strcat(comando,interface);
                          strcat(comando," -s ");
                          //printf("%s\n\n",comando);
                          sprintf(temp,"%d",getOcteto(curPtr->ip,4));
                          strcat(comando,temp);
                          strcat(comando,".");
                          sprintf(temp,"%d",getOcteto(curPtr->ip,3));
                          strcat(comando,temp);
                          strcat(comando,".");
                          sprintf(temp,"%d",getOcteto(curPtr->ip,2));
                          strcat(comando,temp);
                          strcat(comando,".");
                          sprintf(temp,"%d",getOcteto(curPtr->ip,1));
                          printf("temp %s\n",temp);
                          strcat(comando,temp);
                          strcat(comando," -j DROP");
                          printf("\nbloqueando..%s",comando);
                          system(comando);
                        }
                        else
                          printf("\n");
                        curPtr=curPtr->nextPtr;
                }
                printf("NULL\n\n");
        }
}
unsigned long int setIp(int ip1,int ip2,int ip3,int ip4)
{
        unsigned long int ip=0;
        ip=ip|ip1;
        ip=ip<<8;
```

```
        ip=ip|ip2;
        ip=ip<<8;
        ip=ip|ip3;
        ip<<=8;
        ip=ip|ip4;
        return ip;
}


int getOcteto(unsigned long int ip, int octeto)
{
        unsigned long int mask1, mask2, mask3, mask4,temp;
        switch (octeto){
         case 1:  ip<<=24;
                                                ip>>=24;
                                                return (ip);
         case 2:  ip<<=16;
                                                return ip>>24;
         case 3:  ip<<=8;
                                                ip>>=24;
                                                return ip;
         case 4:  return ip>>24;
        }
}

void leerCadena(char cad[]){
        char *token, *tok;
        char temp[22]={'\0'};
        int k=0,i=0;

                while(i<22)
                {
                        temp[i]=cad[i];
                        i++;
                }
                temp[i-1]='\0';

                tok=strtok(temp," ");
                getIp(tok);


        token=strtok(cad," ");
        while(token!=NULL){
         k++;
         //if(k==1)
         //       {
```

```
	//		printf("%s\n",token); //dirs[pos].ips);
	//		i=0;
	//	}
	//	if(k==4)
	//	{
	//		printf("%s\n",token);
	//	}
		if(k==7)
		{
			ttl=getTtl(atoi(token));
			k+=2;
		}
		if(k==8)
		{
			//printf("%d\n",atoi(token));
			ttl=getTtl(atoi(token));
		}
	 token=(strtok(NULL," "));
	}
	insertarNodo(&raizPtr,ip,ttl,1);
	//printf("\ntokens: %d\n",k);
}

void getIp(char cd[]){
				char *tok;
				int ip1,ip2,ip3,ip4;
				int i=0;
				tok=strtok(cd,".");
				//printf("\n");
				while(tok!=NULL){
				 i++;
				 switch (i)
				 {
					case 1: ip1=atoi(tok); break;
					case 2: ip2=atoi(tok); break;
					case 3: ip3=atoi(tok); break;
					case 4: ip4=atoi(tok); break;
				 }
				 tok=(strtok(NULL,"."));
				}
				ip=setIp(ip1,ip2,ip3,ip4);
}
```

```
int getTtl(int value)
{
	if(value>=16&&value<=32)
		return (32-value);
	else if(value>32&&value<=64)
		return (64-value);
	else if(value>64&&value<=128)
		return (128-value);
	else
		return (256-value);
}
```

Referencias

[1] A. Hussain, J. Heidemann and C. Papadopoulos, “A Framework for Classifying Denial of Service Attacks”, USC/Information Sciences Institute, páginas 99-110, Febrero 2003.

[2] V. Paxson, “An analysis of using reflectors for Distributed Denial-of-Service Attacks”, ACM Computer Communications Review (CCR), páginas 38-47, Julio 2001.

[3] R. K. C. Chang, “Defending Against Flooding-Based Distributed Denial-of-Service attacks: A Tutorial”, The Hong Kong Polytechnic University/ IEEE Communications Magazine, Octubre del 2002.

[4] S. Dietrich, N. Long and D. Dittrich, “Analyzing distributed denial of service tools: The shaft case”, In *Proceedings of USENIX LISA’2000*, New Orleans, LA, Diciembre 2000.

[5] D. Dittrich, “Distributed Denial of Service (DDoS) attacks/tools page”, Disponible en: http://staff.washington.edu/dittrich/misc/ddos/.

[6] C. Jin, H. Wang, K.G. Shin, “Hop-Count Filtering: An effective Defense Against Spoofed DDoS Traffic”, Proceedings of the 10th ACM conference on Computer and communications security, Octubre del 2003, pp 30-41.

[7] Ethereal: A Network Protocol Analizer, http://www.ethereal.com/, accesado en Julio del 2005.

[8] The Swiss Education and Research Network. “Default TTL values in TCP/IP, 2002”. http://secfr.nerim.net/docs/fingerprint/en/ttl default.html, accesado en Julio del 2005.

[9] V.Paxon, “End-to-End Routing Behavior in the Internet”, IEEE/ACM Transactions on Networking (TON), Volumen 5 número 5, Octubre de 1997.

[10] National Laboratory for Applied Network Research. “Active measurement project” 1998, http://watt.nlanr.net/ , accesado en Julio del 2005.

[11] The User Mode Linux Kernel Home Page, http://user-mode-linux.sourceforge.net/

[12] Distributed Internet Traffic Generator, D-ITG, http://www.grid.unina.it/software/ITG, accesado en Julio del 2005.

[13] Syn Flooder, Helsinki University of Technology Department of Computer Science, http://www.niksula.cs.hut.fi/~dforsber/synflood/programs/, accesado en Julio del 2005.

[14] R. Mahajan, S.M. Bellovin, S. Floyd, J. Ionnidis, V. Paxon, S. Shenker, "Controlling High Bandwidth Aggregates in the Network", ACM SIGCOMM Computer Communication Review, vol. 32 número 3, Julio del 2002, páginas 62-73.

[15] S. Savage, D. Wetherall, A. Karling, T. Anderson, "Practical Network Support for IP Traceback" Proceedings of ACM SIGCOMM 2000, Agosto de 2000, pp. 296-306.

[16] A.C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, S.T. Kent, W.T. Strayer, "Hash-Based IP Traceback", Proceedings of ACM SIGCOMM 2001. Agosto del 2001, pp. 3-14.

[17] J. Mirkovic, S. Dietrich, D. Dittrich, P. Reiher, "Internet Denial of Service, Attacks and Defense Mechanisms", Prentice Hall, 2005, ISBN: 0-13-147573-8.

[18] J. Mirkovic. D-WARD: Source-End Defense Againts Distributed Denial of Service Attacks, PhD thesis, University of California Los Angeles, Agosto del 2003, http://lasr.cs.ucla.edu/ddos/dward-thesis.pdf

[19] R.Thomas, T. Johnson, J. Croall, B. Mark, **"**NetBouncer: client-legitimacy-based high-performance DDoS filtering**",** DARPA Information Survivability Conference and Exposition, 2003. Proceedings, Volúmen 1, Abril 22-24 2003, pp.14 – 25.

[20] A. Yaar, A. Perrig, D. Song, **"**Pi: a path identification mechanism to defend against DDoS attacks", Proceedings of the IEEE Symposium on Security and Privacy, Mayo 2003, pp. 93 – 107.

[21] Cyber-attacks batter Web heavyweights, CNN.com, http://www.cnn.com/2000/TECH/computing/02/09/cyber.attacks.01/index.html, accesado en Julio del 2005.

[22] 'Immense' network assault takes down Yahoo, CNN.com, http://www.cnn.com/2000/TECH/computing/02/08/yahoo.assault.idg/index.html, accesado en Julio del 2005.

[23] The Linux Kernel Archives, http://www.kernel.org/, accesado en Julio del 2005.

[24] Computer Crime and Security Survey, http://www.gocsi.com/forms/fbi/csi_fbi_survey.jhtml, accesado en Julio del 2005.

[25] Kasuo Tanaka, “An Introduction to Fuzzy Logic For Practical Applications”, Ed. Springer, 1997, ISBN: 0-387-94807-4.

[26] C. P. Pfleeger, “Security in Computing”, Prentice Hall, segunda edición 1997, ISBN: 0-13-337486-6.

Lista de figuras y tablas

Agradecimientos

Quiero agradecer sinceramente al ITESM Campus Monterrey, por haberme apoyado en el financiamiento de mis estudios y por contribuir a mi formación personal, académica y profesional.

A mi asesor Alejandro Parra Briones, por haberme ayudado enormemente en la investigación y realización de esta tesis, y por la gran cantidad de buenos consejos tanto a nivel escolar, profesional y personal.

A mis sinodales Juan Arturo Nolazco Flores y Jorge Carlos Mex Perera por su contribución, opiniones y recomendaciones a esta tesis.

Y a todas esas grandes personas, familiares y amigos que contribuyeron de forma directa o indirecta, pero que sin su apoyo moral, hubiera sido muy difícil lograr esta meta.

Muchas gracias

Julio César Covarrubias Rodríguez

Resumen

Internet es una gran red de computadoras que se interconectan en todo el mundo. En ella podemos encontrar todo tipo de información, desde sencillas páginas informativas hasta complejos sistemas de acceso a bases de datos mediante diversos lenguajes de programación. En Internet, desde hace unos años a la fecha, muchas empresas han ofrecido diversos servicios en línea a sus clientes, desde consulta de información hasta pagos en línea. Internet también ha sido empleada por personas mal intencionadas, cuyo fin es afectar de alguna forma a empresas con presencia en la red.

Un mal uso de la red, entre otros, es la llamada Negación de Servicios o DoS (Denial of Service, por sus siglas en inglés) el cual es un problema, que puede ser muy sencillo o muy complejo, que tiene como finalidad interrumpir los servicios de alguna empresa para impedir que los clientes puedan usar adecuadamente dichos servicios provocando pérdidas económicas a las empresas. Estas pérdidas representan la segunda causa de perdidas en las empresas según reportes del FBI.

Existen mecanismos de defensa que tratan de prevenir, reaccionar o rastrear las fuentes de un ataque, estas técnicas han sido probadas para casos específicos de ataques. Desafortunadamente también existen técnicas de ataques que dificultan la prevención, reacción o rastreo de los atacantes.

En esta tesis, se propone una modificación a un trabajo previo, el cual trata de identificar el tráfico de ataque para impedir su paso hacia la víctima. La modificación propuesta hace uso de lógica difusa para realizar el filtrado del tráfico atacante. Si éste tiene un comportamiento que se identifica como parte de un ataque de DoS, se le aplicarán reglas de filtrado para impedir su paso. Esta aplicación, FLF4DoS, fue desarrollada en Linux y para la simulación y pruebas se utilizó un ambiente virtual y herramientas de generación de tráfico así como de monitores de red para observar el ancho de banda en la simulación de tráfico normal y tráfico atacante así como del uso de FLF4DoS y su funcionamiento al descartar paquetes que han sido identificados como atacantes.

Tabla de Contenido