MCT-L7

Monitoreo y Caracterización del Tráfico de Layer 7

Cecilia Abalde
Sebastián Hauret
Sebastián Montes de Oca
Verónica Peña

Tutor: Ing. Gabriel Gómez.

Facultad de Ingeniería
Universidad de la República

Proyecto de fin de carrera, Ingeniería Eléctrica
Plan 97, Telecomunicaciones

Resumen

El objetivo de nuestro proyecto de fin de carrera fue diseñar una herramienta capaz de monitorear el tráfico de red, identificar en forma pasiva los distintos protocolos a nivel de capa de aplicación y representar gráficamente los resultados obtenidos en diferentes períodos de tiempo.

No será útil realizar la identificación por puertos, ya que nuestra intención es poder reconocer todo tipo de aplicaciones, incluyendo los Peer-to-Peer (P2P). Por lo tanto, indicaremos que posibilidades existen para ello.

La herramienta fue desarrollada sobre el sistema operativo open source Linux y gran parte de la misma fue implementada a nivel del kernel. Se decidió realizar así, con el fin de lograr un bajo consumo de cpu y memoria, y de esta manera presentar menores tiempos de procesamiento y analizar a mayor velocidad el tráfico en tiempo real.

Índice

Estructura de la documentación

Se comienza por una introducción donde se describen algunas de las necesidades actuales de las redes de datos que impulsaron a llevar a cabo este proyecto. Se prosigue con el planteo del objetivo del mismo donde se detalla los principios de diseño que fueron tomados en cuenta.

Luego se describen las herramientas investigadas y estudios complementarios utilizados para argumentar por qué estas herramientas fueron descartadas tomando la decisión de trabajar a nivel de kernel.

Se describe el sistema modular implementado y se explican las funcionalidades de los tres módulos que conforman el sistema: Módulo de captura y análisis, Módulo de recolección de datos y almacenamiento en base de datos y Módulo para la representación gráfica de los datos obtenidos. Dentro de cada uno de ellos se detallan todas las herramientas utilizadas, el funcionamiento de las mismas y cómo interactúan entre si.

Sobre el final de la documentación, se dan los procedimientos que fueron llevados a cabo para testear la herramienta y se muestran los resultados y las conclusiones obtenidos.

1. Introducción

El gran crecimiento de Internet no solo ha incrementado el volumen de tráfico que las redes de comunicaciones deben soportar, sino también ha transformado la naturaleza del mismo. Internet se ha convertido en una infraestructura global de telecomunicaciones capaz de integrar una enorme variedad de aplicaciones desarrolladas por cualquier usuario o empresa. Esta posibilidad brindada por IP es la responsable de su gran crecimiento de los últimos años, y existe una perspectiva de que esta tendencia continuará hacia el futuro.

La infraestructura sobre la cual se sustenta Internet provee un nivel de servicio de best-effort a nivel de capa de red, lo cual significa que todo el tráfico IP generado por diversas aplicaciones debe competir por los recursos disponibles; esto es, el ancho de banda de los enlaces, capacidad de procesamiento y espacio en buffer de los equipos enrutadores. Las distintas aplicaciones compiten sin considerarse los distintos requerimientos que puede tener cada una de ellas, y por lo tanto no se brinda las diferenciaciones que podrían mejorar sus rendimientos. Tampoco se tiene conocimiento sobre qué es lo que los usuarios demandan y cómo va variando esto a lo largo del tiempo. Poder conocer esto último, podría darle información importante a los proveedores de Internet sobre nuevos servicios que podrían ofrecer y los beneficios que obtendrían.

Generalmente, la caracterización del tráfico o la implementación de políticas que restringen el acceso a ciertos servicios de Internet, son realizados por los usuarios finales o empresas mediante firewalls o puntos de acceso, de forma de controlar el consumo excesivo del ancho de banda, así como también mantener protegida sus redes de intrusos o virus que puedan dañar sus equipos. Sin embargo, los servicios de firewall tradicionales utilizan la información obtenida de las capas inferiores del modelo OSI, como ser las direcciones MAC, direcciones IP, flags de las banderas TCP, puertos TCP bien conocidos, etc., para identificar qué aplicación o host está generando los paquetes y qué tipos de flujos atraviesan el firewall. Con esta identificación se definen políticas y acciones a realizar sobre cada paquete o flujo.

Existe una asociación entre las aplicaciones y puertos que es mantenida por la IANA *(Internet Assigned Numbers Authority)*, en la cuál se basan los firewalls y una gran cantidad de equipos para distinguir entre distintas aplicaciones. La asociación está separada en tres grupos: puertos bien conocidos, puertos registrados y puertos dinámicos y/o privados. Todas estas asociaciones son solo recomendaciones, lo cual significa que queda abierta la posibilidad de poder ejecutar las aplicaciones en puertos diferentes a los asignados por la IANA [1].

Basándose en estudios realizados sobre redes de acceso de Internet, gran parte del tráfico actual no es identificable por dicho esquema, y este porcentaje caracterizado incorrectamente se va incrementando con el tiempo [2] [3].

La rápida evolución de las aplicaciones Peer-to-Peer, las hace “inteligentes” e “incontrolables”: son capaces de testear los puertos abiertos de un firewall y redirigir el tráfico hacia ellos, siendo este proceso totalmente transparente para el usuario e imposible de bloquear con los métodos tradicionales. Por ejemplo, estas aplicaciones pueden funcionar por el puerto estándar de navegación Web (HTTP 80), para poder evadir políticas de seguridad y de monitoreo aplicadas por los firewalls. Un caso claro donde se puede ver este comportamiento es por ejemplo con el protocolo de VoIP Skype, el cual pone especial esfuerzo en escapar a las restricciones de los firewalls. Skype se camufla como http o https y utiliza su puerto estándar, para hacer un túnel con dicho protocolo y transmitir su carga útil de esta manera, evitando no sólo el firewall sino también proxies en la capa de aplicación [4].

Esto hace que los firewalls convencionales sean incapaces de rastrear estas conexiones por medio de los métodos tradicionales, y menos aún, ser capaces de brindar políticas que prioricen o limiten el pasaje de paquetes que correspondan a este tipo de protocolos. Por estas razones es que se vuelve necesario hacer un estudio específico de los protocolos que trabajan en la capa de aplicación. Principalmente nos enfocaremos en los P2P, ya que son ellos quienes representan el mayor interés para ser detectados tanto por los administradores de redes como por los proveedores de servicio (ISP).

Existen distintos métodos propuestos para identificar el protocolo utilizado en un flujo de datos. El primero es inspeccionando el payload de los paquetes en busca de patrones, garantizando así con alto grado de certeza, en una gran variedad de casos, la correcta caracterización de ciertos protocolos. Cada aplicación tiene su manera de establecer la conexión, negociar parámetros, etc., y estos son los patrones buscados que se intentan identificar para realizar dicha clasificación. Este método tiene como costo, que al realizar la inspección de cada paquete conlleva un aumento en el procesamiento necesario y tratamiento de los datos en relación a otras herramientas que solo machean por puertos. De todas formas, existen métodos para que no sea necesario analizar todos los paquetes sino un pequeño porcentaje de cada conexión disminuyendo considerablemente el procesamiento necesario. Los mismos serán explicados en detalle más adelante.

Otro método es aprovechar las características estadísticas de los flujos de datos, como pueden ser la cantidad de paquetes transmitidos, tiempo entre arribo, distribución del tamaño de los paquetes durante el transcurso de una conexión, etc. Estudios realizados muestran que distintos tipos de tráfico, como VoIP,
streaming de audio, FTP tienen comportamientos claramente diferenciables que permiten identificar a qué aplicación pertenece el flujo observado. Este método de caracterización también tiene como ventaja que, como no es necesario observar el payload, se pueden caracterizar flujos que utilizan IPsec o en los cuales el payload se encuentra encriptado [5] [6].

2. Objetivo del Proyecto

El objetivo del proyecto consiste en desarrollar un sistema mediante el cual se pueda identificar y cuantificar en tiempo real o desde una traza, el tráfico cursado a través de una red sin alterar el mismo, y además permita visualizar gráficamente el comportamiento de éste en diferentes períodos de tiempo.

A diferencia de otras implementaciones, la identificación del tráfico no consistirá en diferenciar por puertos o dirección IP, sino a partir del payload.

Se buscará que ésta herramienta sea capaz de clasificar la mayoría de los protocolos utilizados a nivel de capa 7, y que a su vez deje abierta la posibilidad de una fácil inserción de nuevos protocolos, para no quedar así limitado a los existentes en la actualidad.

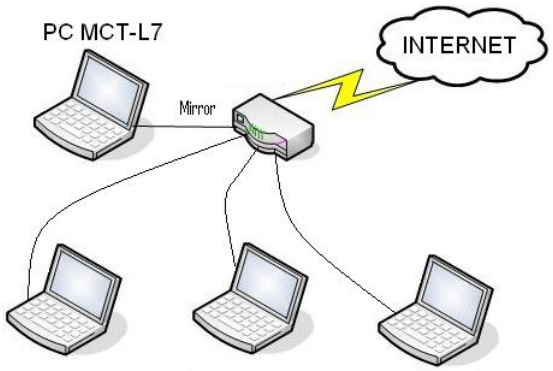
La identificación y cuantificación se deberá realizar de manera confiable, necesitando para ello verificar la precisión de los resultados. Además se analizará la performance en diferentes niveles de tráfico para los distintos protocolos.

Como último punto, será importante especificar los equipos necesarios para el correcto funcionamiento.

Principios de diseño

A continuación se detallan una serie de instancias que fueron tomadas como punto de partida para la elaboración de la herramienta, las cuales consideramos esenciales para la construcción de la misma:

- La herramienta deberá trabajar en tiempo real y en forma pasiva en el análisis de los diferentes flujos que circulan sobre la red, de modo que el tráfico no se vea afectado por la presencia de ésta y evitando que sea un cuello de botella en la red. Con lo cual deberá únicamente actuar como un contador de paquetes o de datos, sin tomar en un principio, políticas administrativas. También deberá poder analizar los datos de una traza guardada en disco.



- Deberá trabajar únicamente a nivel de aplicación para el reconocimiento de los distintos protocolos, evitando de esta forma los análisis tradicionales de los firewall actuales, los cuales utilizan principalmente las capas inferiores para su análisis.

- Trabajará sobre un Hardware básico que consiste en una PC convencional, sin equipamiento específico y sobre un sistema operativo open-source.
- Se necesita que la inserción de nuevos protocolos o la modificación de los ya existentes, se pueda hacer de una forma ágil y relativamente fácil.
- Para la arquitectura de diseño del sistema, consideramos conveniente que la herramienta presente una forma modular, para poder modificar sencillamente cualquiera de los módulos por otros más eficientes, sin alterar el resto del sistema.
- Consideramos que la herramienta debe ser lo suficientemente sencilla para lograr una alta eficiencia del análisis de flujos en redes de gran capacidad, con los requerimientos en el hardware especificados en ítem anteriores.

3. Descripción

Pasos seguidos:

1. Investigar herramientas existentes desarrolladas hasta el momento para ser utilizadas en nuestro software. Estudiar fiabilidad, capacidad y limitaciones de las mismas.
2. Evaluar la utilidad de alguna de ellas para los fines del proyecto. Adicionalmente, de observar que las funcionalidades de alguno de los bloques de código de las herramientas se adaptan a nuestros intereses, se integrará el código del mismo a nuestra aplicación.
3. Descartar herramientas dejando claro el por qué no nos son útiles.
4. Investigar y evaluar trabajar en la detección y clasificación de los protocolos a nivel del Kernel-space o del user-space.
5. Instalación y modificación de un nuevo Kernel agregando las funcionalidades necesarias.
6. Búsqueda de herramientas para llevar una estadística de los protocolos y poder visualizarlos en diferentes períodos de tiempo.
7. Generación de trazas patrones de forma de poder comprobar la precisión de los resultados.
8. Testing y performance. Realización y ejecución de trazas de pruebas con protocolos para evaluar el desempeño de la herramienta.
9. Comparación de los resultados obtenidos con otras herramientas similares.

4. Estudios Previos

Esta sección consta de tres partes donde se describen todos los estudios que se realizaron con el fin de poder discernir con qué tipo de herramienta nos convenía trabajar y con cuales no.

4.1. Formas de clasificación

Como se comentó, nuestro objetivo es poder caracterizar los paquetes o flujos IP según la aplicación que los haya generado. Para esto será necesario estudiar con mayor detalle cómo se comportan los distintos flujos y buscar características que tengan los paquetes que los integran. A partir de este estudio se definirán patrones, los cuales deberán ser fácilmente modificables para adaptarse a los cambios constantes que las aplicaciones realizan en la forma como transmiten sus datos. En particular las aplicaciones P2P que modifican su comportamiento en cada nueva versión. Por esa razón, también es necesario desarrollar métodos que permitan descubrir estos patrones y automatizar la búsqueda de los mismos.

Se pueden encontrar tres maneras de caracterizar el tráfico, las cuales no son exclusivas y es posible utilizarlas simultáneamente, explotando las ventajas que cada una de ellas posee. A continuación se explica en qué consisten estos métodos para luego indicar cómo se complementarían para un análisis más preciso, dinámico y potente:

a) Inspección profunda de paquetes (DPI – deep packet inspection): su cometido es reconocer patrones definiendo características a buscar en los paquetes. Este tipo de reconocimiento es utilizado por los sistemas de detección de intrusos (IDS), para encontrar tráfico malicioso o sospechoso. Este reconocimiento no se reduce a la búsqueda de un solo campo de un paquete, sino que puede definirse que se revise para cada paquete la dirección MAC, el protocolo indicado por IP y alguna bandera de TCP. Estas reglas más específicas se llamarán firmas y caracterizarán a los paquetes que cumplan con ellas.
   Estas firmas se pueden complicar tanto como uno quiera, pero hay que tener en cuenta que las mismas no sean demasiado específicas, ya que serán pocos los paquetes que coincidirán con ellas. Tampoco pueden ser demasiado generales ya que habrá paquetes que serán caracterizados incorrectamente dando lugar a falsos positivos. Debido a lo anterior es posible buscar las características que tienen los paquetes pertenecientes a un stream generado por cierto protocolo de la capa aplicación y hallar estas características tanto en los encabezados Ethernet, IP, TCP y en el payload del mismo. Esta búsqueda aumenta el procesamiento necesario, pero incorpora un método altamente potente para detectar con mayor precisión que aplicación ha generado los paquetes que se inspeccionan.

El desarrollo de estos patrones se realiza fuera de línea, siendo necesaria una etapa de investigación y análisis. La idea es buscar un string o una secuencia de bits en algún lugar específico de un paquete o stream que corresponda a cierta aplicación. Aunque esta tarea no es simple en algunos casos, como los son los protocolos privados o P2P, en la gran mayoría es posible encontrar un patrón para cada una de ellas, ya que al igual que los protocolos de las capas inferiores hay características que debe cumplir cada aplicación para poder comunicarse correctamente con el otro extremo.
Debido a que para su correcto funcionamiento debe poder observar más allá de la información que se provee en la capa de transporte, este método puede suponer una pérdida de privacidad de los usuarios. Se profundizará en este punto cuando se explique en más detalle como se realiza la inspección de los paquetes. También es imposible de esta manera, detectar flujos que estén cifrados como sucede en algunos P2P. Como último punto negativo es que se requiere un alto procesamiento y se dificulta la tarea de reconocimiento cuando aumenta la carga, pero existen métodos que permiten aumentar la eficiencia y que también serán explicados más adelante.

b) Buscar patrones basados en cómo una aplicación se comporta durante su comunicación, los cuales son: tamaño absoluto o relativo de los paquetes que se envían, cantidad de información que se envía por flujo, número de flujos y tasa con la que se crean nuevos flujos por cada aplicación. Esto es útil para diferenciar entre P2P y no P2P. En el primer caso un host se comunica con muchos otros host a través de distintos puertos en forma simultánea, mientras que al tratarse de un servidor Web o de Correo, éste establece varias sesiones con diferentes hosts pero utilizando el mismo puerto.

c) Análisis estadístico: propone una nueva metodología para identificar flujos de transporte en Internet considerando dinámicas en el tráfico de red causadas por las diferentes aplicaciones, puede ser útil para distinguir entre distintos tipos de transmisiones. Por ejemplo, streaming de audio o video, VoIP, chat o transferencia de archivos. Para cada flujo se calculan ciertos indicadores estadísticos como valores medios, mediana, centroide, etc. , y se definen zonas en las que los flujos que caigan dentro de las mismas corresponden a tipos de tráfico similares [5] [7].

Los dos últimos métodos permiten realizar la detección sin necesidad de inspeccionar el payload de los paquetes, evitando invadir la privacidad de los usuarios y disminuyendo el tiempo de procesamiento necesario para realizar la caracterización del tráfico. Sin embargo, deben poder aprender o conocer cuáles son las aplicaciones que generan los flujos que están inspeccionando y en esta etapa de aprendizaje es imperativo utilizar algún DPI o poder contar con tráfico preclasificado contra el que comparar los resultados.

Llegado a este punto, decidimos profundizar el estudio de las herramientas que permiten inspeccionar el payload de los paquetes, ya que éstas podrían entregar resultados que serán útiles para comprobar el correcto funcionamiento de los métodos estadísticos.

Como se dijo anteriormente, estas herramientas deben hacer un procesamiento más exigente y requerirán mayores recursos a la hora de caracterizar altos volúmenes de tráfico. En nuestro caso, pretendemos utilizar una PC convencional y por lo tanto debemos definir de qué forma trabajará la herramienta. Para ello se estudiaron las diferencias entre manejar la información que se encuentra en el payload de los paquetes a nivel de usuario o a nivel del kernel. Posteriormente se detallarán las características de algunas de las herramientas existentes estudiadas, para luego elegir la que más se adapte a nuestras necesidades.

4.2. Espacio de usuario vs. Espacio de Kernel

Con respecto a los analizadores de tráfico para redes de datos, éstos deben ser diseñados para trabajar de la manera más eficiente posible, intentando de esta forma mantener el impacto en la performance de la red lo más pequeño que se pueda. Durante la ejecución de cada proceso a nivel del user-space, la mayoría de los sistemas basados en Unix son regularmente interrumpidos para poder atender procesos de mayor jerarquía, lo cual no sucede en el caso de los "procesos"que corren a nivel del Kernel. A su vez, la transferencia de datos desde el kernel hacia el user-space trae acarreado una serie de operaciones en donde se debe copiar la información a ciertos buffers para que puedan ser leídos en forma correcta por estos procesos. Esto trae como consecuencia una reducción en la performance del sistema. Sin embargo, la eficiencia de estos procesos no es el único aspecto que se debe considerar a la hora de diseñar una herramienta.

Al trabajar a nivel del kernel, se está pagando un alto precio a nivel de seguridad y en la robustez del sistema, debido al hecho de que es sumamente difícil mantener políticas de restricción y de supervisión (políticas de seguridad) en el código ejecutado a nivel del kernel, es que en numerosos casos es necesario la realización del código a nivel de usuario. Estos procesos que corren a nivel del espacio de usuario pueden ser supervisados y mantener políticas restrictivas, pero por supuesto pagando el precio de una gran degradación en la performance. Con respecto a la robustez del proceso frente al sistema, los efectos de imperfecciones y de fallas en herramientas ejecutadas a nivel del kernel traen consecuencias mucho más serias que si se corrieran a nivel del espacio de usuario.

Los sistemas escritos a nivel del modo de usuario utilizan los llamados system calls que provee el kernel del SO para realizar diferentes operaciones como ser escribir o recibir los paquetes de la red, de los cuales se dará una breve explicación y sus inconvenientes en la performance más adelante.

A continuación, resumimos una serie de ventajas y desventajas de por qué deberíamos o no implementar un sistema a nivel de kernel o de usuario.

- Cuando un programa que trabaja a nivel de usuario realiza una system-call, existe lo que llamamos un overhead asociado a la transición de datos desde el user-space al kernel-space. Si programamos todas estas funcionalidades directamente en el kernel, podemos ahorrarnos estos overhead y ganar performace en el sistema.
- Los datos correspondientes a una aplicación que envía o recibe paquetes, son copiados del modo de usuario al kernel y viceversa. Implementando las aplicaciones de red directamente desde el kernel, es posible reducir dicho overhead e incrementar de esta forma la eficiencia al no estar obligados de copiar estos datos al modo usuario.
- Cuando un programa ejecuta una llamada al sistema (system-call) el kernel debe realizar una serie de tareas, entre las cuales se encuentran guardar el contenido de determinados registros, realizar cambios para afrontar los límites de espacio y realizar comprobación de errores en los parámetros de la función que realiza la llamada al sistema, todo esto en adición a la tarea particular que debe cumplir el kernel para atender la llamada del proceso ejecutándose en modo de usuario.

En distintas investigaciones a nivel de alta performace computacional, recomiendan que para alcanzar un alto rendimiento de transferencia a altas velocidades es necesario que estas aplicaciones sean implementadas a nivel del Kernel (ejemplo de estos son sistemas de monitorización de red, NIDS e IPS) [8] [9] [10].

Por otro lado, no todo es color de rosas, y trabajar a nivel de Kernel puede traer serios dolores de cabeza. A continuación se dan algunas reseñas de porqué no siempre es conveniente trabajar a tan bajo nivel.

- La seguridad es una de las preocupaciones principales cuando se trabaja a nivel del kernel, una gran variedad de aplicaciones diseñadas para trabajar en modo de usuario no son recomendadas para que trabajen directamente a nivel del Kernel. Consecuentemente, hay que tener un cuidado cuando se diseñan aplicaciones que trabajen a bajo nivel en cuanto a seguridad se refiere. Por ejemplo, la lectura y escritura de archivos dentro del kernel es usualmente una mala idea, aunque muchas aplicaciones requieren de archivos de entrada y salida.
- Grandes aplicaciones no pueden ser implementadas en el kernel debido a las restricciones memoria y como el kernel la administra.
- Problemas en la ejecución de la herramienta trabajando a nivel del kernel-space puede provocar que el sistema deje de funcionar.

En particular cuando hablamos de aplicaciones de red, los paquetes son transferidos desde la interfaz de hardware hacia el kernel del sistema operativo. La mayoría de las aplicaciones generalmente corren en el user-space, con lo cual se debe realizar necesariamente una transición entre el user y el kernel space para poder tener acceso a los datos. El overhead involucrado en esta transición puede resultar significante, aumentando tanto la latencia del mensaje como también la carga en el procesamiento del CPU. Como se mencionó, cuando una aplicación o proceso realiza una función send() o receive() sobre un paquete, se desencadenan una serie de procesos en el kernel. Durante cada receive() (o send()), el sistema operativo debe cambiar hacia el modo kernel, acceder a un socket, copiar los datos hacia los buffer del kernel reservados para esa aplicación, realizar determinadas actividades relacionadas con ese protocolo (como ser revisar la integridad de los encabezados, el cheksum, ruteo, etc.) y finalmente colocar el paquete completo en una cola para ser transmitido.

Por la suma de todas estas razones, en conjunto con la forma en que será insertada nuestra herramienta en la topología de la red, es que nos pareció necesario buscar o implementar la parte del análisis e identificación de los datos a nivel de Kernel. Las principales razones por las cuales consideramos esto como la mejor opción fueron las siguientes:

- Nuestra herramienta no estaría insertada en el medio del tráfico, con lo cual un posible problema en donde el sistema operativo deje de funcionar debido a una falla de la herramienta, no estaría teniendo ningún tipo de efecto en el transcurso de los datos hacia su destino.
- Realizar el análisis de los datos a nivel de Kernel produciría una menor sobrecarga de procesamiento en el CPU, logrando de esta manera una mejor performance en volumen de tráfico analizado.

- Como observamos los datos de la red de una forma pasiva, sin tener interacción con los mismos, no debemos preocuparnos a la hora de tener una buena seguridad y políticas de restricción desde la interfaz donde nos llega el tráfico a analizar.

Si bien se hizo un estudio de una variedad de herramienta que realizan en mayor o menor medida las tareas que necesitamos, no nos restringimos a aquellas que únicamente trabajaban a nivel del Kernel-space. Aunque si consideramos el nivel de performance que alcanzaban y cuanto de este análisis era realizado a nivel de usuario o del kernel, ya que era un factor importante en la cantidad de tráfico que podíamos alcanzar a analizar.

4.3. Herramientas estudiadas

4.3.1. Snort

Snort es un sniffer de paquetes con el fin de prevenir (NIPS Network Intrusion Prevention System) y detectar intrusos en una red (NIDS Network Intrusion Detection System) [11].

Está disponible bajo licencia GPL, gratuito y funciona bajo plataformas Windows y UNIX/Linux. Dispone de una gran cantidad de filtros y patrones ya predefinidos, así como actualizaciones constantes ante casos de ataques, barridos o vulnerabilidades que vayan siendo detectadas a través de los distintos boletines de seguridad.

Es capaz de analizar el tráfico en tiempo real y puede realizar análisis de protocolos buscando en el contenido del payload de los paquetes. Es usado habitualmente para bloquear o detectar gran variedad de ataques como buffer overflows, escaneos de puertos, ataques a aplicaciones Web, intentos de OS fingerprinting, etc. Implementa un motor de detección de ataques y barrido de puertos que permite registrar, alertar y responder ante cualquier anomalía previamente definida.

Puede funcionar de las siguientes maneras:

- Sniffer: en este modo lee los paquetes de la red y los muestra en una terminal en la pantalla.
- Packet Logger: en este modo se puede registrar el flujo de información guardándolo en el disco para su posterior análisis (off_line).

- Detección de intrusos (NIDS): en este modo se puede analizar el tráfico de red usando reglas definidas por el usuario que macheen los diferentes protocolos y así tomar diferentes acciones sobre los mismos. Cuando un paquete coincide con algún patrón establecido en las reglas de configuración, éste genera un log en el sistema. Así se sabe cuándo, de dónde y cómo se produjo el ataque.

En el modo NIDS se emplea el archivo de configuración snort.conf el cual contiene las reglas definidas que se quieren emplear, o sea que a cada paquete se le aplicarán las reglas definidas en este archivo para saber qué acción se debe tomar.

También está disponible el modo online que obtiene los paquetes desde Iptables en lugar de libpcap. En este modo los paquetes son pasados al user-space y puestos en una cola utilizando la función nf_queue de Netfilter (esto se explicará más adelante). Se puede definir por ejemplo, que sólo se pasen a Snort los paquetes que tengan como destino algún equipo de la red local.

Snort funciona mediante la utilización de un lenguaje basado en reglas que combina los beneficios de la inspección de la firma, la inspección de protocolo, y la inspección basada en anomalías. Tiene una base de datos de ataques que se está actualizando constantemente y a la cual se puede añadir o actualizar a través de Internet.

Las firmas han sido diseñadas específicamente para detectar ataques conocidos, ya que contienen signos distintivos, tales como cadenas de string conocidos o cualquier otro marcado único que puede o no estar relacionado con la explotación de una vulnerabilidad en la realidad. A diferencia de las reglas la vulnerabilidad sí está relacionada con la realidad.

El equipo de soporte de SNORT es el encargado del desarrollo de reglas, llamadas reglas **VRT** (Sourcefire Vulnerability Research Team) que se catalogan en tres tipos:

1. Reglas para suscriptores. Es decir, de pago. Permanentemente actualizado.
2. Reglas para usuarios registrados (basta con crearse una cuenta).
3. Reglas disponibles para usuarios no registrados. Muy desactualizadas.

Los usuarios pueden crear reglas basadas en las características de los nuevos ataques de red y enviarlas a la lista de correo de Snort, para que así todos los usuarios se puedan beneficiar. Esta ética de comunidad y compartir ha convertido a Snort en uno de los IDSes más populares, actualizados y robustos.

Sin embargo, Snort es una herramienta que a pleno funcionamiento consume muchos recursos, no sólo porque trabaja a nivel de usuario sino también que constantemente tiene que actualizar sus reglas para no quedar obsoleto. Por otro lado, esta herramienta esta más enfocada a la detección de intrusos no tanto a la clasificación del tráfico de red como es nuestro objetivo [12].
Por estos motivos fue que se decidió descartar esta herramienta.

4.3.2. Hippie

HIPPIE (Hi-Performance Protocol Identification Engine) es una extensión del kernel de Linux con el fin de clasificar en tiempo lo real los paquetes de un tráfico de red, ya sea desde una interfaz de red en funcionamiento o de alguna otra fuente [13].

Realiza un examen inteligente de paquetes de red y además permite incorporar otros tipos de software para tomar decisiones sobre los paquetes basados en la información que puede proporcionar hippie. A diferencia de otros sistemas de clasificación de tráfico cuyos métodos consisten en buscar coincidencias de byte o de String, permite aplicar algoritmos complejos para la inspección de paquetes.

Su objetivo es brindar un sistema capaz de examinar los paquetes y proporcionar información sobre lo que hay dentro de los mismos en el contexto de los protocolos de Internet que se utilizan. Aunque el propósito de Hippie no es determinar qué hacer con esa información, deja varias posibilidades para poder aprovechar la misma, como por ejemplo:

- **Analizador pasivo de tráfico:** En este caso se instala HiPPIE en el kernel de un sistema Linux y se hace que el tráfico que desea ser analizado pase por una interfaz. Gracias a la ayuda de otras herramientas se puede llegar a obtener una imagen pasivo del uso de esta red, visualizando qué protocolos componen al tráfico de esta red.

- **Filtrado de paquetes:** Creando un sistema Linux con HIPPIE ya sea como bridge o como dispositivo de enrutamiento de tráfico y obligando al tráfico que pase a través de ella, utilizando por ejemplo Netfilter / IPTables para filtrar el tráfico. Se puede aprovechar la capacidad de reconocimiento de hippie para descartar o limitar cierto tipo de tráfico. Esto serviría para filtrar ciertos protocolos, ya sea por políticas administrativas, usos anormales, o para poder dar prioridad a determinados protocolos que hayan sido etiquetados, etc. Hippie es capaz de reconocer los siguientes protocolos: aim, ares, bittorent, dirconn, dns, edonkey, esp, fasttrack, filetrpia, ftp, gnutella, gre, h323, http, icmp, icq, imap, irs, msnim, nntp, novellcp, pop3, pplive, rdp, rstp, sip, Skype, smtp, ssh, sst, strom, winmx, wmedia, xunki y yahooim. Aunque, en alguno de los protocolos indicados anteriormente no es posible detectar correctamente todos los flujos correspondientes.

La clasificación se realiza utilizando firmas que tiene integradas junto al kernel. Estas firmas no están basadas únicamente en expresiones regulares, se pueden definir distintas parámetros a buscar, un string, una secuencia de bytes o inclusive el largo de los paquetes. La inserción de nuevos protocolos o mejoras en estas firmas no se pueda llevar a cabo de forma ágil ni de manera sencilla. Por lo tanto, representa una importante restricción para nosotros y no cumple con uno de los principios de diseño definidos al comienzo del proyecto y por esta razón no la utilizaremos.

4.3.3. Ourmon

Ourmon es una herramienta open-source orientada principalmente a la supervisión y a la detección de anomalías dentro de una red. Siendo utilizado en la mayoría de los casos conectado a un switch con mirror, teniendo acceso de esta manera a todos los paquetes que circulan en la red. A diferencia de otros sistemas, Ourmon utiliza principalmente para analizar, información de los host y no se basa tanto en la información de los flujos de datos sobre la red. Este sistema también es capaz de revisar en el payload de la capa 7, y fue en este punto donde se despertó nuestro interés sobre esta herramienta [14] [15].

La arquitectura de Ourmon consiste en dos partes fundamentales, la llamada front-end part, la cual es motor de análisis que lee los paquetes del kernel del buffer donde se guardan los paquetes levantados de la interfaz Ethernet, y luego pasan por una serie de filtros. Cada filtro, cuenta los bytes o paquetes y los almacena periódicamente en un archivo para ser pasados la segunda etapa del sistema, llamada back-end. Esta etapa es la encargada de desplegar en forma gráfica los datos entregados por el front-end y de crear las gráficas a través de herramientas RRDtool.

Ourmon es capaz de reconocer qué aplicaciones corren sobre el tráfico de una red, sin embargo tiene varias maneras de hacer este reconocimiento. Uno de ellos es actuando sobre la información de las capa de transporte como por ejemplo contabilizando la cantidad de banderas syn y fin que aparecen en los paquetes que envía determinado host. De esta manera puede reconocer si el tráfico de un host está asociado por ejemploy cual a una aplicación P2P, a una conexión normal de Internet o un comportamiento malicioso o anormal del host (por ej. un worm o un virus). Este método de captura no nos interesa en lo más mínimo ya que utiliza capas inferiores para el reconocimiento de aplicaciones y esta orientado sobre todo al reconocimiento de anomalías o comportamientos extraños en los Host. El otro método utilizado para el reconocimiento de aplicaciones sobre los datos, fue introducido en las nuevas versiones de Ourmon. Este método se basa en PCRE las cuales son expresiones regulares escritas en perl utilizadas para machear sobre el payload del tráfico. Estas expresiones buscan los patrones en la capa de aplicación

de los paquetes y marcan los paquetes con un tag asociado al protocolo que identifiquen. Para realizar esto, analiza una determinada cantidad de paquetes (o una determinada cantidad de bytes) enviado por un host no en forma individual, como haría en el caso de estar buscando anomalías, sino que busca estos patrones en una serie de paquetes (o stream de bytes) macheando expresiones regulares sobre el payload de los mismos.

Estas expresiones regulares están basadas en los patrones creados por el equipo de desarrollo de L7-Filter, y lo que buscan son determinados comportamientos característicos de los flujos de datos de una aplicación. El inconveniente principal del sistema Ourmon, es que la cantidad de paquetes o de bytes a analizar están asociados a lo que envía un determinado host (los flujos son identificados por la IP de origen), con lo cual el reconocimiento que se hace es para ese host en particular y no sobre el flujo total de datos que circulan por la red. De esta forma, Ourmon no caracteriza el tipo de aplicaciones sobre la red sino que únicamente nos dice que uno de esos host está traficando una determinada aplicación y cuál es el ancho de banda consumido por ese host.

4.3.4. Ntop

Al comienzo de nuestro proyecto, y luego de un análisis inicial de las herramientas encontradas que se ajustaban a las necesidades del mismo, NTOP fue una de las dos herramientas que mejor se ajustaba y que también abarcaba la gran mayoría de los requisitos propuestos. Por dicha razón, fue que se estudió de manera más profunda su funcionamiento, las distintas capacidades con las que cuenta y la posibilidad de modificar el código fuente para cumplir con nuestras necesidades [16]. A continuación se dan detalles técnicos del funcionamiento de NTOP.

Ntop es una herramienta open-source escrita sobre el lenguaje C bajo licencia publica GNU. Esto significa que el código de NTOP es libre de modificar y el mismo se encuentra disponible en Internet. Los autores originales diseñaron la versión original adecuándose a sus necesidades y luego la misma fue acomodándose y adecuándose, influenciados por programadores externos al grupo inicial, para cumplir con las siguientes metas:

- Portabilidad para la mayoría de las plataformas de UNIX y non-UNIX.
- Simplicidad y eficiencia en aplicaciones dentro del Kernel consumiendo pocos recursos de CPU y memoria.
- Capacidad de presentar la información disponible por medio de caracteres o a través de una interfaz Web.
- El análisis de la salida de la red debe ser rica en contenido y fácil de leer.

El diseño de Ntop sigue la filosofía de UNIX, es decir, las aplicaciones no necesariamente tienen que ser grandes estructuras de código, sino que preferentemente debería estar formado por pequeñas partes e independientes entre sí, cooperando en conjunto para alcanzar metas comunes. El kernel es el responsable de manejar las tareas eficientemente y de proveer a los distintos plugins las facilidades necesarias para utilizar los servicios de este. De esta forma se mantiene la complejidad del plugin baja y se puede focalizar principalmente en la funcionalidad de dicho plugin, mientras que el kernel es el encargado de llevar a cabo las demás funciones. Por medio de estos plugins, el usuario puede activar sólo los que le resulten necesario, dependiendo de distintas situaciones a las que nos enfrentamos. A su vez, muchos de los servicios básicos son proporcionados directamente por el kernel de Ntop, simplificando la implementación de nuevos plugins.

La arquitectura de NTOP presenta el siguiente esquema: un motor que hace de sniffer, el cual recolecta los paquetes de la red y luego se la pasa al analizador de paquetes, el cual procesa esta información. La siguiente etapa recoge esta información y despliega lo pedido de forma apropiada.

Sistema de Sniffer.

Bajo UNIX NTOP utiliza la librería libpcap como interfaz para la captura de paquetes. El filtrado de dichos paquetes se realiza por medio de filtros BPF incluidos en la librería de libpcap. Estos filtros son especificados utilizando simples expresiones similares a las aceptadas por tcpdump.

Las librerías utilizadas para la captura de paquetes tienen buffers internos pequeños, los cuales impiden que las aplicaciones sean capaces de manejar las grandes ráfagas de tráfico. A fin de superar este problema y reducir la pérdida de paquetes, los buffers de Ntop capturan los paquetes. Esto permite que el módulo analizador de paquetes quede desasociado del sniffer, impidiendo de esta forma que se pierdan una gran cantidad de paquetes debido a estas ráfagas.

Analizador de Paquetes.

En esta etapa, dicho proceso analiza un paquete a la vez. Los encabezados de los paquetes son analizados en función de la interfaz de red que se este utilizando (ya que los encabezados son distintos dependiendo de la interfaz en la capa de enlace, como ser Token Ring o cualquiera no Ethernet). La información de los hosts es almacenada en grandes tablas Hash, en donde se almacenan diferentes contadores que mantienen el rastro de los datos enviados y recibidos por distintos hosts. Debido a que es prácticamente imposible saber cual será la cantidad de hosts analizados por Ntop, se vuelve imposible tener una tabla hash lo suficientemente

grande para almacenar a todos ellos. Por esta razón, de ser necesario, Ntop elimina los host de la tabla para poder manejar de forma eficiente los recursos limitados y mantener la performance del sistema.

Ntop es capaz de permitir al usuario especificar diferentes flujos de datos, esto es, un stream de paquetes que machea una regla especificada por el usuario. Estas reglas son especificadas mediante las Expresiones BPF. De esta forma, Ntop permite especificar flujos de datos que tengan un interés particular para el usuario.

Reporte de los Datos.

La versión de Ntop estudiada tiene la posibilidad de presentar los datos de dos maneras posibles.

- Modo interactivo: Ntop corre sobre una terminal basada en caracteres y el usuario puede interactuar con esta terminal ingresando comandos mediante el teclado.
- Modo Web: Ntop actúa como un servidor http permitiendo a los usuarios remotos analizar las estadísticas del tráfico mediante un Web browser.

Ntop fue sido diseñado para ser independiente de la manera en que los reportes de tráfico son mostrados. Independizarse de la manera en que estos reportes son creados da la facilidad de que si se desea presentar estos datos de una nueva manera, este módulo de Ntop es el único que se vería afectado, dejando al resto intacto.

PLUGINS

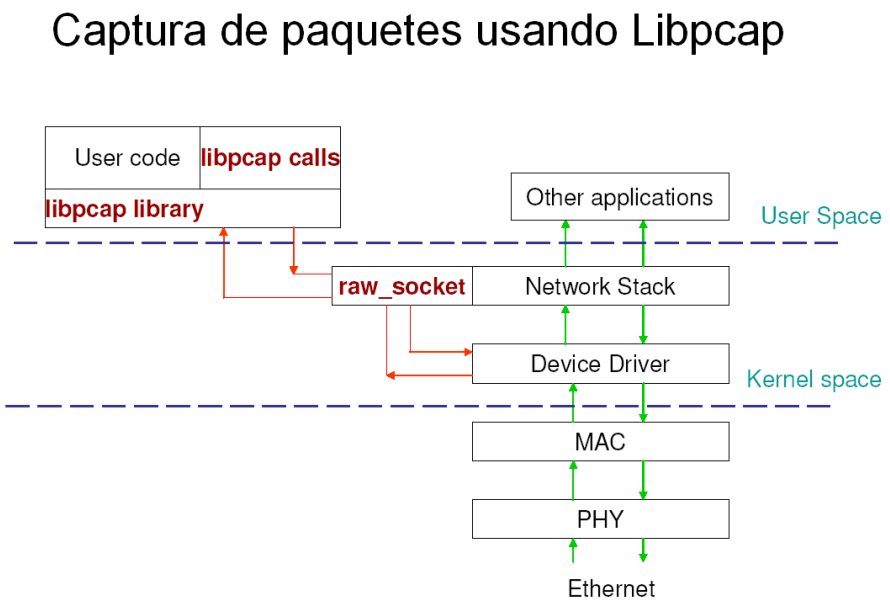
Estos plugins son librerías compartidas con un puntero de entrada bien definido y almacenado en un directorio específico. Al inicio, Ntop lista los plugins almacenados y los carga secuencialmente en orden alfabético. Los usuarios pueden utilizar estos plugins para extender el kernel de Ntop, utilizándolos para distintas funciones e implementando contadores de flujos de tráfico más avanzados los cuales realizan operaciones adicionales además de las básicas.

Librería Libpcap

Libpcap es un interfaz para la captura de paquetes a nivel del user-space. Éste provee un Framework portable para un gran número de SO's utilizado principalmente para el monitoreo de la red. Una gran cantidad de aplicaciones que cumplen con funciones de recolección de estadísticas de red, monitoreo de seguridad, debugging de problemas en la red, etc., utilizan esta librería para poder levantar los paquetes y manipularlos o simplemente para analizarlos. Para este caso en especial

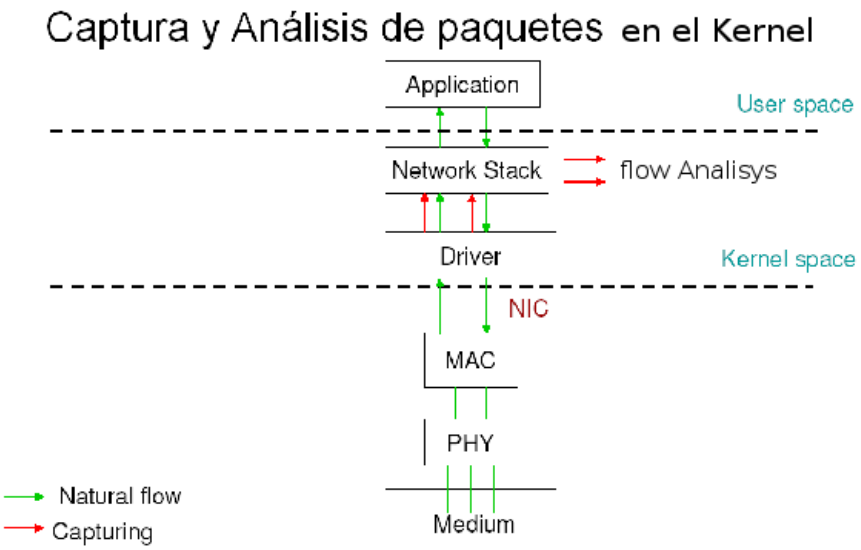
libpcap le provee una API a Ntop, el cual utiliza esta librería para levantar los paquetes hacia sus buffers ubicados a nivel del user-space para luego analizarlos.

Para este sistema, el proceso que se lleva a cabo para levantar un paquete desde la red es el siguiente: La tarjeta de red (NIC) lee el paquete y si está destinado a ella (dirección MAC correspondiente o cualquier paquete en caso que se esté trabajando en modo promiscuo) lo almacena en un pequeño buffer para luego interrumpir al procesador. Luego el paquete es tomado por el kernel de SO y almacenado en un buffer interno. La librería Libpcap lee este buffer y filtra los paquetes (en caso de que esto sea requerido por medio de los filtros BPF), luego pasa los paquetes al sistema Ntop o cualquier aplicación que lo requiere y esté escuchando a nivel usuario. Durante este proceso de lectura y escritura tanto en los buffers del kernel y de la tarjeta como en los buffers de Ntop y de Libpcap, es probable que se puedan producir pérdidas de paquetes debido a un procesamiento en exceso del CPU, no pudiendo atender a todos los procesos que lo requieran. A su vez, como vimos en los problemas de pasaje de información desde el modo kernel al modo de usuario, se produce un procesamiento extra en el CPU por las reiteradas llamadas al sistema para el pasaje de información del kernel al modo de usuario.



Para nuestro caso, nos es imposible poder evitar las pérdidas de paquetes a nivel de la NIC y del Kernel sin realizar un mejoramiento del Hardware (principalmente un mejor procesador, pero también es necesario más recursos, y una tarjeta de red con mejor rendimiento). Sin embargo podríamos elegir un sistema en donde no sea necesario pasar la información del paquete hacia el user-space y en su lu-

gar realizar el análisis del paquete directamente en el kernel, donde lograríamos en teoría una mejor performance.



A pesar de lo anterior, en un principio optamos por intentar introducirnos un poco más en esta herramienta, ya que contaba con una buena cantidad de patrones de análisis para la capa de aplicación y tenía una interfaz gráfica diseñada para ser desplegada en una Web que mostraba en una forma sumamente amigable el tráfico de la red en tiempo real. Nuestra idea era introducirnos en el código de Ntop para poder eliminar una cantidad de funciones y utilidades que éste prestaba pero que no eran de interés para el proyecto, y ver también la posibilidad de desplegar tiempos de captura más largo, ya que Ntop sólo mostraba datos guardados de los últimos 40 días.

El problema a partir de aquí fue la escasa documentación existente y también que no existía un diagrama de flujo para poder introducirnos de una forma adecuada en el código. El otro inconveniente fue nuestra falta de experiencia a nivel de programación en C, en donde nos vimos muy limitados a la hora de introducirnos en el código. Otra gran limitación que tuvimos fue que una gran parte del código que queríamos analizar (por ejemplo el hecho de que se contabilizaran paquetes por puertos o hosts, cosa que no nos interesaba) era muy complicado de erradicar, ya que el código de estas funciones se encontraba en el propio kernel de Ntop, y cambiando algo en este punto se podía perjudicar el funcionamiento del resto del sistema.

Por estas limitantes a la hora de enfrentarnos al código de Ntop, fue que decidimos descartar esta herramienta, sin embargo tomamos valiosas ideas sobre la arquitectura de Ntop, las cuales nos sirvieron para poder desarrollar nuestra herramienta. Sumado al hecho que debido a la manera en que trabaja Ntop (capturando con libpcap), presenta inconvenientes a la hora de analizar el tráfico cuando ocurren grandes ráfagas, en donde la pérdida de paquetes se vuelve considerable. Aunque como vimos Ntop solucionó este problema mediante la aplicación de filtros propios y desasociando los "módulos" de captura y análisis (realizando este último a nivel de usuario), las limitantes a nivel de performance son de un máximo de 200 Mb/s, logrando en este punto sobrecargar el CPU y comenzando a perder paquetes en forma considerable. El otro punto que nos motivó a cambiar de herramienta y optar en particular por L7-Filter (herramienta que será explicada en detalle más adelante), fue que Ntop utilizaba los mismos patrones de análisis para el reconocimiento del tráfico. Sin embargo Ntop realiza el análisis a nivel de usuario, mientras que L7-Filter lo puede hacer directamente en el Kernel. Por esta razón estaríamos ganando en performance y podríamos llegar a mayores volúmenes de tráfico analizado correctamente. Por otro lado, si optábamos por utilizar L7-Filter, debíamos encontrar una manera de poder guardar los datos y luego mostrarlos en una forma amigable al administrador. Estos inconvenientes son desarrollados en capítulos posteriores, en donde también presentamos la solución que encontramos para cada uno de ellos.

Hasta aquí hemos estudiado una variedad de herramientas desarrolladas para diferentes metas, destacando de cada una de ellas, distintas ventajas y desventajas, adaptándose en mayor o menor grado a los principios de diseños elegidos. Si bien algunas de ellas realizan las tareas que nos propusimos, identificando el tráfico por medio de la búsqueda de patrones en la capa de aplicación, también realizan otras tareas, las cuales en nuestro caso, no nos son de gran utilidad. Es más, estas tareas adicionales, entorpecen la eficiencia del sistema.

Al comienzo del proyecto se intentó eliminar estas funciones básicas, con el fin de adaptar la herramienta a los requerimientos de diseño pero se nos presentaron grandes dificultades, ya que muchas de ellas estaban incluidas en el propio kernel de la herramienta, y dada la poca documentación existente y el nivel de código al que debíamos introducirnos, nos vimos imposibilitados en modificar el sistema en el corto plazo.

De todas maneras, por más que no seleccionamos ninguna de ellas para trabajar, pudimos sacar ideas para desarrollar la nuestra, basándonos en algunos programas más específicos e implementándolos en una forma modular. La gran ventaja de la implementación de un sistema modular, es que ante la necesidad de integrar una nueva funcionalidad o cambiar un módulo por uno más eficiente, no sea necesario modificar todo el sistema, sino que se remplaza dicho módulo manteniendo el resto intacto.

5. Descripción del sistema modular

Llegado a este punto, nos vimos en la necesidad de definir que funciones deberían cumplir los diferentes módulos y luego la manera de implementar cada uno de ellos. Dada la experiencia de estudios previos, decidimos componer nuestro sistema con tres módulos principales, donde cada uno tendrá una función bien definida pero interactuando todos en forma conjunta.
Estos módulos seran:

1. Captura y análisis del tráfico.
2. Recolección de datos y almacenamiento en base de datos.
3. Representación de los datos en forma gráfica.

A continuación de describen brevemente las funcionalidades de cada uno de los módulos y luego se detallarán por separado.

Para la captura e identificación del tráfico contamos con un sistema que es capaz de escuchar todo el tráfico que pasa por su interfaz Ethernet. Dicho módulo debió ser capaz de levantar el tráfico destinado a otros hosts para luego poder extraer el payload de los paquetes, examinarlo y aplicar los distintos patrones para su identificación. Este módulo trabaja como un contador de paquetes y/o datos para los protocolos encontrados e identifica los diferentes flujos que circulan por la red. Una vez realizada la identificación y el conteo, los paquetes deben ser descartados para mantener la privacidad de los datos. No es de nuestro interés ver la información, sino simplemente contabilizar cómo se reparte el tráfico.

Una vez realizada la captura, identificación y actualización del conteo del tráfico para los distintos protocolos, tenemos que recolectar esta información para guardarla en una base de datos para su posterior estudio. Es aquí que entra en funcionamiento el segundo módulo de nuestro sistema. Éste debe estar consultando periódicamente la actualización de los contadores del módulo de recolección y análisis, para luego guardar toda esa información. Debíamos mantener una base de datos para cada contador de los diferentes protocolos, para luego consultarla y así desplegar los datos en distintos períodos de tiempo, como ser horas, días, meses, etc. Cuando se diseñó este módulo se tuvo especial cuidado a la hora de elegir el tipo de base de datos a utilizar, ya que la misma no podía crecer de forma indefinida y debía ser lo más eficiente posible para no consumir una gran cantidad del recursos en el sistema.

Hasta aquí hemos realizado todo el trabajo de análisis y almacenamiento, pero no teníamos la posibilidad de ver los resultados de una forma amigable. Por esta razón, se implementó el tercer módulo del sistema, que se basa en la representación en forma gráfica de los datos adquiridos. Esto permite trabajar en una forma sencilla y facilita a los administradores definir políticas sobre el uso de la red o simplemente tener en forma gráfica el porcentaje de los diferentes protocolos sobre el total del tráfico.

A continuación se detallan en mayor profundidad los 3 módulos implementados y dentro de cada uno de ellos, las herramientas utilizadas para lograr sus objetivos.

5.1. Módulo de captura y análisis de datos

Para la captura y análisis de datos optamos por trabajar a bajo nivel en el sistema operativo, para poder utilizar al máximo los recursos del sistema. Es por esta razón que decidimos implementar este módulo en su totalidad, a nivel del Kernelspace.

En esta etapa tuvimos varios inconvenientes a resolver para poder cumplir con los requisitos establecidos en los principios de diseño. Entre ellos decidir cómo íbamos a capturar y cómo debíamos configurar las distintas herramientas para poder trabajar de esa manera. A continuación se nombran las herramientas utilizadas y la función de cada una de ellas.

Para poder levantar los paquetes de la NIC decidimos trabajar con el Framework Netfilter [17] , el cual ya viene integrado en las nuevas versiones del kernel de Linux y está bien testeado. Para poder seguir las conexiones e identificar los distintos flujos de la red, utilizamos connection tracking (conntrack), que es una herramienta de Netfilter. Una vez que capturamos los datos debíamos analizar el payload de los paquetes de un flujo determinado y poder asociarlo con algún protocolo. Para lograr esto último, parcheamos Netfilter con un módulo especial, el cual trabaja también a nivel del Kernel. Este módulo se llama L7-Filter [18] y busca únicamente para machear, el payload de los paquetes y lo compara con distintos patrones, en caso de hallar a qué protocolo corresponde, marca la conexión en las tablas de conntrack de forma de asociar los futuros paquetes pertenecientes a este flujo con este protocolo.

Para poder escribir las reglas de los distintos patrones y poder integrarlas al kernel a través del Framework de Netfilter, se utilizaron las Iptables. Iptables es usado para filtrar, contabilizar, o hacer calidad de servicio de los datos de una red, pero no únicamente a nivel de IP, TCP, sino que también es capaz de analizar encabezados de capas superiores o el payload de los paquetes, como en nuestro caso, gracias a L7-Filter.

Con estas herramientas trabajando en conjunto, es que pudimos capturar paquetes, mantener un seguimiento de las conexiones y los flujos, analizar el payload y hacer un conteo de los mismos. A continuación se dan detalles técnicos más profundos y la forma de funcionamiento de cada una de estas herramientas, para luego explicar cómo se debe configurar cada una de ellas.

5.1.1. Netfilter

Como hemos mencionado, los Firewall tradicionales filtran tráfico basados en el host y en la dirección del puerto principalmente. Cuando los paquetes llegan al firewall, se examinan las cabeceras que contienen el protocolo de las capas inferiores para determinar el origen y el destino del host y las direcciones de los puertos. Una vez que se extraen las direcciones, el firewall consulta una lista de reglas que describe el tráfico que se acepta o se rechaza, y maneja el paquete en consecuencia. Comunmente, esta característica se utiliza para la construcción de conjuntos de reglas que aceptan el tráfico de máquinas específicas, pero rechazan los demás paquetes. Con una regla por defecto diseñada para rechazar todos los paquetes, el administrador del sistema puede construir reglas para aceptar el tráfico de todas las máquinas en su dominio. En tal caso, todo el tráfico procedente de fuera de la intranet no coincidirá con sus reglas, por lo que se eliminará de acuerdo a la regla por defecto.

Netfilter es una herramienta que está integrada en el Kernel de Linux, y debido a que Linux es un entorno operativo libre, está comúnmente disponible en los servidores. Actualmente las políticas de filtrado basadas únicamente en los encabezados de los paquetes están obsoletos. En estos días, los llamados statefull firewall proveen mecanismos avanzados que permiten a los administradores de una red definir políticas mas inteligentes. Intentaremos dar una descripción detallada del sistema que provee el proyecto Netfilter para lograr lo anterior.

Netfilter Framework comprende una serie de Hooks a lo largo del stack de protocolos de red de Linux. Con estos hooks podemos registrar módulos en el kernel que hagan manipulación dentro de las diferentes etapas. Netfilter inserta cinco hooks en la pila de red de Linux para llevar a cabo el manejo de paquetes en diferentes etapas, que son las siguientes:

- PREROUTING: Todos los paquetes, sin excepciones, pasan a través de este hook, el cual se encuentra antes de tomar la decisión de enrutamiento, y después de que se realicen una serie de chequeos por parte del kernel para verificar la integridad de los encabezados IP y TCP.
  Address Translation (NAPT) y redirecciones, es decir, la traducción de red de destino (DNAT), se aplican en este hook.

- LOCAL INPUT: Todos los paquetes que van a la máquina local llegan a este hook. Este es el último hook en el camino de entrada para el equipo local de tráfico.

- FORWARD: Los paquetes no van a la máquina local (por ejemplo, los paquetes que se rutean a través del firewall) llegan a este hook.

- LOCAL OUTPUT: Este es el primer hook en la ruta de acceso de paquetes salientes. Los paquetes que salen de la máquina local siempre dan en este hook.

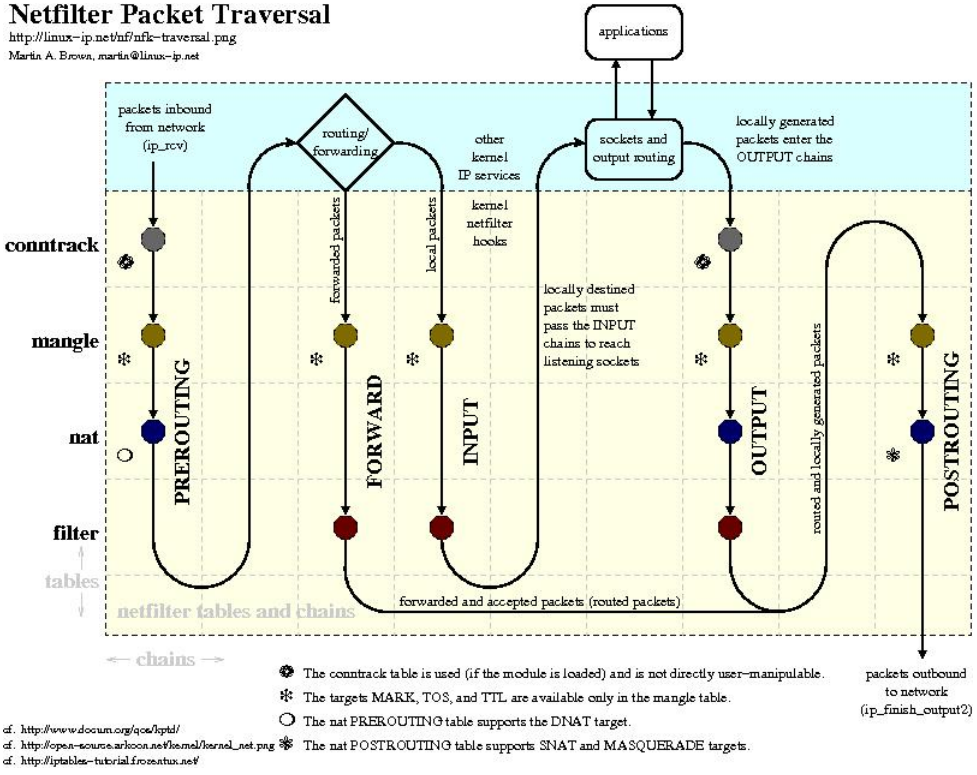
- POSTROUTING: Este hook se lleva a cabo después de la decisión de enrutamiento. Source Network Address Translation (SNAT) está registrado para éste hook. Todos los paquetes que salen de la maquina local llegan a este hook.

Por esta razón podemos modelar tres tipos de flujos de datos dependiendo del destino:

1. Tráfico pasando a través del firewall. En otras palabras, tráfico el cual no está destinado hacia la máquina local. Dicho tráfico sigue el siguiente camino:
   PREROUTING FORWARD POSTROUTING.
2. Tráfico entrante al firewall, por ejemplo tráfico destinado a procesos locales de la máquina donde corre dicho firewall. Para dicho camino seguimos:
   PREROUTING INPUT.
3. Tráfico saliente de la máquina local: dicho camino sería:
   OUTPUT POSTROUTING.

Se pueden registrar lo que se llama callbacks function hacia un hook en particular, los cuales están definidos en las cabeceras de Netfilter. Las funciones callbacks pueden devolver una gran variedad de valores los cuales serán interpretados por el Framework de Netfilter de la siguiente manera:

- ACCEPT: Permite al paquete seguir viajando a través del stack.
- DROP: Descarta al paquete.
- QUEUE: Pasa el paquete al espacio de usuario a través de la instalación de nf_queue. Un programa de espacio de usuario va a hacer para nosotros el manejo de paquetes.
- STOLEN: Se queda con el paquete hasta que algo ocurre, de manera que temporalmente no sigue viaje a través del stack. Esto es usualmente usado para recoger los paquetes IP desfragmentados.
- REPEAT: Fuerza al paquete para que vuelva a entrar al hook.



Resumiendo, el Framework provee un método para el registro de funciones de callbacks que hacen algún tipo de manipulación en el paquete en algunos de los distintos estados detallados con anterioridad. El valor de retorno es usado por el Framework, el cual aplicará las políticas basándose en estos veredictos.

El cometido de un módulo para macheo es inspeccionar cada paquete recibido y decidir cuándo machea o no, de acuerdo a los criterios establecidos, donde estos criterios pueden ser cualquier cosa que se nos ocurra. Los criterios más obvios y más usados en los firewall convencionales es machear a través de las direcciones de origen y destino, así también como por puertos de destino y origen. Para Netfilter/Iptables también existen casos de módulos más avanzados y que machean reglas más específicas como ser por el número de conexiones, machear por system-time trabajando con reglas para que nos permitan limitar determinadas conexiones en algunos horarios del día.

En la gran mayoría de los casos, los módulos no modifican información del paquete. Modificar cualquier parte de los paquetes debería hacerse únicamente en lo que llamamos TARGETS los cuales son definidos más tarde, pero por supuesto hay excepciones.

5.1.2. Connection Tracking

La conexión y el seguimiento durante la inspección de estado

Conection tracking es otro pequeño ladrillo en la cima del Framework Netfilter para poder crear un firewall de manera statefull, basándose no sólo en la información del encabezado del paquete. A lo largo de los años, las aplicaciones convencionales que únicamente revisaban los parámetros de los encabezados de cada paquete han quedado obsoletas ya que son insuficientes para brindar protección frente a ataques especializados, como ser por ejemplo los llamados ataques de denial-of-services.

Básicamente, las connection tracking almacenan información acerca de los estados de las conexiones en una estructura de memoria que contiene la IP de origen y destino, los pares de puertos utilizados, tipo de protocolo, estado y time out. Con esta información es con la cual podemos identificar los diferentes flujos de datos que circulan por nuestra red, y de esta manera se pueden crear reglas de filtrado más inteligentes no sólo sobre los paquetes en sí, sino también aplicarlo a todos los paquetes del flujo. Otra de las ventajas es que permite aplicar reglas para algunos protocolos a nivel de aplicación, como ser FTP, TFTP, IRC y protocolos PPTP, los cuales presentan características difíciles de seguir para un firewall convencional. El cometido principal del sistema de connection tracking no es filtrar paquetes en sí, sino que el comportamiento por defecto de este sistema es guardar la información del flujo y dejar continuar su camino a los paquetes a través del Network stack del S.O.

Estados

Los posibles estados definidos para una conexión son los siguientes:

- NEW: La conexión se está estableciendo. Este estado se alcanza si el paquete es válido, es decir, si pertenece a la secuencia válida de inicialización (por ejemplo, en una conexión TCP, un paquete SYN recibido), y si el firewall sólo ve el tráfico en una dirección (es decir, el firewall aún no ha visto ningún paquete de respuesta).
- ESTABLISHED: La conexión ha sido establecida. En otras palabras,este estado se alcanza cuando el firewall ha visto la comunicación de dos vías.
- RELATED: Se trata de una conexión de espera.
- INVALID: Este es un estado especial que se utiliza para los paquetes que no siguen los comportamientos que se esperan de una conexión. Opcionalmente, el administrador del sistema puede definir las reglas de Iptables para registrar y colocar este paquete.

Cabe destacar que estos estados no tienen nada que ver con los estados de las conexiones del protocolo TCP. El módulo nf_conntrack para Ipv4 (también existe la posibilidad de hacer el seguimiento de las conexiones para Ipv6 y funciona de manera similar que Ipv4) registra cuatro funciones callbacks en varios hooks del Framework Netfilter. Las callbacks se pueden separar en tres tipos distintos: las callbacks para la creación y la búsqueda de las entradas en la lista de conntracks, las callbacks para los paquetes fragmentados y por último los helpers (los cuales se mencionan mas adelante), tomando todas ellas como parámetros, los datos dentro de los encabezados de las capas 3 y 4 del modelo OSI.

Estructura básica

Todo el seguimiento de las connection tracking es realizada por un Framework especial dentro del kernel llamado conntrack. Éste puede ser cargado ya sea como un módulo o directamente integrado al kernel de Linux. Aunque es siempre requerido cuando se desea trabajar con un servicio de Nat, en nuestro caso fue fundamental para poder trabajar en conjunto con el módulo L7, pudiendo así hacer el reconocimiento de los protocolos de capa de aplicación y mantener un estado de los flujos presentes en la red.

Este módulo está implementado con una tabla hash para mejorar en forma sustancial la performance del sistema. Cada entrada (Bucket) de la tabla hash tiene asociado una double-linked-list de tuplas hash. Hay dos tuplas hash para cada conexión (una en cada sentido de la dirección de los paquetes). Cada una de estas tuplas representa la información más importante de cada conexión como ser las IP de origen y destino e información de la capa 4 del modelo (TCP o UDP). A su vez cada una de estas tuplas está embebida en una tupla hash. De aquí en más llamaremos conntrack a esta estructura, la cual forma una estructura de tres capas para la identificación de cada flujo que circula por la red. Estas conntrack lo que hacen esencialmente es guardar el estado de cada uno de los flujos identificados y registrarlo en esta estructura especial para lograr una búsqueda relativamente fácil y rápida.

Hasta aquí tenemos definido la estructura donde se almacenarán las conexiones, ahora debemos saber cómo hará el sistema para encontrar una conexión cuando llegue un paquete. Para esto, cada vez que llega un paquete al sistema, se utiliza una función Hash para calcular la posición en que se encuentra la tupla hash que representa la conexión a la que pertenece el paquete. Para realizar esta función, se toman como parámetros de entrada a la función, los datos relevantes de los encabezados del paquete de la capa de red y de la capa de transporte (la función hash utilizada es la llamada función hash de jelkins), como resultado, la función nos devolverá la ubicación del bucket donde se debe buscar la conexión. Luego actualizará el estado de la conexión, o en caso de no encontrarse (primer paquete

de la conexión) se generara una nueva entrada en el conntrack [19].

Resumiendo, los pasos seguidos por este módulo ante la llegada de un paquete son los siguientes:

- Llega un paquete
- Se le aplica una función hash respecto a la tupla de origen, destino, protocolo y puertos.
- Ese hash se emplea como índice para acceder a la linked-list que corresponda. Esta operación siempre tiene el mismo costo de procesamiento.
- Se recorre la linked-list con las entradas de conntrack hasta encontrar la conexión. Dependerá del tamaño de esta lista, el tiempo que llevará encontrar la entrada buscada.

Los callbacks de nf_conntrack_in son registrados en el hook PREROUTING ya que como vimos, en éste es donde se reciben todos los paquetes que circulan por la red, y también se registran en el hook OUTPUT los paquetes generados por procesos locales.

Esto significa que Iptables hará todo el mantenimiento de los distintos estados de una conexión en la cadena PREROUTING. Por ejemplo si enviamos un paquete desde la máquina donde corre nuestro sistema para iniciar una conexión, el estado de dicha conexión va a cambiar hacia NEW en la cadena OUTPUT y cuando reciba el paquete de retorno, éste será analizado en la cadena PREROUTING, y en dicha cadena se cambiará el estado de esta conexión a ESTABLISHED (sin embargo en nuestro caso sólo inspeccionaremos en forma pasiva el flujo, con lo cual no nos interesan los paquetes debidos a procesos locales). Una vez que ingresa un paquete al sistema, se hace un chequeo previo para verificar que los encabezados del paquete sean correctos (checksum, etc) y luego el sistema busca una entrada en el sistema de conntrack que machee con la información del paquete recibido.

Estados de la máquina

Una Connection tracking se realiza para permitirle al Framework de Netfilter saber cómo está el estado de una conexión específica y en nuestro caso particular servirá al módulo L7-Filter a asociar los distintos paquetes con algunas de las conexiones existentes y poder así reconocer los distintos protocolos en cada flujo de datos. Los firewall que implementan este tipo de filtrado son los llamados statefull firewall. De esta forma, estos firewall son generalmente más seguros que uno no-statefull (stateless) firewall ya que nos permite escribir un conjunto de reglas más estrictas y dinámicas.

Dentro de Iptables, los paquetes pueden ser relacionados a distintas conexiones en cuatro estados distintos. Estos estados a los que nos referimos describen el estado actual de una conexión, los cuales son conocidos como NEW, ESTABLISHED, RELATED e INVALID. Con estos estados es sumamente sencillo controlar quién o a qué se le permitirá iniciar nuevas sesiones para el caso en que se quiera hacer filtrado o calidad de servicio o simplemente tener un control de la cantidad de conexiones activas o nuevas que se crean.

Entradas en la tabla CONNTRACK

Estas entradas se pueden encontrar en /proc/net/nf_conntrack o pueden ser visualizadas y manipuladas a nivel de usuario mediante el sistema de conntrack-tools de una manera más amigable, en donde se pueden ver las entradas actuales de la tabla, agregar nuevas conexiones manualmente, borrar conexiones específicas o manipular las existentes de distintas formas.

Las entradas de la tabla tienen un formato especial en donde se almacenan todos los datos relevantes de las conexiones y en caso de estar en funcionamiento el módulo L7-Filter, marcará la conexión con el nombre del protocolo con el cual este módulo haya reconocido los datos. A continuación damos un ejemplo de cómo se vería una entrada para una conexión particular:

```
tcp 6 117 ESTABLISHED src=192.168.1.6 dst=192.168.1.9
sport=32775
dport=22 src=192.168.1.9 dst=192.168.1.6 sport=22
dport=32775
[ASSURED] use=2 l7proto=ssh
```

El ejemplo muestra toda la información que el módulo de conntrack mantiene para saber cuál es el estado actual de dicha conexión. Primero que nada guarda el tipo de protocolo de capa de transporte de la conexión, el cual para este caso es TCP y va seguido por el código o versión del mismo en decimales. Después tenemos un contador el cual nos da una idea de cuánto tiempo tiene de vida la entrada dentro de la tabla de conntrack y la misma se va decrementando regularmente a menos que se vea más tráfico perteneciente a la conexión. Seguido de esta información viene el estado actual de la conexión, en donde debemos tener en cuenta que el valor interno del estado de una conexión es ligeramente distinto a los usados por Iptables. Por esta razón es que no deben ser confundidos entre sí. El valor ESTABLISHED nos dice que lo que estamos mirando es una conexión que se ha establecido por una comunicación en dos vías. Luego podemos encontrar información acerca de la IP de origen y destino seguido por los puertos correspondientes de origen y destino respectivamente. Las entradas en las connection tracking pueden tomar una

serie de diferentes valores dependiendo del tipo de protocolo que circula por dicha conexión (TCP, UDP, ICMP). Por último se ve cuál es el protocolo de capa 7 al que fue asignada dicha conexión, en este caso SSH.

Helpers and expectation

Algunos protocolos de la capa de aplicación presentan ciertas características especiales, las cuales son sumamente difíciles de seguir, uno de estos casos es el protocolo utilizado para el intercambio de archivos FTP. Cuando este protocolo trabaja en modo pasivo, utiliza el puerto 21 para intercambiar los datos de las operaciones de control entre el servidor y el cliente, pero luego utiliza para la transferencia de los datos, alguno de los puertos TCP del 1024 al 65535 en lugar de utilizar el puerto 20 (el cual utiliza cuando trabaja de manera activa o tradicional). Esto quiere decir, que si bien estrictamente las dos conexiones son distintas, ambas están relacionadas entre sí. Para poder llevar esto a cabo es necesario obtener información adicional para poder reconocer y de ser necesario filtrar este tipo de protocolos de una forma adecuada.

El sistema de connection tracking, define estructuras adicionales para hacer el seguimiento de estas conexiones, estos mecanismos se llaman helpers y permiten al sistema reconocer cuándo una conexión en realidad está relacionada o deriva de otra conexión (en FTP estas conexiones toman el nombre de padre-hijo). Para realizar esto, se define el concepto de expectation. Una expectation es una conexión la cual es probable que ocurra en un período corto de tiempo. Estos Helper buscan una serie de patrones en los paquetes de las conexiones padres para identificar a las que derivan de esta. Para el caso que venimos manejando de FTP, se busca el puerto en donde se hará la transferencia de los datos en la conexión de control (puerto TCP 21). Si este patrón es encontrado, se crea una nueva expectation y se espera por la llegada de los paquetes de esta conexión hija. En el caso de nuestro sistema, utilizamos varios Helpers para poder asociar distintas conexiones con sus conexiones de control originales, como ser el caso de los protocolos de VoIp H323 y SIP, los protocolos de transferencia IRC y FTP y TFTP. Para los casos particulares de los VoIp, identifica las conexiones de voz con las respectivas conexiones de señalización que las originaron (de esta forma podemos identificar el protocolo RTP de una llamada con el protocolo de señalización que la originó). De esta manera L7-Filter podrá reconocer las conexiones padres, buscar los distintos patrones en su payload y en caso de identificar alguna con algún protocolo, también identificará la conexión hija de esta con el mismo protocolo [20].

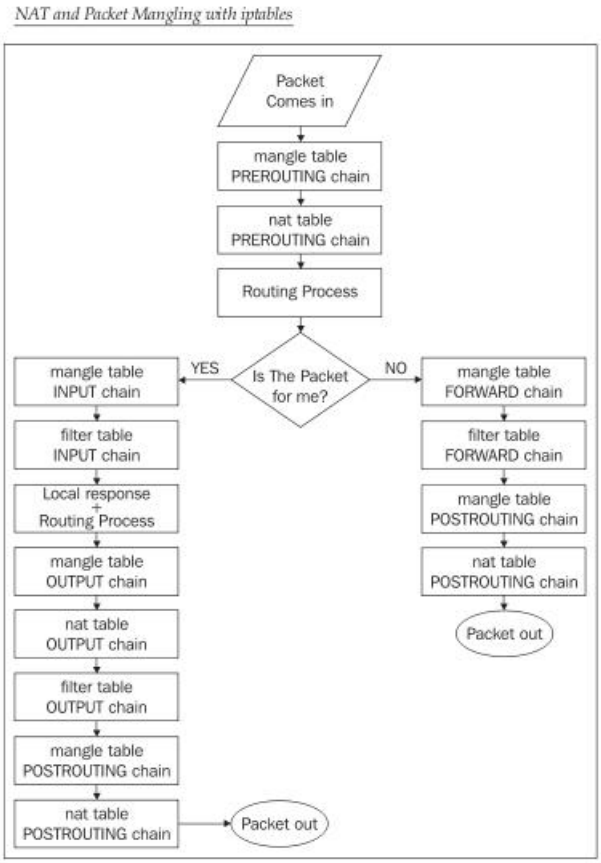
5.1.3. Iptables

Iptables es esencialmente un filtro IP y como lo indica su nombre, actúa en la capa 3 del modelo de referencia OSI. Sin embargo, tiene también la habilidad de trabajar en la capa 4, y en nuestro caso lo utilizaremos para realizar filtrado a nivel de capa 7. Si la implementación de un filtro IP sigue estrictamente la definición, este sería solamente capaz de filtrar paquetes basados en su encabezado IP (Dirección origen, dirección destino, TOS/DSCP/ECN, TTL, Protocolo, etc.). Debido a que la implementación de Iptables no es estricta respecto a esta definición, es posible clasificar paquetes basados en encabezados de capas superiores y buscar expresiones regulares en el payload de los paquetes. Esto último es posible gracias al módulo L7-Filter el cual es detallado más adelante.

Iptables permite al administrador del sistema definir reglas acerca de qué hacer con los paquetes de red pero no puede seguir el flujo de los paquetes, o juntar la información enviada en distintos paquetes de forma de contar con todos los datos enviados. Sí se puede mantener un estado de las conexiones e identificar qué paquetes pertenecen a la misma por medio de los números de puertos y direcciones IPs tanto de origen como destino. En este punto justamente es donde entra en acción el sistema de seguimiento de conexiones (connection tracking) previamente explicado.

A continuación se define qué es una tabla, una cadena, etc. y dentro de cada una de esas definiciones se nombran las diferentes opciones. Solamente se detallan las utilizadas por nosotros para implementar nuestro sistema. En el Anexo B se definen las demás opciones que hay dentro de Iptables.

Existen cuatro tablas definidas en Iptables y cada una tiene un propósito específico: filter, nat, mangle y raw. Reglas de filtrado son aplicadas a la tabla filter, reglas NAT son aplicadas a la tabla nat, reglas especializadas que alteran el contenido del paquete son aplicadas en la tabla mangle, y excepciones para conntrack se aplican en la tabla raw [21].



De las tablas mencionadas anteriormente, en el contexto de este proyecto nosotros utilizamos la tabla mangle. Esto es debido a que la misma es la tabla diseñada para operaciones avanzadas y es responsable de ajustar las opciones de los paquetes, por tanto todos los paquetes pasan a través de ella. Gracias a esta tabla y a la cadena PREROUTING definida más adelante, se logrará analizar en su totalidad todo el flujo de datos que pasa por la red, sin importar que el mismo sea o no destinado a la máquina con esa capacidad de análisis.

La tabla mangle contiene las cinco cadenas predefinidas las cuales se nombran a continuación:

- PREROUTING : Todos los paquetes que logran entrar a este sistema, antes de que el ruteo decida si el paquete debe ser reenviado o si tiene destino local.

- INPUT : Todos los paquetes destinados a este sistema pasan a través de esta cadena.
- FORWARD : En esta cadena de la tabla mangle se puede manipular los paquetes luego de haber realizado las decisiones de ruteo iniciales y antes de realizar las últimas decisiones justo antes de enviar el paquete.
- OUTPUT : Todos los paquetes creados en este sistema pasan a través de esta cadena.
- POSTROUTING : Todos los paquetes que abandonan este sistema pasan a través de esta cadena.

Además de las cadenas ya incorporadas, el usuario puede crear todas las cadenas que quiera dentro de cada tabla, las cuales permiten agrupar las reglas en forma lógica.

Cada cadena contiene una lista de reglas que definen qué hacer con cada paquete. Cada regla tiene un conjunto de macheos gracias a los cuales se puede especificar un destino o target que le indica a Iptables qué acción tomar sobre el paquete en caso de que cumpla con el macheo. Un macheo de Iptables es una condición que debe ser cumplida por un paquete para que Iptables procese al mismo de acuerdo a lo que especifique el target de la regla. Los target pueden ser aceptar, descartar, negar los paquetes u otras acciones más particulares. El target es útil para definir qué es lo que se quiere hacer con los paquetes que cumplan ciertas condiciones.

Cuando un paquete se envía a una cadena, se lo compara siguiendo el orden que tienen las reglas de esta cadena. La regla especifica qué propiedades debe tener el paquete para que la misma coincida. Si la regla no coincide, el procesamiento continúa con la regla siguiente. Si la regla, por el contrario, coincide con el paquete, las instrucciones de destino de las reglas se siguen (y cualquier otro procesamiento de la cadena normalmente se aborta). Algunas propiedades de los paquetes sólo pueden examinarse en ciertas cadenas (por ejemplo, contabilizar paquetes que salgan por cierta interfaz de red no es válido en la cadena INPUT). Algunos destinos sólo pueden usarse en ciertas cadenas y/o en ciertas tablas (por ejemplo, el destino SNAT sólo puede usarse en la cadena de POSTROUTING de la tabla NAT).

Luego de haber explicado brevemente las diferentes tablas, las posibles cadenas predefinidas y la utilidad de las reglas dentro de esas cadenas, entraremos más en detalle sobre cómo los paquetes atraviesan las distintas cadenas, y en qué orden. También se indicará el orden en que las tablas son atravesadas. Para ello se tomará como ejemplo el caso del tráfico destinado a la máquina local. Conocer esto será muy valioso a la hora de definir nuestras reglas, por ejemplo es esencialmente necesario saber en qué lugar se debe definir una regla cuando ésta puede modificar el ruteo de los paquetes.

Cuando un paquete entra a la máquina, pasa por la tarjeta para luego dirigirse al correspondiente driver del dispositivo en el kernel. Luego el paquete empieza una serie de pasos en el kernel para finalmente ser enviado a alguna aplicación, ser reenviado hacia otro host, etc. Para el caso en que el tráfico es destinado al localhost, se siguen los pasos mostrados en la siguiente tabla:

Paso	Tabla	Cadena	Descprición
1			En el cable
2			Ingresa a la interfaz
3	raw	PREROUTING	Esta cadena es utilizada para manejar los paquetes antes de que connection tracking intervenga. Puede ser usadopor ejemplo para indicar que una conexión específica no sea manejada por connection tracking
4			Este paso se da cuando el código de connection tracking toma lugar. Ver Estado de máquina
5	mangle	PREROUTING	Esta cadena es normalmente usada para manipular paquetes por ejemplo, cambiar el campo TTL.
6	nat	PREROUTING	Esta cadena es normalmente utilizada para DNAT, es recomendable evitar filtrado en esta cadena ya que es salteada en algunos casos.
7			Decisión de ruteo, el paquete tiene como destino el local host o debe ser forwardeado y hacia adonde
8	mangle	INPUT	Esta cadena se puede usar para manipular paquetes, luego de haber sido ruteados, pero antes de ser realmente enviados al proceso en la máquina.
9	filter	INPUT	Aquí es donde realizamos el filtrado para todo el tráfico entrante destinado hacia el local host. Notar que todo el tráfico entrante destinado para el local host atraviesa esta cadena.
10			Proceso local o aplicación que recibe el paquete.

Dentro del ANEXO B además de definir en su totalidad los términos utilizados en Iptables para las distintas opciones, hay dos ejemplos más de cómo se atraviesan las tablas y cadenas para casos diferentes.

Netfilter e Iptables

Durante los años, Iptables ha madurado en un formidable firewall con la mayoría de las funcionalidades disponible en los firewall comerciales. Por ejemplo, Iptables ofrece, inspección de paquetes en la capa de aplicación, limitador de velocidad, un poderoso mecanismo de especificar políticas de filtrado. Las mayores distribuciones de Linux incluyen Iptables, y muchas piden al usuario que especifique políticas de Iptables desde el instalador.

Las diferencias entre el término Iptables y Netfilter es una fuente de confusión en la comunidad Linux. El nombre del proyecto oficial para todo el filtrado y manejo de paquetes provisto por Linux es Netfilter, pero este término también refiere al Framework dentro del Kernel de Linux que puede ser usado para adosar funciones dentro del stack de red en varias etapas. Iptables es quien usa el Framework Netfilter para enganchar dichas funciones, diseñadas para realizar operaciones sobre los paquetes (como filtrado o conteo) dentro del stack de red. Se puede pensar de la siguiente manera, Netfilter como proveedor del Framework en el cual Iptables crea la funcionalidad de firewall. El término Iptables también refiere a la herramienta en el espacio de usuario que pasa los comandos y le comunica las políticas del firewall al kernel. Términos como tablas, cadenas, maches y targets dan sentido en el contexto de Iptables.

Netfilter no filtra el tráfico en sí mismo, sino que sólo permite a otras funciones que pueden filtrar el tráfico y sean incorporadas en el correcto lugar dentro del kernel. El proyecto Netfilter también provee bastantes piezas de infraestructura en el kernel, como connection tracking y logging; cualquier política de Iptables usa estas facilidades para realizar procesamiento especializado de paquetes.

En nuestro caso nos interesa qué porción del tráfico corresponde a las distintas aplicaciones, poder contar los paquetes o bytes generados por ellas nos permitirá lograr nuestro objetivo. Por lo tanto, utilizaremos Iptables para esta tarea. Como se dice anteriormente, Iptables puede referirse a varias cosas, entre ellas al comando de Linux que nos permite desde el nivel de usuario, conocer por ejemplo comó aumenta la cantidad de paquetes HTTP que han pasado por nuestro equipo.

Chain PREROUTING (policy ACCEPT 165 packets, 11751 bytes)

pkts	bytes	target prot opt	in	out	source	destination
64	4030	all – any	any	anywhere	anywhere	LAYER7 l7proto http

Podemos ver que han llegado a la cadena PREROUTING 165 paquetes cuyo tamaño acumulado es de 11751 bytes, de los cuales 64 paquetes son de HTTP con un tamaño total de 4030 bytes.

5.1.4. L7_Filter

L7-Filter es un módulo de Linux que sirve para clasificar los paquetes que circulan por una red de datos privada o de Internet, con la salvedad que a diferencia de la mayoría de los clasificadores actuales, L7-Filter no busca en los valores de los puertos de la capa de transporte ni en las direcciones de IP de la capa de red, sino que busca expresiones regulares en la capa de datos de aplicación para determinar el protocolo que está siendo utilizado en un determinado flujo de datos.

Este módulo ésta implementado para interactuar con el Framework Netfilter, ya sea trabajando a nivel del propio kernel del SO (como un módulo o directamente integrado a éste) o para interactuar con Netfilter pero desde el nivel de usuario. El hecho de que L7-Filter trabaje en asociación con Netfilter nos da la posibilidad de realizar diferentes acciones con el tráfico, utilizarlo para introducir marcas en los diferentes flujos para luego realizar control de calidad sobre el tráfico identificado, bloqueo de aplicaciones, etc.

A pesar de estas posibilidades, L7-Filter no está pensado, y no es recomendado por los propios desarrolladores del proyecto, para trabajar haciendo packet shapping o como solución de firewall. Para el caso de nuestro proyecto en particular, utilizaremos este módulo únicamente para reconocer los protocolos de aplicación en cada flujo y llevar un conteo a lo largo del tiempo de la característica y la distribución de los distintos protocolos en el tráfico total de la red. Esto es debido a que trabajaremos en forma pasiva sobre el tráfico y no es nuestra intención en principio tomar acciones de ningún tipo sobre el mismo.

Como mencionamos, existen actualmente dos versiones de L7-Filter:

- Kernel-Space: fue la que se desarrolló primero y la que está mejor testeada. Es complicada de instalar y en ocasiones causa que algunos sistemas dejen de funcionar. Esta versión utiliza expresiones regulares simples para los patrones.
- User-Space:esta versión está en lo primeros niveles de desarrollo. Es relativamente fácil de utilizar y no puede hacer que el sistema se vuelva inestable y deje de funcionar. Puede utilizar completamente las expresiones regulares grep-style de GNU.

Para nuestro sistema optamos por trabajar con la versión de kernel-space, para poder lograr que la herramienta fuera más eficiente, ahorrándonos procesamiento innecesario y consumir menor cantidad de recursos del sistema. El principio de funcionamiento de la versión en user-space es marcar los paquetes que son enviados por Netfilter hacia el nivel de usuario, analizarlos y luego volver a marcar el paquete con el protocolo al que pertenecen para devolverlo y dejar así la opción de que Netfilter realice distintas tareas con ellos de ser necesario. Para esto, Netfilter

cada vez que le llega un paquete al Hook donde se desea que L7-Filter analice el tráfico, lo pone en un sistema de colas para poder pasarlo al user-space para su análisis. L7-Filter toma este paquete y debe consultar al sistema de conntrack por el estado de dicha conexión, con lo cual debe hacer una llamada al sistema para adquirir estos datos (recordemos que el sistema de connection tracking trabaja al igual que Netfilter directamente sobre el Kernel de Linux). Luego de analizado el paquete, L7-Filter devuelve el paquete nuevamente a Netfilter en el kernel-space con una marca del protocolo correspondiente en caso de que halla logrado reconocer algún patrón o sin marcar en caso contrario, luego el paquete sigue el transcurso normal a través de los Hook de Netfilter.
Todo este sistema de lectura y escritura del Kernel hacia el user-space y viceversa, genera un procesamiento innecesario como vimos en capítulos anteriores. Para el caso de la versión que trabaja directamente sobre el Kernel, los patrones son integrados en el kernel, y todo el análisis se hace sobre éste, ahorrándonos todo el overhead y procesamiento innecesario que genera la versión de user-space.

También nos apoyamos en las afirmaciones del equipo de desarrollo de la herramienta en cuanto a la eficiencia y estabilidad de las distintas versiones, donde explican que la versión de user-space está en su etapa de inicio y no ha sido lo suficientemente testeada. Por el contrario la versión para el kernel-space a pesar de estar testeada y ser estable en su funcionamiento, presenta mayores dolores de cabeza a la hora de la instalación y mayor dificultad para escribir nuevos patrones debidos a la dificultad de su sintaxis.

L7-Filter utiliza para agregar sus reglas la sintaxis estándar con la cual trabaja Iptables. Debimos tener en cuenta la forma en que trabaja Iptables a la hora de elegir las reglas y las tablas en donde debíamos agregar las mismas para que el módulo l7-Filter reconozca bien el tráfico. Los patrones deben ser capaces de poder ver todo el flujo de datos y principalmente ambos sentidos de la conexión para poder machear e identificar un flujo en forma correcta. Por la cadena de Iptables donde se agreguen las reglas deben pasar paquetes en ambos sentidos.

Que es lo que L7_Filter ve y hace

Si hemos ingresado las reglas de Iptables en el lugar correcto, L7-Filter verá los paquetes de datos que van en ambas direcciones (del cliente al servidor y viceversa) y utilizará conntrack para caracterizar los flujos de red que pasan a través de la máquina, almacenando en buffer el payload de los N primeros paquetes de la conexión para su estudio. De esta manera, a modo de ejemplo, en el inicio de la conexión de control del protocolo FTP lo primero que se ve en el payload de los paquetes es “221 server ready”, después ÜSER bob", luego "331 send password", luego "PASS frogy-jeje", y así se continúa con los siguientes mensajes especificados por el protocolo. De no poder ver ambos lados de la conexión L7-Filter no

lograría una buena eficiencia a la hora de machear ya que muchos de los patrones buscados no coincidirían.

En una primera instancia, con el ejemplo especificado para FTP, el patrón intenta primero machear sobre "221 server ready", luego sobre "221 server readyUser bob", luego "221 server readyUSER bob331 send password", etc. Por lo tanto, lo que podemos intentar buscar para este protocolo sería algo como "220.*user.*331", estos son parámetros o palabras claves que aparecen en todas las conexiones del protocolo FTP. Como el módulo conntrack rastrea también sesiones UDP e ICMP como conexiones, este trabajará con ellos de la misma forma que lo hace con TCP.

Generalmente las características buscadas para identificar un protocolo con un flujo de datos se encuentran al inicio de la conexión, en donde se negoció el inicio de la conexión del protocolo de aplicación y los recursos que utilizará la aplicación para el posterior envío de los datos. Por esta razón, y para ahorrar tiempo de procesador y recursos, l7-Filter sólo busca en los primeros paquetes (como default se utilizan 10 paquetes como máximo, luego se da como desconocido el flujo) o en los primeros 2kbyte del payload de cada conexión. Si se buscara en todos los paquetes de la conexión por estos patrones, se estarían consumiendo una gran cantidad de recursos del sistema como ser CPU y memoria y no se obtendría una eficiencia considerable.

El payload se irá almacenando en un buffer formando un stream de datos por cada flujo. Sobre los datos almacenados en este buffer se recorrerán los patrones hasta que alguno de ellos coincida y se reconozca que el flujo de datos ha sido generado por cierto protocolo. Cada macheo hecho a esa altura será aplicado para el resto de los paquetes de la conexión utilizando conntrack. Si luego de los 10 paquetes o que se llene el buffer del payload, L7-Filter aún no ha asociado un protocolo a este flujo, el mismo se dará como desconocido y no se seguirán revisando el resto de los paquetes que arriben de ese flujo. De esta forma, al examinar sólo los primeros paquetes de una conexión, l7-Filter es razonablemente eficiente a pesar de usar un algoritmo más complicado que la mayoría de los clasificadores de paquetes tradicionales.

Como hemos explicado en el ejemplo anterior, para el protocolo FTP lo que se intenta reconocer dentro del flujo es lo que podríamos llamar el propio “encabezado” de este protocolo. En general cada protocolo en la la capa de aplicación genera lo que seria un encabezado adicional en el modelo de capas de red de Internet. Este encabezado ubicado dentro del payload del encabezado TCP (donde definimos como payload de un encabezado específico , a todos los datos que vienen después de este hasta el final de la trama), es el utilizado por la aplicación para negociar entre los dos extremos los distintos parámetros que serán utilizados para trabajar en forma correcta. Al igual como ocurre con el encabezado IP, en donde el encabezado TCP forma parte de su payload, en la capa de aplicación podría pensarse de una

manera similar, en donde dentro del payload del encabezado TCP existe otro encabezado perteneciente a la aplicación que está corriendo el usuario. Desde este punto de vista, podemos decir que los verdaderos datos útiles del usuario estarían dentro del "payload" de este encabezado perteneciente al protocolo de aplicación, y al realizar la inspección mediante DPI (en este caso por los patrones del módulo L7-Filter) no estamos inspeccionando realmente los datos del usuario, sino que estaríamos buscando los encabezados de los protocolos de aplicación, sin violar la integridad de los verdaderos datos del usuario.

Patrones

L7-Filter provee una serie de archivos donde se encuentran los patrones con sus respectivas expresiones regulares y algunos comentarios sobre la eficiencia y rapidez del patrón, de esta manera el usuario no tiene la necesidad de proveer las expresiones regulares a través de línea de comando. Una vez que sean agregados los patrones usando las reglas de Iptables, estos se guardarán en el correcto lugar del kernel y se comenzará a analizar los paquetes para los diferentes flujos en la red.

Los archivos donde se guardan las expresiones regulares presentan un formato especial el cual es descrito a continuación, también se mencionará a grandes rasgos cómo es la expresión regular de un patrón y las características que debe presentar para ser considerado como un patrón bueno.

- Formato Básico:
  - nombre del protocolo en una línea
  - expresión regular que defina el protocolo en la siguiente línea.

El nombre del archivo debe corresponder con el nombre del protocolo (si es ftp entonces el nombre del archivo debe ser "ftp.pat") y las líneas que comienzan con "#" y líneas en blanco son ignoradas. De esta forma por ejemplo, el archivo para vnc.pat puede ser:

vnc
ˆ rfb 00[1-9]\.00[0-9]\x0a$ [1]

- Un patrón debe ser lo suficientemente específico, pero no demasiado.
  - Ejemplo 1 : el patrón para "bear" para Bearshare no es lo suficientemente específico. Este patrón puede machear una gran variedad de conexiones no-Bearshare. Por ejemplo un pedido HHTP para *http://bear.com* sería macheado por este patrón.

[1] expresión regular -"V8 (a.k.a. Henry Spencer, a.k.a. kernel) Regular Expressions "

  - Ejemplo 2 : el patrón "`220.*ftp.*(\[.*\]|\(.*\))`" para FTP es demasiado específico. No todos los servidores mandan () o [] luego de 220. De hecho, los servidores no están obligados a enviar el string "ftp" todo el tiempo, aunque la gran mayoría lo hace.

- El patrón debe usar la mínima cantidad de poder de procesamiento imprescindible. Si es posible reducir el número de instancias de *,+ y | en un patrón, éste debería hacerlo.

- El patrón debe completar el macheo en la menor cantidad posible de los primeros paquetes de una conexión.Por ejemplo el patrón:
  "\^220[\x09-\x0d -~]* \x0d \x0aUSER[\x09-\x0d - ~]* \x0d \x0a331"para FTP no macheará hasta recibir al menos el tercer paquete de la conexión.
  Mientras que el patrón: "~220[\x09-\x0d - ~]*ftp"
  debería machear en el primer paquete de la conexión.

Cómo actualizar definición de protocolos

La definición de los protocolos es simplemente un texto con un formato específico, el cual fue definido anteriormente en detalle. La modificación de estos archivos se puede realizar cambiando el archivo mismo o modificando la sintaxis dentro del archivo existente.

Si se agrega un patrón o se actualiza una definición de un protocolo, es necesario borrar y volver a ingresar la regla correspondiente de Iptables. Esto se debe a que las expresiones regulares de los patrones son integradas a un espacio de memoria del kernel al momento de agregar la regla correspondiente utilizando Iptables.

Problemas con L7_Filter

Algunos de los problemas que enfrentamos con l7-Filter o con otras herramientas de reconocimientos de patrones mediante el análisis del payload son:

- Falsos positivos (un protocolo que luce igual que otro).
- Falsos negativos (algunas aplicaciones pueden tener un comportamiento oscuro para enmascararse y hacerse pasar por otro protocolo).
- También es posible, que una conexión sea macheada por más de un patrón.

Los patrones son testeados en el orden especificado en las reglas de Iptables. Luego de que una conexión machea con alguna de las reglas, L7-Filter no continúa testeando el resto de los patrones sobre esta conexión. Por esta razón, el orden en que ingresamos las reglas de Iptables también debe ser tomado en cuenta, ya que alterando el orden original podría provocar algunos cambios en la identificación de los flujos.

Otras observaciones a tener en cuenta a la hora de elegir el orden en la ubicación de un patrón dentro de la lista de protocolos a identificar, es la velocidad de reconocimiento de dicho patrón, de esta forma estaríamos ahorrando tiempo de procesamiento y logrando mayor eficiencia a la hora de identificar las distintas aplicaciones dentro de un tráfico de datos. Otro ítem a tener en cuenta es la precisión de un patrón al reconocer en forma correcta una determinada aplicación, y no perder precisión a la hora de identificar un flujo con falsos positivos o falsos negativos. Estas características de precisión de los patrones se estudian con mayores detalles en secciones posteriores.

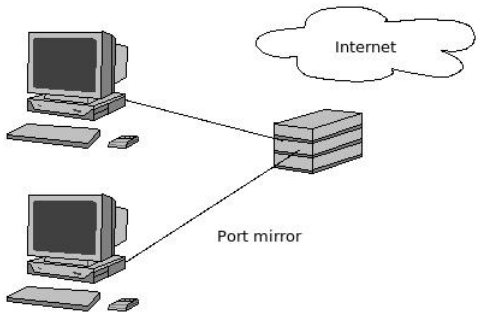
Algunos de los problemas existentes para detectar ciertas aplicaciones, es que al ser bloqueados, son capaces de switchear entre TCP y UDP, abrir nuevas conexiones para cualquier operación trivial, usar cifrado o emplear otras tácticas evasivas. En el caso de utilizar protocolos con cifrado de los datos, como es el caso del protocolo SSL, reconoceríamos este protocolo, pero no podríamos reconocer cuál protocolo de aplicación estaría corriendo encima de SSL.

5.1.5. Configuración del módulo

Hasta aquí hemos introducido una serie de herramientas y definiciones acerca de cómo es el funcionamiento actual del firewall de Linux y cuales son las funcionalidades que posee el mismo. Se ha intentado explicar con bastante detalle Netfilter, Iptables y L7-Filter, ya que a partir de ellos podremos caracterizar el tráfico.

L7-Filter no viene integrado en los releases de Linux y fue necesario recompilar el kernel luego de aplicar una serie de parches a las fuentes de Linux, Netfilter e Iptables (ver Anexo B). Esto permitió contar con un nuevo módulo en Iptables desde el cual era posible indicarle al kernel, a través del Framework Netfilter que realice una inspección más exhaustiva de los paquetes. Este módulo llamado L7-Filter va juntando el payload de un número finito de paquetes pertenecientes a una conexión, para luego buscar si se verifican las expresiones regulares de las reglas de Iptables cargadas, con el flujo que se está analizando. En caso de que se llegue al máximo de paquetes buscados, L7-Filter se resigna y cataloga a ese flujo como unknown, en el caso contrario en que sí pueda encontrar quién a generado los paquetes que ha analizado antes de llegar al máximo, asocia esos paquetes con el protocolo que corresponda, así como también a todos los demás paquetes que pertenecen al mismo flujo. Aquí es donde entra el poder de conntrack y una de las utilidades que más necesita L7-Filter para caracterizar la mayor cantidad de paquetes y flujos y poder disminuir notoriamente los niveles necesarios de procesamiento. Gracias a conntrack, L7-Filter puede manejar cargas que de otra manera le sería imposible. Más adelante indicaremos hasta dónde hemos podido llegar con esta herramienta antes de comenzar a descartar paquetes o consumir la CPU.

Como dijimos, todos los paquetes que ingresan a una PC pasan por la cadena PREROUTING y para que L7-Filter funcione correctamente, debe poder ver los dos lados del flujo. En nuestro caso definimos la siguiente topología:



Vamos a trabajar en modo pasivo, escuchando en un mirror de un switch. Debido a esto recibiremos ambos lados de la conexión por la misma interfaz y por

consiguiente pasarán todos los paquetes por la cadena PREROUTING. Sólo alcanzará con definir en esta cadena los protocolos que deseamos buscar, pero todavía existe un problema y es que estos paquetes no están destinados a la MAC de la PC en la que se ha agregado el módulo L7-Filter, y serán descartados antes de que puedan ser tomados por los hooks de Netfilter.

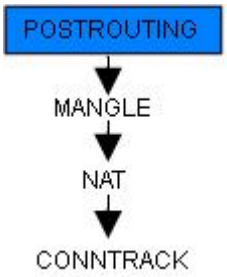
No alcanzó con definir que la interfaz trabaje en modo promiscuo para que los paquetes sean tomados por Netfilter y fue necesario crear un bridge "tonto" entre la interfaz de entrada y otra interfaz de salida. Al crear un bridge, se indica que los paquetes que ingresan por una interfaz sean redireccionados hacia la otra interfaz perteneciente al bridge, pero éste, por defecto va aprendiendo qué hosts se encuentran de cada lado del bridge de forma de redireccionar los paquetes por la interfaz donde se encuentra el destino, es similar a un switch y evita propagar paquetes por la red de forma innecesaria. Para realizar esto, mantiene una tabla asociando las direcciones MACs de los hosts con las interfaces por la que se puede llegar a dicho host. Nosotros recibimos todo el tráfico por la misma interfaz por lo que trabajar de esta manera sólo deja pasar los primeros paquetes hacia el otro lado. Luego de "aprender" comienza a descartarlos y por lo tanto no llegan a los hooks de Netfilter.

La solución para esto fue crear un bridge "tonto" que no aprendiera donde se encontraban los host y redireccionará todos los paquetes por la otra interfaz asociada al bridge sin importar cuál sea su destino. Esto se logró seteando en 0 el tiempo de permanencia de las entradas en la tabla del bridge. De esta forma pudimos hacer que los paquetes no sean descartados y que puedan pasar por los hooks PREROUTING – FORWARD - POSTROUTING, donde Iptables junto a L7-Filter estarán escuchando. Para establecer el bridge necesitábamos contar con dos interfaces, y mantener el link en ambas interfaces. Utilizamos un loop Ethernet en la interfaz de salida de forma de lograr esto, conectando Rx con Tx.

Otra consideración a tener en cuenta es que se debe evitar contabilizar doble la cantidad de paquetes y bytes macheados. Por ejemplo, si se ha conectado la interfaz eth0 al port mirror de un switch, los paquetes ingresarán por esta interfaz para luego atravesar las cadenas de Netfilter y salir por la interfaz eth1. Como hemos colocado un loop ethernet en esta interfaz, si configuramos la misma en half duplex, cada vez que la tarjeta envíe un paquete, no estará escuchando al mismo tiempo y no recibirá los mismos paquetes que envía. Si no se realizara esto, los paquetes volverían a ingresar y atravesarían nuevamente todas las cadenas de Netfilter en el sentido opuesto y se contabilizarían nuevamente.

Otra forma de evitar lo anterior es descartar los paquetes que ingresan por la interfaz eth1 utilizando el match de Iptables, "physdev". Pero encontramos que la primer manera no requiere agregar una nueva regla ni tampoco ningún tipo de

procesamiento.[2]



Se intentó realizar lo anterior descartando los paquetes en alguna de las cadenas, como por ejemplo POSTROUTING, pero esto empeoró la caracterización de L7-Filter. La razón de ello es que en las distintas cadenas existe un orden en el cuál se chequean las tablas. Si se dropean los paquetes utilizando la tabla mangle o la tabla NAT, estos no llegarán a pasar por conntrack,quien tiene como tarea en esta cadena confirmar que se ha dado una comunicación en dos sentidos e ingresar la entrada correspondiente en su tabla [22].

Una vez llegado a este punto estábamos en condiciones de comenzar a realizar pruebas e investigar la correcta caracterización de los protocolos, pero nos topamos con un problema, y es que no existen o no pudimos obtener trazas bien caracterizadas o trazas patrones que permitan tener un punto de comparación con los resultados que nos entregue L7-Filter a través de los contadores de Iptables. Por lo tanto tuvimos la necesidad de generar nuestras propias trazas patrones. Seleccionamos distintos tipos de aplicaciones, entre las que se encuentran Web, Correo, VoIp, P2P, transferencia de archivo, Stream de música y video, acceso remoto y chat. De esta manera intentamos cubrir las aplicaciones más demandadas actualmente por los usuarios. Todas estas trazas se generaron de forma controlada e intentamos que la mayoría de los paquetes que integran cada una de ellas correspondan a un único protocolo.

[2]En el ANEXO E se explica cómo configurar cualquiera de estas dos formas

5.2. Módulo de recolección de datos y almacenamiento en base de datos

La utilidad de este módulo es recolectar y guardar información desplegada por Iptables para luego ser mostrados en diferentes períodos de tiempo, o simplemente guardar dicha información para su posterior análisis.
Dado lo anterior, durante el estudio de diversas herramientas, nos enfocamos a que las mismas fueran capaces de consultar periódicamente los contadores de las Iptables y guardaran la información en bases de datos.

Otras dos características importantes que se tuvieron en cuenta a la hora de seleccionar la herramienta, fueron las siguientes:

- Las bases de datos creadas para cada protocolo no debían crecer indefinidamente, evitando así que las mismas se tornaran difíciles de mantener y además consumieran muchos recursos del sistema.
- Se pudiera crear con facilidad una nueva base de datos para un nuevo protocolo a analizar.

La herramienta que mejor se adaptó a nuestros requerimientos fue COLLECTD. La misma trabaja como demonio recopilando información de distintas variables del sistema. Estos datos son guardados en unas Bases RRDs presentando una estructura circular la cual es detallada a continuación.

5.2.1. RRD-Tools Round Robin Data Base

RRD hace referencia a Round Robin Database, la cual es una manera de trabajar con un número fijo de datos y un puntero al elemento de referencia. Esto se puede pensar como un círculo en cuyos bordes se almacenan un número determinado de datos, y desde el centro del círculo una flecha que haría de puntero hacia alguno de los datos. Si se lee o se escribe un elemento en la base, este puntero se mueve hacia el siguiente. De esta forma, la base de datos no posee un elemento inicial o uno final, y luego de un tiempo, los lugares disponibles en dicha base se acabarán. Dicho proceso automáticamente irá reutilizando los lugares a medida que tenga nuevos datos para almacenar, sobrescribiendo los viejos.

Con la forma de trabajo detallada en el párrafo anterior, se puede asegurar que la base datos para cada protocolo no crecerá nunca en tamaño. Como resultado, no requerirá mantenimiento alguno ni consumirá demasiados recursos del sistema, logrando así que se cumpla uno de los requerimientos fijados durante la selección de la herramienta.

Los datos que son capaces de almacenar este tipo de bases son prácticamente cualquiera que tenga la característica de ser una serie de datos temporal. Esto último hace que sea posible guardar información obtenida a partir de las Iptables para los distintos protocolos a lo largo del tiempo [23].

Cómo trabaja RRDtools

Adquisición de los datos:

Cuando se están monitoreando los valores de los contadores de las Iptables, es conveniente tener los datos disponibles en un intervalo de tiempo constante. Desafortunadamente no siempre se tendrá disponible estos datos en el momento exacto en el que se requieran. Por lo tanto, RRDtools permite actualizar el archivo de registro en cualquier momento que se desee, luego interpolará automáticamente el valor enviado por el data-source en el momento del último Time Slot oficial y escribirá este valor en la base de datos. El valor original que se ha proporcionado también se almacena y se tiene en cuenta a la hora de realizar la próxima interpolación.

Función Consolidacion:

Uno puede guardar información en intervalos de 1 minuto, pero podría ser de gran interés saber el desarrollo de los datos a lo largo del último año. Esto se puede realizar fácilmente almacenando los datos en intervalos de 1 minuto a lo largo de todo el año. Sin embargo, realizarlo de esta manera ocuparía un espacio en disco considerable, y a su vez tomaría mucho tiempo cuando se desee analizar los datos obtenidos al crear una gráfica para cubrir todo el año. RRDtools ofrece una solución a esto a través de su herramienta data-consolidation. Cuando se setea una RRData-base, se puede definir el tamaño del intervalo de tiempo para el cual se hará la consolidación de los datos y cual será la función de consolidación (promedio, picos máximos, mínimos, el último, total, etc.).

RRA Round Robin Archives:

Los valores de los datos de la misma consolidación son almacenados en el mismo archivo Round Robin (RRA). Ésta es una manera muy eficiente de guardar estos datos durante un determinado período de tiempo, utilizando una cantidad constante de espacio en disco.
Esto trabaja de la siguiente manera: si se desean almacenar 1000 valores en intervalos de 5 minutos, RRDtools destinará un espacio para guardar estos 1000 valores y un espacio para almacenar el header. Sobre este header se almacenará el puntero de

la base, indicando en qué valor del slot en el área de almacenamiento se está escribiendo. Los nuevos valores que se irán escribiendo en la base, se irán almacenando en la RRA de la manera Round Robin vista anteriormente. Con esto, automáticamente limitamos el historial de valores a los últimos 1000 capturados. Para cada RRD se pueden setear distintos RRA con diferentes funciones de consolidación y distintos tamaños de almacenamiento.

El uso de las RRA garantizan que las RRD no crezcan en el tiempo, y que los datos viejos sean automáticamente eliminados. Sin embargo, utilizando dichas funciones de consolidación se puede almacenar información por largos períodos de tiempo, bajando gradualmente la resolución de las muestras a lo largo del eje de tiempo. A su vez, con estas funciones de consolidación se puede guardar el tipo de dato que se desee dentro del intervalo de tiempo especificado.

RRDtools nos permite generar reportes tanto numéricos como gráficos basados en los datos almacenados en los distintos RRDs. Sin embargo, para nuestro caso en particular optamos por no usar estas herramientas debido a que las mismas si bien son configurables, no presentan la información en forma muy amigable. Además necesitábamos poder aplicarle a los valores obtenidos, diferentes funciones como por ejemplo, para poder ver el porcentaje de tráfico de cada protocolo p2p sobre el total del tráfico, etc. Esto último se logró realizar de forma sencilla utilizando la herramienta CACTI la cual se explica en detalle dentro del módulo de representación gráfica.

Cómo crear una nueva base de datos

A la hora de crear una nueva base de datos Round Robin, en concreto se debe especificar la frecuencia de actualización, la naturaleza de los datos, las funciones que forman los archivos y la resolución de estos. A continuación se detalla el significado de los datos anteriores y los posibles tipos:

- La frecuencia de actualización o Step indica cada cuánto tiempo se deben ingresar datos nuevos a la base de datos.
- La naturaleza de los datos se puede definir como:
  - COUNTER: contador, siempre se incrementa, registra el incremento/intervalo de tiempo como por ejemplo, paquetes/s o bytes/s.
  - GAUGE: indicador, registra el valor tal como lo medimos, por ejemplo, uso de la CPU, número de usuarios, etc.
  - DERIVE: contador, puede decrecer.
  - AVSOLUTE: valor absoluto, contador que se resetea tras su lectura.

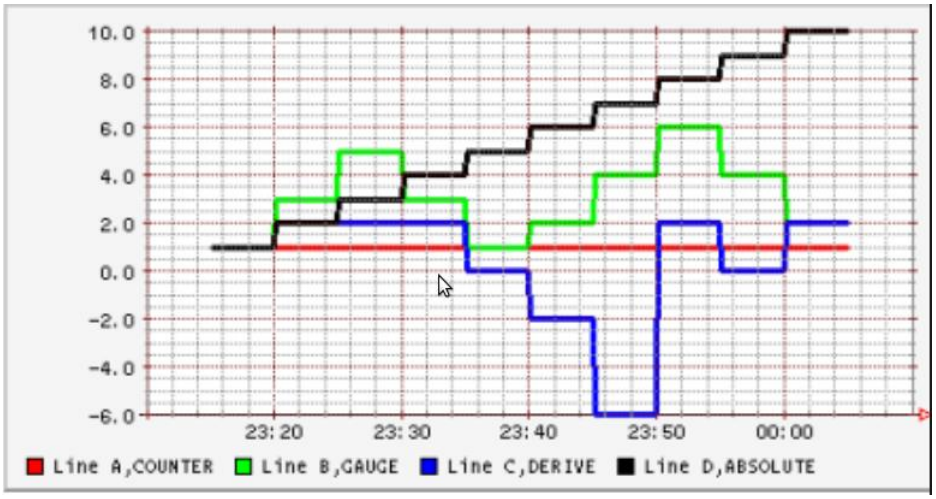
Lo veremos más claro con el siguiente ejemplo:
Se crea la base de datos all.rrd en la cual se define una variable de cada tipo:

rrdtool create all.rrd –start 978300900
DS:a:COUNTER:600:U:U
DS:b:GAUGE:600:U:U
DS:c:DERIVE:600:U:U
DS:d:ABSOLUTE:600:U:U
RRA:AVERAGE:0.5:1:10

Insertamos los datos:
La primer columna indica la marca de tiempo, las siguientes son los valores para cada una de las cuatro variables. O sea, para el tiempo 978301200 se inserta en la base el valor 300 para el tipo COUNTER, 1 para GAUGE, 600 para DERIVE, 300 para ABSOLUTE y así sucesivamente separados un tiempo de 300 segundos como se puede ver en la primer columna. Estos 300 segundos es la llamada frecuencia de actualización o el StepSize.

rrdtool update all.rrd
978301200:300:1:600:300
978301500:600:3:1200:600
978301800:900:5:1800:900
978302100:1200:3:2400:1200
978302400:1500:1:2400:1500
978302700:1800:2:1800:1800
978303000:2100:4:0:2100
978303300:2400:6:600:2400
978303600:2700:4:600:2700
978303900:3000:2:1200:3000

Si se grafica este ejemplo veremos lo siguiente:



Se puede observar que para la variable A (COUNTER), aunque insertamos datos que van aumentando, se dibuja una línea recta. Esto representa el incremento constante.
La línea B, GAUGE, representa los valores tal como los recogemos.
La línea C, DERIVE, se comporta como la A pero permite decremento.
La línea D, ABSOLUTE, para los mismos datos que la A, representa un incremento mayor, ya que supone que se resetea el contador.

Las funciones de consolidación para los archivos RRA indican qué función aplicar sobre los datos originales para obtener valores de una RRA con menor resolución. Estas funciones pueden ser las siguientes:

- AVERAGE
- MIN
- MAX
- LAST

Normalmente interesa guardar la media de los valores recogidos, pero también puede ser útil el mínimo o máximo valor del período.
Para crear diferentes archivos RRAs dentro de cada base de datos, se debe determinar lo siguiente:

- cada cuánto tiempo tomar muestras nuevas (Step).
- cada cuántas muestras aplicar la función de consolidación (pdp_per_row).
- cuantos de estos valores consolidados guardar en cada RRA (rows).

El Timespan, que es largo de la RRA en segundos, queda determinado por estos últimos valores: Timespan=Step * pdp_per_row * rows

En resumen, una base de datos RRD se compone de:

- Data Source
  La sintaxis para especificar una fuente de datos o data source (DS) en RRD es la siguiente:

  DS:nombre:COUNTER|GAUGE|ABSOLUTE|DERIVE:frecuencia:vmin:vmax

  - DS: clave para data source
  - nombre: variable donde recogeremos los datos
  - tipo: COUNTER,GAUGE,ABSOLUTE,DERIVE
  - frecuencia: especifica el número de segundos que pueden pasar entre cada muestra sin que se asuma como nulo.
  - vmin,vmax: valor mínimo y máximo aceptable. Si el valor recogido está fuera del rango, se le asignará el valor nulo.
- Round robin archives (RRA)

  RRA:AVERAGE|MIN|MAX:percent:pdp_per_row:rows

  - RRA: clave para round robin arhive
  - function: función de consolidación a aplicar sobre los datos.
    AVERAGE, MIN, MAX, LAST
  - percent: define qué parte del intervalo de consolidación puede estar formado por datos NAN (intervalos sin tráfico).
  - pdp_per_row: define la cantidad de valores originales que se utilizan para calcular la función de consolidación.
  - rows: especifica la cantidad de valores consolidados que se deben guardar en la base de datos.

5.2.2. Collectd

Como se mencionó durante la descripción de este módulo, se decidió utilizar esta herramienta para recolectar y guardar información obtenida con las Iptables sobre la cantidad de bytes y paquetes de cada protocolo.

A continuación se describe brevemente el funcionamiento de la misma y las ventajas que presenta para nuestros objetivos.

Collectd es un demonio que recibe estadísticas del sistema sobre el cual está corriendo y almacena dicha información dejándola disponible de varias maneras. Estas estadísticas pueden ser luego usadas para distintos propósitos específicos como por ejemplo, ubicar problemas de performance debido a apariciones de bottlenecks en una red de datos, tener una predicción futura de la carga del sistema, o simplemente mostrar en una gráfica actualizada en tiempo real o cada determinado tiempo las distintas variables del sistema que tengamos interés en monitorear [24].

En nuestro caso, necesitamos saber qué protocolos están siendo utilizados y el porcentaje de cada uno de ellos sobre el total del tráfico. Esto se puede lograr fácilmente habilitando dos plugins ya incluídos en el archivo de configuración collectd.conf y realizando una configuración muy sencilla sobre cada uno de ellos. Más adelante hablamos sobre estos plugins y detallamos sus respectivas configuraciones.

¿Por qué collectd?

Hay una variedad muy basta de proyectos free open source con funcionalidades similares a collectd, pero hay algunas diferencias esenciales por las cuales creemos que este demonio se adapta de manera correcta a nuestro sistema.

La primera característica de esta herramienta es que está escrita completamente en C, lo cual mejora la portabilidad y la performance. Esto permite que corra en distintos tipos de sistemas operativos. A su vez, incluye optimizaciones y utilidades para manejar miles de conjuntos datos.

Viene con una gran variedad de plugins los cuales van desde casos muy simples hasta casos muy específicos y aplicaciones avanzadas. El uso de los diferentes plugins influye en que el demonio principal no tenga dependencias externas y hace aún más fácil su inserción en los distintos sistemas.

Otra característica importante es que collectd está actualmente en un estado activo de desarrollo, con un equipo de soporte y de desarrolladores, y con una buena

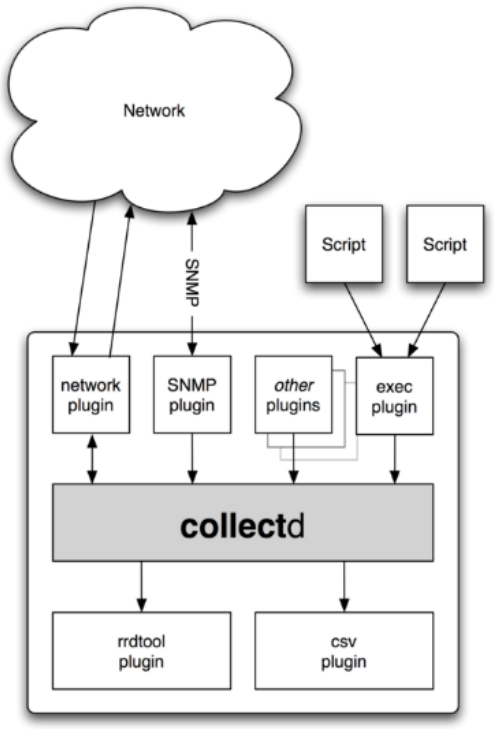
documentación oficial.

A nivel de performance, debido a que collectd corre como demonio, no consume tiempo arrancando y cargándose una y otra vez. Una vez puesto en marcha, prácticamente ningún mantenimiento es necesario sobre el mismo para su correcto funcionamiento. El demonio principal en sí no tiene ningún tipo de funcionalidad real, aparte de la carga, consulta y presentación de plugins. La configuración se intenta mantener lo más sencilla posible. Mas allá de qué plugins es necesario habilitar y configurar, no es necesario cambiar o agregar nada más. Sin embargo este se puede modificar y adaptar a nuestras necesidades tanto como sea necesario.

El verdadero poder de esta herramienta son eventualmente los plugins, los cuales se dividen principalmente en dos grandes grupos: input y output plugins. Los input plugins son consultados periódicamente. Se adquiere el valor actual de lo que se quiera monitorear y se entrega este valor al demonio. Un ejemplo de este tipo de plugins es el CPU plugin, el cual lee los valores actuales del cpu-counters en sus diferentes modos (user, system, nice, etc.) y despacha estos contadores al demonio.

Los Output plugins obtienen los valores enviados por el demonio y efectúan alguna acción con ellos. Las aplicaciones más comunes son escribir archivos RRD, CSV, enviar los datos a través de la red a un equipo remoto, etc. Por supuesto que no todos los plugins entran perfectamente en una de estas dos categorías anteriores. Por ejemplo, el Network plugin, es capaz de enviar y recibir valores y además, se abre un socket a la inicialización y envío de los valores cuando los recibe y no se dispara al mismo tiempo en que los plugin de entrada empiezan a leer.

Diagrama COLLECTD



En este diagrama se puede observar el funcionamiento de la herramienta Collectd. Básicamente está constituida por plugins los cuales al habilitarlos generan diferentes acciones dependiendo del tipo que sean. Por ejemplo si se habilita el plugin rrdtool se pueden crear bases de datos RRD's y además definir en cada uno de ellas la cantidad de archivos RRAs con sus respectivos tamaño de almacenamiento, etc.

Para nuestro problema puntual, no nos serán de gran utilidad la mayoría de las funcionalidades capaces de implementar mediante los distintos plugins. Básicamente configuraremos el demonio para trabajar solamente con los necesarios que son el plugin rrdtool y el plugin Iptables los cuales se detallan a continuación.

Plugin rrdtool

Este plugin será utilizado para crear las diferentes bases RRD y sus respectivas RRAs asociadas. Collectd se encargará luego de almacenar en estas RRD la información obtenida mediante el plugin Iptables, el cual a su vez es explicado más adelante.

La configuración que se debe realizar en este plugin es definir el step, el heartbeat y las diferentes RRAs que se van a guardar en cada base de datos. Para ver más detalle sobre la configuración de este plugin, ver ANEXO de Instalación y configuración.
A continuación se muestra un ejemplo de 3 RRAs para terminar de entender las diferencias entre los parámetros que se deben tener en cuenta al crearlas.
El Step se seteó en 10 segundos.

RRA	Function	pdp_per_row	rows	Timespan (seg)	Resolution
0	avg/min/máx	1	360	1 hora	10 segundos
1	avg/min/máx	90	96	1 día	15 minutos
2	avg/min/máx	180	336	1 semana	30 minutos

El significado de cada uno de estos parámetros fue explicado en el punto 6.1.2.

Plugin rrdtool

Este plugin está desarrollado para poder guardar en bases de datos el incremento de paquetes y bytes por intervalo de tiempo de cada protocolo que machea con las reglas agregadas en Iptables.

Utiliza la librería libiptc, lo cual significa que el demonio habla directamente con el kernel. Esta librería es la misma que utiliza Iptables para modificar y consultar las cadenas, reglas y los contadores de paquetes y bytes.

Al habilitar este plugin, se comienza a guardar la información mencionada anteriormente en las bases de datos RRD creadas por el plugin rrdtool. La configuración es muy simple, alcanza con agregar las diferentes tablas y cadenas de las cuales se quiere guardar información. Para nuestro caso en particular solamente necesitamos guardar la información de la tabla mangle y de la cadena
PREROUTING, por lo cual debemos agregar la siguiente línea: Chain mangle PREROUTING.

Por cada tabla y cadena configurada en este plugin, automáticamente se crea una carpeta. Dentro de estas carpetas se guardan las dos bases de datos creadas para cada protocolo ingresado en las reglas de Iptables. Una de estas bases contendrá el

incremento de bytes/segundos y la otra, el incremento de paquetes/segundos.

Es importante que cuando se ingresen las reglas de Iptables, se agregue el "comment" haciendo referencia a dicho protocolo para luego poder discriminar en cuales RRDs se encuentran los datos asociados al mismo.

A continuación se muestra un ejemplo tomando los protocolos bittorrent y http de cómo se deben ingresar las reglas:

Iptables -t mangle -A PREROUTING -m layer7 - -l7proto http -m comment - -comment "http-Prerouting"
Iptables -t mangle -A PREROUTING -m layer7 - -l7proto bittorrent -m comment - -comment "bittorrent-Prerouting"

Para este ejemplo, las RRDs creadas son las siguientes:

- ipt_bytes-Prerouting-http.rrd
- ipt_packets-Prerouting-http.rrd
- ipt_bytes-Prerouting-bittorrent.rrd
- ipt_packets-Prerouting-bittorrent.rr

Si se ejecuta en un terminal el comando rrdtool info nombre-base.rrd se puede ver como fue creada esta base de datos y las diferentes RRAs asociadas. Estos parámetros coinciden con la configuración realizada en el plugin rrdtool. Además se puede observar el tipo de datos que guarda que es COUNTER, lo cual hace que los valores que se guardan sea el incremento de cada protocolo en el tiempo tanto en bytes como en paquetes.

5.3. Módulo para la representación gráfica de los datos obtenidos

Como se mencionó anteriormente, nuestro sistema está compuesto de tres grandes módulos. Cada uno de ellos contiene funcionalidades bien definidas pero a su vez interactúan todos entre sí. Para la representación gráfica de la información obtenida en el módulo anterior necesitábamos una herramienta capaz de graficar los archivos .rrd creados por collectd, debido a que esta última herramienta no incluye esta funcionalidad. Durante esta etapa se estudiaron varias herramientas que cumplieran con esas características, llegando a la conclusión que la herramienta que se adapta mejor es Cacti [25].

Cacti es una herramienta diseñada para aprovechar el poder de almacenamiento y la funcionalidad de graficar que poseen las RRDtool. Esta herramienta es desarrollada en PHP y provee un pooler ágil, plantillas de gráficos avanzadas, múltiples métodos para la recopilación de datos, manejo de usuarios, y una interfaz de usuario muy fácil de usar. Los métodos más utilizados para obtener los datos a graficar son mediante Scripts externos o consultas SNMP. Sin embargo, también se puede indicar la ruta en donde se encuentran las bases RRD existentes que contienen los datos a ser graficados en diferentes períodos de tiempo. De esta última manera es como necesitamos que trabaje en nuestro caso, debido a que Collectd ya crea esas bases de datos.

Otra de las grandes ventajas que presenta Cacti, es la posibilidad de aplicar funciones matemáticas a los datos para ser visualizados de diferentes maneras. Ya cuenta con varias funciones muy útiles llamadas CDEFs y además se pueden crear nuevas. Para poder graficar el porcentaje de cada protocolo con respecto al total del tráfico, fue necesario crear CDEFs de manera de realizar los cálculos correspondientes sobre cada uno de ellos. Cacti tiene la posibilidad de crear distintos Templates para datos, gráficos y host, y utilizarlos para todos los datos que cumplan con las mismas características. Esto hace que la configuración para agregar nuevos protocolos a visualizar, sea muy sencilla y rápida. Simplemente se crea una nueva fuente de datos, indicando la ruta en donde se encuentra la base de datos asociada y se seleccionan los diferentes Templates ya definidos anteriormente para los demás protocolos.

Si necesitamos cambiar alguno de los parámetros configurados, alcanza con modificar los templates y esos cambios serán aplicados a todos los datos que tengan asociados dichos templates. Esto último es una gran ventaja dada la cantidad de protocolos que se pueden llegar a querer visualizar.
Otra característica que presenta Cacti es la visualización en forma de árbol, lo cual es muy útil si se quiere dar un orden jerárquico. Además, cuenta con la funcionalidad de manejo de usuarios haciendo posible agregar usuarios con permisos para modificar solamente ciertas áreas de Cacti o que solamente puedan ver las gráficas, etc.

Con lo dicho anteriormente, se cumplen los requerimientos que se había especificado a la hora de seleccionar la herramienta para este módulo.

Para configurar los diferentes Templates, se deben tener en cuenta varios parámetros nombrados a continuación:

Tal como se detalló durante la explicación del módulo de recolección y almacenamiento de los datos, se pueden definir varios archivos RRAs configurando el plugin
rrdtool de Collectd. Esto hace que dentro de cada base de datos, sea posible guardar la información en diferentes períodos de tiempo.
Tomando como ejemplo el protocolo bittorrent, si ejecutamos el comando rrdtool info en un terminal, se despliega toda la información de dicha RRD y las RRAs asociadas:

rrdtool info "ipt_packets-bittorrent-PREROUTING.rrd"
filename = "ipt_packets-bittorrent-PREROUTING.rrd"
rrd_version = "0003"
step = 10
last_update = 1268507108
ds[value].type = "COUNTER"
ds[value].minimal_heartbeat = 20
ds[value].min = 0.0000000000e+00
ds[value].max = 1.3421772800e+08
ds[value].last_ds = "0"
ds[value].value = 0.0000000000e+00
ds[value].unknown_sec = 0
rra[0].cf = "AVERAGE"
rra[0].rows = 1200
rra[0].pdp_per_row = 1
rra[0].xff = 1.0000000000e-01
rra[0].cdp_prep[0].value = NaN
rra[0].cdp_prep[0].unknown_datapoints = 0
rra[1].cf = "MIN"
rra[1].rows = 1200
rra[1].pdp_per_row = 1

Los parámetros a tener en cuenta para configurar el Data template para nuestros datos son:

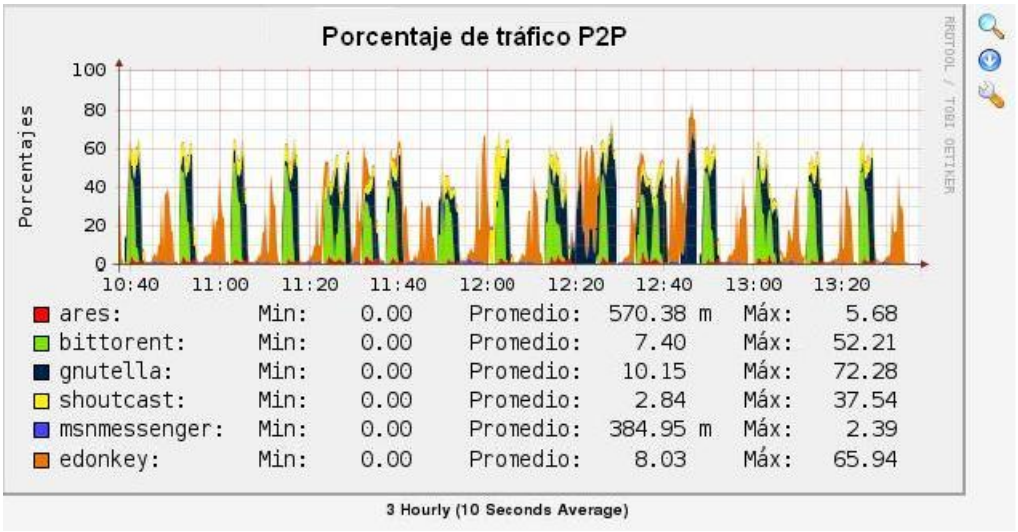
- cada canto tiempo se guarda un nuevo dato (Step).
- el heartbeat .
- el tipo de datos que se guardan (COUNTER).

- el DS (value).

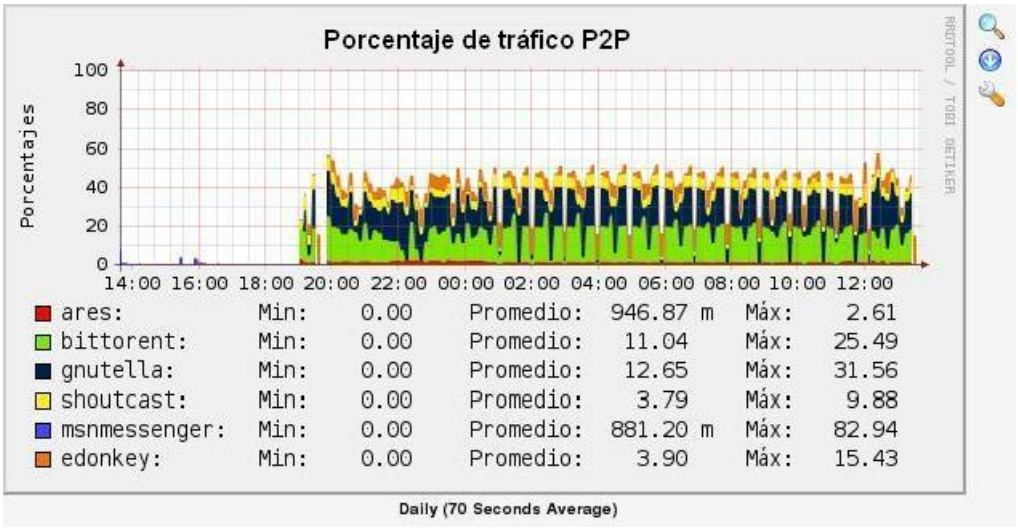
Para configurar los template asociados a las diferentes gráficas, se deben tener en cuenta:

- las funciones de consolidación (AVERAGE, MIN y MAX).
- las RRAs asociadas a las bases de datos: cada cuántas muestras aplicar dicha función de consolidación (pdp_per_row), cuántos de estos valores guardar en cada RRA (rows), etc.

A continuación se muestran dos gráficas, en las cuales se ve el porcentaje de tráfico p2p en relación al tráfico total. La diferencia entre ellas es el tiempo en que se están observando los datos. En la primera se visualiza una muestra cada 10 segundos durante un total de 3 horas y en la segunda, el promedio de cada 7 muestras en un intervalo de tiempo igual a un 1 día.



En la siguiente gráfica, se puede observar dicho porcentaje durante un día, promediando cada 7 muestras.



Para más información sobre la configuración de esta etapa se puede consultar el ANEXO de Instalación y configuración, donde se detalla paso a paso la configuración a realizar en cada herramienta.

5.4. Resumen

Como ya se ha mencionado en varias oportunidades, la base de este sistema MCT-L7 es el Framework de Netfilter. Gracias a Netfilter se pueden interceptar y manipular los paquetes de la red en los diferentes estados de procesamiento.

Para poder seguir las conexiones e identificar los distintos flujos de la red, utilizamos el connection tracking system, conocido como sistema de seguimiento de conexiones que es otro subsistema que ofrece el proyecto de Netfilter. Este último se utiliza, como bien lo dice el nombre, para seguir las conexiones y así permitir al núcleo llevar la cuenta de todas las conexiones o sesiones lógicas de red para de este modo, relacionar todos los paquetes que puedan llegar a formar parte de esa misma conexión.

Gracias al módulo especial L7-Filter, se puede analizar el payload de los paquetes de un flujo determinado e identificarlo con algún protocolo y en caso de ser así, se guarda esta marca en las tablas de conntrack para asociar los restantes paquetes de esa conexión con ese protocolo sin tener que seguir analizando el payload de cada uno de ellos.

Para poder llevar la información de la cantidad de paquetes y bytes de cada protocolo obtenida durante el seguimiento de conexiones, se usan las Iptables. Al ingresar las reglas de Iptables, solamente se tomó en cuenta la tabla mangle debido que es la tabla en la cual se pueden manipular los paquetes y la cadena PREROUTING ya que es por la cual pasan todos los paquetes sin excepciones.

Para recolectar y guardar los datos sobre los contadores de cada protocolo, se utiliza la herramienta Collectd, la cual consulta periódicamente los contadores que va llevando Iptables y los guarda en bases de datos RRDs.

Para graficar se utilizó la herramienta Cacti la cual consulta los valores guardados en las bases de datos creadas por Collectd y los despliega en pantalla.

6. Verificación del funcionamiento de MCT-L7

Hasta aquí contábamos con nuestra herramienta funcionando con los tres módulos en conjunto y preparada para ser puesta en funcionamiento. Sin embargo, no teníamos un conocimiento claro de las capacidades del sistema completo, sino que sólo contábamos con una idea aproximada de los requerimientos de las herramientas obtenidas de las distintas especificaciones descriptas en sus documentaciones respectivas. Por esta razón es que realizaremos una serie de pruebas para poder obtener datos más específicos sobre las diferentes capacidades del MCT-L7.

Las pruebas se basan principalmente en corroborar cuáles son las limitaciones del módulo de captura y análisis, el cual no sólo es de primordial importancia, sino que es el que debe presentar mayor estabilidad. El módulo de almacenamiento en base de datos también es muy importante, debido a que errores en este punto provocaría errores en la información que luego se desplegará. Sin embargo, por los estudios en la documentación de las herramientas de este módulo, estas no requieren un gran consumo de recursos, ya que que sólo entrarán en acción para guardar datos en determinados períodos de tiempo. El módulo de representación gráfica consideramos que no era vital para el correcto funcionamiento del sistema ya que en una primera instancia esta herramienta estaba diseñada para trabajar de forma pasiva sobre los datos. Considerando que la herramienta está diseñada para trabajar sobre una red Ethernet, tenemos que obligatoriamente considerar las ráfagas de tráfico características de esta tecnología. Por esta razón, ante la probabilidad de aparición de ráfagas de tráfico que alcancen a saturar al sistema, es más importante que el sistema atienda los pedidos de recursos del módulo de análisis y del módulo de almacenamiento, quitando recursos al módulo gráfico. Esto está pensado así ya que, aunque perdamos respuesta del módulo gráfico por unos instantes, los datos no se perderán en el sistema. Luego de estas ráfagas y de la estabilización de la red, el módulo podrá desplegar los datos en forma correcta de todas formas.

Por la propia arquitectura del sistema y la decisión de ubicar el módulo de análisis dentro del propio kernel del sistema operativo, estamos priorizando a éste frente al resto de los módulos del sistema. Por estas razones es que enfocamos la mayoría de las pruebas a verificar las respuestas que este módulo tiene frente a distintas situaciones.

Analizamos en una primera instancia, cuál era la capacidad de este módulo de poder reconocer en forma correcta distintos tipos de tráfico y cómo era el comportamiento de los distintos patrones. Deseábamos tener una idea de qué es lo que realmente reconocen cuando se establece el flujo y qué es lo que necesitan ver de éste para identificar con buena precisión, e hicimos principal hincapié en los protocolos peer-to-peer ya que son la motivación principal del desarrollo de este sistema. Como segunda instancia intentamos dar datos sobre las limitaciones del sistema frente a distintos niveles de carga, y cuánto afectaban en la performance

la variación de los diferentes parámetros del sistema. Aquí probamos con distintos equipos y con distintas configuraciones de parámetros en el sistema operativo para finalmente dar recomendaciones de los puntos que generan el cuello de botella en el sistema.

6.1. Consideraciones previas

L7_Filter posee una gran variedad de patrones creados por sus desarrolladores y por aportes de distintos contribuyentes de la red, no olvidemos que nos enfocamos en trabajar con herramientas open source. Por esta razón es que la base de datos con los patrones se ha creado con el aporte de muchos colaboradores, los cuales generan los distintos patrones buscando e inspeccionando en las distintas trazas generadas por los protocolos, y buscando características que diferencien a éste del resto de las aplicaciones. El hecho de que los patrones se creen con el aporte de los contribuyente de la red, sumado a que muchas veces el código de la aplicación se mantiene bajo licencia sin que se cuente con la documentación adecuada (por ejemplos RFC's, etc.), genera que los patrones no siempre sean confiables o que no reconozcan exactamente todo el tráfico de una aplicación. Por esta razón es que los desarrolladores del módulo de L7-Filter decidieron generar una serie de categorías que definan las características del patrón, entre las que se encuentran la rapidez con la que reconoce, la precisión del mismo para reconocer en forma acertada o en confundirse en la identificación, generando falsos positivos. Debido a estas causas es que intentamos comprobar estas especificaciones, estudiando el comportamiento que tienen algunos patrones. Generamos algunas trazas controladas para poder dar detalles y contrastar con los desarrolladores de L7-Filter los resultados obtenidos. Intentamos decir también para qué casos el patrón, de una determinada aplicación se comporta en forma correcta y explicar brevemente qué es lo que busca dentro del flujo.

L7_Filter posee actualmente 110 patrones generales, los cuales pueden agruparse en distintas familias y son los que se recomiendan cuando se utiliza este módulo . A su vez también existen otros patrones de menor interés y algunos patrones especiales para reconocer tipos de archivos que se transportan sobre HTTP y FTP, como ser archivos perl, winRAR,flash, zip ,.pdf, mp3 entre otros. Estos tipos de patrones si bien pueden ser de interés en algunos casos, no los consideramos aquí, ya que necesitan un tratamiento distinto del resto y bajan la performace del sistema en gran medida. Nos enfocamos en algunos de los patrones más generales y más utilizados sobre Internet como FTP, SMTP, HTTP etc, protocolos de VoIP y algunos de los protocolos P2P y de mensajería más utilizados en estos días.

Los protocolos elegidos para trabajar fueron los siguientes:

P2P	VoIP y Mensajería	Tradicionales
Bittorrent	SIP y RTP	SSH
Gnutella	Skype	FTP
eDonkey	IRC	HTTP
Shoutcast	Msn messenger	TELNET
Ares		VNC
		POP3 y SMTP

Para la descripción de un patrón se definen una serie de características que muestran las distintas cualidades en la eficiencia de este, entre ellas se encuentra la velocidad de reconocimiento, qué porción del tráfico reconoce, etc. Se define un parámetro de calidad del patrón el cual especifica que tan bien se entiende el protocolo (no olvidemos que no todas las aplicaciones tienen disponible su código y se vuelve difícil rearmar el flujo de datos o la estructura de comunicación que éste usa), hasta qué punto ha sido testeado, en cuáles situaciones y qué fracción del tráfico de la aplicación puede ser reconocida en forma correcta. El otro parámetro de interés, es la velocidad con la que puede reconocer cada patrón, con esto nos podemos beneficiar colocando los patrones más rápidos al inicio de la tabla de reglas, ahorrando recursos y ganando en eficiencia cuando se trabaja con grandes cantidades de tráfico (para una descripción de estos parámetros ver Anexo A).

6.2. Herramientas complementarias utilizadas

Durante las distintas pruebas que realizamos, fueron necesarias una serie de herramientas que nos permitieron generar trazas definiendo escenarios controlados, verificar la correcta caracterización de las mismas y realizar pruebas de carga para poder estimar cuáles son las condiciones extremas que pueden ser soportadas.

El procedimiento general de las pruebas fue generar, en un ambiente controlado, trazas de distintas aplicaciones. El objetivo de esto fue tener un punto de comparación para poder afirmar con bajo nivel de incertidumbre si nuestra herramienta caracteriza correctamente los protocolos elegidos. Tuvimos especial cuidado en evitar que dentro de una misma traza se encuentren flujos generados por distintas aplicaciones. Sin embargo lo anterior no fue posible en algunos casos como se indicará más adelante. Dada la variedad de protocolos que debimos generar, fue necesario instalar y configurar distintos clientes y servidores, como por ejemplo al momento de generar tráfico SIP, tuvimos que instalar una central SIP y capturar el tráfico intercambiado con dos softphones. De forma similar debimos proceder para cada uno de los protocolos y dependiendo de la manera en que trabajaba cada uno de ellos, esperábamos distintos niveles confiabilidad en las trazas. A continuación indicamos cuáles fueron las herramientas utilizadas.

Para el tratamiento de las trazas se utilizaron las siguientes herramientas:

- **Wireshark:** Es un analizador de protocolos que provee una funcionalidad similar a la de tcpdump, a su vez agrega una interfaz gráfica y otras opciones para organizar y filtrar la información. Es posible monitorear todo el tráfico que pasa a través de una red o examinar una traza de captura previa guardada en disco. Se puede analizar la información capturada, dando reportes y sumarios por cada paquete. Incluye una librería con una gran variedad de filtros para ver en detalle lo que nos interesa y la posibilidad de reconstruir un flujo de datos de una sesión TCP o UDP [26].

- **Tcpdump:** Esta herramienta permite esencialmente al usuario, capturar los paquetes que circulan por una red para ser luego analizados con otras herramientas. Este sistema se basa en la librería libpcap de Linux (la librería correspondiente para Windows sería winpcap). Con esta herramienta fue posible capturar los distintos tráficos utilizando una serie de filtros, para guardar luego los datos de los paquetes en un archivo en disco. Con estos datos guardados es que pudimos capturar y analizar en forma amigable el tráfico, utilizando luego esta información para la descripción de los patrones. Se utilizaron también una serie de herramientas de Tcpdump que permiten modificar la información de la traza (aleatorizar las IP, cambiar puertos entre otras posibilidades) así también como reinsertar en la red, de forma controlada, las trazas guardadas en disco [27].

- **Bit-Twist:** Utilizamos esta herramienta para simular tráfico en la red. Esta herramienta es un potente generador de paquetes Ethernet basada en libpcap, diseñado para complementar a la herramienta tcpdump. Bittwist permite regenerar el tráfico capturado y guardado en disco, volviéndolo a insertar en la red. Además contiene un editor de archivo de seguimiento que le permite cambiar el contenido de un archivo de una traza. Presenta similitudes con las herramientas adicionales que posee Tcpdump [28].

- **OpenDPI:** Esta herramienta posee una biblioteca de software diseñada para clasificar el tráfico de Internet de acuerdo a aplicaciones, realizando una inspección profunda de paquetes (DPI) sobre el payload del tráfico. OpenDPI es capaz de levantar una traza guardada en disco y analizar el contenido de ésta en busca de distintas aplicaciones. Sin embargo la gran limitante de este sistema es que no puede analizar el tráfico en tiempo real, solo puede levantar datos desde una traza. Se utilizó a OpenDPI para comparar los resultados con los obtenidos con la nuestra y tener ideas de cuán bien reconocía [29].

- **HTTPperf:** Esta herramienta fue diseñada para poder medir el rendimiento de un servidor web, mediante el envío de una tasa fija de peticiones al servidor para poder medir su rendimiento. Para nuestros propósitos, esta herramienta nos permitió abrir una gran cantidad de conexiones TCP para

simular un determinado número de usuarios. Nos permitió llenar la tabla de conntrack con una gran cantidad de conexiones simulando que se encuentran conectados muchos usuarios [30].

- **APACHE:** Es un servidor web , lo utilizamos para en conjunto con HTTP-perf para poder simular gran cantidad de conexiones [31].
- **Conntrack-tools:** Es una herramienta desarrollada a nivel de la capa de usuario para poder tener control sobre el módulo de conntrack. Nos permitió ver de una forma amigable los flujos establecidos, así también como operar sobre el conntrack [32].

Aparte de estas herramientas utilizadas a lo largo de todas las pruebas, también se utilizaron otras para generar trazas en situaciones controladas, por ejemplo se utilizaron servidores simulando centrales SIP (simulador 3CX), se utilizaron servidores FTP, servidores web, aplicaciones de acceso remoto, etc.

6.3. Pruebas Realizadas

Aparte de estas herramientas utilizadas a lo largo de todas las pruebas, también se utilizaron otras para generar trazas en situaciones controladas, por ejemplo se utilizaron servidores simulando centrales SIP (simulador 3CX), se utilizaron servidores FTP, servidores web, aplicaciones de acceso remoto, etc.

6.3.1. Verificación del funcionamiento en modo pasivo

En esta etapa de las pruebas verificamos el correcto funcionamiento del módulo de captura y análisis trabajando en modo pasivo, escuchando y analizando todo el tráfico de la red. Fue necesario comprobar esto en una primera etapa para asegurar que al colocar el sistema en la boca del port-mirror de un switch, no cambiaría en nada el proceso de análisis de las herramientas que intervienen en el módulo. Una vez que esto estuvo asegurado, pudimos pasar a las siguientes etapas y trabajar sobre el correcto funcionamiento de los patrones y las características que buscan, así también como el nivel de performance para diferentes niveles de tráfico.

Las herramientas que integran este módulo están diseñadas para trabajar insertas en el tráfico, configurando el sistema como router y ubicándolo en la frontera de la red. Es por esto que nos vimos obligados a utilizar el bridge (entre otras cosas), para lograr que Netfilter pueda levantar los paquetes, pero sin conocer a priori si el resto de las herramientas (Iptables , L7-Filter o conntrack) continuaban funcionando en forma correcta. Lo que nos propusimos realizar fue recrear dos topologías de red insertando MCT-L7 en distintas posiciones, verificando que capture en forma correcta ya sea en forma pasiva o en forma activa.

- En primera instancia, se utilizó una arquitectura de red en donde se ubicó nuestra herramienta en el medio del tráfico y se generaron de forma controlada, las trazas de distintos protocolos de aplicación guardando una copia en disco de la misma.
- Para la segunda instancia de esta prueba, se realizó una arquitectura de red en la cual la PC con MCT-L7 se encontraba escuchando en forma pasiva, sin interferir en los datos, en donde le llegaban copias del tráfico que estaba circulando por la red. En esta etapa, se utilizaron las trazas guardadas de la etapa anterior insertándolas en la red mediante la herramienta Bit-Twist.

A continuación se describen las topologías utilizadas

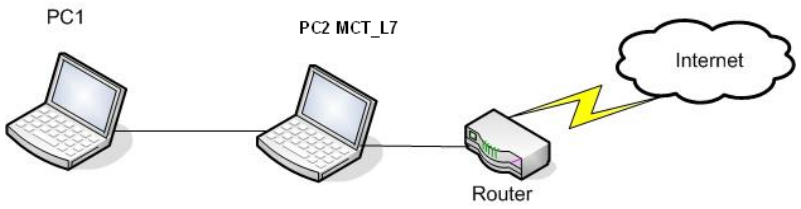
Topología 1

En esta etapa generamos las trazas para los diferentes protocolos, utilizando MCT-L7 en medio del tráfico. Con esta topología podemos asegurar que el módulo de análisis no sólo va a capturar el tráfico, sino que podrá analizarlo en forma correcta. Utilizaremos éstos datos para contrastar luego con los obtenidos en la siguiente topología.

Si bien en esta instancia no deseábamos corroborar la veracidad de los resultados del módulo L7-Filter, sí debíamos asegurar que los paquetes capturados pertenecieran en su mayoría al flujo de una determinada aplicación, y que el patrón de L7-Filter reconociera en forma correcta qué tipo de tráfico había en la red. Por esta razón, generamos tráfico utilizando aplicaciones tradicionales y fácilmente detectables por cualquier sistema. Para esta etapa decidimos utilizar los datos de las trazas generadas con los protocolos ssh, vnc y telnet. Se escogieron estas trazas como referencia o como trazas patrones debido a que L7-Filter asegura que reconoce fácilmente estos protocolos y con poca probabilidad de error. Otra ventaja de estos protocolos es que si trabajan en su puerto por defecto pueden ser también analizados por wireshark y OpenDPI, asegurando que los resultados del módulo sean correctos en ambas topologías.

Además de estas trazas, se generaron otros protocolos para comparar los resultados y la coincidencia de los datos, sin embargo para dar el visto bueno de cuál era la configuración necesaria nos basamos en las tres trazas mencionadas.

La figura a continuación muestra como fue la topología con MCT-L7 analizando en medio del tráfico y trabajando como router, no como bridge (como acotación: debimos setear la variable ip_forward del kernel en 1 para trabajar de esta manera).

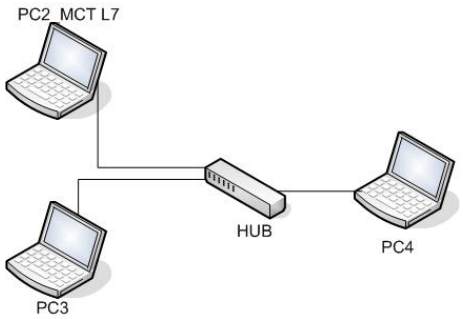


En algunos de los casos , nos vimos en la necesidad de acceder a Internet para poder generar trazas, en los casos que no era necesario, se eliminó esta salida de forma de mantener lo más controlado posible los distintos flujos de datos existentes.

La PC1 fue la máquina encargada de generar las trazas utilizando una o varias PC's auxiliares para levantar las aplicaciones (por ejemplo con SIP utilizamos un servidor proxy SIP en la PC1 y dos máquinas auxiliares con el cliente sip).

Topología 2

El procedimiento para detectar si la herramienta funciona correctamente en modo pasivo, se realizó con la arquitectura de red que se muestra en la siguiente figura:



Para simular en un principio la boca de mirror del switch optamos por utilizar un Hub, el cual copia en todos sus puertos el tráfico que le llega. Se conectaron todas las PC's a dicho Hub y se utilizaron las trazas generadas durante la actividad 1 con el fin de poder probar el funcionamiento de MCT-L7 trabajando en modo pasivo (con el bridge configurado). Las PC3 y PC4 son las que insertan nuevamente las trazas capturada en la instancia previa y guardadas en disco, utilizando para esto la herramienta Bit-Twist.

Para que los paquetes enviados logren atravesar el bridge es necesario, que ninguno de ellos tenga como direcciones MAC origen o destino las interfaces de dicha PC (recordemos que en la instancia anterior la maquina con MCT-L7 trabajaba como router, con lo cual los paquetes eran enviados a ella directamente). Esto

se debe a que a pesar de setear en cero el tiempo que se mantiene la información en la tabla del bridge, siempre existen en la tabla las entradas correspondientes a las interfaces locales (las direcciones MAC de eth0 y eth1).

Una observación previa al analizar las trazas, es que en muchas casos se puede ver que los paquetes contabilizados por L7-Filter como unknown más los reconocidos en las reglas no suman la totalidad de paquetes de la traza. Esto no es un error en el sistema sino que está asociado a la manera en que L7-Filter define como unknown un flujo. Los primeros paquetes de un flujo no pueden ser contabilizados debido a que L7 filter no los ha asociado a ningún protocolo, si éste falla en caracterizar el flujo, entonces el resto de los paquetes de la conexión sí los identificará como unknown.

Protocolo SSH

Para generar este protocolo se realizó una conexión SSH con un servidor remoto y se genero una traza.

A continuación se presentan los resultados para ambas trazas.

Resultados Obtenidos

Traza 1: Se incluye la conexión, 276 paquetes

Protocolo	Topología 1	Topología 2	OpenDPI	Wireshark
SSH	269	269	268	188
unknown/TCP	0	0	7	88

Protocolo Telnet

Para probar el comportamiento del patrón para este protocolo, se generaron dos trazas, una donde se estableció una sesión telnet utilizando el puerto estándar definido por este protocolo, y otra donde se usó un puerto cualquiera. El objetivo por el cual se cambió el puerto estándar por otro, fue probar que nuestra herramienta era capaz de determinar el protocolo de todas formas y hacer notar la diferencia con los datos que obtiene wireshark inspeccionando únicamente los puertos utilizados.

Resultados Obtenidos

Traza 1: Puerto 23, 199 paquetes

Protocolo	Topología 1	Topología 2	OpenDPI	Wireshark
Telnet	190	190	187	119
dns	6	6	0	6
unknown/TCP	0	0	6	74

Traza 2: Puerto 100, 222 paquetes

Protocolo	Topología 1	Topología 2	OpenDPI	Wireshark
Telnet	208	208	205	0
dns	11	11	6	11
unknown	0	4	11	77
TCP-data	—	—	—	134

Protocolo VNC

Utilizamos para la generación de este protocolo una configuración sencilla de dos máquinas con un programa de escritorio remoto, abrimos una sesión entre las 2 computadoras y capturamos el tráfico generado.

Resultados Obtenidos

Traza 1: 22821 paquetes

Protocolo	Topología 1	Topología 2	OpenDPI	Wireshark
VNC	22818	22818	22817	19625
unknown	0	0	4	327

Conclusiones de esta etapa

Si bien generamos otros protocolos y los testeamos en ambas topologías funcionando correctamente, elegimos a estos tres como referencia porque son en principio sumamente fácil de generarlos de modo controlado, evitando la introducción de flujos de otras aplicaciones. Podemos corroborar que el sistema cumple con el comportamiento esperado según los resultados obtenidos. Demostramos que configurando el sistema como describimos en secciones anteriores es posible trabajar en forma pasiva. No sólo el sistema Netfilter puede levantar los paquetes en forma correcta sino que también L7-Filter puede analizarlos y conntrack es capaz de ingresar a su tabla los distintos flujos presentes en la traza. La otra prueba fehaciente del buen funcionamiento de L7-Filter es al cambiar el puerto en que trabaja el protocolo Telnet, observamos que sólo MCT-L7 y OpenDPI (ambas herramientas realizando DPI sobre los datos) pueden reconocer el protocolo sin importar el

puerto (como era de esperar), mientras que wireshark ve a todo el tráfico como TCP.

Con estas pruebas podemos afirmar que es posible colocar la herramienta en un port-mirror sin afectar la correcta caracterización a través de los patrones, con lo cual estamos en condiciones de testear cuán bueno son éstos y qué es lo que realmente capturan en algunos de los casos. Esto se verá en la siguiente sección.

6.3.2. Estudio de los distintos patrones

Con los resultados obtenidos en la sección anterior estamos seguros que el comportamiento del sistema escuchando en un port-mirror, es equivalente al que presenta si este estuviera trabajando en el camino del flujo. Con esto establecido estamos en condiciones de comprobar el funcionamiento de los patrones y en cada uno de ellos estudiar qué es lo que reconocen y cuál es su exactitud a la hora de identificar un patrón en un flujo sin equivocarse de protocolo.
Como segunda instancia del estudio de los patrones y entendido su funcionamiento, intentaremos ver cuáles son las posibilidades de que alguno de los patrones produzcan circunstancias de falsos positivos o falsos negativos.

Los protocolos los separamos en tres grandes grupos: protocolos de VoIP, tradicionales y bien comportados, y por último, aplicaciones p2p. Los protocolos que despiertan nuestro mayor interés a estudiar son los identificados con los grupos de VoIP y p2p, debido a la popularidad que van adquiriendo estos protocolos con el correr del tiempo. Los protocolos VoIP vienen creciendo en funcionalidades y cada vez más empresas poseen herramientas de este tipo para desarrollar sus funciones. Estos protocolos poseen como limitante la cantidad de recursos que requieren para funcionar correctamente, y por lo tanto necesitan mantener bajo control determinados parámetros de la red (ancho de banda, delay, jitter, etc). El otro grupo de interés lo integran los protocolos p2p, los cuales están generando un gran dolor de cabeza tanto a los ISP's como a los administradores. Más detalles de estos protocolos p2p y los problemas o motivaciones por lo que se intentan mantenerlos bajo control están explicados mas adelante en esta sección.

Si bien los protocolos tradicionales son sumamente conocidos, existen diversas formas de reconocerlos por distintas técnicas (sobre todo debido a su buen comportamiento), en la mayoría de los casos se pueden reconocer por los puertos que utilizan, pero es necesario tener una idea lo más precisa posible de cuales son los mensajes que se intercambian en la capa de aplicación, esto es debido a que muchas de las aplicaciones p2p o skype intentan camuflarse con estos protocolos (la gran mayoría intenta camuflarse con HTTP). Debido a esto es que necesitamos conocer bien de cerca estos patrones para poder confiar en ellos cuando se revisen las características de las aplicaciones P2P.

Protocolos Tradicionales

Protocolo SSH

Para probar la veracidad de este protocolo se realizó una conexión SSH con un servidor remoto y se generaron dos trazas, una de ellas contiene el establecimiento de la conexión y la otra no.

A continuación se presentan los resultados para ambas trazas.

Resultados Obtenidos

Traza 1: Se incluye la conexión, 276 paquetes

Protocolo	MCT-L7	OpenDPI	Wireshark
SSH	269	268	188
unknown	0	7	88

Traza 2: No se incluye la conexión, 110 paquetes

Protocolo	MCT-L7	OpenDPI	Wireshark
SSH	0	0	76
unknown	99	109	34

Protocolo Telnet
Resultados Obtenidos

Traza 1: Puerto 23, 199 paquetes

Protocolo	MCT-L7	OpenDPI	Wireshark
Telnet	190	187	119
unknown	0	0	74

Traza 2: Puerto 100, 222 paquetes

Protocolo	MCT-L7	OpenDPI	Wireshark
Telnet	208	205	0
unknown	0	6	77
TCP-data	—	—	134

Protocolo HTTP

Conocer el comportamiento de eco aplicación y saber como actúa su patrón fue de gran utilidad cuando trabajamos con los protocolos p2p debido al comportamiento que éstos presentan. Según los detalles brindados por el equipo de desarrollo de L7-Filter, catalogan a este patrón como superset, es decir que el patrón podría confundirse y caracterizar algún flujo como http sin serlo realmente.

Resultados Obtenidos

Traza: 1215 paquetes

Protocolo	MCT-L7	OpenDPI	Wireshark
ssl	30	30	14
HTTP	1136	1150	336
unknown	0	35	865

Protocolo VNC

Resultados Obtenidos

Traza 1: 22821 paquetes

Protocolo	MCT-L7	OpenDPI	Wireshark
VNC	22818	22817	19625
unknown	0	4	3196

Protocolos de Correo

Protocolo POP3

Resultados Obtenidos

Traza 1: 8074 paquetes

Protocolo	MCT-L7	OpenDPI	Wireshark
POP3	8065	8059	4859
unknown	0	11	3215

Protocolo SMTP

Resultados Obtenidos

Traza: 5602 paquetes

Protocolo	MCT-L7	OpenDPI	Wireshark
SMTP	5592	5587	2788
unknown	0	8	2814

Conclusiones

Los patrones para estos protocolos funcionan correctamente según los clasifica L7-Filter, por lo que era de esperar que en las pruebas realizadas también se hayan clasificado correctamente. La prueba en donde se quita el inicio de la conexión de un protocolo fue realizada para varios patrones (como ser POP3· y SMTP) obteniéndose resultados similares. En estos casos donde se eliminan éstos paquetes iniciales, las expresiones regulares buscadas no serán vistas, marcando al flujo como desconocido. Sólo mostramos los resultados para la traza de SSH porque el resto no adiciona ninguna información extra, solo confirman lo obtenido con ésta, lo mismo ocurre con el comportamiento con la traza de Telnet cuando se le cambia el puerto.

Protocolos de transferencia de archivos

Protocolo FTP

Este protocolo presenta comportamientos distintos dependiendo del modo en que se utilice, con lo cual se debe tratar en forma especial en el sistema. En los clientes actuales de FTP se puede trabajar de dos formas distintas frente al servidor. Uno de estos modos de conexión es el conocido como modo activo (modo tradicional de operación) , el cual trabaja mandando los parámetros de control por el puerto TCP 21 mientras que los datos se envían por el puerto TCP 20. En el otro modo de conexión, conocido como modo pasivo, se establece la conexión de control por el puerto 21 y sobre ésta se negocia un puerto aleatorio por el cual se intercambiarán los datos. El problema de esto es que con un sistema de reconocimiento DPI como el nuestro , si bien podemos reconocer la conexión de control, no habría forma en principio de reconocer la transferencia de los datos inspeccionando el payload, y para los casos en que el reconocimiento de las aplicaciones se realiza por puerto (Wireshark), es aún peor.

Para solucionar esto, el sistema de connection tracking utiliza una funcionalidad especial para manejar, reconocer y asociar estas conexiones a la conexión de control que las originó. Para manejar esto, el sistema de conntrack utiliza los llamados helpers (los cuales fueron explicados en la sección de connection tracking en el módulo de análisis). Para este protocolo en particular y el protocolo TFTP (similar a FTP) se utilizan los siguientes helpers.

- **nf_conntrack_ftp**
- **nf_conntrack_tftp**

Con estos helpers cargados (estos se cargan como módulos dentro del kernel), el módulo de análisis puede asociar las conexiones de transferencia de los datos en

los puertos aleatorios con sus respectivas conexiones de control y marcar el flujo sobre estos puertos como FTP.

Para corroborar este funcionamiento se generó una traza accediendo a un servidor ftp vía web, en el primer caso se cargaron los módulos anteriores y en el segundo no.

Resultados Obtenidos

Traza 1: FTP con módulos, 2977 paquetes

Protocolo	MCT-L7 con helpers	OpenDPI	Wireshark
FTP-control	2744	2744	41
FTP-datos	—	—	1702
DNS	12	12	12
HTTP	186	198	18
unknown	35	23	1204

Traza 2: FTP sin módulos, 2977 paquetes

Protocolo	MCT-L7 sin helpers	OpenDPI	Wireshark
FTP-control	47	2744	41
FTP-datos	—	—	1702
DNS	12	12	12
HTTP	182	198	18
unknown	2736	23	1204

Conclusiones generales de estos patrones Tradicionales

Con las pruebas realizadas y la variaciones hechas a las trazas, se pueden sacar conclusiones valiosas de determinados protocolos y también algunas conclusiones generales del sistema. En primera instancia podemos apreciar algunas de las ventajas de trabajar con la inspección DPI, en donde se pueden reconocer los flujos aunque estos intenten cambiar de puertos y despistar con estas tácticas. Esto se vio demostrado con la traza de Telnet, en donde comparamos los resultados de los datos cuando la aplicación trabaja en su puerto natural y cuando lo hace en un puerto elegido por nosotros al azar. Aquí vemos que las herramientas tradicionales, Wireshark en nuestro caso, reconocen bien el flujo cuando la aplicación Telnet corre sobre el puerto estándar 23, mientras que fallan en reconocer cuando éste cambia de puerto, cosa que no sucede con MCT-L7. Este protocolo sumamente conocido y de gran difusión podría sortear sin problemas un firewall que intente bloquearlo sólo con el simple hecho de que ambas puntas se pongan de acuerdo y trabajen en un puerto distinto al conocido (esto es generalmente lo que hacen los clientes p2p, investigan los puertos abiertos para traficar en éstos sin ser detectados).

Otra conclusión importante que se desprende de las pruebas es en realidad una desventaja de trabajar con estos sistemas DPI. Cuando trabajamos con el protocolo SSH, se eligió una sesión en la cual el sistema reconoció bien todo el tráfico, el siguiente paso fue tomar la misma traza pero quitándole los primeros paquetes, en los cuales se estableció la conexión. El resultado fue que MCT-L7 no asignó el flujo a ningún protocolo dándolo como desconocido, mientras que Wireshark asoció correctamente gran parte del tráfico. Esto se puede explicar de la siguiente manera: el patrón está creado para buscar determinadas expresiones que aparecen al inicio de la sesión ssh, éstas pueden ser palabras o valores hexadecimales en algún orden específico en el payload. Si se pierde alguno de los primeros paquetes del flujo puede suceder que el sistema no pueda asociar ninguna de las aplicaciones (debido a que perdió estas palabras claves) con los patrones cargados antes de llegar al número máximo de paquetes que tiene permitido analizar por flujo.
Si bien en situaciones normales el sistema debería tener siempre a su disposición la totalidad de los paquetes que circulan por la red, puede que se pierdan algunos de los paquetes del inicio de una conexión debido por ejemplo a ráfagas de tráfico en donde el sistema no sea capaz de manejar todo lo que ve perdiendo determinados paquetes vitales para la identificación de ese flujo.

El otro punto interesante que se puede sacar de las pruebas realizadas es la utilidad de los helpers de conntrack, en donde se pueden comparar los resultados obtenidos cuando se carga el módulo y cuando éste no esta en funcionamiento. Si bien se reconocen las conexiones de control y se reconocen los flujos de ésta, el sistema no puede relacionarlas con las conexiones “child”(flujos con los datos) que se originarán. Es por esto que los datos de estas conexiones no son reconocidos sin los

módulos. En contraste la herramienta wireshark reconoce en ambos test la misma cantidad de datos, dejando escapar aquellos datos pertenecientes a los flujos que no corren sobre los puertos estándar. Este comportamiento también se observará más adelante cuando estudiemos el comportamiento del protocolo SIP y las conexiones de voz RTP asociadas.

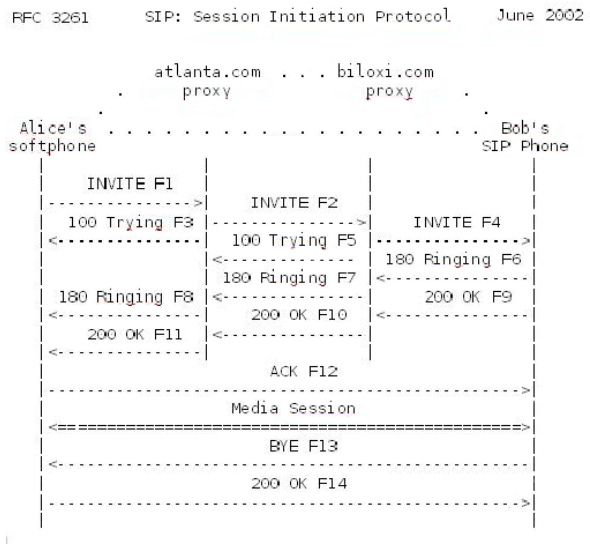
En general y como era de esperar para este apartado, los protocolos que podemos catalogar como tradicionales poseen un patrón muy confiable, el cuál puede identificar con gran precisión la aplicación correspondiente. Este comportamiento confiable por parte de los patrones de estas aplicaciones era lógico debido a la existencia de mucha documentación relativa a éstos (la mayoría están especificados en RFC's) y esto facilita la creación de expresiones regulares precisas.

Protocolo de VoIP

Protocolo SIP y RTP

SIP *(Session Initiation Protocol - Internet telephony)* es un protocolo desarrollado con la intención de ser el estándar para el establecimiento, modificación y finalización de sesiones interactivas de usuario donde intervienen elementos multimedia como: video, voz, mensajería instantánea, juegos en línea y realidad virtual. Se complementa con el protocolo RTP *(Real-time Transport Protocol)*, el cual es el verdadero portador del contenido de la voz y el video que intercambian los participantes en una sesión establecida por SIP. El protocolo SIP adopta el modelo cliente-servidor y es transaccional: el cliente realiza peticiones que el servidor atiende y genera una o más respuestas. El servidor responde ya sea rechazando o aceptando esa petición en una serie de respuestas que llevan un código de estado que brindan información acerca del estado de las peticiones, si éstas fueron resueltas con éxito o si se produjo un error.

Para su funcionamiento se utiliza un modelo transaccional de pedidos y respuestas similar al protocolo HTTP. Cada transacción consiste en un pedido a un servidor el cual devuelve al menos una respuesta. El ejemplo tradicional para explicar este protocolo es el siguiente (Incluido en la RFC's): la transacción comienza con un cliente Alice el cual envía un petición de INVITE a otro usuario Bob registrado en el proxy-server, el INVITE contiene una serie de datos utilizados para la localización, el tipo de sesión a inicializar,etc. Los proxy's server son los encargados de la localización de los distintos usuarios en la red, una vez que es localizado el usuario en la red se intenta entablar la sesión como una llamada local. Luego de establecida la señalización, se negocian los puertos y otros parámetros para dar paso al protocolo RTP, el cual es el encargado de transportar la voz. Estos establecimiento de apertura y de cierre de sesiones siguen una secuencia bien conocida de mensajes los cuales están definidos en sus correspondientes RFC's (RFC 3261 para sip y RFC 1889 para rtp).
A continuación se da un ejemplo gráfico del establecimiento de una sesión SIP y la utilización de RTP en la conversación.



Los mensajes enviados por SIP entre los clientes y el servidor siguen el siguiente formato:

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards:70
To: Bob <sip:bob@biloxi.com>
From:Alice<sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq:314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type:application/sdp
Content-Length: 142

Estos serán los formatos de mensajes que buscará el patrón dentro del payload para poder identificar un flujo con el protocolo SIP, buscando los mensajes REGISTER (pedidos de registros del cliente al servidor), INVITE y CANCEL (establecimiento de sesión), aunque existen otros mensajes definidos en SIP, se buscarán solo estos ya que son los que se dan al inicio de un flujo.

Para RTP existe otro patrón definido en L7-Filter, el cuál utiliza una expresión regular distinta, e identificará el flujo RTP pero sin asociarlo a la sesión sip establecida. El inconveniente que presenta este patrón es que los encabezados utilizados por RTP son muy cortos y compactos, lo que dificulta en gran medida crear una buena expresión regular. El patrón para este protocolo busca unos pocos bits que se repiten en todos los encabezados, sin embargo la mayoría de los bits del encabezados son números aleatorios que representan el índice de los paquetes, su timestamp , sincronización,etc. Este formato del encabezado hace que sea realmente difícil encontrar una buena expresión regular, y a su vez que ésta no sea muy precisa para

no generar falsos negativos.

Sin embargo, el protocolo RTP generalmente viene asociado con el establecimiento de una sesión SIP, con lo cual sería deseable poder asociar ambos protocolos cuando RTP deriva de una sesión SIP. Para esto se utiliza al igual que para el protocolo FTP, un módulo helper en el conntrack, lo cual ayuda a predecir cuál será el nuevo flujo negociado en los parámetros de la sesión SIP, a través del cual irá RTP llevando la voz de la llamada. Esto no solo logra asociar ambos flujos y etiquetarlos como SIP, sino que hace al protocolo más fácil y preciso en su detección.

Para probar este protocolo configuramos una red privada en donde levantamos un proxy sip y varios clientes (softphones). Se generó una traza la cual contiene todo el intercambio de datos incluyendo el registro de los teléfonos al servidor SIP y el corte de la llamada.
Para la traza capturada se comprobó el funcionamiento de ambos patrones y la utilización del helper del conntrack para asociar ambos flujos de datos en la tabla de connection tracking.

Modulo helpers : **nf_conntrack_sip**

Resultados Obtenidos

Traza 1: 1639 paquetes

	MCT_L7			
Protocolo	Sin módulos	Con módulos	OpenDPI	Wireshark
SIP	40	1506	0	40
RTP	1464	0	1455	1466
dns	16	16	42	42
unknown	24	24	51	9

Protocolo Skype

Skype es un protocolo peer-to-peer VoIP desarrollado en el 2003, el mismo fue creado para permitirle a los usuarios realizar llamadas de voz y de enviar mensajes de textos con otros clientes del Skype. En esencia es muy similar a otros protocolos de este tipo como ser MSN y aplicaciones de Yahoo en cuanto a las capacidades y servicio que este ofrece. Sin embargo las técnicas y protocolos con los que está desarrollado son diferentes, produciendo mejores resultados. La red de Skype está formada por dos entidades diferenciadas, unos llamados Super-nodes y los clientes. Los clientes, cuando quieren hacer uso de los recursos o simplemente anunciarse en la red, deben pasar por un proceso de login contra un servidor el cual los registrará y mantendrá en su base de datos por un tiempo determinado.

Cada cliente realiza en forma contínua una serie de acciones entre las que se encuentra, escuchar en puertos particulares esperando por llamadas entrantes, a su

vez mantiene una lista de las direcciones de algunos de los super-nodes que se encuentran en su "neighborhood", mantiene una lista de contactos, utiliza mensajes encriptados de end-to-end. Otra acción sumamente importante y por las cuales vuelve a este protocolo tan atractivo y un dolor de cabeza para los administradores, es su habilidad para detectar y evadir NAT's y firewall's. El protocolo no ha sido publicado por los propietarios de skype lo cual es una de las principales causas por las que sea sumamente complicado desarrollar patrones de reconocimientos. La principal diferencia entre Skype y otros protocolos skype es que skype opera su red como si fuera un protocolo peer-to-peer, en lugar de los modelos tradicionales de clientes-servidor de los protocolos de telefonía tradicionales. El directorio de los usuarios de la red de skype se encuentra totalmente descentralizado y distribuido a lo largo de todos los nodos de la red. Esta característica vuelve a este protocolo muy escalable y permite que crezca con facilidad sin costo de infraestructura.

El cliente skype no tiene un puerto default, sino que elije unos puertos TCP y UDP particulares en el que escuchará y recibirá las llamadas de otros clientes, estos puertos son elegidos al azar cuando se instala el cliente en el host, en este punto es donde se vuelve imposible reconocer este protocolo a nivel de las capas inferiores. Además de estos puertos, también utiliza los puertos 80 y 443 (HTTP y HTTPS respectivamente). Otra de las características de skype es la utilización de protocolos de encriptación para la voz (AES Advanced Encryption Standard) y para los paquetes de señalización (utiliza RC4 para estos mensajes). Todas estas características junto con la imposibilidad de de acceder al código, vuelven sumamente difícil el desarrollo de un patrón característico para este protocolo.

A pesar de todas estas características de evasión que presenta skype, se han desarrollado algunos patrones que puede reconocer más o menos bien estos flujos,. Existen para L7 dos tipos de patrones para reconocer los diferentes flujos de skype, uno dedicado a encontrar las llamadas generados entre clientes de skype, y otro para las llamadas de un cliente de skype hacia una PSTN o una red de telefonía externa. El patrón para las llamadas entre usuarios de skype busca un caractér general que se presenta cuando el usuario no está haciendo nada o cuando estácon una conversación activa, el inconveniente que presenta el patrón es la generalidad de éste, ya que otros flujos pueden ser macheados con éste, sin embargo es un protocolo muy rápido y reconoce con pocos paquetes y poco procesamiento si el flujo es skype o no.

Resultados Obtenidos

Traza : llamada entre dos clientes de skype, 42989 paquetes

Protocolo	MCTL_L7	OpenDPI	Wireshark
skypeout	31	—	0
skypetoskype	42078	—	0
UDP-data	—	—	42045
unknown	671	42913	926

Conclusiones protocolos VoIP

Para el caso de los protocolos sip y rtp observamos para que para nuestra red controlada y dedicada, ambos patrones pueden identificar en forma correcta los flujos de ambos protocolo. Para el caso en que se trabajo con el módulo cargado, se reconoció como si todos los paquetes caracterizados como RTP en la primera experiencia, en esta fueron reconocidos como partes de la sesión SIP por los helpers del conntrack. De no estar cargado el helper, de todas formas se puede reconocer en forma correcta tanto los paquetes pertenecientes a SIP como los de la voz, pertenecientes a RTP. Sin embargo en las especificaciones del patrón dados por el equipo de desarrollo, aclara que aunque la expresión regular para RTP no está profundamente testeada, por el contexto se puede asegurar que ésta posiblemente confunda ciertos flujos y se las asigne a este protocolo. Por estas razones es que recomiendan utilizar el helpers de contrack para reconocer este protocolo, trabajar de esta forma para reconocer trae una serie de ventajas:

- En primer lugar, para reconocer el protocolo RTP no se utiliza directamente la expresión regular especificada por el equipo de desarrollo de L7-Filter, la cual está catalogada como undermatch, sino que utiliza las utilidades del conntrack y al patrón de SIP ,el cual es mucho más específico, para reconocer y asociar flujos RTP.
- La otra ventaja, es el ahorro de recursos al utilizar el sistema para reconocer RTP a las conexiones pertenecientes a SIP e identificarla con las conexiones expected del conntrack. El procesamiento utilizado por el conntrack es sumamente menor que el utilizado por el módulo si tiene que buscar la expresión con el patrón dentro del payload.

Para el caso de skype, se puede ver que que el patrón puede reconocer bien estos flujos y prácticamente todos los paquetes fueron reconocidos con este protocolo. Esto demuestra que la efectividad del patrón es buena para machear en forma positiva los flujos que realmente son de skype. Pero qué pasa con el resto? La generalidad de este patrón hace que mucho del tráfico presente en una red de Internet caiga dentro de los parámetros que busca este patrón, por esto es que se producen problemas de falsos positivos, flujos reconocidos como skype que no son skype en realidad. Estos problemas son producidos por la generalidad del patrón(debido a

ser una aplicación bajo licencia),y por la poca cantidad de paquetes que necesita este para poder reconocer como positivo un flujo de skype. Debido a todo esto es que este patrón es bueno si se quiere reconocer la existencia del protocolo skype sobre la red, pero tendremos un pequeño margen de error si lo que queremos es medir la cantidad del flujo de este protocolo. Más datos sobre estos resultados son explicados en la siguiente etapa de pruebas.

Como observaciones generales de estos protocolos podemos decir que en ambos casos los patrones trabajaron en forma correcta reconociendo la gran mayoría de los flujos y asociándolos a estas aplicaciones. Tener la posibilidad de reconocer en forma correcta estos protocolos, podría llevar en un futuro a elaborar políticas para realizar calidad de servicio o en el caso de skype llegar a limitar el ancho de banda o directamente a limitar su uso. Como observación particular aconsejamos el uso del helper de conntrack para asociar los flujos RTP a las sesiones SIP, en donde con esto podríamos reconocer cuánto del tráfico pertenece realmente a la utilización de SIP y a su vez se lograría una mejor utilización de los recursos del sistema.

Protocolo P2P

El análisis e identificación de este conjunto de aplicaciones ha despertado un particular interés en los últimos años debido a los grandes problemas que causan en las redes de las diferentes empresas y también a nivel de sus ISP's. La causa de esto es por el gran crecimiento en los últimos años debido al incremento de popularidad y de usuarios que estos protocolos han adquirido. La agresividad a la hora de generar conexiones y el alto ancho de banda que estos protocolos generan en las redes actuales ha provocado el interés de los administradores de mantener restringido o limitado a estos protocolos, ya que disminuyen la performance de las aplicaciones críticas de las empresa (uso de correo o de Internet, etc). También a nivel de los ISP's proveedores de banda ancha, se ha vuelto muy costoso este tipo de aplicaciones y se desearía degradar en cierta medida el ancho de banda consumido por estas, para poder así bajar los costos de sus enlaces de interconexión con otros ISP's. (actualmente más del 60 % del ancho de banda en un ISP es utilizado para la transferencia de archivos de video, música o software de algún tipo) [33] [34].

Sin embargo, lograr identificar estos protocolos se ha vuelto una tarea sumamente difícil, ya que estos intentan escapar a estas restricciones y han evolucionado y encontrado tácticas para abrirse paso a través de los firewalls, entre las que se encuentran traficar los datos por puertos no estándar, encriptar el payload, camuflarse en puertos estándar de otras aplicaciones, etc.
Una de las particularidades en la manera de actuar de estos protocolos y que dificulta la manera de reconocerlos por los métodos tradicionales, es que si el servidor de la red se encuentra saturado, estos protocolos tienen la capacidad de poder intercambiar archivos entre sí, volviéndose los clientes una especie de cliente y servidor

al mismo tiempo. Esto elimina la posibilidad de que el servidor se vuelva el cuello de botella a la hora de intercambiar información y permite que un archivo pueda descargarse de múltiples fuentes. La suma de todo lo anterior dificulta la búsqueda de este tipo de tráfico en las capas inferiores contando únicamente con el análisis de las conexiones que estos generan.

Para reconocer estos protocolos, inspeccionaremos en el payload en busca de determinados patrones utilizados cuando dos usuarios intentan compartir una archivo en la red. Una restricción existente a la hora de poder crear estos patrones son las faltas de documentación específica, la velocidad con la que cambia el código de la aplicación en cada nueva versión para los que son open source, o la dificultad de acceso al código o para realizar ingeniería inversa en el caso de los protocolos propietarios. El otro gran inconveniente al buscar patrones en el payload, es la imposibilidad de identificar en forma correcta cuando se utiliza algún sistema de encriptación para los datos, o cuando el protocolo utiliza algún sistema de control para establecimiento y otro para el envío de datos (como es el caso de Ares).

Generalmente, estas aplicaciones presentan dos fases a la hora de hacer el pedido para compartir un archivo, estas son:

Señalización: En esta etapa, el cliente busca por el archivo que requiere y determina cuáles son los peers que se encuentran disponibles para iniciar la transferencia. En algunos de los protocolos, el cliente no establece ninguna comunicación directa con el peer que proveerá dicho archivo.
Download – En esta etapa el cliente establecerá contacto con el o los peers con los cuales intercambiará la información.
Este tipo de establecimiento de “conexión” son los que buscaremos para poder identificar el protocolo.

Aquí analizamos una serie de protocolos p2p y de mensajería, los cuales consideramos son algunos de los más conocidos y utilizados, y presentan algunos de los casos más problemáticos a la hora de reconocerlos. Los protocolos que estudiaremos son los siguientes:

Ares	Bittorrent	shoutcast
Gnutella	msnmessenger	IRC
Edonkey		

Generaremos trazas para cada uno de los protocolos y analizaremos como actúan cada uno de los patrones dentro del flujo de paquetes, buscando reconocer la capacidad de no caer en el reconocimiento por falsos positivos ni falsos negativos, mencionando también cuáles son los inconvenientes encontrados en cada uno de ellos.

Protocolo Ares

Ares es uno de los sistemas de distribución de archivos de Internet más usados en América debido al rápido crecimiento de su red en popularidad y por su simpleza. Se basa en una estructura de red descentralizada y un sistema de búsqueda por broadcasting. Posee un sistema muy rápido de búsqueda y es muy sencillo de usar, volviéndolo muy atractivo para múltiples usuarios [35].
En particular este protocolo presenta un sistema de establecimiento de conexión para la negociación de los diferentes parámetros que utilizará y a su vez maneja un sistema de encriptacion simple para el intercambio de los datos.

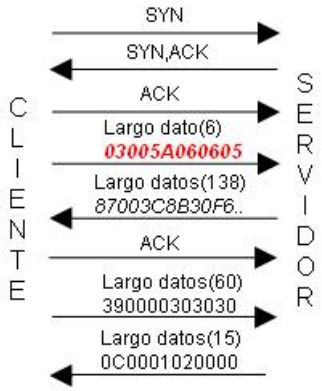
Resultados Obtenidos

Traza 1: ares1, 20960 paquetes

Protocolo	MCT_L7	OpenDPI	Wireshark
Ares	17732	0	0
unknown	11211	14829	14881
Ipv4 fragment	—	—	6079

Los datos obtenidos por la traza corresponden a los valores esperados, en donde sólo se reconocen algunos paquetes con este patrón, los cuales pertenecerían al establecimiento de conexión de ares. Debido al uso de encriptación en las conexiones para enviar los datos, es que el patrón reconoce únicamente los mensajes intercambiados entre el servidor y el cliente durante las conexiones de control. Con la definición actual del patrón no nos es posible realizar un conteo del flujo total o real utilizado por Ares (establecimiento y transferencia de archivos), sino que sólo tenemos la posibilidad de bloquear el tráfico de ares, lo cual no es de nuestro interés. Si bien este patrón está catalogado como bueno y a la vez rápido, no nos es de mucha utilidad, ya que deja escapar las conexiones de datos que es lo que realmente nos interesa contabilizar.

Un punto que se debe tener en cuenta en este protocolo, es que el tamaño de algunos paquetes es superior a 1500 bytes siendo necesario que se de-fragmenten en el recorrido hacia el destino. Esto ocasiona que se reciban varios paquetes que se volverán a ensamblar antes de pasarlos a Netfilter. Ésta contabilizará el tamaño total del paquete fragmentado y se indicará en los contadores de Iptables que ha llegado un único paquete. Aquí observaremos diferencias con Wireshark quien sí contará todos los paquetes aunque los mismos estén fragmentados. Debido a esto se llegarán a resultados más reales si tomamos en cuenta la cantidad de bytes indicados por Iptables y no la cantidad de paquetes.



Durante el establecimiento de la conexión se pasan los siguientes mensajes entre cliente-servidor, hemos visto que este comportamiento se da en diferentes trazas generadas. El dato que se busca se encuentra marcado en rojo, éste comienza con el byte 03 en hexa y termina con los bytes 06, 06 y 05, aunque el patrón es un poco más general y busca únicamente el último byte (05). Indicamos los tamaños de los paquetes en el flujo mostrado ya que estos también se mantienen en las distinta trazas que hemos generado y podrían ser utilizados para identificar a estos flujos.
Entonces, el patrón buscado por L7-Filter machea con el primer paquete con datos de la conexión, por esa razón se indica que este patrón permite una rápida caracterización. En caso de que no se pueda analizar este paquete porque haya mucha carga o el flujo no corresponda a una conexión de Ares, no tendría sentido seguir analizando los paquetes posteriores del flujo hasta llegar al máximo de paquetes que se definió para L7-Filter. En la versión actual, se definen la misma cantidad de paquetes a buscar por flujo para todos los protocolos.

Protocolo Bittorrent

Bittorrent es uno, sino el más popular de los protocolos para transferencia de archivos en Internet, donde se estima que entre un 35-60 % por ciento del p2p pertenece a esta aplicación de intercambio de archivos. La red de este protocolo consiste en clientes y un servidor centralizado. Los clientes se conectan entre sí directamente para intercambiar fragmentos de la totalidad de un archivo, mientras que la tarea del servidor consiste únicamente en controlar estas conexiones. La tarea de localizar el archivo no depende del servidor de la red, sino que se deja en manos del usuario, el cual buscará lo que se llama el archivo torrent en la web, este archivo posee la información del servidor de la red de Bittorrent (llamado tracker), y contiene la información necesaria del archivo pedido. El peer debe conectarse con este tracker para obtener la información del archivo y una lista de los posibles peers a los cuales puede ir a pedir fragmentos de lo que desea descargar. Los clientes establecen las conexiones por fuera de la red, sin embargo, deben consultar periódicamente a estos tracker para obtener nueva información acerca de los peers

a los cuales puede ir a pedir más fragmentos. Por esta razón, es que a diferencia de Ares y otros sistemas, no existe una señalización asociada a este protocolo.

Para identificar el tráfico asociado a este protocolo, el patrón no buscará las consultas HTTP en la web del torrent del archivo ni las conexiones con el servidor, sino que se concentrará en reconocer las conexión entre los clientes mientras intercambian los datos asociados a un torrent. El cometido de concentrarse en la búsqueda de las conexiones de transferencia de datos, es que los paquetes intercambiados por cliente-servidor son insignificantes comparados con la carga cuando se baja un torrent, así es que el patrón se focaliza en la búsqueda de estas conexiones.

Resultados Obtenidos

Traza : bittorrent, paquetes 24881

Protocolo	MCT_L7	OpenDPI	Wireshark
bittorrent	17951	12212	7479
edonkey	25	—	—
HTTP	56	22	20
UDP-data	—	—	10093
unknown	6131	12611	7259

Este patrón si bien tiene un bajo nivel de falsos negativos cuando identifica, presenta una baja performance al intentar detectar los caracteres dentro del payload. Esto es debido a que necesita de unos cuantos paquetes de la conexión para poder dar como reconocido un flujo de datos, generando en el sistema mayor procesamiento y consumo de recursos. Este protocolo se encuentra dentro de los caracterizados como undermatch, y la razón de esto es que es imposible escribir un patrón que reconozca en forma completa todo el tráfico de esta aplicación dentro de la red. Se decidió optar por intentar reconocer las conexión entre cliente-cliente, sacrificando las conexiones entre el cliente y el servidor.

Para el caso en que el flujo de datos cuando se intercambia un archivo viaje cifrado, el patrón se ve imposibilitado de reconocer parámetros identificativos del protocolo. Esta es una de las principales desventajas del sistema de análisis DPI. Sin embargo, se clasifica el patrón de bittorrent dentro de los protocolos considerados como undermatch, debido a que es difícil o imposible escribir un patrón que coincida con todas las conexiones que se establezcan.

Protocolo Gnutella

Gnutella es otro ejemplo de aplicaciones p2p que presenta una topología de red del tipo distribuida donde cada cliente de la red realiza tanto tareas asociadas a un cliente o a un servidor. El protocolo Gnutella también puede funcionar como una red centralizada donde un servidor atiende las peticiones de varios cliente. Estos clientes-servidores se conectan a la red Gnutella por medio del protocolo TCP y luego proveen al usuario de la capacidad de realizar consultas por archivos y desplegar los resultados. Al mismo tiempo también atienden peticiones de otros usuarios o clientes de la red, en donde chequean sus datos locales y responden a estas peticiones. Al poseer una red distribuida de clientes-servidores implementando Gnutella, la red se vuelve muy resistente a los fallos, ya que la operación en la red no se vería interrumpida por el bloqueo o salida de línea de los clientes.

El protocolo gnutella define una manera especial de comunicación en la capa de aplicación entre los clientes perteneciente a la red. Define una serie de "encabezados" dentro del payload que utiliza como manera de comunicación entre los distintos clientes y una serie de reglas para el intercambio de la información. Algunos de los mensajes definidos sirven para descubrir nuevos hosts y para unirse a la red, mensajes para buscar información o archivos los cuales son respondidos con la información necesaria para poder hacer frente a estos pedidos y comenzar el intercambio. Gnutella también provee un sistema para poder escapar a las restricciones aplicadas por un firewall sobre un peer que se encuentre detrás de este. Esto es lo que hace que este protocolo sea tan escurridizo a las reglas y restricciones aplicadas por los firewalls tradicionales y donde se puede notar la capacidad de nuestro sistema.

Cuando se desea transferir un archivo entre dos peers, estos establecen una conexión por fuera de la red, es decir establecen una conexión directa, los datos nunca son transferidos sobre la red Gnutella. El protocolo para realizar el intercambio entre ambos peers es mediante HTTP, en donde se envían una serie de datos de la forma:

GET /get/<File Index>/<File Name>/ HTTP/1.0\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
User-Agent: Gnutella\r\n\r\n

Luego de mandar la información necesaria entre los peers con mensajes similares al anterior sobre HTTP utilizando el protocolo UDP, los dos peers están preparados para comenzar la descarga. Estos mensajes simulan una especie de establecimiento de conexión o hand-shake donde se intercambia el indice del fragmento del archivo total que se quiere bajar (se bajan una serie de fragmentos del archivo para luego ser reensamblado, estos fragmentos se pueden bajar de dife-

rentes peers), y otros parámetros necesarios.

Estos mensajes de conexión de control de la red y los mensajes para la transferencia de archivos son los que se van a buscar con el patrón [36].

Resultados Obtenidos

Traza : gnutella, 12075 paquetes

Protocolo	MCT_L7	OpenDPI	Wireshark
genutella	11768	11767	7343
unknown	164	294	4842

A pesar de utilizar sentencias de HTTP para transmitir los datos, la expresión regular definida en el patrón no devuelve datos erróneos en caso de que también se intente detectar HTTP. Por lo tanto no se observaron falsos positivos en las distintas pruebas realizadas con este protocolo.

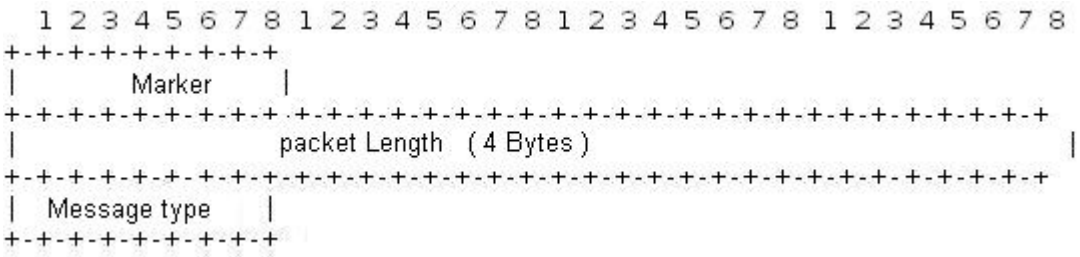
Protocolo Edonkey

La red eDonkey combina las mejores virtudes de las redes centralizadas y descentralizadas. Al tomar estas características de las redes descentralizadas, hace muy difícil el control por parte de los administradores y es aquí donde radica la verdadera fortaleza de este diseño, asegurando que si alguno de los host es sacado de línea por alguna política restrictiva la red seguiría su curso normal sin verse afectada en lo más mínimo. Sin embargo,está demostrado que este tipo de topologías hace que las búsquedas no sean muy escalables e inundan la red con mensajes cuando alguno de los peers desea buscar un archivo, siendo en este punto donde la características de tener una red centralizada se vuelve más atractiva. Con esta arquitectura híbrida de redes centralizas y descentralizadas a la vez, eDonkey distribuye una gran cantidad de servidores a lo largo de Internet, asegurando con estos mejores mecanismos de búsqueda y una gran robustez de la misma en caso de falla en alguno de estos servidores.

En los nuevos clientes de este protocolo, se provee la posibilidad de realizar obfuscation (ocultar las estructuras conocidas del protocolo) y encriptado de los datos , lo que hace en cierta medida que la búsqueda de patrones sobre el payload se vuelva inútil o baje considerablemente su performance de reconocimiento. Cada cliente se conecta mediante TCP a uno de los servidores de la red para realizar la búsqueda de los archivos, aquí es donde se produce la fase de señalización, y a continuación el peer enviará los pedidos de búsqueda al servidor. La conexión con los demás peer, se realiza en forma directa con estos, utilizando también TCP para abrir la conexión, y luego pide al otro extremo los fragmentos del archivo que necesite. Durante estos establecimientos de conexión del protocolo eDonkey, se

envían una serie de encabezados propios del protocolo, los cuales van dentro del payload del paquete IP. Esta estructura o marcas que realiza eDonkey son los que se buscarán por parte del patrón para poder reconocer los flujos y poder contabilizarlos.

Estos paquetes presentan el siguiente formato dentro del payload:



Según diferentes desarrolladores que trabajaron sobre la expresión regular de eDonkey para reconocer los paquetes de señalización y intercambio de datos pertenecientes al protocolo, recomiendan entre otras cosas, considerar las siguientes características:

1. La marca eDonkey es el primer byte luego del encabezado IP+TCP ,donde el valor de ésta es siempre 0xe3 hexadecimal.

2. El número que se da en los 4 bytes siguientes es igual al tamaño de paquete completo, excluyendo los bytes del encabezado IP+TCP y de 5 bytes extras.

Estas marcas más algunas otras adicionales son las utilizadas por el patrón dentro de los datos para reconocer el protocolo [37] [38].

Resultados Obtenidos

Traza : eDonkey, 6361 paquetes

Protocolo	MCT_L7	OpenDPI	Wireshark
edonkey	4697	4292	361
kugoo	1634	0	0
unknown	28	2027	7651

Si bien este patrón funciona y reconoce de forma adecuada los flujos que poseen contenido eDonkey, la estructura general del patrón hace que éste también pueda llegar a machear en flujos donde se encuentra esta marca en forma aleatoria, aunque por las especificaciones de los desarrolladores del patrón esto no sucede con frecuencia.

Conclusiones protocolos P2P

Este tipo de protocolos generan las mayores complicaciones a la hora de ser detectados, pero a pesar de ello se observó que es posible reconocer alguna de las etapas que deben atravesar para poder funcionar. Por ejemplo, para Gnutella los resultados que se obtienen son bastante interesantes pudiendo caracterizar correctamente cerca del 100 % del tráfico generado. En el resto de los casos, el porcentaje de tráfico caracterizado no es tan alto pero se detectan ciertas conexiones importantes, para Ares se reconocen todas las conexiones que realizan los clientes con el servidor posibilitando mantener bajo control la cantidad que se podrían establecer. En el caso de bittorrent y eDonkey se caracteriza entre un 70 y 80 % del tráfico generado, aunque en el segundo caso, una parte del tráfico es caracterizado como otro protocolo P2P. Este protocolo es llamado kugoo y es utilizado principalmente en china. Debido al funcionamiento de estas herramientas y al poco conocimiento de su diseño, podría existir la posibilidad que efectivamente los clientes como por ejemplo eDonkey2000, compartan las redes con otros protocolos P2P similares y por esa razón se reconocieron paquetes como kugoo.

Es importante recalcar que estas trazas fueron generadas por nosotros y se intentaron realizar de forma controlada para garantizar que el tráfico generado corresponda únicamente al protocolo correspondiente. Esto en algunos casos no es tan directo dado que no es conocido como funcionan dichas aplicaciones y se podrían generar otros tipos de flujos además de los esperados.

En el caso de bittorrent, el tráfico no caracterizado podría corresponder a la transferencia de paquetes entre el cliente y servidor, lo cual no está contemplado en el patrón.

Protocolo Msnmessenger

Una sesión MSN messenger ocurre en dos etapas, primero se realiza una conexión con un NS (notification server), que es el encargado de proveer el servicio de presencia, es decir, es el encargado de mantener la información de un cliente en particular y de todos sus contactos. Por otro lado, el protocolo MSN Client es el encargado de mandar los mensajes entre dos clientes. Los NS que mencionamos anteriormente, son los que permiten conectarse a los servidores SB (switchboard) quienes proveen servicios de mensajería instantánea. Luego de que un cliente se loguea, se crea una nueva conexión por la cual el servidor envía a través de HTTP y SSL información de los contactos del cliente correspondiente y le permitirá establecer conversaciones con ellos.

La expresión regular para este protocolo se divide en tres sub-expresiones, las cuales son excluyentes y buscan los distintos flujos que intervienen. La primera

expresión corresponde al establecimiento de la conexión entre el cliente y el servidor. Este patrón reconoce en el payload la palabra VER seguido de un número entre 0 y 9, a continuación busca la palabra msnp seguido de un número que puede llegar a ser de dos cifras y una cantidad de caracteres hexadecimales que pueden encontrarse dependiendo del caso. Por último, reconoce la palabra crv0 y los hexadecimales 0d y 0a. Aquí sólo se detectará como MSN el flujo que corresponde al logueo en el servidor y no a la información enviada por éste con los datos de los contactos.

La segunda expresión corresponde al intercambio de mensajes entre dos usuarios, en particular busca los mensajes generados por el que solicita la conversación. Esta expresión busca la palabra usr seguida de un 1, luego un string de caracteres seguida de una secuencia de números y por último los hexadecimales 0d y 0a. En las trazas que generamos encontramos flujos que macheaban con esta expresión, pero en muchos casos ocurría que no era mandatario que el valor que se encuentra a continuación de usr sea 1, sino que podía ser cualquier número entero. Por esta razón definimos un nuevo patrón para mejorar la caracterización, sin afectar el funcionamiento de la herramienta debido a falsos positivos que puedan generarse dado el decremento en la especificidad del patrón.

Por último, la tercer expresión es similar a la anterior pero busca los flujos que corresponden al usuario que recibe la invitación a formar parte de la conversación. El formato de mensaje cambia la expresión usr por ans manteniendo el resto sin cambiar. También modificamos esta sub-expresión para que se reconozca el flujo como msn aunque el entero luego de ans sea distinto de 1.

Resultados Obtenidos

Traza : msn, 4328 paquetes

Protocolo	Patrón de L7_Filter	Patrón definido	OpenDPI	Wireshark
dns	74	74	74	74
ssl	703	703	121	398
http	826	826	784	176
msnmessenger	174	2143	2858	419
unknown	2108	176	339	3261

Se observan claramente las ventajas de definir un nuevo patrón para este protocolo, esto fue posible gracias a la documentación que se encuentra sobre este patrón y la fácil detección de los mensajes que se intercambian en las diferentes etapas de la conexión. En caso de transferir archivos a través de msn, el patrón utilizado por L7-Filter no reconoce esta transferencia si se utilizan las últimas versiones de msn messenger. Esta fue otra modificación que realizamos para poder aumentar la cantidad de paquetes caracterizados, agregando a la expresión regular

que revise también si se encuentra la secuencia MSNSLP/1.0 200 OK en el payload de alguno de los flujos.

Patrón original:
ver [0-9]+ msnp[1-9][0-9]? [\x09-\x0d -]*cvr0\x0d\x0a$
OR usr 1 [!-]+ [0-9.]+\x0d\x0a$
OR ans 1 [!-]+ [0-9.]+\x0d\x0a$
Patrón modificado:
ver [0-9]+ msnp[1-9][0-9]? [\x09-\x0d -]*cvr0\x0d\x0a$
OR usr *[0-9]* + [!-]+ [0-9.]+\x0d\x0a$
OR ans *[0-9]* + [!-]+ [0-9.]+\x0d\x0a$
OR *MSNSLP/1.0 200 OK*

Para más referencias del significado de cada una de las expresiones regulares, consultar el Anexo_3.

Protocolo Shoutcast

El propósito de Shoutcast es la de brindar una plataforma de software capaz de realizar streaming de audio (mp3) sobre la red de Internet, en donde el uso principal dado a esta aplicación es para crear o escuchar sobre Internet broadcast de audio. Shoutcast es un protocolo que utiliza un modelo de cliente-servidor, a diferencia de la mayoría de los protocolo p2p de la actualidad, en donde los componentes de la red (clientes y servidores) se comunican con otros para intercambiar datos de audio y datos acerca de este streaming de audio (como ser los títulos de las canciones y el nombre de la estación de radio). Lo que caracteriza a Shoutcast es que el streaming de audio es on-demand, es decir que el audio se comienza a escuchar antes que el archivo entero de audio haya sido bajado totalmente. Se usa HTTP como protocolo de transporte aunque es capaz también de utilizar multicast como alternativa. El protocolo para el streaming de audio utiliza un formato especial para los llamados metadata (que es la información acerca de la datos, en este caso el audio) y para las respuesta en el cual se les coloca una etiqueta ICY, los cuales son leídos por el software cliente para el reconocimiento de los datos.

El modo de operar de esta aplicación es la siguiente: los servidores de Broadcast de Shoutcast suelen estar en una lista dentro de un archivo del programa cliente del usuario, estos son simples archivos de texto que contienen las URL's de los servidores. Cuando se visita una URL utilizando algún browser (por ejemplo Mozilla), este retorna una página con la información y el estado del servidor, además del streaming de audio. En estas conversaciones y pedidos al servidor, protocolo pone un tag en los datos sobre HTTP y esto es lo que podemos llegar a reconocer. Un ejemplo de un pedido de un cliente seria el siguiente

GET / HTTP/1.0
ICY 200 OK
*icy-notice1:
This stream requires <a href="http://www.winamp.com/»*
*Winamp
*
*cy-notice2:SHOUTcast Distributed Network Audio Server/posix v1.0b
*
icy-name: Great Songs
icy-genre: Jazz
icy-url: http://shout.serv.dom/
icy-pub: 1
icy-br: 24
<data><songtitle><dat

Estas etiquetas ubicadas dentro del payload seran buscadas por el patrón para poder reconocer el flujo de datos con este protocolo. De esta forma el patrón buscará en los pedidos de los cliente y también las respuestas del servidor de streaming.
Este patrón intenta buscar en primer lugar los pedidos HTTP que parezcan ser un pedido a Shoutcast por un streaming de audio, el segundo sector del patrón (segundo branch, los cuales están separados por los símbolos pipe) buscará por la respuesta del servidor [39] [40]..
Patrón L7:
ˆ get /.*icy-metadata:1|icy [1-5][0-9][0-9] [\x09-\x0d -]*(content-type:audio|icy-)

Resultados Obtenidos

Traza : shoutcast, 5420 paquetes

Protocolo	MCT_L7	OpenDPI	Wireshark
shoutcast	5316	—	0
MPEG	—	5322	—
HTTP	14	18	6
DNS	10	10	10
unknown	0	61	5404

Protocolo IRC

El protocolo IRC (Internet Relay Chat) se ha diseñado para usarse como conferencia basada en texto. El protocolo IRC se ha desarrollado en sistemas que usan el protocolo de red TCP/IP, aunque no es imperativo que ésta sea la única forma en que funcione. El IRC es en sí mismo un sistema de tele conferencia que (a través del modelo cliente-servidor) es adecuado para funcionar en muchas máquinas en una forma distribuida. Una configuración típica incluye un único proceso (el servidor) que conforma un punto central para que los clientes (u otros servidores) se conecten a él, realizando los envíos y multiplexado de mensajes requeridos, así como otras funciones.

Los mensajes de protocolo deben extraerse de la cadena contigua de octetos. La solución es asignar dos caracteres, CR y LF como separadores de mensajes. Los mensajes vacíos se ignoran de forma silenciosa, lo que permite el uso de la secuencia CR-LF entre mensajes sin problemas.

El mensaje extraído se divide en las componentes <prefijo>, <comando>y lista de parámetros formada por componentes <parámetro intermedio>o <parámetro final>
La representación BNF para esto es:
<mensaje>::= [':' <prefijo><ESPACIO>] <comando><parámetro><crlf>
<prefijo>::= <nombre de servidor>| <nick>['!' <usuario>] ['@' <host>]
<comando>::= <letra> <letra>| <número><número><número>
<ESPACIO>::= ' ' ' '
<parámetro>::= <ESPACIO>[':' <parámetro final>| <parámetro intermedio><parámetro>]
*<parámetro intermedio>::= <Cualquier secuencia de octetos *no vacía* que no incluya ESPACIO, NUL, CR o LF, el primero del cual no puede ser ':'>*
*<parámetro final>::= <Cualquier secuencia, posiblemente *vacía* que no incluya NUL, CR o LF>*
<crlf>::= CR LF
El patrón desarrollado por L7 filter buscará esencialmente en el payload los mensajes del usuario en donde manda las etiquetas NICK y USER, e identificara este flujo con IRC sin importar el orden en que aparezcan.[**?**] [**?**]

(nick[\x09-\x0d -]*user[\x09-\x0d -]*:
OR user[\x09-\x0d -]*:[\x02-\x0d -]*nick[\x09-\x0d -]*\x0d\x0a *)*

Nos interesó este protocolo ya que varios de los clientes P2P incorporan una interfaz de chat que utiliza irc. Esta traza se generó utilizando el cliente P2P eDonkey2000, que tiene incorporado un servicio de mensajería que utiliza el protocolo irc, es por esa razón que se contabilizaron flujos correspondientes a bittorrent, gnutella y eDonkey.

Resultados Obtenidos

Traza : irc, 2966 paquetes

Protocolo	MCT_L7	OpenDPI	Wireshark
bittorrent	124	140	70
genutella	956	947	23
edonkey	189	46	126
irc	688	681	362
unknown	24	7	2385

Este protocolo de mensajería no presenta un gran obstáculo para los proveedores o los administradores ya que los anchos de banda que manejan estos servicios no son comparables con los utilizados por los protocolos p2p, por esta razón es que si bien se ha analizado su funcionamiento y la calidad de las capturas, no se entró en mayores detalles.

Comparación entre las versiones Kernel-space y Users-space de L7-Filter

En esta etapa haremos énfasis en la comparación entre las dos versiones del módulo de L7-Filter, utilizando las trazas creadas para verificar la precisión de las expresiones regulares pero utilizándolas a nivel del kernel y a nivel del user-space. Ambas versiones utilizan las mismas expresiones regulares pero de distinta forma, utilizando distintas librerías para revisar el payload en busca de los patrones. Compararemos ambas formas de análisis y la eficiencia con que utilizan estas expresiones para marcar los flujos y haremos algunos comentarios sobre su modo de funcionamiento.

Para comparar las versiones, utilizamos las trazas creadas para verificar los patrones a nivel del kernel en la etapa anterior, utilizaremos algunas de las trazas de aplicaciones tradicionales y las p2p que presentan un comportamiento aceptable en cuanto al comportamiento de su protocolo.

Tomamos como verdaderos los datos obtenidos por los patrones utilizando la versión del Kernel. Esto se hará de este modo dado que fueron comparados con otros sistemas obteniendo resultados similares en ambos casos, además nos sustentaremos en los datos aportados por el equipo de desarrollo del módulo L7-Filter, los cuales especifican que los resultados obtenidos por la versión del kernel, están lo suficientemente testeados como para tomarlos como verdaderos. Sin embargo para la versión del módulo a nivel de usuario, se especifica que la misma está en etapa de desarrollo y propensa a modificaciones debido a las apariciones de distintos bugs, los cuales algunos han sido ya corregidos.

Las trazas utilizadas y los resultados obtenidos fueron los siguientes:

Bittorrent		
Protocolo	User	Kernel
bittorrent	11310	17951
edonkey	23	25
http	19	—
unknown	13498	6131

Edonkey		
Protocolo	User	Kernel
edonkey	2876	4967
kugoo	0	1634
http	0	0
unknown	3474	28

FTP		
Protocolo	User	Kernel
dns	6	12
FTP	41	2744
http	93	186
unknown	2838	35

vnc		
Protocolo	User	Kernel
vnc	19624	22818
unknown	3197	3

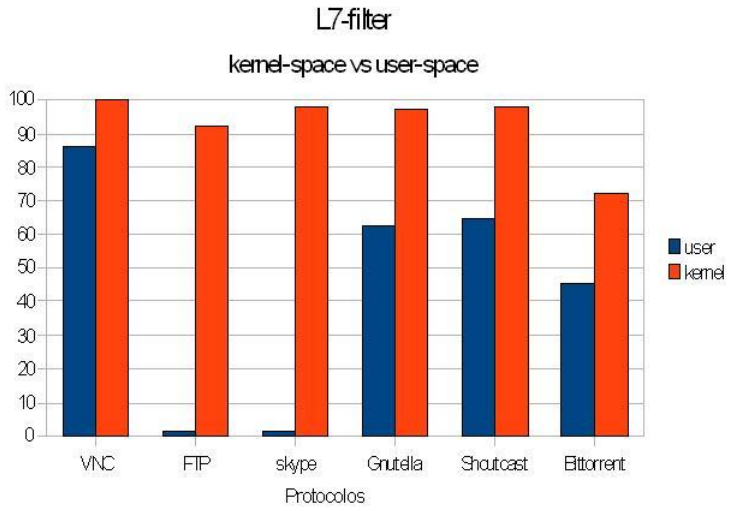
Gnutella		
Protocolo	User	Kernel
Gnutella	7568	11768
unset	4372	—
unknown	128	164

IRC		
Protocolo	User	Kernel
dns	3	—
bittorrent	72	124
Gnutella	547	956
edonkey	17	186
IRC	415	688
unknown	1768	24

Shoutcast		
Protocolo	User	Kernel
dns	9	10
shoutcast	3496	5310
http	5	14
unknown	1919	0

Skype		
Protocolo	User	Kernel
skype	540	42078
dns	2	4
ssl	8	0
unknown	42723	671

En la siguiente gráfica comparamos para cada protocolo cuál es la cantidad de paquetes reconocidos por las dos versiones de L7-Filter. Se puede ver que en todos los casos, el patrón para la versión del kernel trabaja de manera más precisa que al utilizarlo en la versión de usuario.



Conclusiones

Comparando los resultados obtenidos en ambas versiones, notamos claramente que los patrones, si bien utilizan las mismas expresiones regulares, trabajan en forma más precisa en la versión del kernel. Esto se debe a las bibliotecas utilizadas en cada versión para inspeccionar el payload en busca de las coincidencias en el payload. Otro dato interesante a destacar es que la versión a nivel de usuario utiliza un método diferente para mantener un control de flujos de los paquetes de la red. Este módulo no utiliza el sistema de conntrack del kernel para mantener el control de los flujos, sino que lo hace de una manera particular a nivel de la capa de usuario. Trabajando de esta forma no usa las utilidades provistas por el sistema de conntrack (particularmente los helpers del conntrack) para el seguimiento de los flujos de datos relacionados. Esto se ve reflejado al analizar la traza FTP en donde la versión de usuario sólo puede reconocer los paquetes pertenecientes al flujo de control de la aplicación, pero no las conexiones relacionadas a éste, por donde circulan los datos del usuario. Otro punto a destacar es la sobrecarga que exige con las configuraciones utilizadas del sistema operativo para levantar los paquetes de la NIC, es que hacer el análisis de los datos cuesta al sistema mayor cantidad de recursos.

Si bien existen técnicas nuevas para bajar el tiempo y el procesamiento de pasar los paquetes al nivel de usuario utilizando estructuras de socket especiales y recogiendo la información directamente desde la NIC. Estas estructuras pasan los

paquetes directamente al nivel de usuario, sin hacer el recorrido normal a través de los driver y los buffers del kernel. La precisión que tienen los patrones a nivel de usuario es aún muy baja comparada con el obtenido con la versión del kernel, lo cual hace que el haber optado por esta última versión haya sido acertado.

6.3.3. Determinación de Falsos Positivos y Falsos Negativos

En esta etapa se buscó dar una idea de cómo es el comportamiento de los distintos patrones cuando se ponen en conjunto para diferenciar las distintas aplicaciones en la red. Como hemos mencionado y verificado en el apartado anterior, no todos los patrones reconocen de igual forma, en algunos casos no es fácil encontrar un comportamiento definido para crear una expresión regular eficiente. Muchos de éstos se han creado, sobretodo para aplicaciones bajo licencia o sin especificaciones, revisando trazas de capturas y analizando manualmente el payload en busca de algún patrón característico de la aplicación. Esto genera que algunos patrones sean muy generales o que exista la posibilidad de que se confundan en el análisis y generen lo que llamamos falsos positivos, el otro inconveniente de estas expresiones es la aparición de falsos negativos (los cuales fueron estudiados en el apartado anterior).

Luego de realizar un estudio de los patrones y revisar las expresiones regulares podemos concluir que el orden en que éstos aparezcan en las reglas de Iptables puede afectar (en un pequeño porcentaje), esto es debido a que no todos los patrones necesitan inspeccionar la misma cantidad de paquetes y algunos pueden encontrar su expresión regular en el payload de un flujo de otra aplicación para la cual su patrón aún no logró identificar.

Los encargados de mantener el módulo L7-Filter detallaron una escala para identificar y caracterizar el nivel de precisión de los patrones y si éstos identifican flujos en forma errónea.

Se caracterizaron los patrones de la siguiente manera:

- **Overmatching:** Es difícil o imposible escribir un patrón para este protocolo de forma fiable que sólo coincida con el protocolo previsto. Podría pasar que el uso de este modelo de como resultado falsos positivos, por lo que sería aconsejable utilizarlo junto con otras reglas, como ser el puerto o alguna IP característica.
- **Undermatching:** Es difícil o imposible escribir un patrón para este protocolo que coincida con todas las conexiones.
- **Superset:** Este patrón machea el tráfico que también sería macheado por el patrón de otras aplicaciones. Si el mismo está por delante de uno de estos patrones en las reglas de Iptables, el tráfico no será macheado por los otros.
- **Subset:** Este patrón machea tráfico que es un subconjunto del tráfico macheado por algún otro protocolo.

Para realizar esto, hicimos un recuento de las características de los patrones y los colocamos en las reglas de Iptables ordenados en escalas según su precisión, cantidad de paquetes aproximada que necesita ver del flujo para reconocer la aplicación y la capacidad de identificar el flujo sin equivocarse. Debido a que Iptables aplica las reglas en orden secuencial en el que aparecen , los patrones más rápidos y específicos irán al comienzo, dejando a los más generales y con mayor probabilidad de error en último lugar en la tabla.

En primera instancia utilizamos las trazas generadas para algunas de las aplicaciones que estábamos seguros que eran identificadas en forma correcta por sus respectivos patrones. Las trazas utilizadas fueron algunas de las que usamos en el apartado anterior para corroborar la veracidad; se utilizaron las que el porcentaje de caracterización fue más alto utilizando trazas de aplicaciones tradicionales, VoIP y p2p.

Las trazas utilizadas fueron:

bittorrent	pop3	sip + rtp	smtp
vnc	gnutella	ftp	http

Se optaron por agregar a las reglas de Iptables algunos de los protocolos más utilizados y que más tráfico producen en las redes hoy en día. Una vez hecho esto, intercambiamos el órden las reglas y agregamos ciertos patrones en distintas posiciones para poder identificar la cantidad de falsos positivos que producen algunos de éstos, y en qué medida afectaba lo reconocido por los demás. En una segunda instancia se utilizó una traza capturada de Internet generadas por algunos usuarios en una red pequeña, para realizar las mismas pruebas sobre tráfico cotidiano de Internet.

Caso 1

En este caso se ingresaron al principio de las reglas de Iptables, los protocolos cuyos patrones son calificados como superset, como es el caso de http. Luego se agregaron todas las reglas para los protocolos P2P y por último los protocolos cuyos patrones están muy bien testeados y se sabe que siempre funcionan, como el caso de POP3, VNC, etc. De esta manera se intentó detectar falsos positivos producidos por las primera regla.
Orden de las reglas ingresadas:

1	http	14	napster
2	dns	15	ssh
3	dhcp	16	irc
4	shoutcast	17	vnc
5	ares	18	pop3
6	bittorrent	19	smtp
7	gnutella	20	ftp
8	edonkey	21	telnet
9	msn-filetransfer	22	sip
10	msnmessenger	23	rtp
11	ssl	24	netbios
12	pcanywhere	25	tftp
13	imap	26	unknown

Caso 2

En este caso se dejaron las reglas de Iptables en el mismo orden que en caso 1, pero se agregó al comienzo de la tabla las correspondientes al protocolo skype (skypeout y skypetoskype). El objetivo para este caso fue mostrar que los patrones de esta aplicación son demasiado genéricos y por lo tanto se debe tener precaución en el orden en que se ubican en la tabla para no generar falsos positivos. Al realizar pruebas de funcionamiento, generalmente notábamos que al probar el sistema con tráfico variado (y en ocasiones con trazas generadas para un protocolo específico en un ambiente controlado) si estaban las reglas de skype veíamos paquetes que macheaban con estas. Esto nos llamó la atención ya que estábamos seguros que el proceso de skype no estaba corriendo, revisando el patrón y la lista de correo de este módulo, encontramos comentarios sobre éste en especial y algunos otros, en los que se explicaba que este comportamiento era debido a las generalidades de sus expresiones regulares. Es por esto que utilizaremos en especial a este protocolo y daremos recomendaciones de uso luego de los resultados [39].

Caso 3

En este caso se cambiaron las reglas de skypeout y skypetoskype al final de la tabla, y luego se verificó cual era el resultado de colocarlos en esta posición. Con esto podemos tener una noción de cuánto afecta el orden de ingreso de las reglas en Iptables, y dar recomendaciones a los usuarios para lograr mejor precisión.

Caso 4

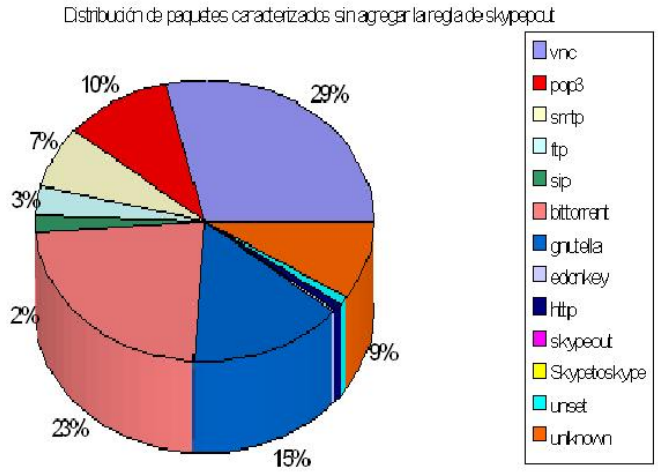
Para este caso se dejaron los patrones de skype y se quitaron las demás reglas. Con esto queríamos comparar cual era el resultado de los reconocimientos de las etapas anteriores y si afectaba o no que estuvieran las demás reglas junto con las de skype.

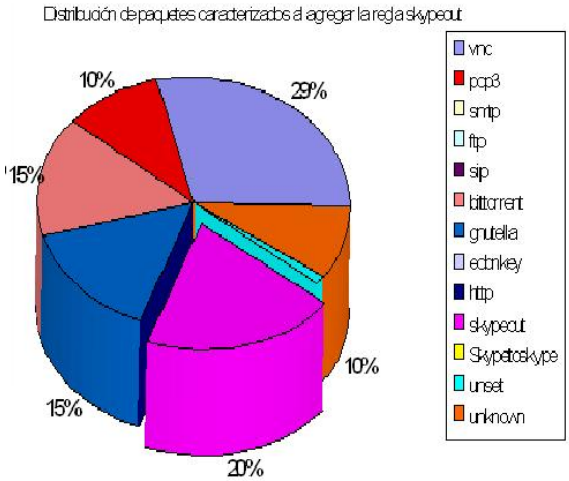
Resultados Obtenidos

Se muestra la tabla de resultados obtenidos en cada caso:

Protocolo	Caso 1	Caso 2	Caso 3	Caso 4
ssh	0	0	0	0
vnc	22818	22818	22817	0
pop3	8065	8065	8065	0
smtp	5592	0	5592	0
ftp	2744	0	2744	0
dns	16	16	16	0
telnet	0	0	0	0
sip	1506	0	1506	0
shoutcast	0	0	0	0
ares	0	0	0	0
bittorrent	18007	12165	18007	0
gnutella	11768	11768	11768	11768
ssl	30	30	30	0
edonkey	25	0	25	0
http	576	25	513	0
rtp	0	153	0	0
dhcp	14	0	14	0
irc	0	0	0	0
unset	807	789	791	966
unknown	7208	7520	7171	62383
skypeout	0	15828	118	15828
skypetoskype	0	0	0	0
total	79176	79177	19177	19177

Con estas trazas también se corrieron cerca de 30 patrones más para comprobar si algunos de estos también arrojaban algún comportamiento anómalo, mostrando datos falsos. Entre los protocolos analizados se encontraban algunos aplicaciones p2p como por ejemplo Kazaa, Xunlei, directconnect, fasttrack, pplive(video streaming), pcanywhere, entre otros menos conocidos y algunas aplicaciones de juegos online.





Conclusiones

La aparición de flujos reconocidos como skypeout confirma la suposición original acerca de la generalidad y de lo impreciso de este patrón. En las gráficas de las Figuras se puede observar claramente cómo al agregar este patrón en las reglas de Iptables se reconoce un 20 % por ciento del tráfico como skype, algo que en los hechos sabemos no es cierto debido a que usamos trazas conocidas, las cuales fueron utilizadas para testear otros protocolos con buenos resultados. Si bien no todos los protocolos se vieron afectados por la presencia del patrón skypeout, existen algunos de ellos que sí redujeron los flujos que reconocieron, estos son los casos smtp, sip y rtp, ftp. Esto se debió a que los patrones de las aplicaciones mencionadas no lograron reconocer sus flujos antes que skypeout. Por otro lado algunos protocolos como bittorrent o http, si bien lograron reconocer algunos de los flujos, no lograron machear la totalidad de lo que debían. Dichos flujos reconocidos en forma errónea (falsos positivos) por skype, probablemente pertenezcan tanto a los flujos de http o de bittorrent.

El otro dato interesante es que en los demás protocolos, no se presentaron casos de falsos positivos considerables, si bien en algunas de las pruebas hechas sobre tráfico normal de Internet, algunos datos presentaban un leve cambio en los resultados, éste no era considerable ni con el tráfico total reconocido por el patrón y menos aún con el tráfico total con el cual se estaba testeando.

Como conclusión, podemos aconsejar la no utilización del protocolo skypeout, si lo que deseamos es tener datos aproximados del tipo de aplicación que circula por la red. Sin embargo el resto de los patrones presentaron un buen comportamiento, inclusive el patrón skypetoskype (el cual era considerado sospechoso en un principio). El orden de las reglas no afectará en gran medida (al menos para los protocolo elegidos) las estadísticas del tráfico que circula, aunque es preferible colocar los patrones más rápidos y con menos errores en los primeros lugares para poder ahorrar recursos del sistema en los casos extremos de carga.

6.3.4. Testeo de performance del sistema

En esta sección indicaremos los resultados obtenidos al realizar pruebas de performance, debido a no disponer de tráfico real, como en la sección anterior fue necesario generar nuestras propias trazas, así como también tuvimos que definirnos distintos métodos para poder enviarlas. Como se mencionó, para poder caracterizar el tráfico se necesitan varias herramientas funcionando concurrentemente, cada una ellas jugando un papel esencial en el correcto funcionamiento de la solución global, pero a su vez consumiendo recursos en el hardware limitando el funcionamiento a cierto nivel máximo de procesamiento.

Los paquetes que ingresan al sistema de caracterización deben atravesar diferentes controles para ser atendidos, entre ellos para chequear la integridad de los encabezados (funciones de checksum) de las capas inferiores o en las diferentes etapas dentro del Kernel. Finalmente, se podrá llegar a representar gráficamente cómo se distribuye el tráfico entre los distintos protocolos de aplicación. Indicaremos en qué medida afectan cada uno de los módulos definidos anteriormente e intentaremos dar una guía de en qué puntos es necesario mejorar la performance y cómo sería posible hacerlo. Algunos de los usos que se le podría dar a una herramienta DPI serían, controlar, priorizar o simplemente investigar el tráfico que cursa a través de la red de un ISP, pero para ello es necesario lograr procesar altos niveles de carga en tiempo real. Con los recursos que dispusimos y el hardware que habíamos definido en los principios de diseño, logramos alcanzar analizar de una forma correcta un tráfico cercano a los 190Mbps sin pérdida de paquetes y verificando los flujos contra 30 patrones que consideramos incluyen las aplicaciones más utilizadas en la actualidad.

Módulo 1

Comenzaremos analizando el módulo de caracterización y análisis, siendo este módulo el más restrictivo y el de mayor vitalidad en la arquitectura de la herramienta. Este requiere de un alto procesamiento para asignar cada flujo con el tipo de aplicación que lo está generando. Aquí interactúan una serie de herramientas y cada una ellas tiene sus requerimientos que afectan en la performance y limitan la tasa capaz de procesar. Este módulo depende íntegramente de la carga de la red, y particularmente del tipo de tráfico presente, para poder trabajar en forma correcta. Por esta razón es por la cual este punto será el cuello de botella para la implementación de la herramienta. Cuando nos referimos al tipo de tráfico presente, queremos decir que la performance se verá fuertemente afectada por la velocidad con que se abren nuevas conexiones, cuanto mayor sea ésta, mayor procesamiento será necesario.

Conntrack:

Como se ha mencionado conntrack se encarga de actualizar y mantener la lista de flujos que están atravesando el equipo. Para poder actualizar o ingresar nuevas entradas a dicha lista es necesario que previamente se realicen una serie de operaciones para cada paquete que ingresa al kernel generando un consumo de los recursos disponibles, como ser CPU y un espacio de memoria donde guardar la información de cada conexión. Para identificar un paquete con una conexión (aquí no hablamos de conexiones TCP, sino que entendemos por conexiones a los distintos flujos definidos por los puertos e IP's de origen y destino, más el protocolo de capa 4 sea TCP/UDP), se utiliza una función hash especial. Esta función toma como parámetros de entrada, la información correspondiente a la capa de transporte y de red del paquete (puertos, Ips, protocolo, etc), retornando un índice. Luego se busca con este índice la ubicación del bucket en donde se encuentra la conexión correspondiente al paquete en una tabla hash indexada. En cada bucket de esta tabla hash, se pueden guardar más de una conexión (esto depende de la cantidad de conexiones abiertas en el flujo de datos y de la cantidad de buckets definidos para la tabla hash) las cuales están ordenadas a su vez en una lista encadenada (linked list), guardándose información de ambos sentidos de las conexiones.

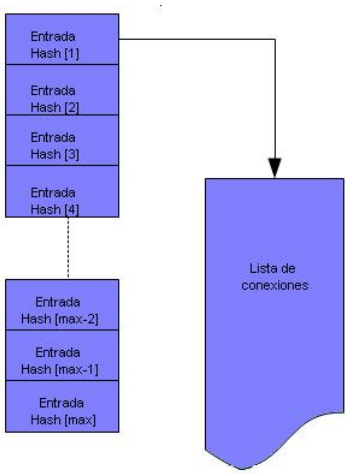
La distribución de las conexiones en este sistema de tres capas constituye una forma inteligente y rápida de ordenar en el kernel el sistema de connection tracking. La eficiencia de utilizar la función de hash para poder indexar las conexiones constituye el pilar en que se basa el conntrack para alcanzar una buena performance ya que esta función genera siempre el mismo tiempo de procesamiento (no teniendo que recorrer toda la lista de conexiones almacenadas en el conntrack, sino sólo las pertenecientes a dicho bucket). Sin embargo, luego de obtenido este índice y de encontrado el bucket donde se debe buscar la conexión, es necesario recorrer la lista de conexiones pertenecientes a éste, lo cuál sí afectará la performance a medida que la cantidad de tuplas en esta lista crezca.

La cantidad de índices Hash es configurable en Linux, aunque varía la manera de modificar este parámetro dependiendo de la versión de Kernel que se esté utilizando, a partir de 2.6.20 se puede controlar este parámetro de la siguiente manera:

 # echo $HASHSIZE >/sys/module/nf_conntrack/parameters/hashsize

También se define cuál es la cantidad máxima de conexiones simultáneas que se pueden registrar entre todas las listas para ser manejadas por netfilter en la memoria del kernel. Este valor por lo tanto influye en la cantidad máxima de memoria que podrá ser utilizada y también es posible setearlo en Linux. A partir del kernel 2.6.20.

 # echo $CONNTRACK_MAX >/proc/sys/net/ipv4/netfilter/nf_conntrack_max



La elección de estos valores depende de la utilización que se le de al equipo, en nuestro caso pretendemos mantener la mayor cantidad de conexiones posibles y a su vez, que la búsqueda de ellas no se enlentezca. En el caso de llegar al máximo de conexiones, éstas idealmente se distribuirán de igual medida entre las distintas listas, con lo cuál estadísticamente, costaría lo mismo hallar distintas conexiones en diferentes listas.

Pretendemos manejar muchas conexiones simultáneas, por lo que vamos a aumentar el valor de CONNTRACK_MAX, pero al hacerlo, estaremos aumentando la cantidad de entradas que se registrarán por cada Beckett, afectando la performance y obligando a netfilter recorrer un array más extenso. La forma de solucionar esto sería aumentar también la cantidad de índices, aumentando el valor de HASHSIZE, ya que calcularlo tiene siempre el mismo costo. En el caso limite en que las entradas de conntrack se distribuyan uniformemente, nos servirá tomar los valores de HASHSIZE y CONNTRACK_MAX iguales para maximizar la performance de la búsqueda [41] .
De esta manera disminuímos el procesamiento necesario, pero se incrementa el espacio de memoria que será utilizado, este aumento no es significativo ya que definiendo la cantidad de hash y el tamaño de conntrack máximo de la manera que se indicó, y sabiendo que cada entrada de conntrack utilizá alrededor de 308 bytes cuando se llega al máximo de conexiones permitidas. Este valor depende de la arquitectura del equipo y la versión del kernel. Eso quiere decir que en caso de reservar 512 MB de memoria para esta tarea, podrían manejarse:

*Cantidad máxima de conexiones simultáneas = (512 * 1024^2) / 308 = 1.743.087*

Para poder verificar estos valores utilizamos wireshark, httperf, tcprewrite y tcpreplay, y procedimos de la siguiente manera:

- Por medio de httperf generamos varios flujos en los que se mantenían las direcciones IPs de ellos, pero cambiaban los puertos.
- Guardamos el tráfico anterior utilizando wireshark y luego quitamos de la traza guardada todos los paquetes que tengan la bandera FIN = 1.
- Con tcprewrite duplicamos la traza cambiando las IPs origen y destino. Repetimos este paso varias veces de forma de aumentar la cantidad de flujos que se enviarán.
- Por último, utilizamos tcpreplay para enviar estas trazas hacia la PC que caracteriza el tráfico.

Con el procedimiento anterior logramos abrir 150.000 conexiones simultáneas y hacer que las mismas se mantengan en las listas de conntrack hasta que se cumpla un timeout. Según lo explicado más arriba, se deberían haber utilizado:

*Cantidad máxima de conexiones simultáneas = (150.000 * 308) / 1024ˆ2 = 44 MB*

La memoria que se utilizó en nuestro caso estuvo por encima del valor anterior llegando a 65 MB. Este valor está por encima de lo esperado, pero a pesar de ello, no genera una limitante para manejar grandes cantidades de conexiones tomando en cuenta, las memorias con las que cuentan los equipos actuales.

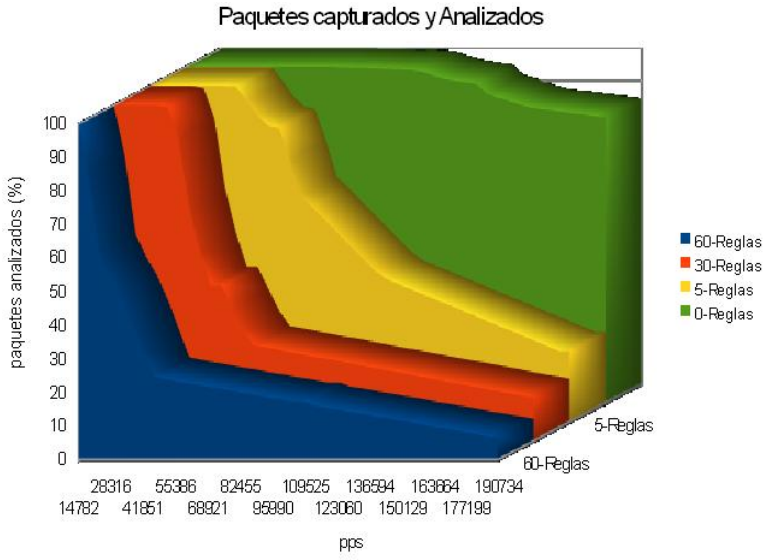
Resumiendo, conntrack brinda un mecanismo que facilita el seguimiento de las conexiones y brinda la posibilidad de que L7-Filter no tenga que analizar todos lo paquetes de cada flujo. Además es altamente eficiente, generando un consumo de memoria mínimo tomando en cuenta la memoria RAM que dispone cualquier PC en la actualidad.

Iptables, netfilter y L7-Filter

Como hemos mencionado, Iptables se ha desarrollado para utilizarse como el firewall de Linux y junto a Netfilter son capaces de indicar cuáles serán las políticas o controles que se realizarán sobre el tráfico. Son altamente eficientes pudiendo manejar grandes niveles de tráfico pero la performance depende de la cantidad de reglas que se configuren, ya que las mismas se chequean secuencialmente dentro de cada cadena. Hay distintas soluciones para mejorar esto [42] en donde se proponen diferentes algoritmos para mejorar la búsqueda. Al utilizar L7-Filter, se deben

chequear los flujos de la misma manera hasta encontrar cual es el patrón que corresponde. Este chequeo causa problemas al aumentar la cantidad de tráfico que es necesario procesar o al aumentar la cantidad de reglas que se definan. También afecta la tasa con que se abren nuevas conexiones, ya que se debe actuar sobre los primeros paquetes de cada una de ellas, esto es algo que empeora la performance en caso de analizarse tráfico con un alta tasa de flujos. Este tipo de flujos son los generados usualmente en las aplicaciones P2P, ya que continuamente están abriendo conexiones con los demás pares.

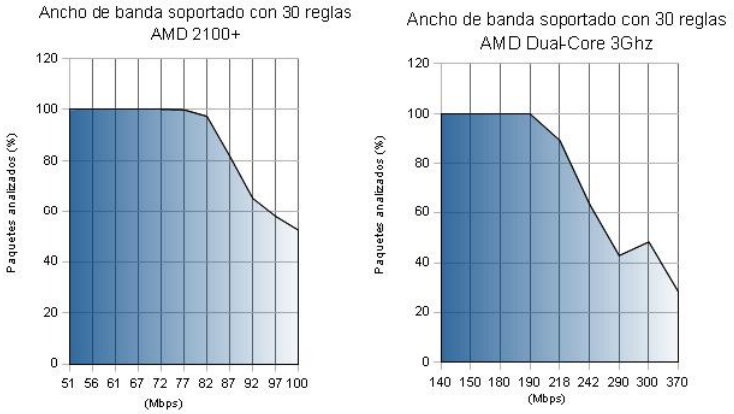
Fue necesario generarnos nuestras propias trazas para probar el funcionamiento de L7-Filter en función del número de reglas y la cantidad de paquetes por segundo que puede procesar. Estas trazas ocupaban 500 MB (801000 paquetes) e incluían principalmente tráfico P2P y HTTP. Para enviarlas utilizamos tcpreplay, indicando una tasa de paquetes constante. De esta manera exigimos en mayor medida a la herramienta dando como resultado los valores pico que podría llegar a soportar antes de que se comience a caracterizar incorrectamente. A continuación están los resultados obtenidos, para distintas tasas de paquetes enviados. Se observa como el número de reglas afecta a la cantidad de paquetes por segundo que es posible procesar y en consecuencia también afecta a la caracterización de estos flujos.



Los datos de la gráfica anterior se obtuvieron utilizando una computadora con procesador Dual Core AMD Athlon(tm) II X2 250 (3 Ghz), al configurar que se buscara entre 60 patrones de L7-Filter fue posible procesar hasta una tasa de 14000pps, con 30 reglas se alcanzó a 32000pps, mientras que para 5 reglas se pueden procesar sin pérdidas cerca de 46000pps. Cabe aclarar que esta tasa de paquetes corresponde a un valor cercano a los 230 Mbps. Por último, se intentó encontrar cuál era el máximo de paquetes que se lograban capturar independiente-

mente de L7-Filter, según los resultados obtenidos, este máximo correspondería a 100.000 pps (o 500Mbps dada la traza que enviamos). Este valor obtenido puede engañar a simple vista y puede no ser del todo cierto, dado que a estas velocidades pueden existir múltiples factores que generen pérdidas desde el momento que se intentan enviar las trazas. Por otro lado, permite afirmar que los resultados anteriores sí pueden ser tomados como válidos e indican que el responsable de generar mayor procesamiento y limitar la cantidad de paquetes que se pueden analizar es L7-Filter.

Además del equipo anterior se utilizó otra computadora con procesador AMD Athlon(tm) XP 2100+ y 512 MB RAM. Se compararon los resultados obtenidos por ambos equipos al utilizar 30 reglas y se encontró que se duplica la capacidad de procesamiento.



Durante estas pruebas, se relevó el consumo de CPU, de memoria y la cantidad de interrupciones (IRQ). En el caso de la memoria se identificó que la misma no se ve comprometida, aún en el equipo con 512 MB. Esto era de esperarse ya que la memoria es utilizada principalmente por conntrack. [3]
En cambio, un aumento en la cantidad IRQ's afecta considerablemente y genera un alto consumo de CPU, a medida que aumentamos la cantidad de paquetes que llegan a la tarjeta de red. Para cada paquete, la tarjeta debe interrumpir al procesador para que éste procese el paquete y lo pase al stack de red para que sea analizado. En el caso del equipo Dual-core se observa un incremento en la performance ya que mientras que un procesador atiende las interrupciones, el otro procesador realiza operaciones para reconocer el tráfico.

[3]En caso de poner la herramienta en producción, hay que tener cuidado en parar el demonio correspondiente al log de linux si se habilitó la opción "Layer 7 debugging output" cuando se compiló el kernel

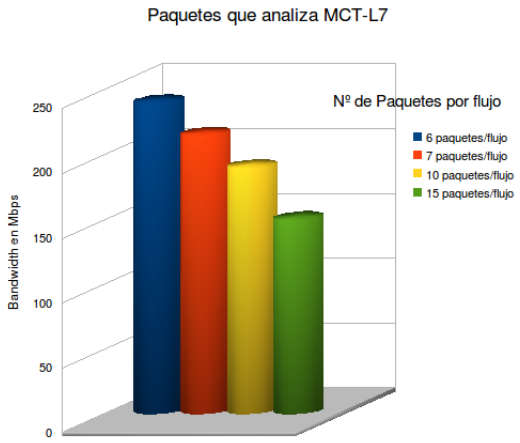
CPU	%Usr	%Nice	%Sys	%Irq	%Softirq	%Wait	%Idle
Ambos	8	0	32	10	39	0	11
1	16	0	64	0	2	0	18
2	0	0	0	19	77	0	4

Como se observa la carga de CPU se reparte principalmente entre softirq y system, la primera ha aparecido a partir de la versión de 2.5 del kernel de Linux y permite una mejor escalabilidad para sistemas multiprocesador (SMP) dando una nueva forma de manejar las tareas que se deben realizar luego de que ha ocurrido una interrupción. Existe un número definido de softirqs y cada una de ellas tiene cierta afinidad con alguno de los procesadores existentes, por esa razón se observa al procesador 2 encargado principalmente de ellas. Mientras tanto, el procesador 1 se encuentra ejecutando acciones a nivel del sistema (kernel-space) [43].

Una forma de disminuir la cantidad de interrupciones y por lo tanto mejorar la performance del sistema, es utilizando tarjetas que soporten NAPI. Estas tarjetas reducen la cantidad de interrupciones guardando los paquetes en un buffer evitando realizar interrupciones por cada paquete que llega a la tarjeta de red.

Como se ha mencionado en varias oportunidades, L7-Filter observa una cantidad de paquetes predefinida por flujo, por defecto este valor es 10 pero nos interesa encontrar como varía la caracterización al modificarlo ya que un cambio en este parámetro podría afectar considerablemente la performance del sistema. Manteniendo la tasa de paquetes enviados, en un valor cercano al máximo que se puede procesar antes de comenzar a perder paquetes, se fue variando el parámetro layer7_numpacket que se encuentra en el /proc.

Layer7_numpackets	Ancho de Banda (Mbps)
6	240
7	215
10	190
15	150



Al aumentar la cantidad de paquetes a analizar por flujo, disminuye la performance del sistema, por lo tanto queremos que este valor sea el mínimo posible antes de que comience a afectarse la caracterización. Utilizando las trazas que se generaron en la parte anterior para una diversidad de protocolos, se observa que cada uno de ellos requiere distintas cantidades de paquetes de forma de caracterizar correctamente. Se observa claramente que ssh, shoutcast, irc, http necesitan poder observar hasta el séptimo paquete de una conexión para identificar de forma correcta.

	Layer7_numpackets		
Protocolo	5	7	10
ssh	0	269	269
vnc	22818	22818	22818
pop3	8065	8065	8065
smtp	5592	5592	5592
ftp	2866	2866	2866
dns	29	29	29
telnet	190	190	190
sip	4104	4104	4101
shoutcast	0	5316	5316
ares	1772	1772	1772
bittorrent	18075	18075	18075
gnutella	12717	12717	12717
ssl	30	30	30
http	1115	1115	1215
rtp	6	6	6
dhcp	13	13	13
irc	0	682	682

Módulo 2

Si bien el módulo que restringe en mayor medida la performance del sistema es el de captura y análisis, nos interesa también dar algunos datos relativos al comportamiento y consumo de recursos de los otros dos.

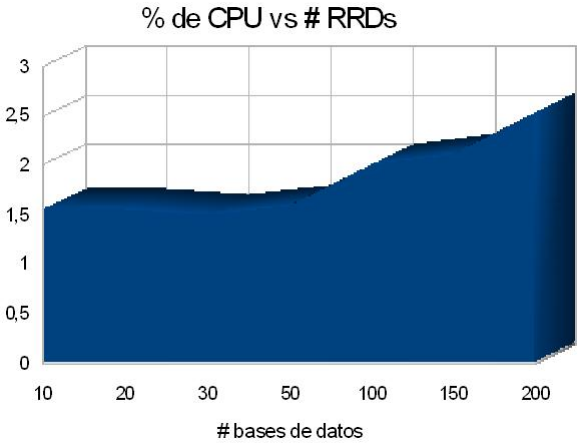
El módulo 2, se basa principalmente en consultar las variables de Iptables en el kernel de forma periódica, para luego guardarlas en sus correspondientes bases de datos. Una de las características fundamentales que nos llevó a utilizar Collectd para esta tarea, fue la manera en que interactúa con el Kernel cuando realiza las consultas de las variables. Esto es importante ya que no trabaja a nivel del user-space para chequear los contadores sino que lo hace directamente al kernel, evitándose así las sobrecargas de comunicación entre ambos niveles y elevando considerablemente la performance del sistema.

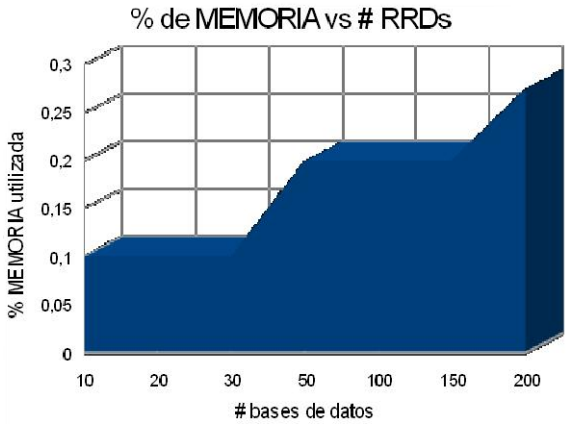
Como fue explicado dentro de la sección del módulo 2, necesitamos habilitar dos plugins para trabajar. Uno de ellos es el plugin rrdtool, utilizado para configurar los diferentes archivos RRAs, el otro plugin utilizado fue el de Iptables, encargado de consultar periódicamente los contadores en el kernel y guardar el incremento de bytes y paquetes de cada protocolo ingresado en la tabla de reglas. Este plugin para Iptables utiliza la librería libiptc, que es la misma que utiliza Iptables para modificar o consultar las reglas en las diferentes cadenas y tablas. Mediante las funciones previstas por esta librería, el demonio conntrack puede acceder a las variables del sistema de forma directa, pasándole como datos la ubicación de la regla dentro de las cadenas y tablas en donde se encuentra, y qué contador se quiere consultar (paquetes y/o bytes). El hecho de que trabaje con esa librería, hace que el demonio interactúe en forma directa con el kernel sin la necesidad de recurrir a las system call.

Con respecto a las bases de datos, gracias a la forma como trabajan las RRD podemos mantener siempre bajo control el espacio en disco utilizado. Cuando se quiere agregar una nueva RRD para guardar la información de un protocolo, dicha base se crea utilizando una cantidad constante de espacio en disco. En nuestro caso, cada nueva bases de datos creada ocupa un espacio de 144,9 KiB, por cada protocolo son dos RRD (una conteniendo la información en paquetes y la otra en bytes) consumiendo en un total de 290 KiB por cada uno.

A medida que aumentamos la cantidad de aplicaciones que deseamos reconocer, aumenta también la cantidad de consultas que debe realizar collectd al sistema y también la cantidad necesaria de escrituras en disco que se deben realizar.

A continuación mostramos dos gráficas en las cuales se puede ver cómo influye la cantidad de bases de datos creadas contra el porcentajes de cpu y de memoria utilizados por collectd para consultar las variables en el Kernel y guardarlas luego en las RRD's.





Con esto resultados, podemos concluir que este módulo no utiliza recursos en el sistema suficientes como para que se note una disminución en la performace. El espacio en disco consumido por las bases de datos es insignificante para las capacidades que manejan los discos de hoy en día y la manera de consultar collectd al sistema hace que este módulo sea muy eficiente.

Módulo 3

Como fue mencionado en varios puntos en esta documentación, este módulo es utilizado simplemente para mostrar en forma amigable la información guardada por Collectd en las RRD's.

CACTI necesita de un servidor web (utilizamos Apache2 en nuestro caso) para poder trabajar y levantar su interfaz web, y para acceder a ésta se necesita utilizar algún navegador como por ejemplo Internet Explorer, Firefox, etc. Todos estas aplicaciones necesitan consumir recursos en el sistema para funcionar en forma correcta, tanto a nivel de CPU como de memoria. En esta sección veremos cómo es el comportamiento del sistema a nivel de performance y cuál es su comportamiento ante la escasez de recursos cuando se trabaja con la interfaz gráfica de CACTI levantada en la misma máquina que captura.

El procedimiento seguido para medir la performace del sistema, fue nuevamente enviar trazas conocidas a distintas velocidades verificando los valores obtenidos para cada una, y cuáles eran los recursos consumidos por el sistema en esos momentos. Se midió tanto la carga del procesador, la memoria consumida, y en particular se siguió el consumo de recursos de algunos procesos en particular. El objetivo de estas pruebas fue verificar cómo es la distribución de carga del procesador entre el user-space, los procesos del sistema y la atención a periféricos (NIC). A medida que aumentábamos la tasa de paquetes a la que enviábamos la trazas, el nivel de utilización del CPU para los procesos a nivel de usuario aumentaba debido al consumo de los procesos que necesitaba CACTI para funcionar. Los datos obtenidos para algunos de los procesos principales fueron los siguientes:

El explorador llego a consumir hasta un 20 % de CPU o inclusive más dependiendo de las tareas que se realicen y hasta un 9 % de la memoria libre, mientras que la interfaz gráfica (proceso Xorg), consume aproximadamente un 7 % de CPU y un 4 % de memoria.

Cuando se alcanzaban los 45 Mb de tráfico se notaba cómo dejaba de responder la interfaz gráfica del sistema operativo, sin embargo en las lecturas del nivel de carga del CPU, aún se atendían procesos pertenecientes al user-space. Aunque la interfaz gráfica del CACTI no trabaja en forma correcta debido al procesamiento de los datos desde la NIC, los módulos de captura y análisis y el demonio collected seguían trabajando sin inconvenientes. Cuando alcanzábamos entre los 50 y 55 Mb, la interfaz gráfica del sistema operativo dejaba de responder, sin embargo el módulo de captura funcionaba aún sin perder paquetes y reconociendo los distintos flujos en forma correcta. Cuando alcanzábamos los 60 Mb las lecturas del consumo del CPU mostraba como en ocasiones dejaba de atender las operaciones pertenecientes al espacio de usuario para atender únicamente las peticiones del sistema operativo y de las IRQ del sistema. Los contadores de Iptables mostraban como el sistema

empezaba a perder algunos paquetes y a identificar en forma errónea el tráfico.

Como conclusión podemos destacar la ventaja de haber diseñado nuestro módulo de análisis para trabajar directamente desde el Kernel. Esto se ve reflejado cuando el procesador se ve saturado por peticiones tanto del nivel de usuario como del Kernel y periféricos, sin embargo éste atiende las peticiones con mayor prioridad (procesos pertenecientes al kernel). Es por esto que si bien vemos como la interfaz gráfica de CACTI se congela y no reacciona, el módulo de análisis aún funciona en forma correcta obteniéndose resultados precisos.

Esta ventaja de trabajar con la interfaz gráfica a nivel de usuario y mantener el módulo de captura dentro del propio Kernel se ve reflejada ante la posible aparición de ráfagas de tráfico. En este caso el CPU dejara de atender los procesos que necesita CACTI para funcionar, pero sí atenderá los procesos de mayor prioridad. Entre estos procesos se encuentran el módulo de L7-Filter , Netfilter, etc. Con este comportamiento, si una de estas ráfagas sobrepasa la capacidad de procesamiento del sistema, éste atenderá únicamente a los paquetes y al módulo de captura, logrando identificar de forma correcta, mostrará los datos más adelante cuando se liberen los recursos necesarios.

7. Comentarios finales y trabajo a futuro

Hemos diseñado un sistema al que llamamos MCT-L7, que permite identificar y representar gráficamente cómo se distribuye el tráfico generado por las aplicaciones utilizadas actualmente. Este sistema busca patrones en el payload de los paquetes utilizando expresiones regulares y contabilizando el número de paquetes o bytes que cumplen con estos patrones, luego almacena los datos para que sea posible mostrarlos de una forma amigable.

Tuvimos que generar tráfico correspondiente a diferentes aplicaciones de forma de probar si es adecuado utilizar expresiones regulares para caracterizarlo. Se consiguieron resultados bastante alentadores en la mayoría de los casos llegando a caracterizar correctamente cerca del 100 % del mismo, como es el caso de Gnutella. Sin embargo, esto no sucede en todos los casos y en particular no funcionará correctamente para los protocolos que utilicen algún tipo de cifrado para transmitir sus datos (Ares es un ejemplo de ello). Se han encontrado expresiones regulares que no estaban actualizadas y fue posible mejorarlas o adecuarlas fácilmente, obteniendo mejores resultados (Msn Messenger). Por lo tanto, RegEx es un buen método con el cual es factible reconocer a qué aplicación corresponden los distintos flujos, y brinda la posibilidad de crear patrones a medida que surjan nuevas aplicaciones.

Una de las desventajas de este método, es que al inspeccionar en profundidad los paquetes, puede considerarse que se está afectando la privacidad de los usuarios, dado que estamos observando información contenida en el payload. Sin embargo, no toda la información en el payload corresponde a datos del usuarios, también hay información que es generada por las aplicaciones y esto es lo que buscamos. Por lo tanto, es discutible hasta que punto este método afectará la privacidad.

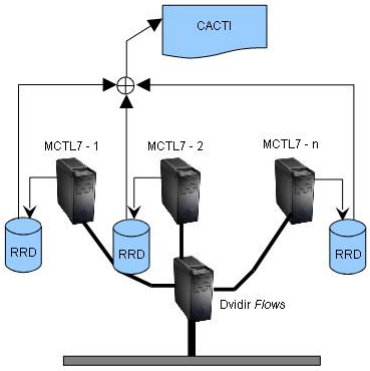
Otra desventaja es el gran consumo de CPU que se necesita. Por ello hemos definido máximos alcanzables utilizando dos equipos con características diferentes. Se demostró que es posible lograr tasas superiores a los 200Mbps con una computadora convencional y una tarjeta Gigabit ethernet, analizando los primeros paquetes de cada flujo. Es importante recalcar que al no disponer de tráfico real, se utilizaron trazas no lo suficientemente grandes como para demostrar cuál sería el comportamiento de la herramienta a altas tasas de tráfico luego de atravesar el transitorio inicial. En dicho transitorio se debe realizar el mayor esfuerzo en cuanto a procesamiento, debido a que durante este tiempo se deben analizar casi todos los paquetes que se reciben.

Por lo tanto, podemos considerar que hemos encontrado un máximo ante ráfagas, y en condiciones normales sería posible alcanzar un valor aún mayor. Profundizando aún más, verificamos que se llega a los mismos resultados de identificación

si se busca en los primeros 7 paquetes de cada flujo en lugar de los primeros 10, como recomienda el equipo de desarrollo de L7-Filter. Sin embargo al analizar menos paquetes, estamos aumentando el ancho de banda que podemos alcanzar en más de un 10 %, llegando a los 215 Mbps. Para tener un punto de comparación, cabe mencionar que el ancho de banda actual que tiene contratado Antel es de 11 Gbps [44], por lo que utilizando una PC de escritorio sin ningún equipamiento particular y con una inversión mínima es posible caracterizar casi el 2 % de dicho tráfico.

En la actualidad existen equipos comerciales que afirman ser capaces de caracterizar decenas de Gbps detectando el 99 % del tráfico P2P correctamente [45] [46]. Estos equipos tienen altos requerimientos de hardware que aumentan considerablemente su costo. Por otro lado, existen desarrollos en los cuales manteniendo bajos requerimientos de hardware, implementan mejoras en el kernel del sistema operativo. Éstos modifican la manera como se realiza la captura de paquetes y el pasaje de éstos desde la NIC hasta el espacio de usuario, disminuyendo la cantidad de escrituras y lecturas necesarias (PF_RING [47]). Gracias a lo anterior, si bien es posible capturar altas tasas de paquetes, todavía es necesario caracterizarlos, lo cual es la gran limitante. Para esto último, se proponen desarrollos que explotan la capacidad de contar con sistemas multiprocesador y con la capacidad multi-threading de Linux. Lo que se busca en estos casos, es repartir los distintos flujos hacia diferentes procesadores de forma de que cada uno de ellos procese una porción del tráfico total, aumentando considerablemente el ancho de banda que es posible procesar y caracterizar. De esta manera se logra procesar tasas del orden del Gbps contando simplemente con un equipo Quad core [48] [49].

Razonando de manera similar, y aplicando estas ideas para aumentar la performance del sistema, debemos contar con un equipo que sea capaz de repartir los flujos hacia distintas computadoras, cada una ellas corriendo MCT-L7. Es importante notar que se debe distribuir todos los paquetes perteneciente a un flujo hacia un único pc con MCT-L7 o un único procesador, donde cada uno se encargará de analizar una porción del tráfico total. La información obtenida de todos ellos podrá acumularse y desplegarse de manera similar que si se trabajara con un único sistema MCT-L7. El inconveniente principal cuando se desea analizar tráfico perteneciente a un ISP o a una red grande es que no siempre están disponible los dos sentidos del flujo. Esto limita la capacidad de identificación de los sistemas que se basan en DPI como es nuestro caso, degradando significativamente la precisión de reconocimiento de las aplicaciones.



Proponemos como trabajos a futuro, encontrar una forma eficiente de separar los flujos para que éstos puedan ser distribuidos hacia distintas máquinas que utilicen el sistema MCT-L7. La dificultad consiste en encontrar una forma de "rutear" estos flujos en forma balanceada para equilibrar la cantidad de información que procesen los sistemas de identificación. Este equipo que trabajará como un separador de flujo podría ser desarrollado utilizando el sistema de connection tracking estudiado, el cual es capaz de almacenar las distintas conexiones en sus tablas e identificar una gran cantidad de información muy eficientemente. Una posible forma de realizar esto sería rutear a partir de la tupla (IP origen, IP destino, puertos y protocolo), realizando una operación que incluya estas variables y tome decisiones hacia donde redirigir los paquetes dependiendo del resultado de ésta. Como vimos, conntrack trabaja de esta manera, calculando un índice a partir de las variables anteriores, con el que decide donde debe guardar la información de la conexión. Ya que conntrack requiere poco procesamiento se podría avanzar un paso más, dando distintos destinos dependiendo del índice calculado.

Otro aspecto a mejorar sería la forma en que se recorren las reglas en el módulo de identificación. Iptables utiliza un algoritmo que depende en gran medida del número de reglas definidas y genera en muchos casos comparaciones innecesarias. En este punto, se pueden implementar mejoras en como se definen las reglas, utilizando por ejemplo un sistema de árbol. Se podrían agrupar los patrones según ciertas características, utilizando en primer lugar las reglas más simples y que consumen menos recursos. Otra posibilidad es definir patrones más generales que puedan aplicar rápidamente con uno o dos paquetes e identificar alguna característica en los flujos. Luego, dependiendo de esta característica identificada, aplicar solo los patrones que tienen buenas posibilidades de encontrar su ExpReg en el payload, logrando así la estructura de árbol mencionada [50].

Existe una gran cantidad de trabajos sobre estos puntos actualmente, dado el gran interés en contar con mayor información sobre que es lo que realmente atraviesa las redes de datos. Además de diseñar un sistema que integre varias aplicaciones de forma de lograr el objetivo planteado, hemos intentado estudiar en profundidad una herramienta que utiliza DPI dando sus ventajas y desventajas e indicando que es viable y eficaz utilizar una herramienta con estas características para conocer más acerca del tráfico de Internet.

Referencias

[1] *http://www.iana.org/assignments/port-numbers*.

[2] T. KARAGIANNIS, A. BROIDO, M. FALOUTSOS, AND K. CLAFFY. *Transport Layer Identification of P2P Traffic*.In IMC'04, Taormina, Italy,October 25-27, 2004

[3] S. SEN, O. SPATSCHECK, AND D. WANG. ACCURATE, SCALABLE IN-NETWORK

[4] S. A. BASET, H. SCHULZRINNE. *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol*. In Proc. IEEE Infocom 2006,2006 .

[5] INTENATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY (IJCSNS). *An efficient Flow Analysis Scheme for identifying Various Application in IP-based Networks* . Vol 8 No.12, December 2008

[6] G. GOMEZ , P. BELZARENA. *Early traffic classification using Support Vector Machines* - 2009.

[7] *http://www.ipoque.com/resources/white-papers*

[8] A. HESS, T. GINGOLD, S. R. GARZON, G. SCHAFER *Performance comparison between user and kernel space implementation* - 2009.

[9] SMALL TRAFFIC PERFORMANCE OPTIMIZATION FOR 8255X ETHERNET CONTROLLER - INTEL 2003.

[10] HTTP://WWW.LINUXJOURNAL.COM/ARTICLE/7660?PAGE=0,0 - 2005.

[11] *http://www.snort.org/*.

[12] M. A. RODRÍGUEZ GARCÍA MIGUEL , A.ORENES FERNÁNDEZ *Snort: Análisis de la herramienta. Versión practica*.

[13] *http://hippie.oofle.com/*.

[14] *http://ourmon.cat.pdx.edu/ourmon/info.html#l7match*

[15] *http://www.filewatcher.com/p/ourmon2.5.tbz.274879/mrourmon/etc/ourmon.conf.html*.

[16] *http://www.notp.org/*

[17] *www.netfilter.org*

[18] *l7-filter.sourceforge.net*

[19] FILTRADO DE ALTO RENDIMIENTO CON LINUX *http://talika.eii.us.es/ javier/netfilter.pdf*

[20] P. NEIRA AYUSO *Netfilter's connection tracking system 131.106.3.253/publications/login/2006-06/pdfs/neira.pdf*

[21] *Designing And Implementing Linux Firewalls And QoS* - 2006

[22] *http://www.aptalaska.net/ jclive/IPTablesFlowChart.pdf.*

[23] *http://oss.oetiker.ch/rrdtool/*

[24] *http://collectd.org/*

[25] *http://www.cacti.net/*

[26] *http://www.wireshark.org/*

[27] *http://www.tcpdump.org/*

[28] *http://bittwist.sourceforge.net/*

[29] *http://www.opendpi.org/*

[30] *http://httperf.comlore.com/*

[31] *http://www.apache.org/*

[32] *http://www.netfilter.org/projects/conntrack-tools/index.html*

[33] *http://www.myce.com/news/ISPs-say-P2P-bandwidth-costs-too-much-1_3-billion-in-2003-5939*

[34] *http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009*

[35] *http://torrentfreak.com/bittorrent-still-king-of-p2p-traffic-090218/*

[36] *The Gnutella Protocol Specification v0.4 - http:// rfc-gnutella.sourceforge.net/developer/stable/index.html*

[37] D. WANG, O. SPATSCHECK, S. SEN, ACCURATE, SCALABLE IN-NETWORK IDENTIFICATION OF P2P TRAFFIC USING APPLICATION SIGNATURES - 2004.

[38] DAVID ALEXANDRE MILHEIRO DE CARVALHO, TOWARDS THE DETECTION OF ENCRYPTED PEER-TO-PEER FILE SHARING TRAFFIC AND PEER-TO-PEER TV TRAFFIC USING DEEP PACKET INSPECTION METHODS. University of Beira Interior - Covilhã, Portugal 2009

[39] *http://sander.vanzoest.com/talks/2002/audio_and_apache/*

[40] *http://www.icecast.org*

[41] *http://www.wallfire.org/misc/netfilter_conntrack_perf.txt*

[42] *http://people.netfilter.org/kadlec/nftest.pdf*

[43] I'LL DO IT LATER: SOFTIRQS, TASKLETS, BOTTOM HALVES, TASK QUEUES, WORK QUEUES AND TIMERS

[44] *http://www.infoycom.org.uy/?q=node/4170*

[45] *http://www.ipoque.com*

[46] *http://www.qosmos.com/*

[47] *http://www.ntop.org/PF_RING.html*

[48] T. LACERDA, S. FERNANDES, A. OLIVEIRA, D. SADOK, J. KELNER *Performance-driven Development of Deep Packet Inspection Systems on Commodity Platforms* -2009.

[49] J. WANG, H. CHENG, B. HUA, X. TANG *Practice of Parallelizing Network Applications on Multi-core Architectures*– Junio 2009.

[50] S.DHARMAPURIKAR , F. YU, S. KUMAR, P. CROWLEY, J. TURNER *Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection* - September 11-15, 2006, Pisa, Italy.

ANEXO A

8. Anexo A: Pruebas

En la siguiente tabla se muestran los diferentes criterios que utiliza la herramienta L7_filter para clasificar los protocolos que elegimos y luego detallamos cada uno de esos parámetros.

Name	Vel_Kspace	Vel_Uspace	Calidad	Grupos	Notas
SIP	F	F	4	Telèfono Internet	A,B
FTPVoIP IETF PS	S	NSF	5	Documents retrieval/ IEFT S	–
HTTP	S	NSF	5	Documents retrieval/ IEFT DS	C
RTP	F	F	3	Streaming video/ IETF S	A,B
Telnet	VF	F	4	Acceso Remoto/ obsoleto / IETF S	–
VNC	VF	F	5	Acceso Remoto	–
SSH	VF	F	5	Acceso Remoto/ Secure / IETF DS	–
SMTP	NSF	F	1	Mail / IETF S	–
POP3	F	F	1	Mail / IETF S	–
skypeout	S	NSF	3	VoIP / P2P/ propietario	A
skypetoskype	VF	F	3	VoIP / P2P / propietario	A
Applejuice	VF	F	5	P2P	–
Ares	VF	F	4	P2P / Open Source	B
Bittorent	S	NSF	4	P2P / Open Source	B
Gnutella	NSF	NSF	4	P2P / Open Source	–
msnmessager	S	NSF	4	Chat / propietario	–
Shoutcast	S	NSF	4	Streaming audio	–
irc	F	F	4	P2P	–

Descrpición de las abreviaciones:

- IETF PS: IETF proposed estándar
- IETF S: IETF standar
- IEFT DS: IETF draf estándar
- A: Overmaching
- B: Undermaching

- C: Superset
- VF: Very Fast
- F: Fast
- NSF: Not so fast
- S: Slow

Categorías:

Calidad

La çalidad"da una idea aproximada de lo bien que funciona el modelo. Es una medida que contempla varias cosas, incluyendo:

- qué tan bien se entiende el Protocolo.
- hasta qué punto el modelo ha sido probado.
- en qué variedad de situaciones el modelo ha sido probado.
- qué fracción de tráfico es identificada correctamente. Para más detalles, lea el archivo de patrones o de entrada wiki del protocolo.

La calidad puede ser clasificada en:

5 Muy Bien: Funciona siempre

4 Bien: Funciona dentro de lo que se conoce

3 Regular: Probablemente funcione

2 Malo: Puede ser que funcione como puede que no

1 Muy Malo: Probablemente no funcione

Velocidad

La velocidad mostrada en la primer columna es cuando se utiliza la herramineta en espacio de kernel y la segunda es la velocidad cuando se utiliza en espacio de usuario.

- Fast: 3-10 segundos.
- Not so fast: 10-100 segundos.
- Slow:>100 segundos.

Grupos

Los protocolos son marcados en uno o varios "grupos", en donde algunos grupos se refieren a qué tipo de objetivos tiene cada protocolo. Los misos permiten tratar un conjunto de protocolos de la misma manera, sin necesidad de que el usuario seleccione cada protocolo de manera individual. Otros grupos indican si un protocolo se documenta en un IETF RFC, ya sea normalizado por cualquier organismo oficial o un no-estándar, utilizado principalmente por los programas de código abierto, o de propiedad. Esto da una idea de la volatilidad de estos protocolos.
Por ejemplo, para estándares IETF es muy improbable que cambie el comportamiento o se rompan los patrones de l7-filter de repente. Sin embargo, los protocolos propietarios pueden cambiar en cualquier momento sin previo aviso y la naturaleza de los cambios pueden ser un secreto muy bien guardado.

Notas

- Patrón Overmatching: es difícil o imposible escribir un patrón para este protocolo de forma fiable que sólo coincida con el protocolo previsto. Podría pasar que el uso de este modelo rinda resultados positivos falsos, por lo que probablemente sólo debería utilizarlo junto con otras características, como el puerto o IP.

- Patrón Undermatching: es difícil o imposible escribir un patrón para este protocolo que coincida con todas las conexiones.

- Superset: este patrón machea el tráfico que también sería macheado por otros protocolos. Si el mismo está por delante de uno de estos patrones en las reglas de Iptables, el tráfico no será machado por los otros protocolos.

- Subset: Este patrón machea tráfico que es un subconjunto del tráfico macheado por algún otro protocolo.

ANEXO B

9. Anexo B: Iptables

Detalles técnicos de L7_Filter

Básicamente la idea de l7-filter es simple: utiliza expresiones regulares para machear conexiones de la capa de datos de aplicación para determinar que protocolo esta siendo usado. L7-Filter sólo mira los primeros 10 paquetes o los primeros 2 Kbyte del payload de una conexión, en donde buscará en dichos paquetes los patrones y palabras claves que identifican el establecimiento o la negociación de parámetros del protocolo, como ser, el establecimiento de la conexión. Esto es así, ya que no tendría sentido a nivel de eficiencia del motor de búsqueda y además prácticamente sería irrealizable buscar patrones en todos los paquetes que circulan en la red. De esta forma, al examinar sólo los primeros paquetes de una conexión, l7-filter es razonablemente eficiente a pesar de usar un algoritmo más complicado que la mayoría de los clasificadores de paquetes tradicionales.
En lugar de que el usuario tenga que proveer las expresiones regulares a través de línea de comando, l7-filter provee una serie de archivos donde se encuentran los patrones. Por ejemplo para machear con el protocolo "Http" el usuario ejecuta:

Iptables ... -m layer7 - - l7proto http

y a continuación Iptables lee estas expresiones regulares que definen al protocolo "http" en la dirección /etc/l7-protocols/*/http.pat.

En las siguiente líneas mostramos un Pseudo Código del módulo del Netfilter sobre el kernel de Linux trabajando con L7-Filter:

```
/*This is the estándar Netfilter match function, which
returns true on a match and false otherwise.
Obviously, many details are ignored here. */
int match(packet, protocol)
   if(regular expression for protocol is not compiled)
     compile it and put it in a list of compiled
regexps;
   else
     fetch the compiled pattern from the list;
   if(already classified this connection)
     if(classification matches one we're looking for)
       return true;
     else
       return false;
   if(seen too many packets with no match)
     return false;
   Append application layer data to data buffer;
   if(data buffer matches regexps for the protocol
we're looking for)
```

```
    Mark the connection as identified;
    return true;
  else
    return false;
```

Iptables

Iptables es esencialmente un filtro IP y como lo indica su nombre, actúa en la capa 3 del modelo de referencia OSI. Sin embargo, tiene también la habilidad de trabajar en la capa 4, y en nuestro caso lo utilizaremos para realizar filtrado a nivel de capa 7. Si la implementación de un filtro IP sigue estrictamente la definición, este sería solamente capaz de filtrar paquetes basados en su encabezado IP (Dirección origen, dirección destino, TOS/DSCP/ECN, TTL, Protocolo, etc.). Debido a que la implementación de Iptables no es estricta respecto a esta definición, es posible clasificar paquetes basados en encabezados de capas superiores y buscar expresiones regulares en el payload de los paquetes. Esto último es posible gracias al módulo L7-Filter el cual es detallado más adelante.

Iptables permite al administrador del sistema definir reglas acerca de qué hacer con los paquetes de red pero no puede seguir el flujo de los paquetes, o juntar la información enviada en distintos paquetes de forma de contar con todos los datos enviados. Sí se puede mantener un estado de las conexiones e identificar qué paquetes pertenecen a la misma por medio de los números de puertos y direcciones IPs tanto de origen como destino. En este punto justamente es donde entra en acción el sistema de seguimiento de conexiones (connection tracking) previamente explicado.

A continuación se define qué es una tabla, una cadena, etc. y dentro de cada una de esas definiciones se nombran las diferentes opciones. Solamente se detallan las utilizadas por nosotros para implementar nuestro sistema.

Una tabla es un constructor de Iptables que delinea amplias categorías de funcionalidades, como filtrado de paquetes, traducción de direcciones de red (NAT) o manipulación de paquetes. Cada tabla tiene un propósito específico, existen cuatro tablas: filter, nat, mangle y raw. Reglas de filtrado son aplicadas a la tabla filter, reglas NAT son aplicadas a la tabla nat, reglas especializadas que alteran el contenido del paquete son aplicadas en la tabla mangle, y excepciones para conntrack se aplican en la tabla raw

Atravesando las tablas y cadenas

En esta sección se discutirá cómo los paquetes atraviesan las distintas cadenas, y en qué orden. También se indicará el orden en que las tablas son atravesadas. Conocer esto será muy valioso a la hora de definir nuestras reglas, por ejemplo es

esencialmente necesario saber en que lugar es necesario definir una regla cuando esta puede modificar el ruteo de los paquetes.

Cuando un paquete entra a la máquina, pasa por la tarjeta para luego dirigirse al correspondiente driver del dispositivo en el kernel. Luego el paquete empieza una serie de pasos en el kernel para finalmente ser enviado a alguna aplicación, ser reenviado hacia otro host, etc.

CASO: Tráfico destinado al local host

Paso	Tabla	Cadena	Descprición
1			En el cable
2			Ingresa a la interfaz
3	raw	PREROUTING	Esta cadena es utilizada para manejar los paquetes antes de que connection tracking intervenga. Puede ser usadopor ejemplo para indicar que una conexión específica no sea manejada por connection tracking
4			Este paso se da cuando el código de connection tracking toma lugar. Ver Estado de máquina
5	mangle	PREROUTING	Esta cadena es normalmente usada para manipular paquetes por ejemplo, cambiar el campo TTL.
6	nat	PREROUTING	Esta cadena es normalmente utilizada para DNAT, es recomendable evitar filtrado en esta cadena ya que es salteada en algunos casos.
7			Decisión de ruteo, el paquete tiene como destino el local host o debe ser forwardeado y hacia adonde
8	mangle	INPUT	Esta cadena se puede usar para manipular paquetes, luego de haber sido ruteados, pero antes de ser realmente enviados al proceso en la máquina.
9	filter	INPUT	Aquí es donde realizamos el filtrado para todo el tráfico entrante destinado hacia el local host. Notar que todo el tráfico entrante destinado para el local host atraviesa esta cadena.
10			Proceso local o aplicación que recibe el paquete.

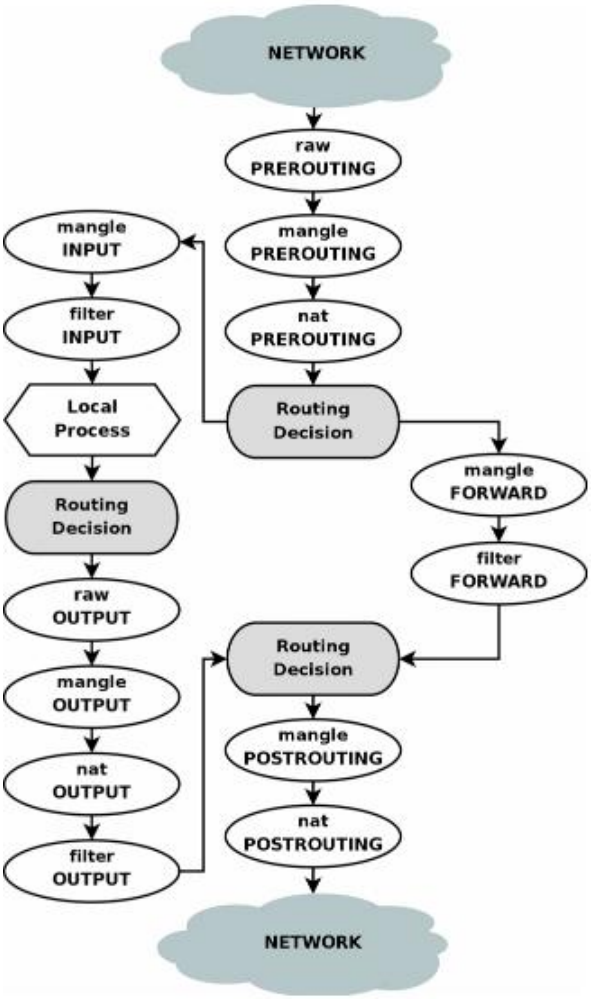
CASO: Local host como origen

Paso	Tabla	Cadena	Descprición
1			Proceso local o aplicación que genera el paquete.
2			Decisión de ruteo, que dirección de origen e interfaz se usará. Se definirá todo la demás información necesaria.
3	raw	OUTPUT	Aquí es donde se puede realizar trabajo antes de que connection tracking intervenga para los paquetes localmente localmente generados. Por ej,puedes marcar conexiones que no quieres que sean detectadas por el connection tracking.
4			Aqui es donde connection tracking interviene para los paquetes generados localmente, por ejemplo cambie de estado de NEW a ESTABLISHED
5	mangle	OUTPUT	Aquí es donde se manipulan los paquetes, se aconseja no filtrar en esta etapa ya que puede tener efectos colaterales
6	nat	OUTPUT	Esta cadena puede ser usada para hacer NAT en paquetes salientes del firewall
7			Nueva decisión de ruteo, en caso de que las previas mangle y nat hayan modificado como el paquete debía ser ruteado.
8	filter	OUTUT	Aquí se filtran los paquetes que salen del local host luego de haber sido ruteados, pero antes de ser realmente
9	mangle	POSTROUTING	La cadena POSTROUTING en la tabla mangle es usado principalmente cuando queremos manipular paquetes antes que abandone nuestro host, pero luego de haber realizado la decisión de ruteo. Esta cadena aplica tanto para paquetes que hayan atravesado la cadena como también para paquetes creados por el propio firewall.
10	nat	POSTROUTING	Aquí es donde se realiza SNAT. Es recomendable no realizar filtrado aquí debido a que puede tener efectos colaterales.
11			Sale por alguna interfaz
12			Se encuentra en el cable

CASO: Paquetes reenviados

Paso	Tabla	Cadena	Descprición
1			En el cable.
2			Ingresa a la interfaz.
3	raw	PREROUTING	Esta cadena es utilizada para manejar los paquetes antes de que connection tracking intervenga.Puede ser usado por ejemplo para indicar que una conexión especifica no sea manejada por connection tracking.
4			Este paso se da cuando el código de connection tracking toma lugar. Ver Estado de máquina
5	mangle	PREROUTING	Esta cadena es normalmente usada para manipular paquetes, por ejemplo,cambiar el campo TTL.
6	nat	PREROUTING	Esta cadena es principalmente usada para DNAT. Es recomendable evitar filtrado en la cadena ya que se saltea saltea en alguno de los casos.
7			Decisión de ruteo, el paquete tiene como destino el local host o si debe ser forwardeado y hacia a donde.
8	filter	FORWARD	La cadena FORWARD de la tabla de mangle se puede manuipular los paquetes luego de haber realizado las decisiones de ruteo iniciales y antes de realizar las últimas decisiones decisiones de ruteo justo antes de enviar el paquete.
9	filter		El paquete es ruteado hacia la cadena FORWARD. Solo paquetes forwardeados pasarán por aquí, y aquí se hará todo el filtrado. Notar que TODO el tráfico forwardeado pasara por aquí, no solo en una dirección, por lo que se debe tener en cuenta esto al momento de definir las reglas.
10	mangle	POSTROUTING	Esta cadena es usada por específicos tipos de paquetes manipulando lo que queramos luego de tomar las decisiones de ruteo, pero antes de que el paquete abandone la máquina.
11	nat	POSTROUTING	Esta cadena debería ser usada para SNAT, evitar utilizarla para filtrar.
12			Sale de la interfaz
13			En el cable nuevamente.

Diagrama de las cadenas de Iptables



Tablas

Una tabla es un constructor de Iptables que delinea amplias categorías de funcionalidades, como filtrado de paquetes, traducción de direcciones de red (NAT) o manipulación de paquetes.

Existen cuatro tablas: **filter, nat, mangle** y **raw**. Reglas de filtrado son aplicadas a la tabla **filter**, reglas NAT son aplicadas a la tabla **nat**, reglas especializadas que alteran el contenido del paquete son aplicadas en la tabla mangle, y reglas que deberían funcionar independientemente del subsistema connection-tracking de Netfilter son aplicadas en la tala **raw**.

Mangle: La tabla Mangle es principalmente utilizada para manipular paquetes. Se pueden modificar los valores de los campos de los encabezados, como pueden ser TOS (Type Of Service), TTL (Time To Live), etc. También es posible marcar los paquetes para luego realizar distintos tipos de ruteo o limitaciones de ancho de banda usando estas marcas.
Esta tabla también esta definida para todas las cadenas por lo que sin importar cual sea el origen o destino de los paquetes, se pueden definir reglas que utilicen esta tabla en alguna de las cinco cadenas definidas.

Nat: La tabla Nat solo debería ser utilizada para realizar NAT (Network Address Translation) en diferentes paquetes. O sea, se usaría únicamente para modificar el campo origen y destino de los paquetes. Solo el primer paquete de un stream toca esta tabla, al resto de los paquetes automáticamente se les realiza la misma acción. Esta tabla esta definida en PREROUTING, OUTPUT y POSTROUTING.

Raw: La tabla Raw es usada principalmente para una sola cosa, setear una marca en los paquetes que no deben ser considerados por conntrack. Esto se realiza utilizando el target NOTRACK.
Si una conexión es marcada con NOTRACK, conntrack simplemente no registrará la conexión.
Esta tabla sólo se puede usar en las cadenas PREROUTING y OUTPUT. En las otras cadenas no es necesaria esta tabla ya que estos son los únicos lugares en los que puedes lidiar con los paquetes antes de que conntrack comience a actuar.

Filter: La tabla Filter es usada para realizar el filtrado de los paquetes. Podemos machear ciertas condiciones de los paquetes y filtrarlos de la manera que queramos. Este es el lugar donde realmente se ejecutan acciones sobre los paquetes y podemos tirarlos o aceptarlos dependiendo de su contenido. Esta es la tabla por defecto (si no se ha utilizado la opción -t otra tabla), Se puede utilizar en las cadenas INPUT, FORWARD y OUTPUT.

Cadenas

Cada tabla tiene su propio conjunto de cadenas incorporadas, pero el usuario también puede definir cadenas, por lo tanto se pueden construir un conjunto de reglas que estén relacionadas por una característica similar. Las cadenas que están incorporadas son **INPUT, OUTPUT, PREROUTING, FORWARD, POSTROUTING.**

Cadenas definidas por el usuario:

También se pueden definir cadenas definidas por el usuario. Si un paquete ingresa a la cadena INPUT en la tabla filter, podemos especificar un salto (jump) hacia otra cadena en la misma tabla. La nueva cadena debe ser definida por el usuario, no puede ser una cadena integrada en Iptables como INPUT o FORWARD. Si consideramos un puntero en una regla en la cadena que se está ejecutando, el puntero irá bajando de regla en regla que se encuentren definidas hasta que las reglas de la cadena hayan terminado. Una vez que haya pasado esto, la política por defecto de la cadena integrada será aplicada para todos los paquetes que no hayan cumplido las reglas.

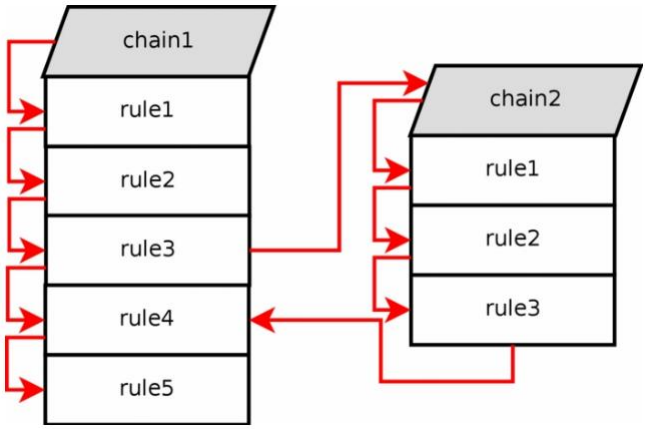


Figura 1: Cadenas definidas por el usuario

Si una de las reglas que se machea apunta a otra cadena definida por el usuario especificado en jump, el puntero saltará hacia esta cadena y continuará atravesando esta cadena desde arriba a abajo. Las cadenas definidas por los usuarios no tienen políticas por defecto al final de la cadena, sólo cadenas integradas tienen esto. Esto puede realizarse definiendo reglas al final de la cadena para los paquetes que machean. Si ninguna regla es macheada en cadenas definidas por los usuarios, el comportamiento por defecto será que el puntero retorne a la cadena original a una regla por debajo de donde había saltado.

Macheos

Cada regla de Iptables tiene un conjunto de macheos gracias a los cuales se puede especificar un destino o target que le indica a Iptables qué acción tomar sobre el paquete en caso de que cumpla con el macheo. Un macheo de Iptables es una condición que debe ser cumplida por una paquete para que Iptables procese el paquete de acuerdo a lo que especifique el target de la regla.

Target

Como se dijo anteriormente, Iptables soporta target que provocan una acción cuando un paquete machea una regla. Los target pueden ser aceptar, descartar, negar los paquetes u otras acciones más particulares. El target es útil para definir que es lo que se quiere hacer con los paquetes que cumplan ciertas condiciones.

ANEXO C

10. Anexo C: Expresiones Regulares

Una expresión regular, generalmente también llamada patrón, es una expresión que describe cadenas de caracteres sin la necesidad de enumerar sus elementos. Se usan en operaciones de apareamiento o comparación, permitiendo realizar búsquedas o sustituciones de gran complejidad.

En casi cualquier lenguaje de programación están disponibles las expresiones regulares y aunque su sintaxis es relativamente uniforme, cada uno de ellos tiene su propio dialecto. Básicamente se construyen utilizando caracteres del alfabeto sobre el cual se define el lenguaje, utilizando operadores unión, concatenación y clausura de Kleene [4].

Si sabemos de antemano la cadena exacta a buscar, no es necesario utilizar un patrón complicado, sino que podríamos usar como patrón la exacta cadena que se busca. Sin embargo, el poder de las expresiones regulares radica precisamente en la flexibilidad de los patrones, que pueden ser confrontados con cualquier palabra o cadena de texto que tenga una estructura conocida.

Caracteres y Metacaracteres

Como se ha mencionado anteriormente, un patrón puede estar formando por un conjunto de caracteres (un grupo de letras, números o signos) o por meta caracteres. Los meta caracteres son interpretados en su significado especial y no como los caracteres que normalmente representan.
El conjunto de meta caracteres para expresiones regulares es el siguiente:

\^ $. [ ] { } | ( ) * + ?

A continuación se describirá su utilización y finalidad:

Contra barra "\"

La contra barra se utiliza para "marcar.[el] carácter que le sigue de forma que este adquiera un significado especial o deje de tenerlo. O sea, se utiliza siempre en combinación con otros caracteres.

Cuando se coloca la barra inversa seguida de cualquiera de los meta caracteres, estos dejan de tener su significado especial y se convierten en caracteres de búsqueda literal, por ejemplo si escribimos \?, estamos representando el carácter ? tal cual,

[4] operación unaria que se aplica sobre caracteres o sobre un conjunto de cadenas de caracteres y representa el conjunto de las cadenas que se pueden formar tomando cualquier número de cadenas del conjunto inicial, posiblemente con repeticiones, y concatenándolas entre sí.

sin significado adicional.

Así como también puede darle significado especial a caracteres que no lo tienen, por ejemplo en los patrones utilizados por L7_filter se utiliza la contra barra seguida de la letra x (\x).
Esto se utiliza para representar caracteres ASCII o ANSI si conoce su código.

Delimitadores "$ z "^"

El metacaracter “$” representa el final de la cadena de caracteres o también el final de la línea. Por ejemplo la expresión regular "\x0d$.^encontrara todo lo que termine con el número hexadecimal 0d.

La funcionalidad del metacaracter "^"depende de si se lo utiliza individualmente o en conjunto con otros caracteres especiales. Si se lo emplea como carácter individual representa el inicio de la cadena, o sea que la expresión regular "^ ssh"todo lo que comience con la palabra ssh.
Si se lo utiliza en conjunto con los meta caracteres, por ejemplo la siguiente expresión "[^\w]"permite encontrar cualquier carácter que no se encuentre dentro del grupo indicado, es decir cualquier carácter que no sea alfanumérico o un espacio.

Punto "."

El metacaracter punto representa cualquier carácter excepto aquellos que representan un salto de línea. Es muy útil para encontrar caracteres que no conocemos, pero se debe tener en cuenta que se podrían obtener resultados no deseados. En lugar de buscar cualquier carácter, se puede restringir buscando cualquier carácter alfanumérico o cualquier dígito o cualquier no-dígito o cualquier no-alfanumérico.

Corchetes Rectos "[]"

La función de este metacaracter es representar çlases de caracteres", es decir, se agrupan caracteres en grupos o clases. Esto es útil cuando se quiere buscar dentro de un grupo de caracteres. También se puede especificar un rango de búsqueda utilizando el guión ".

Adicionalmente, los meta caracteres pierden su significado y se convierten en literales cuando se encuentran dentro de los corchetes. Por ejemplo, "\d", es útil para buscar cualquier carácter que represente un dígito. Sin embargo esta denominación no incluye el punto "."que divide la parte decimal de un número. Para buscar cualquier carácter que representa un dígito o un punto podemos utilizar la expresión regular "[\d.]", o sea que dentro de los corchetes el punto representa un caractér literal y no un meta caractér, por lo que no es necesario antecederlo con

la barra inversa. El único carácter que es necesario anteceder con la barra inversa dentro de los corchetes es la propia barra inversa.

Llaves "{}"

Las llaves sirven a encontrar elementos que se repiten, pero hay que saber de antemano cuantas veces es que se repite. Si no se conoce con claridad cuantas veces se repite lo que se busca o su grado de repetición es variable, nos conviene utilizar otro metacaractér.
Comúnmente las llaves son caracteres literales cuando se utilizan por separado en una expresión regular. Para que adquieran su función de meta caracteres es necesario que encierren uno o varios números separados por coma y que estén colocados a la derecha de otra expresión regular, por ejemplo la siguiente expresión "\d2"busca dos dígitos contiguos.

Barra "|"

Este metacaracter sirve para indicar una de varias opciones. Por ejemplo, la expresión regular “^(+ok | -err)” busca que comience con +ok o con -err.

Paréntesis "()"

Los paréntesis sirven para agrupar caracteres de forma similar a como lo hacen los corchetes, sin embargo existen varias diferencias fundamentales entre los grupos establecidos por medio de corchetes y los grupos establecidos por paréntesis:

- Los caracteres especiales conservan su significado dentro de los paréntesis.
- Los grupos establecidos con paréntesis establecen una .^{et}iqueta.^{o} "punto de referencia".
- Utilizados en conjunto con la barra "|"permite hacer búsquedas opcionales.
- Utilizado en conjunto con otros caracteres especiales ofrece funcionalidad adicional.

Asterisco "*"

El asterisco sirve para encontrar una cadena que se repetido 0 o más veces.

Por ejemplo, en esta expresión regular [0-9]*, con esta expresión se puede no encontrar nada así como también se puede llegar a encontrar el número 99999.
Se debe tener cuidado con el comportamiento del asterisco, ya que este por defecto trata de encontrar la mayor cantidad posible de caracteres que correspondan con el patrón que se busca.

Signo de suma "+"

Se utiliza para encontrar una cadena que se encuentre repetida 1 o más veces.
Si utilizamos este meta carácter en conjunto con el signo de pregunta, podemos limitar hasta donde se efectúa la repetición.

Signo de interrogación ¿ "

Se utiliza para encontrar una cadena que se encuentre repetida cero o una 1.

ANEXO D

11. Anexo D: Protocolos

Protocolo SSH

SSH (Secure SHell) es el nombre del protocolo y del programa que lo implementa. Sirve para acceder a máquinas remotas a través de una red, permitiendo manejar por completo la computadora mediante un interprete de comandos. También permite copiar datos de forma segura (tanto ficheros sueltos como simular sesiones FTP cifradas), gestionar claves RSA de manera de no tener que escribirlas al momento de conectar a los dispositivos y pasar los datos de cualquier otra aplicación por un canal seguro tunelizado mediante SSH.

Para la seguridad, SSH usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación, vaya de manera no legible y ninguna tercera persona pueda descubrir el usuario y contraseña de la conexión ni lo que se escribe durante toda la sesión.

Protocolo SIP y RTP

SIP (Session Initiation Protocol - Internet telephony) es un protocolo que fue desarrollado con la intención de ser el estándar para el establecimiento, modificación y finalización de sesiones interactivas de usuario donde intervienen elementos multimedia como: video, voz, mensajería instantánea, juegos en línea y realidad virtual.

La sintaxis de sus operaciones se asemeja a la de los protocolos utilizados en los servicios de páginas Web y de distribución de e-mails, HTTP y SMTP respectivamente. Esta similitud se debe a que SIP fue diseñado para que la telefonía se vuelva un servicio más en las redes de paquetes.

Una de las principales aplicaciones de este protocolo es la telefonía y por lo tanto uno de sus objetivos fue aportar un conjunto de las funciones de procesamiento de llamadas y capacidades presentes en la red pública conmutada de telefonía.

Se complementa con el protocolo RTP (Real-time Transport Protocol), que es el verdadero portador para el contenido de voz y video que intercambian los participantes en una sesión establecida por SIP.

Las funciones básicas que el protocolo incluye son:

- Determinar la ubicación de los usuarios, aportando movilidad.

- Establecer, modificar y terminar sesiones multipartitas entre usuarios.

Este protocolo adopta el modelo cliente-servidor y es transaccional: el cliente realiza peticiones que el servidor responde ya sea rechazando o aceptado esa petición en una serie de respuestas que llevan un código de estado que brindan información acerca de si las peticiones fueron resueltas con éxito o si se produjo un error. Una transacción es constituida por la petición inicial y todas sus respuestas, las cuales utilizan por defecto el puerto 5060.

A continuación se detallan los distintos agentes que interactúan en este protocolo, así como también los mensajes que son intercambiados.

Agentes de Usuario

Los Agentes de usuario, son los que emiten y consumen los mensajes del protocolo SIP, es decir los video teléfonos, teléfonos, clientes de software (softphone) o cualquier otro dispositivo similar.

Este protocolo no se ocupa de la interfaz entre estos dispositivos y el usuario final, sino que sólo le interesa los mensajes que estos generan y cómo es su comportamiento al recibir determinados mensajes.

Los agentes de usuario se comportan como:

- Clientes (UAC: User Agent Clients), cuando realizan una petición.
- Servidores (UAS: User Agent Servers), cuando reciben una petición.

También existen otras entidades que intervienen en el protocolo: Servidores de Registro y servidores Proxy y Redirectores.

Servidores de Registro

SIP permite establecer la ubicación física de un usuario determinado, o sea en qué punto de la red está conectado, valiéndose del siguiente mecanismo de registro:

- Cada usuario tiene una dirección lógica de la forma usuario@dominio, que es invariable respecto de la ubicación física del usuario.
- La dirección física (denominada "dirección de contacto") es dependiente del lugar en donde el usuario está conectado (de su dirección IP).
- Cuando un usuario inicializa su terminal, al conectar un teléfono o al ejecutar el software de telefonía SIP, el agente de usuario SIP que reside en dicho terminal envía una petición con el método REGISTER a un Servidor de Registro, informando a qué dirección física debe asociarse la dirección lógica del usuario.

- El servidor de registro realiza entonces dicha asociación (binding), que tiene un período de vigencia y si no es renovada, caduca. También puede terminarse mediante un desregistro.

Servidores Proxy y de Redirección

Se recurre a los servidores para encaminar un mensaje entre un agente de usuario cliente y un agente de usuario servidor. Estos pueden actuar de dos maneras:

1 Como Proxy, encaminando el mensaje hacia destino,

1 Como Redirector, generando una respuesta que indica al originante la dirección del destino o de otro servidor que lo acerque al destino.

Formato de los mensajes

Los mensajes que se intercambian en el protocolo SIP pueden ser:

1 Peticiones: tienen una línea de petición (se indica el propósito de la petición y el destinatario de la petición), una serie de encabezados y un cuerpo.

2 Respuestas: tienen una línea de respuesta, una serie de encabezados y un cuerpo.

Las peticiones tienen distintas funciones, donde el propósito de una petición está determinado por lo que se denomina el Método de dicha petición, que no es más que un identificador del propósito de la petición. En la línea de respuesta se indica el código de estado de la respuesta, que es un número que indica el resultado del procesamiento de la petición.

Los encabezados de peticiones y respuestas se utilizan para diversas funciones del protocolo relacionadas con el encaminamiento de los mensajes, autenticación de los usuarios, entre otras. La extensibilidad del protocolo permite crear nuevos encabezados para los mensajes agregando de esta manera funcionalidad.

El cuerpo de los mensajes es opcional y se utiliza entre otras cosas para transportar las descripciones de las sesiones que se quieren establecer, utilizando la sintaxis del protocolo SDP.

Flujo de establecimiento de una sesión

El flujo habitual del establecimiento de una sesión mediante el protocolo SIP, donde todos los servidores actúan como proxy, es el siguiente:

- Un usuario ingresa la dirección lógica de la persona con la que quiere comunicarse, puede indicar al terminal también las características de las sesión que quiere establecer (voz, voz y video, etc.), o estas pueden estar implícitas por el tipo de terminal del que se trate.

- El agente de usuario SIP que reside en el terminal, actuando como UAC envía la petición (en este caso con el método INVITE) al servidor que tiene configurado. Este servidor se vale del sistema DNS para determinar la dirección del servidor SIP del dominio del destinatario. El dominio lo conoce pues es parte de la dirección lógica del destinatario.
- Una vez obtenida la dirección del servidor del dominio destino, encamina hacia allí la petición. El servidor del dominio destino establece que la petición es para un usuario de su dominio y entonces se vale de la información de registración de dicho usuario para establecer su ubicación física. Si la encuentra, entonces encamina la petición hacia dicha dirección.
- El agente de usuario destino si se encuentra desocupado comenzará a alertar al usuario destino y envía una respuesta hacia el usuario originante con un código de estado que indica esta situación (180 en este caso). La respuesta sigue el camino inverso hacia el originante.
- Cuando el usuario destino finalmente acepta la invitación, se genera una respuesta con un código de estado (el 200) que indica que la petición fue aceptada. La recepción de la respuesta final es confirmada por el UAC originante mediante una petición con el método ACK (de Acknowledgement), esta petición no genera respuestas y completa la transacción de establecimiento de la sesión.

Normalmente la petición con el método INVITE lleva un cuerpo donde viaja una descripción de la sesión que quiere establecer, esta descripción es realizada con el protocolo SDP. En ella se indica el tipo de contenido a intercambiar (voz, video, etc.) y sus características (códecs, direcciones, puertos donde se espera recibirlos, velocidades de transmisión, etc.) que se conoce como .ºferta de sesión SDP".

La respuesta a esta oferta viaja, en este caso, en el cuerpo de la respuesta definitiva a la petición con el método INVITE. La misma contiene la descripción de la sesión desde el punto de vista del destinatario. Si las descripciones fueran incompatibles, la sesión debe terminarse (mediante una petición con el método BYE).

Al terminar la sesión, que lo puede hacer cualquiera de las partes, el agente de usuario de la parte que terminó la sesión, actuando como UAC, envía hacia la otra una petición con el método BYE. Cuando lo recibe el UAS genera la respuesta con el código de estado correspondiente.

Si bien se ha descrito el caso de una sesión bipartita, el protocolo permite el establecimiento de sesiones multipartitas. También permite que un usuario esté registrado en diferentes ubicaciones pudiendo realizar la búsqueda en paralelo o secuencial entre todas ellas.

Protocolo HTTP

HTTP (HyperText Transfer Protocol) es el protocolo de transferencia de hipertexto usado en cada transacción Web, definiendo para comunicarse, la sintaxis y la semántica que utilizan los elementos software de la arquitectura web.

Sigue el esquema petición-respuesta entre cliente-servidor y es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. Como las aplicaciones web necesitan mantener estado, se utilizan los cookies, que es la información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de "sesión", y también permite rastrear usuarios ya que los cookies pueden guardarse en el cliente por tiempo indeterminado.

Transacciones HTTP

Una transacción HTTP está formada por un encabezado seguido, opcionalmente, por una línea en blanco y algún dato. En el encabezado se especificará por ejemplo la acción requerida del servidor, el tipo de dato retornado, o el código de estado.

El uso de campos de encabezados enviados en las transacciones HTTP le dan gran flexibilidad al protocolo ya que permiten que se envíe información descriptiva en la transacción, permitiendo así la autenticación, cifrado e identificación de usuario.

El servidor puede excluir cualquier encabezado que ya esté procesado, como Authorization, Content-type y Content-length. Además puede elegir excluir alguno o todos los encabezados en el caso de que al incluirlos se excede algún límite del ambiente de sistema. Podemos ejemplificar lo anteriormente mencionado las variables HTTP_ACCEPT y HTTP_USER_AGENT.

- HTTP_ACCEPT. Los tipos MIME que el cliente aceptará, dado los encabezados HTTP. Otros protocolos quizás necesiten obtener esta información de otro lugar. Los elementos de esta lista deben estar separados por una coma, como lo dice la especificación HTTP: tipo, tipo.
- HTTP_USER_AGENT. El navegador que utiliza el cliente para realizar la petición. El formato general para esta variable es: software/versión biblioteca/versión.

El servidor envía al cliente:

- Un código de estado que indica si la petición fue correcta o no. Los códigos de error típicos indican que el archivo solicitado no se encontró, que la petición no se realizó de forma correcta o que se requiere autenticación para acceder al archivo.
- La información propiamente dicha. Como HTTP permite enviar documentos de todo tipo y formato, es ideal para transmitir multimedia, como gráficos, audio y video. Esta libertad es una de las mayores ventajas de HTTP.
- Información sobre el objeto que se retorna.

Protocolo Telnet

El protocolo Telnet (TELecommunication NETwork) sirve para acceder mediante una red a otra máquina, y así manejarla remotamente. La máquina a la que se accede debe tener un programa especial que reciba y gestione las conexiones. El acceso es sólo en modo terminal, es decir sin gráficos y el puerto que se utiliza generalmente es el 23.

Presenta un problema de seguridad, ya que todos los nombres de usuario y contraseñas necesarias para entrar a las máquinas viajan por la red como texto plano (cadenas de texto sin cifrar). Esto facilita que cualquiera que espíe el tráfico de la red pueda obtener los nombres de usuario y contraseñas, y así acceder también a todas esas máquinas. Por esta razón fue que se dejó de usar ya casi totalmente, cuando apareció y se popularizó el SSH, que puede describirse como una versión cifrada de telnet.

Protocolo VNC

VNC (Virtual Network Computing) es un programa de software libre basado en una estructura cliente-servidor la cual permite tomar el control del ordenador servidor remotamente a través de un ordenador cliente.

Permite que el sistema operativo en cada computadora sea distinto: Es posible compartir la pantalla de una máquina de çualquier"sistema operativo conectando desde cualquier otro ordenador o dispositivo que disponga de un cliente VNC portado.

Protocolo POP3

El protocolo POP3 (Post Office Protocol) se utiliza en clientes locales de correo para obtener los mensajes de correo electrónico almacenados en un servidor remoto ya fue diseñado para recibir correo, no para enviarlo.

POP3 utilizaba un mecanismo de firmado sin cifrado, aunque en la actualidad POP3 cuenta con diversos métodos de autenticación que ofrecen una diversa gama de niveles de protección contra los accesos ilegales al buzón de correo de los usuarios.

Para establecer una conexión a un servidor POP, el cliente de correo abre una conexión TCP en el puerto 110 del servidor. Cuando la conexión se ha establecido, el servidor POP envía al cliente POP una invitación y después las dos máquinas se envían entre sí otras órdenes y respuestas que se especifican en el protocolo.

Como parte de esta comunicación, al cliente POP se le pide que se autentifique, donde el nombre de usuario y la contraseña del usuario se envían al servidor POP. Si la autenticación es correcta, el cliente POP pasa al estado de transacción. En este estado se pueden utilizar órdenes:

- LIST: Muestra todo los mensajes no borrados con su longitud.
- RETR: Solicita el envío del mensaje especificando el número y no se borra del buzón.
- DELE: Borra el mensaje especificando el número para mostrar, descargar y eliminar mensajes del servidor, respectivamente.

Los mensajes definidos que van a ser eliminados no se quitan realmente del servidor hasta que el cliente POP envía la orden QUIT para terminar la sesión. En ese momento, el servidor POP pasa al estado de actualización, fase en la que se eliminan los mensajes marcados y se limpian todos los recursos restantes de la sesión.
De esta manera, es posible conectarse manualmente al servidor POP3 haciendo Telnet al puerto 110, siendo esto muy útil cuando se envía un mensaje con un fichero muy largo que no se quiere recibir.

Una de las ventajas de POP3, es que no se tienen que enviar tantas ordenes entre servidor-cliente para la comunicación entre ellos. Además funciona adecuadamente aunque no se tenga una conexión constante con Internet o con la red que contiene el servidor de correo.

Protocolo SMTP

El protocolo SMTP (Simple Mail Transfer Protocol) permite la transferencia de correo de un servidor a otro mediante una conexión punto a punto, funciona en línea, encapsulado en una trama TCP/IP. El correo se envía directamente al servidor de correo del destinatario.

Se basa en el modelo cliente-servidor, donde un cliente envía un mensaje a uno o varios receptores. La comunicación entre el cliente y el servidor consiste enteramente en líneas de texto compuestas por caracteres ASCII, donde el tamaño

máximo permitido para estas líneas es de 1000 caracteres.

El protocolo SMTP funciona con comandos de textos enviados al servidor SMTP utilizando de manera predeterminada el puerto 25. Luego de enviado cada comando, se envía una respuesta del servidor SMTP compuesta por un número y un mensaje descriptivo.

Las respuestas del servidor constan de un código numérico de tres digitos, seguido de un texto explicativo. El número va dirigido a un procesado automático de la respuesta por autómata, mientras que el texto permite que un humano interprete la respuesta. En el protocolo SMTP todas las órdenes, réplicas o datos son líneas de texto, delimitadas por el carácter <CRLF>. Todas las réplicas tienen un código numérico al comienzo de la línea.

Conexión al inicio del protocolo

Cuando se emplea el protocolo TCP el servidor SMTP escucha permanentemente al puerto 25, en espera de algún cliente que desea enviarlo. El protocolo de aplicación SMTP inicia el comando HELO (para abrir una sesión con el servidor), seguido de la identificación del cliente, el servidor lo acepta con un código «250 OK».

Envío de mensajes

Una vez iniciado el protocolo, se realiza el envío de mensajes desde el cliente al servidor, mediante el siguiente proceso.

Envío del sobre

En primer lugar se transmite la dirección del buzón del origen del mensaje, mediante el comando MAIL FROM (indica quien envía el mensaje) y si el servidor acepta, envía el mensaje 250 OK. Luego se transmite la dirección de destino, mediante el comando RCPT TO (indica el destinatario del mensaje) y el servidor confirma con 250 OK, o si el destinatario no existe envía 550 Failure.

Envío del contenido del mensaje

El cliente informa al servidor de que va a enviar el mensaje mediante el comando DATA (para indicar el comienzo del mensaje, éste finalizará cuando haya una línea únicamente con un punto), si el servidor está dispuesto envía 354, todas las líneas que el cliente envía a partir de este momento se consideran parte del contenido del mensaje, al final del mensaje se considera enviando el «.», cuando el servidor recibe el fin del mensaje confirma con 250 OK.

Cierre de la conexión

Una vez enviado todos los mensajes, el cliente puede cerrar la conexión mediante el comando QUIT (para cerrar la sesión), caso contrario la máquina que recibió los mensajes sea quien las envíe con el comando TURN (solicita al servidor que intercambien los paquetes), el servidor confirma con 250 OK, dando una sección que se inicia con el comando HELO.

Protocolo FTP

FTP (File Transfer Protocol) es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP basada en la arquitectura cliente-servidor. Este protocolo permite, independientemente del sistema operativo que se use, conectarse a un servidor desde un equipo cliente para descargar archivos desde él o para enviarle archivos utilizando normalmente el puerto de red 20 y el 21 respectivamente.

Ofrece máxima velocidad en la conexión pero no la máxima seguridad, ya que todo el intercambio de información, desde el login y password del usuario en el servidor hasta la transferencia de cualquier archivo, se realiza en texto plano sin ningún tipo de cifrado. Por este motivo es posible que un atacante pueda capturar este tráfico, acceder al servidor y apropiarse de los archivos transferidos.

Modelo FTP:

En el modelo, el intérprete de protocolo de usuario, inicia la conexión de control en el puerto 21. Además genera órdenes FTP estándar que se transmiten al proceso del servidor a través de la conexión de control.

Como respuestas a estas ordenes, se envían respuestas estándar desde el interprete de protocolo del servidor al del usuario por la conexión de control. Estas órdenes especifican parámetros para la conexión de datos (puerto de datos, modo de transferencia, tipo de representación y estructura) y la naturaleza de la operación sobre el sistema de archivos (almacenar, recuperar, añadir, borrar, etc.).

El proceso de transferencia de datos de usuario u otro proceso en su lugar, debe esperar a que el servidor inicie la conexión al puerto de datos especificado (puerto 20 en modo activo o estándar) y transferir los datos en función de los parámetros que se hayan especificado.

La conexión de datos es bidireccional, es decir, se puede usar simultáneamente para enviar y para recibir, y no tiene por qué existir todo el tiempo que dura la conexión.

En el próximo diagrama, se puede observar que la comunicación entre cliente y servidor es independiente del sistemas de archivos utilizado en cada ordenador. O sea, no importa que sus sistemas operativos sean distintos, porque las entidades que se comunican entre sí, son los interpretes de protocolos y los procesos de transferencia de datos, los cuales usan el mismo protocolo estandarizado.

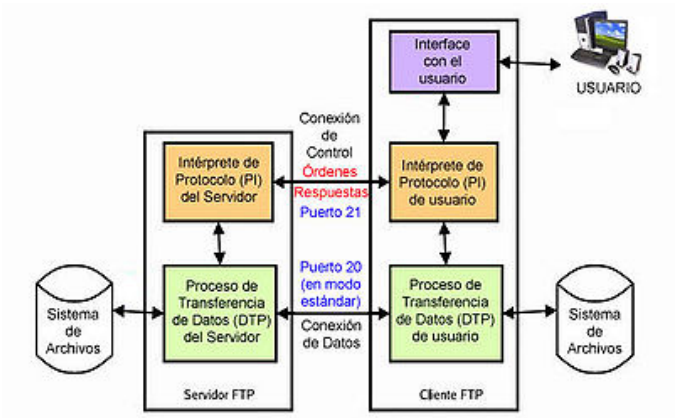


Figura 2: Diagrama de comunicación FTP

Servidor FTP:

Un servidor FTP es un programa especial que se ejecuta en un equipo servidor normalmente conectado a Internet y cuya función es permitir el intercambio de datos entre diferentes servidores.

Las aplicaciones más comunes de los servidores FTP suelen ser el alojamiento web, en el que sus clientes utilizan el servicio para subir sus páginas web y sus archivos correspondientes; o como servidor de backup (copia de seguridad) de los archivos importantes que pueda tener una empresa.

Cliente FTP:

Un cliente FTP es un programa que se instala en la PC del usuario, la cual emplea este protocolo para conectarse a un servidor FTP y transferir archivos, ya sea para descargarlos o para subirlos.

Para utilizar un cliente FTP, se necesita conocer el nombre del archivo, el ordenador en que reside (servidor, en el caso de descarga de archivos), el ordenador al que se quiere transferir el archivo (en caso de querer subirlo nosotros al servidor), y la carpeta en la que se encuentra.

Modos de conexión del cliente FTP :

FTP admite dos modos de conexión del cliente:

- Modo Activo: el cliente envía comandos tipo PORT al servidor por el canal de control al establecer la conexión.
- Modo Pasivo: el cliente envía comandos tipo PASV.

En ambos modos, el cliente establece una conexión con el servidor mediante el puerto 21, donde se establece el canal de control.

Antes de cada nueva transferencia, tanto en el modo Activo como en el Pasivo, el cliente debe enviar otra vez un comando de control (PORT o PASV, según el modo en el que haya conectado), y el servidor recibirá esa conexión de datos en un nuevo puerto aleatorio (si está en modo pasivo) o por el puerto 20 (si está en modo activo).

Modo Activo

En este modo el servidor siempre crea el canal de datos en su puerto 20, mientras que en el lado del cliente el canal de datos se asocia a un puerto aleatorio mayor que el 1024.

Para esto, el cliente manda un comando PORT al servidor por el canal de control indicándole ese número de puerto, de manera que el servidor pueda abrirle una conexión de datos por donde se transferirán los archivos y los listados, en el puerto especificado.

La máquina cliente debe estar dispuesta a aceptar cualquier conexión de entrada en un puerto superior al 1024, lo cual implica un gran problema de seguridad.

Modo Pasivo

Cuando el cliente envía un comando PASV sobre el canal de control, el servidor FTP le indica por el canal de control, el puerto (mayor a 1023 del servidor) al que debe conectarse el cliente.

El cliente inicia una conexión desde el puerto siguiente al puerto de control hacia el puerto del servidor especificado anteriormente.

Protocolo Ares

Ares es un programa P2P de distribución de archivos. En sus orígenes trabajaba con la red Gnutella pero desarrolló su propia red independiente y descentralizada, montada sobre una arquitectura de red P2P que ofrece un sistema de búsqueda de tipo broadcasting.

El funcionamiento de la red es como un P2P (una red que no tiene clientes ni servidores fijos, sino un que es un conjunto de nodos que se comportan simultáneamente como clientes y servidores de los demás nodos). Una de las razones por las cuales suele ir más rápido que otros programas de redes P2P, es por que el método usado consiste en dar mayor prioridad a aquellos nodos cuyo porcentaje de descarga completa sea menor.

Características del programa

- Previsualización de archivos multimedia.

  Breves colas de espera: su flexible algoritmo tiene como prioridad poner en primer lugar en una cola remota de espera, a usuarios que tienen menos porcentaje en una descarga. De esta manera se logra que las colas remotas sean muy breves para aquellos usuarios que inician una descarga.

  Salas de conversación en línea: una de las características típicas de este programa es la capacidad para crear salas de conversación descentralizadas para servir como punto de encuentro entre varias personas que comparten los mismos gustos o simplemente para conocer a más personas que comparten la misma nacionalidad.

- Compatibilidad con el protocolo BitTorrent: Soporta el protocolo BitTorrent, permitiendo gestionar y descargar por defecto el contenido de los archivos .torrent haciendo que Ares interactue como un cliente BitTorrent y obtenga contenido mediante este protocolo. Ares retira el torrent de la red al completarse, lo cual va en contra de la filosofía bittorrent.

- Compatibilidad con radio en Internet SHOUTcast: El soporte experimental de radio Internet Soutcast permite sintonizar directamente desde el reproductor multimedia de Ares cualquier estación de radio emitiendo desde SHOUTcast.

- BiBiblioteca de gestión: Contiene una sección donde se comparten y organizan los archivos descargados. Incluye dos vistas, normal y extendida. En la extendida los archivos compartidos se dividen en categorías; en la normal se muestra los directorios compartidos como los organizó el usuario.

- Descarga de un mismo archivo desde múltiples fuentes: se evita colas remotas de espera, es decir se descargan "pedazos" del mismo archivo de diferentes usuarios con el mismo archivo al mismo tiempo, esto quiere decir que si al usuario a quien está descargándosele un archivo está ocupado, hay más de uno que lo respaldará permitiendo al usuario reanudar la descarga y descargar con mayor velocidad.

- Mensajería instantánea entre dos usuarios: Dos usuarios de Ares pueden comunicarse por medio de un mensajero instantáneo que no requiere de la conexión a ninguna sala pública de chat, sino que al descargar contenido de una persona o viceversa puede enviársele un mensaje instantáneo al otro usuario si se tiene activada la opción de recibir mensajes instantáneos.

- Compartir archivos detrás de un cortafuegos: En las versiones superiores a la 1.9.0, es posible compartir datos entre dos usuarios de Ares que estén usando

un cortafuegos.

Protocolo Bittorrent

El objetivo de esta aplicación es proporcionar una forma eficiente de distribuir un mismo fichero a un gran grupo de personas, forzando a todos los que descargan un fichero a compartirlo también con otros. Primero se distribuye por medios convencionales un pequeño fichero con extensión .torrent; este fichero es estático, por lo que a menudo se encuentra en páginas web o incluso se distribuye por correo electrónico. El fichero 'torrent' contiene la dirección de un "servidor de búsqueda", el cual se encarga de localizar posibles fuentes con el fichero o parte de él.

Dicho servidor se encuentra centralizado y provee estadísticas acerca del número de transferencias, el número de nodos con una copia completa del fichero y el número de nodos que poseen sólo una porción del mismo.

El fichero o colección de ficheros deseado es descargado de las fuentes encontradas por el servidor de búsqueda y, al mismo tiempo que se realiza la descarga, se comienza a subir las partes disponibles del fichero a otras fuentes, utilizando el ancho de banda asignado a ello. Ya que la acción de compartir comienza incluso antes de completar la descarga de un fichero, cada nodo inevitablemente contribuye a la distribución de dicho fichero. El sistema se encarga de premiar a quienes compartan más, a mayor ancho de banda mayor el número de conexiones a nodos de descarga que se establecerán.

Cuando un usuario comienza la descarga de un fichero, BitTorrent no necesariamente comienza por el principio del fichero, sino que se baja por partes al azar. Luego los usuarios se conectan entre sí para bajar el fichero y si entre los usuarios conectados se dispone de cada parte del fichero completo (aún estando desparramado), finalmente todos obtendrán una copia completa de él. Por supuesto, inicialmente alguien debe poseer el fichero completo para comenzar el proceso. Este método produce importantes mejoras en la velocidad de transferencia cuando muchos usuarios se conectan para bajar un mismo fichero.

Cuando no existan ya más nodos con el fichero completo conectados al servidor de búsqueda, existe la posibilidad de que el fichero no pueda ser completado.

Protocolo Gnutella

Gnutella es un proyecto de software distribuido para crear un protocolo de red de distribución de archivos entre pares, sin un servidor central. A diferencia de otras redes de intercambio de fichero, como es el caso de eDonkey, Gnutella es una red

P2P pura, donde todos los nodos tienen la misma función, peso e importancia dentro de la red. El funcionamiento de la red pasa por tres fases:

1 Entrada. En esta fase un nuevo nodo se conecta a otro que ya esté dentro de la red. Cómo se encuentra un nodo ya conectado está fuera del protocolo, pero normalmente los clientes Gnutella se distribuyen con una lista de nodos que se espera estén siempre conectados y se escoge alguno al azar. Un nodo cualquiera puede estar conectado a varios nodos, y recibir conexiones de nuevos nodos formando una malla aleatoria no estructurada.

2 Búsquedas. Cuando un nodo desea buscar un fichero, le envía un mensaje a todos los nodos a los que está conectado. Estos buscan localmente si lo ofrecen, y a la vez reenvían la búsqueda a todos los nodos a los que ellos están conectados. Esta estrategia de difusión se llama inundación de la red, y existen mecanismos para evitar reenvíos infinitos y bucles. Cuando una petición llega a un nodo que ofrece el fichero, se contesta directamente al nodo que inició la búsqueda.

3 Descargas. La descarga se realiza directamente desde los nodos que contestaron a la búsqueda del fichero. Los ficheros pueden partirse en varios trozos servidos por diferentes nodos, y los clientes suelen incluir un sistema de comprobación final de la integridad del fichero.

La inundación producida por la fase de búsqueda es la debilidad más importante de este protocolo. Si hay muchas búsquedas a la vez, la red se llena de mensajes de búsqueda que los nodos se envían entre ellos. Además, este algoritmo de búsqueda no garantiza que el fichero sea finalmente encontrado incluso aunque algún nodo de la red lo tenga. Aún así, el hecho de que no exista un servidor central de búsqueda, como en el caso de eDonkey, hacen que este protocolo sea más robusto en caso de caídas de nodos.

Protocolo Msn

MSN messanger es un programa de mensajería instantánea creada en 1999 pero actualmente descontinuado. Inicialmente fue diseñado para sistemas Windows por Microsoft y luego se lanzo una versión disponible para Mac OS. Se cambiaron de nombre muchos servicios y programas existentes de MSN, con lo que Messenger fue renombrado a "Windows Live Messenger.ª partir de la versión 8.0, como parte de la creación de servicios web denominados Windows Live por Microsoft.

Bajo la denominación se engloban realmente tres programas diferentes:

- MSN Messenger: Es un cliente de mensajería instantánea y su nombre se utiliza para referenciar todos los programas de mensajería de Microsoft.

- Windows Messenger: Viene incluido con Windows XP y se trata de un cliente de mensajería instantánea básico que no soporta muchas características (avatares, imágenes, etcétera). Sin embargo, es capaz de conectarse al Servicio de comunicaciones y de Exchange Instant Messaging usados por algunas empresas y permite controlar una máquina de forma remota de forma similar al NetMeeting.
- MSN Web Messenger: Proporciona características similares al MSN Messenger en un navegador conectado a Internet. Su utilidad reside en que se puede conectar con una cuenta de correo desde un ordenador que no tenga el programa instalado.

Los usuarios de GNU/Linux han sido dejados aún más al margen, necesitando software de terceros para iniciar una sesión y acceder a su perfil almacenado en los servidores de MSN Messenger. Aquel software de terceros es usualmente una de los muchos clientes alternativos de mensajería instantánea como aMSN, Pidgin o Kopete.

Protocolo Edonkey

La red eDonkey es un sistema descentralizado, en su mayoría basada en servidores peer-to-peer donde el intercambio de archivos de red es más adecuado para compartir entre los usuarios archivos grandes, proporcionando para esos archivos una disponibilidad a largo plazo. Como la mayoría de las redes de intercambio de archivos es descentralizada, y ya que no hay ningún eje central de la red, los archivos no se almacenan en un servidor central sino que se intercambian directamente entre usuarios.

La parte del servidor de la red es propiedad freeware y hay dos familias de software de servidor para la red edonkey: la original que ya no se mantiene de MetaMachine, escrito en C++, de código cerrado y propietario; y por otro lado eServer. Este último, fue escrito desde cero puramente en lenguaje C, también de código cerrado y propietario,que está disponible de forma gratuita y para varios sistemas operativos y arquitecturas de computadora.

Protocolo Shoutcast

Shoutcast es una tecnología de streaming auditiva freeware que utiliza la codificación MP3 o AAC de contenido auditivo y utiliza http como protocolo para transmitir radio por Internet.

Radio por internet

A diferencia de muchos sitios que solo ofrecen radio por Internet, SHOUTcast fomenta la creación, por parte de sus usuarios,de nuevos servidores de radio por Internet gracias al software para servidores provisto por ellos. El formato de salida es leído por múltiples programas cliente, incluyendo los productos Nullsoft Winamp, Apple iTunes y Windows Media Player. Con este software, cualquier usuario puede crear y adaptar un servidor para sus propias necesidades.

Esta tecnología requiere que sea el propio usuario el que proporcione el ancho de banda necesario para alimentar las peticiones de los usuarios, lo que implica que si se quiere enviar un stream de alta calidad, se tenga que considerar una conexión ADSL o superior.

Cuando un usuario baja, instala y opera los códecs necesarios para iniciar un streaming, también es añadido al catálogo de SHOUTcast, que contiene cerca de 9.000 servidores de radio por internet, clasificados por género, por ancho de banda de sus transmisiones y por el número de usuarios que la escuchan y que pueden servir al mismo tiempo.

El funcionamiento de este sistema es el siguiente: cuando un usuario abre una página cuyo documento HTML, PHP ó JAVA contiene el servicio de conexión IP, por puerto que esté conectado al servicio de SHOUTcast, hace una petición al servicio DNS que lo enlaza a los servidores que se encuentran regularmente en tres ciudades de Texas. Una vez que la petición esta realizada y el código es autentificado se genera un enlace de respuesta al solicitante y envía paquetes no cifrados.

El sistema puede ser contabilizado por número de nodos y no por número de oyentes. Este nodo puede contener tantos clientes (usuarios en red) como se deseen, pero la contabilización se limita a un nodo sin importar los oyentes reales en línea, los oyentes por enlaces inalámbricos, por cuestiones de banda limitada, no pueden ser detectados, o sea que si hay un nodo el cual solo contiene clientes en enlace inalámbrico WIFI no serán detectados por esta tecnología.

Protocolo Irc

Irc es un protocolo de comunicación en tiempo real basado en texto, que permite debates entre dos o más personas. Se diferencia de los protocolos de mensajería instantánea en que los usuarios no deben acceder a establecer la comunicación de antemano, de tal forma que todos los usuarios que se encuentran en un canal pueden comunicarse entre sí, aunque no hayan tenido ningún contacto anterior.

Los usuarios del IRC utilizan una aplicación cliente para conectarse con un servidor, en el que funciona una aplicación IRCd (servidor de IRC) que gestiona los canales y las conversaciones.

Protocolo Skype

Skype es un programa de software que permite a los usuarios hacer llamadas telefónicas a través de internet a otros usuarios skype de forma gratuita, o a cambio de un precio, podrá hacerlas a teléfonos celulares y fijos. A esto se le suma que cuenta con otras características adicionales entre las cuales se incluye: mensajería instantánea, transferencia de archivos, servicio de mensajes cortos, videoconferencias y sobre todas las cosas la capacidad de burlar los firewalls.

Este protocolo se basa en VoIP, no está a disposición del público y las aplicaciones que utilizan este protocolo son de fuente cerrada. La principal diferencia entre los clientes de VoIP Skype, es que skype funciona basándose en un modelo peer-to-peer, en lugar de basarse en un modelo cliente-servidor. El directorio de usuario de skype es totalmente descentralizado y distribuido entre los nodos de la red, esto implica que la red puede ser fácilmente escalable sin una compleja y costosa infraestructura centralizada.

Una de las características más importantes que posee este protocolo, es garantizar la comunicación, utilizando claves RSA para la negociación y AES (Advanced Encryption Standard) para cifrar las conversaciones. Sin embargo, el registro de usuarios no requiere prueba de identidad, es decir que se puede utilizar el sistema sin tener que relevar la identidad de la vida real para los demás usuarios. Esta falta de autenticación significa que no hay una garantía de los que están en comunicación con quienes dicen que son en la vida real. La gran desventaja es que es fácil utilizar el nombre personal de alguien de confianza, como por ejemplo su apodo, y hacer que alguien revele información.

ANEXO E

12. Anexo E: Instalación y Configuración

Instalación Kernel, Iptables y L7_Filter

Para la instalación del Kernel, Iptables, Netfilter y L7-Filter, nos basamos en la URL:
http://www.howtoforge.com/how-to-set-up-a-linux-layer-7-packet-classifier-on-centos5.1.

Cave destacar que los pasos y errores encontrados durante la instalación dependen del sistema operativo y de las diferentes versiones que se instalen de kernel, Iptables, Netfilter, l7-Protocols.
En este caso detallamos la instalación en un notebook Dell, Pentium Dual Core con sistema operativo Ubuntu 8.10.

Previamente se debe hacer un update y tener instalados patch y ncurses-dev.
En caso de no ser así ejecutar:

```
apt-get update
apt-get install patch
apt-get install ncurses-dev
```

Pasos de la instalación:

1. Descargar los paquetes requeridos por línea de comando dentro del directorio /usr/src/:

1.1. L7-Filter kernel
wget http://downloads.sourceforge.net/l7-filter/netfilter-layer7-v2.19.tar.gz

1.2. L7-Filter userspace
wget http://downloads.sourceforge.net/l7-filter/l7-filter-userspace-0.7.tar.gz

1.3. L7-Filter Protocol definitions
wget http://downloads.sourceforge.net/l7-filter/l7-protocols-2008-04-23.tar.gz

Es importante que la version de L7-Filter que bajemos siempre sea la ultima disponible.

1.4. Linux Iptables 1.4.0 s
wget http://www.netfilter.org/projects/Iptables/files/Iptables-1.4.0.tar.bz2

1.5. Linux Kernel 2.6.26
wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.26.tar.bz2

2. Instalacion de L7-Filter

2.1. Dentro del directorio /usr/src/ descomprimimos los paquetes:

```
cd /usr/src
tar -xvf linux-2.6.26.tar.bz2
tar -xvf netfilter-layer7-v2.19.tar.gz
```

2.2. Aplicamos el correspondiente parche al Kernel de Linux
cd linux-2.6.26
patch -p1 <../netfilter-layer7-v2.19/kernel-2.6.27-layer7-2.19.patch

2.3. Instalamos y aplicamos el parche para Iptables 1.4.0.
Ejecutamos los siguientes comandos:
cd /usr/src
tar -xvf Iptables-1.4.0.tar.bz2
cd Iptables-1.4.0
patch -p1
<../netfilter-layer7-v2.19/Iptables-1.4-for-kernel-2.6.20forward-layer7-2.19.patch
chmod +x extensions/.layer7-test
make KERNEL_DIR=/usr/src/linux-2.6.26
make install KERNEL_DIR=/usr/src/linux-2.6.26

3. Instalación de protocolos. Se ejecutaron los comandos:
cd /usr/src
tar -xvf l7-protocols-2008-04-23.tar.gz
cd l7-protocols-2008-04-23
mkdir /etc/l7-protocols
cp protocols/* /etc/l7-protocols

4. Instalación y compilación del nuevo kernel de linux
cd /usr/src
cd linux-2.6.26
make menuconfig
make all
make modules_install
make install

Cuando ejecutamos make menuconfig seleccionamos lo siguiente:

"Network packet filtering framework(Netfilter)"(Networking→Networking option)
"Netfilter connection tracking support"
→ Network packet filtering framework(Netfilter)
→Core Netfilter Configuration
“Connection tracking flow accounting” (on the same screen)
Finally, "Layer 7 match support"

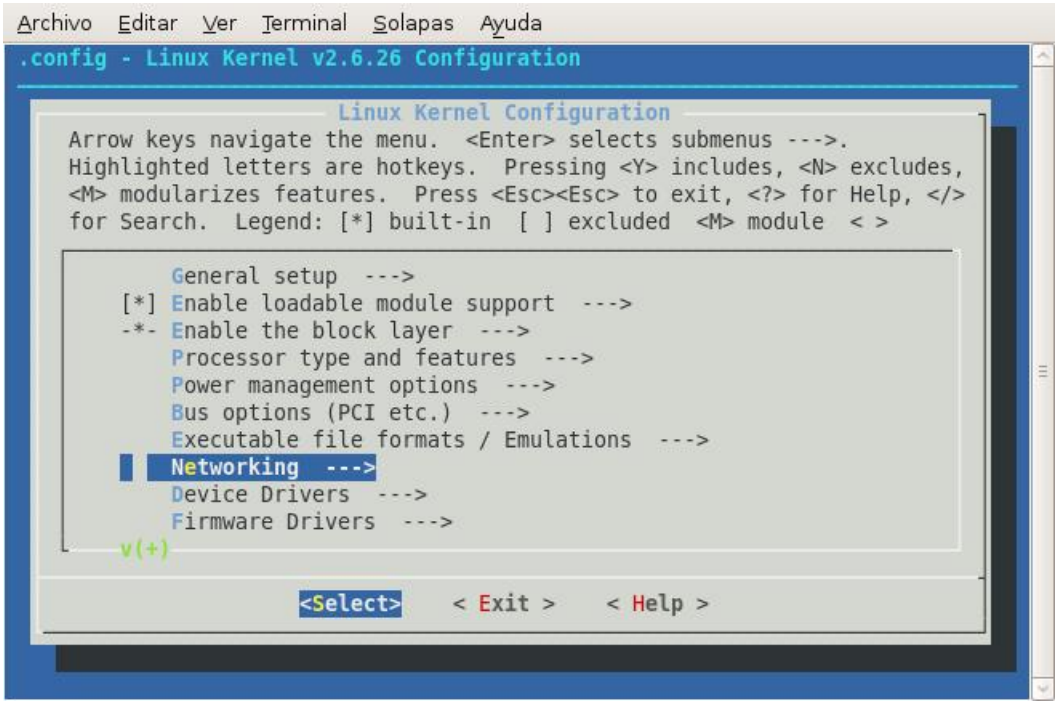


Figura 3: paso 1

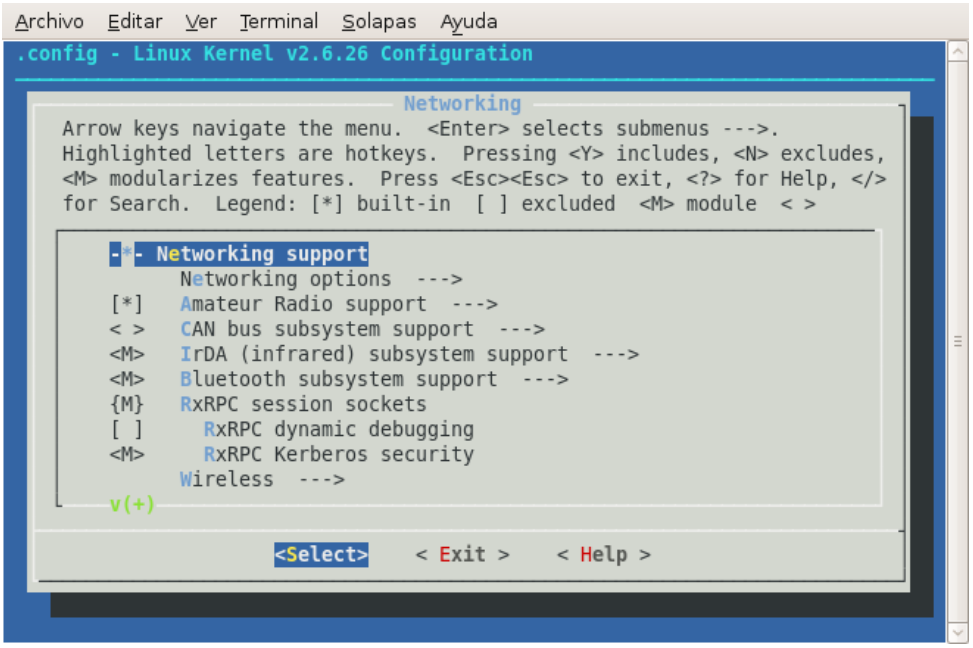


Figura 4: paso 2

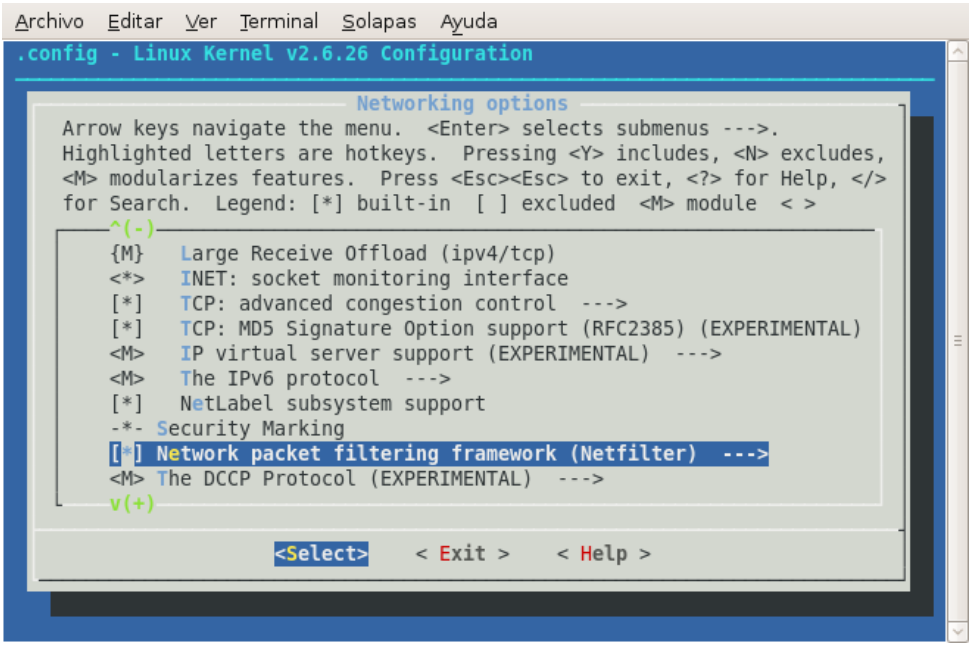


Figura 5: paso 3

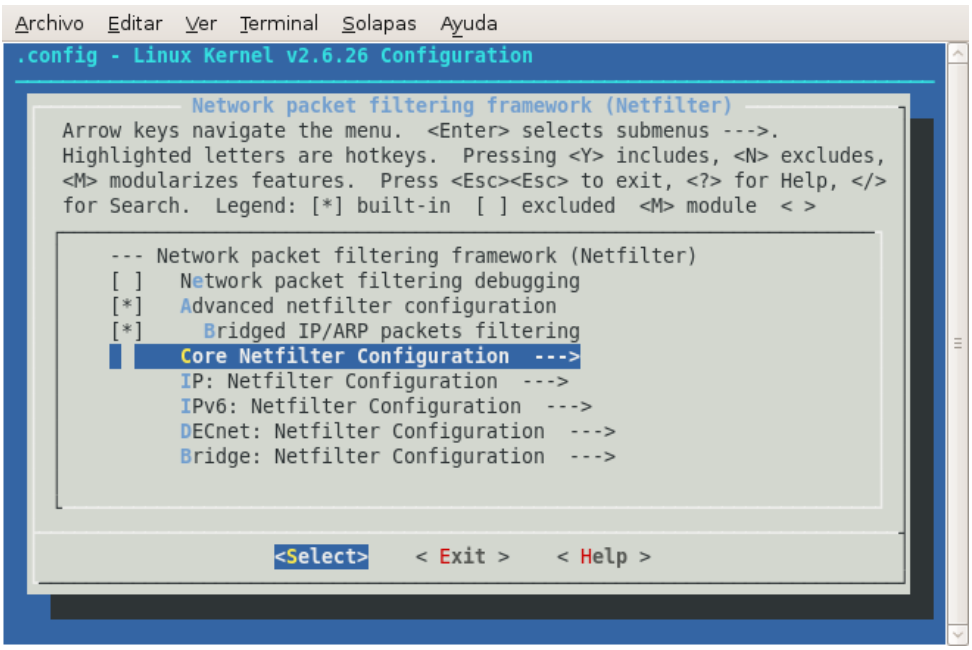


Figura 6: paso 4

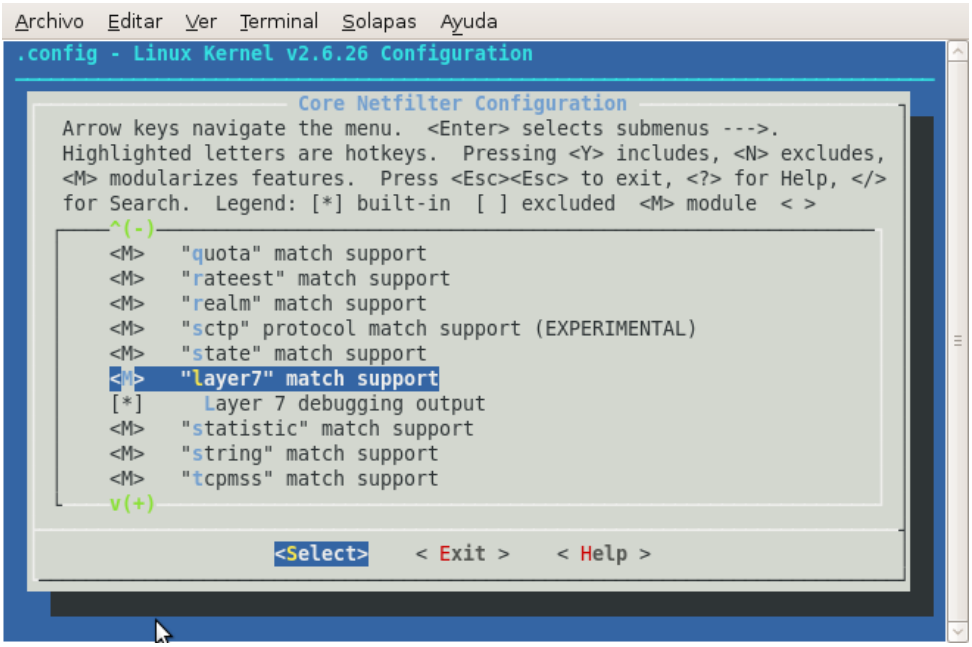


Figura 7: paso 5

5. Dentro de /boot/ generar el archivo de arranque del kernel:

apt-get install mkinitramfs
mkinitramfs -o initrd.img-2.6.26 2.6.26

Al ejecutar el comando anterior (mkinitramfs -o nombre versión) se genera el archivo de arranque que luego se debe configurar en menu.lst como se muestra en el siguiente paso.

6. Dentro de /boot/grub/ modificar el archivo menu.lst de la siguiente manera:

title	title
uuid	eecf99b0-4d53-4270-8263-312b8aacf4d8
kernel	boot/vmlinuz-2.6.26 root=UUID=eecf99b0-4d53-4270-8263-312b8aacf4d8
initrd	/boot/initrd.img-2.6.26
quiet	
make install	

Instalación de y configuración de COLLECTD

Requerimiento previo: Tener instalada la herramienta rrdtool. En caso de no ser así ejecutar el siguiente comando:

apt-get install rrdtool

Tomamos como referencia la url: *http://collectd.org/wiki/index.php/Main_Page*
Los pasos seguidos fueron los siguientes:

1. Se instaló la herramienta
apt-get install collectd

2. Dentro de /etc/collectd/ se configuró el archivo collectd.conf de la siguiente manera:
FQDNLookup true
BaseDir /var/lib/collectd//esta línea se descomentó
LoadPlugin Iptables //esta línea se descomentó
LoadPlugin rrdtool //esta línea se descomentó

```
<Plugin Iptables \>// se agregó la tabla y la cadena
  Chain mangle PREROUTING
<\Plugin>

<Plugin rrdtool \>
  DataDir /var/lib/collectd/rrd"
  CacheTimeout 120
  CacheFlush 900
  # The following settings are rather advanced
  # and should usually not be touched:
  StepSize 10
  HeartBeat 20
  # RRARows 1200
  # RRATimespan 158112000
  # XFF 0.1
<\Plugin>
```

Para poder graficar los datos obtenidos por Collectd, se debe dar permisos de lectura y escritura a las carpetas donde están contenido las bases de datos.

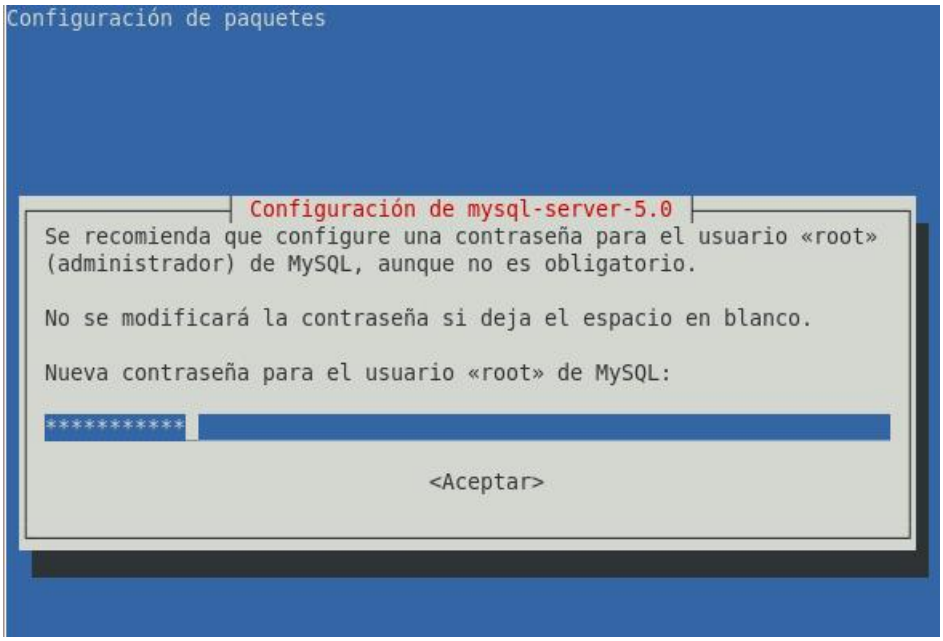
Instalación de la herramienta CACTI

Requerimientos previos:

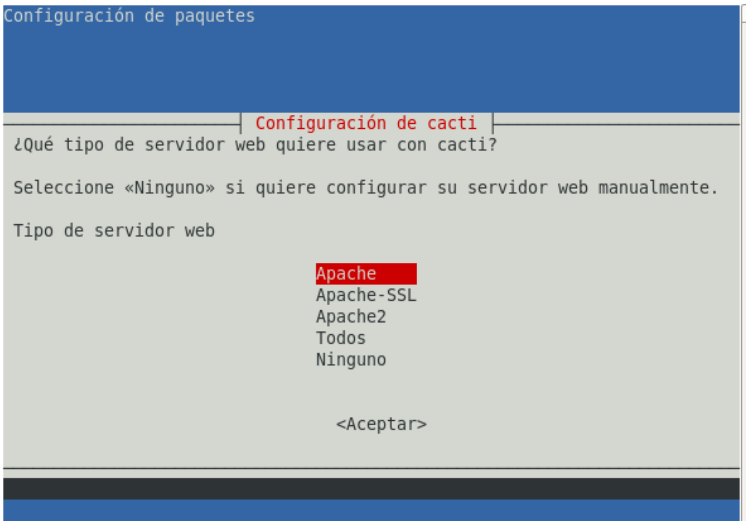
- Servidor web apache2: *apt-get install apache2*
- Módulo php: *apt-get install php5*
- Libreria para que apache2 y php se entiendan correctamente: *apt-get install libapache-mod-php5*
- Se instaló la herramienta cacti ejecutando el comando: *apt-get install Cacti*

Se muestran a continuación imágenes relevantes durante la instalación:

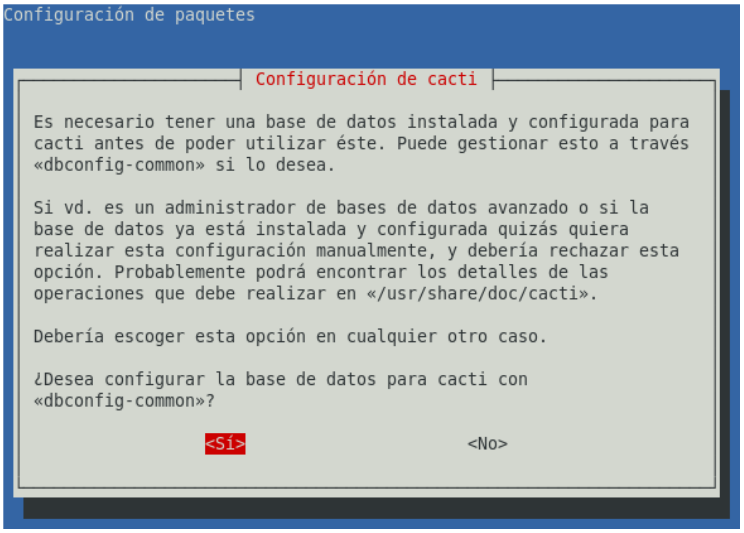
Solicitud de la contraseña para la base de datos MySQL



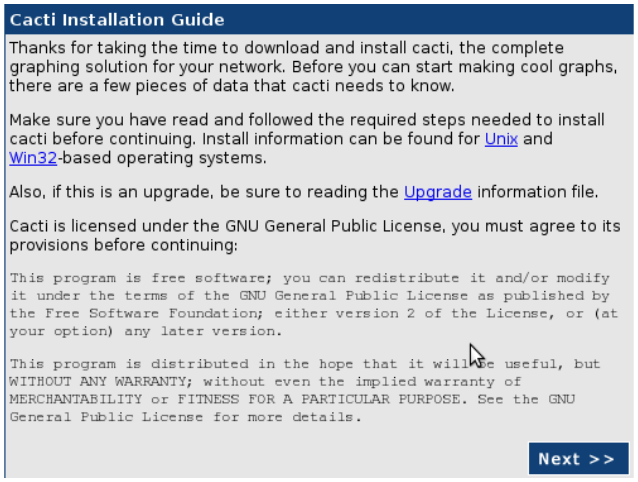
Confirmar contraseña de mysql
Configuración libphp-adodb: sólo debemos seleccionar aceptar.
Seleccionar el tipo de servidor que tenemos instalado: Apache2



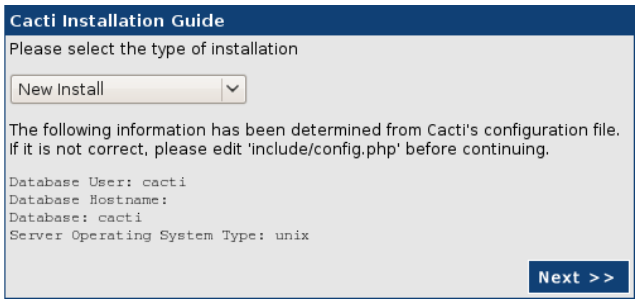
Seleccionar **SI** en la siguiente ventana para que se utilice la base de datos configurada anteriormente:



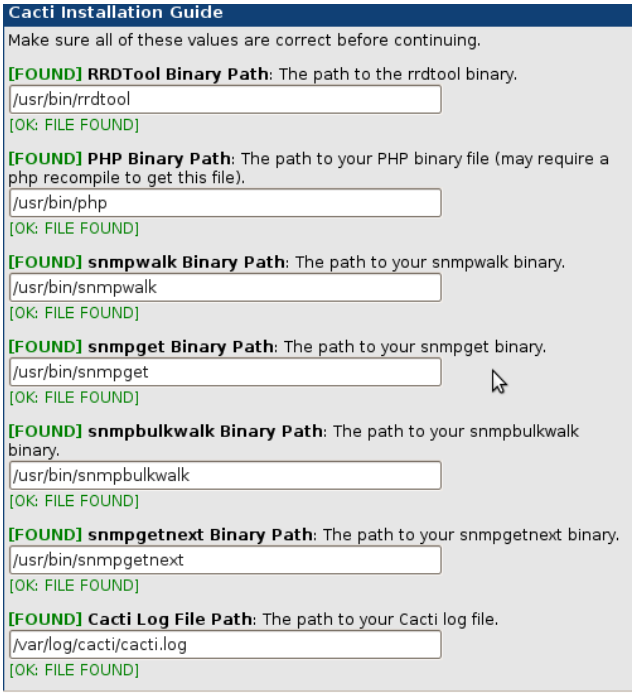
Aparecerá la siguiente ventana, en la cual debemos hacer click en Next:



Seleccionar nueva instalación y luego hacer hacer click en **Next**



A continuación, se muestra la ubicación de los diferentes archivos:



Finalmente, aparecerá la ventana para loguearnos. El username/password por defecto es admin/admin; luego el wizard requerirá ingresar nuevo username y password:

Los pasos a seguir para visualizar en diferentes gráficas, el porcentaje de cada protocolos respecto al total del tráfico, se detalla a continuación.

Pasos a seguir para agregar nuevos gráficos:

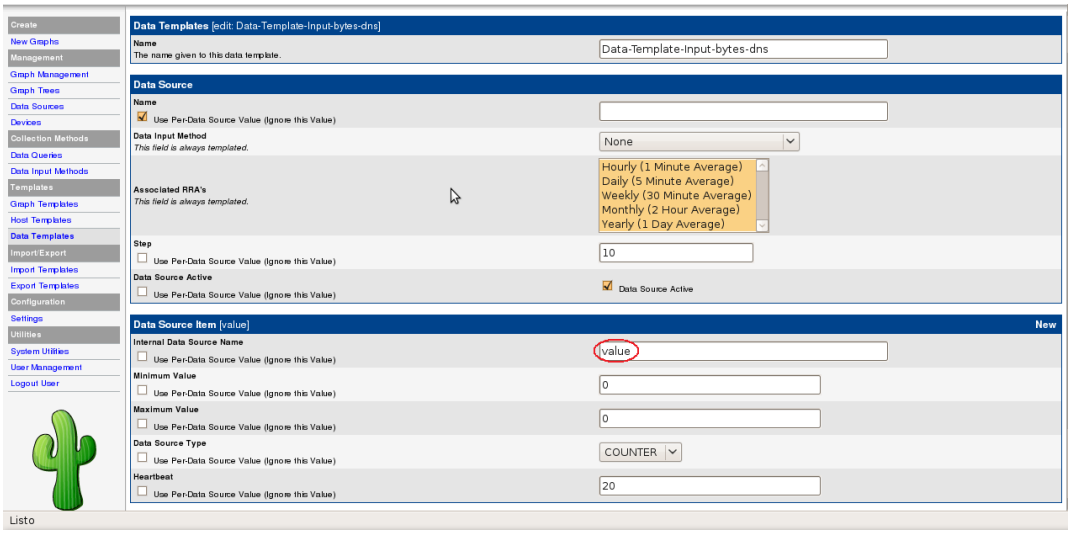
Crear Data Template

En este primer paso de la configuración, es importante que se tengan en cuenta las características de las fuentes de datos a visualizar. Para crear un nuevo Data Template y luego poder asociarlo a nuestro tipo de bases de datos, se debe especificar correctamente, el método a utilizar para obtener los valores, el Step, el Heartbeat, el tipo de datos a guardar y la DS.
Todos estos parámetros, se pueden obtener fácilmente ejecutando en un terminal el comando rrdtool info nombre.rrd.
Esto último hace que se despliegue toda la información sobre cómo fue creada dicha base de datos y las respectivas RRAs asociadas. Para entender mejor lo dicho anteriormente, tomamos como ejemplo la RRD creada por collectd para guardar el

incremento de paquetes/segundos del protocolo bittorrent y mostramos la información desplegada:

rrdtool info "ipt_packets-bittorrent-PREROUTING.rrd"
filename = "ipt_packets-bittorrent-PREROUTING.rrd"
rrd_version = "0003"
step = 10
last_update = 1268507108
ds[value].type = "COUNTER"
ds[value].minimal_heartbeat = 20
ds[value].min = 0.0000000000e+00
ds[value].max = 1.3421772800e+08
ds[value].last_ds = "0"
ds[value].value = 0.0000000000e+00
ds[value].unknown_sec = 0
rra[0].cf = "AVERAGE"
.....

Como se puede ver en la siguiente imagen, los parámetros a tener en cuenta en este paso son: Step, Heartbeat, el tipo de datos a ser guardados y el data source.

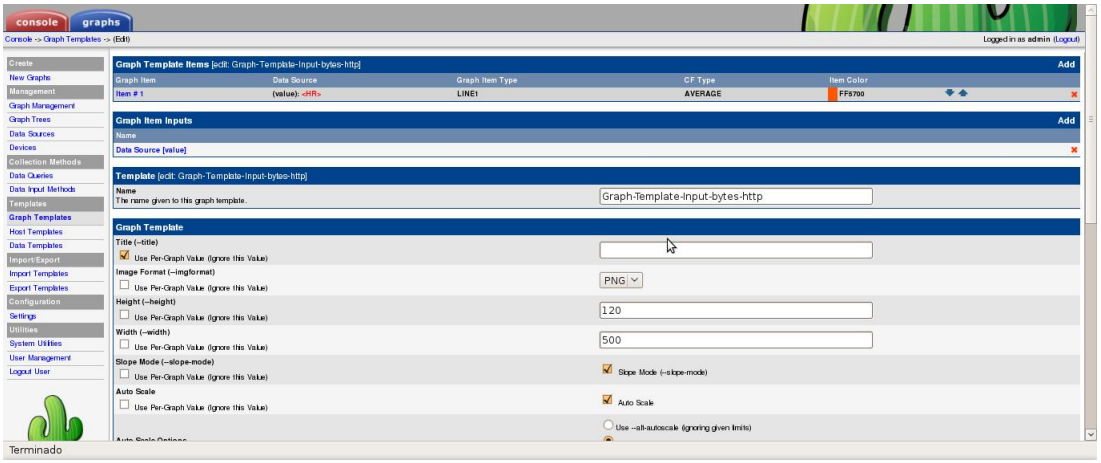


Campos a completar:
Data Templates:
Name: nombre del nuevo Data Template.
Data Source:
Name: escribir nombre o seleccionar en 'ignorar este valor'.
Data Input Method: None

Associated RRA's: seleccionar las diferentes RRAs que se quieren visualizar.
Step: 10, indica cada cuánto tiempo guarda un nuevo dato en la RRD.
Data Source Active: hacer click en Data Source Active.
Data Source Item [value]:
Internal Data Source Name: value (ds[value])
Minimum Value: dejar el valor por defecto 0.
Maximum Value: idem Minimum Value.
Data Source Type: COUNTER (ds[value].type = "COUNTER")
Heartbeat: 20 (ds[value].minimal_heartbeat = 20).

Crear Graph Template

Para crear un nuevo Graph Template se debe ir a la solapa Graph Template y hacer click en Add Graph Template.



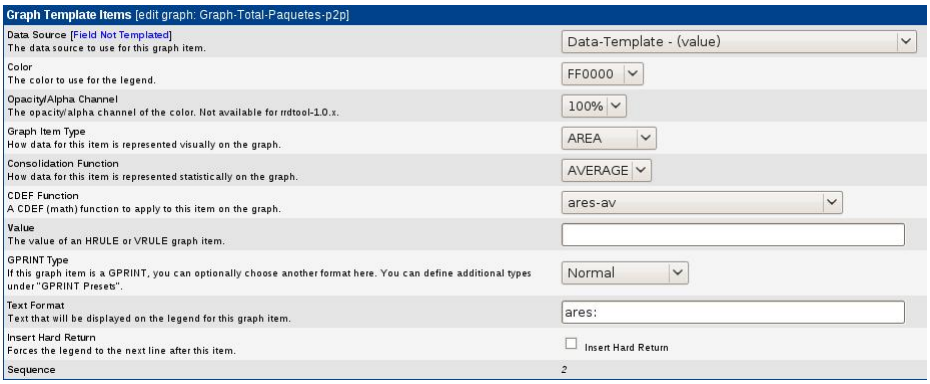
Campos a completar:
Graph Template Items:
Data Source: seleccionar el Data Template creado.
Color: color asignado para cada protocolo.
Graph Item Type: Area, Stack, etc
Consolidation Function: AVERAGE, MIN, MÁX.
CDEF Function: función creada para poder desplegar el porcentaje del tráfico de cada protocolo respecto al total.
GPRINT Type: Normal, Load Average, Exact Number. Se puede seleccionar opcionalmente otro formato para los datos.
Graph Ítem Inputs: se debe agregar un ítem por cada protocolo para poder asociar los ítems creados anteriormente a un mismo protocolo.
Template:
Name: nombre haciendo referencia al Graph Template creado.
Graph Template:

Title: título para este template o seleccionar 'ignorar este parámetro'.
Image Format: seleccionar formato PNG.
Auto scale: no seleccionar esta opción.
Las demás opciones dejarlas por defecto.

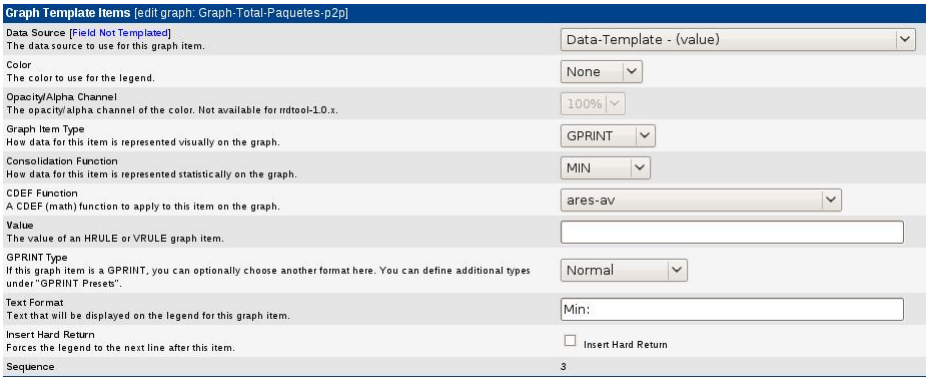
Una vez creado el Graph Template, se debe agregar un Graph Template Item dentro del cual a su vez, son agregados los diferentes ítems. En estos ítems, es donde se define, además de las características gráficas de cada uno de los protocolo, qué es lo que se quiere ver en dicha gráfica. En nuestro caso, se agregaron por cada protocolo, 4 ítems. Los mismos fueron configurados de forma tal que gráficamente, se vea el promedio del porcentaje de dicho protocolo sobre el total del tráfico, y además se despliegue en pantalla, el valor máximo, mínimo y promedio. Para poder realizar lo dicho anteriormente, se debe seleccionar para el primer ítem asociado a cada protocolo, la forma en que se quiere ver gráficamente la información. En los otros tres ítems creados para cada protocolo, se debe especificar la leyenda desplegada en pantalla y seleccionar la función de consolidación aplicada. Un detalle importante es que además de agregar los ítems para cada protocolo, se debe agregar otro, el cual será utilizado luego, para poder seleccionar la base de datos correspondiente al tráfico total y así poder realizar el correspondiente cálculo para obtener el porcentaje de cada protocolo sobre el total. Además no hay que olvidarse de seleccionar en todos los ítems menos en el último (asociado al tráfico total), la función CDEF creada anteriormente para poder hallar dichos porcentajes. Cómo crear los CDEFs se detalla más adelante.

A continuación se muestra la configuración de cada uno de los ítems nombrados anteriormente, tomando como ejemplo el protocolo ares.

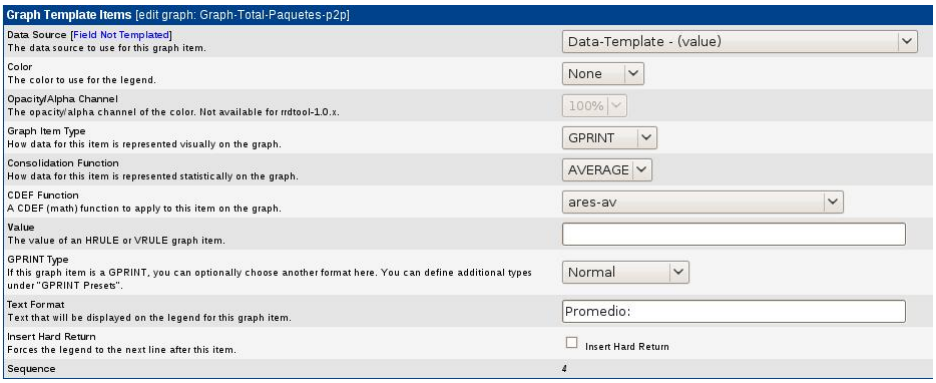
Ítems creados dentro del Graph Template ítem:
Ítem #1:



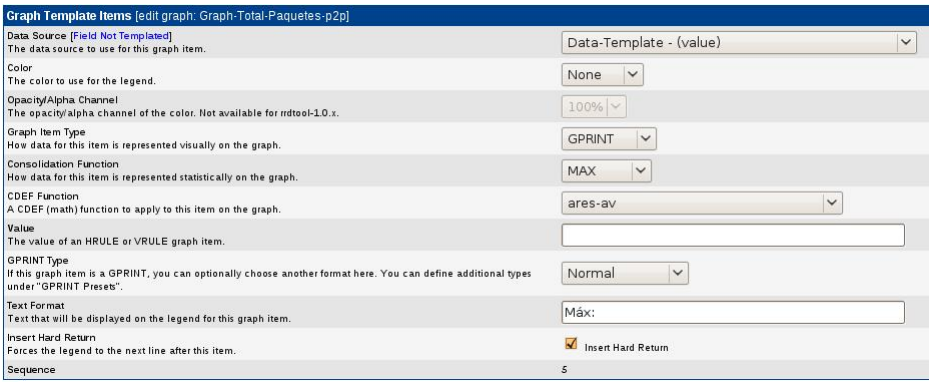
Ítem #2:



Ítem #3:



Ítem #4:



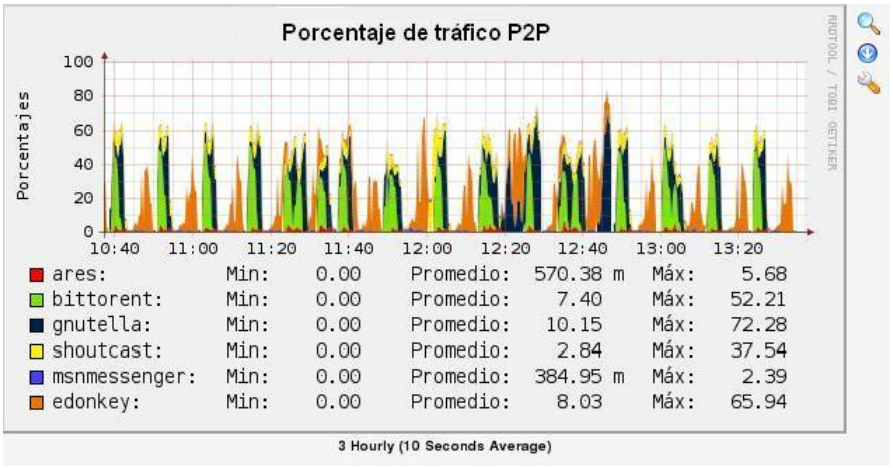
Item #1:
Data Source: Data Template creado
Color: seleccionar el color para la gráfica
Grah Item Type: AREA
Consolidation Function: AVERAGE
CDEF Function: ares-av (CDEF creado para este protocolo)
Gprint Type: nomal
Text Format: ares

Item #2:
Data Source: Data Template creado
Color: seleccionar none
Grah Item Type: GPRINT
Consolidation Function: MIN
CDEF Function: ares-av
Gprint Type: nomal
Text Format: Min:

Item #3:
Data Source: Data Template creado
Color: seleccionar none
Grah Item Type: GPRINT
Consolidation Function: AVERAGE
CDEF Function: ares-av
Gprint Type: normal
Text Format: Promedio:

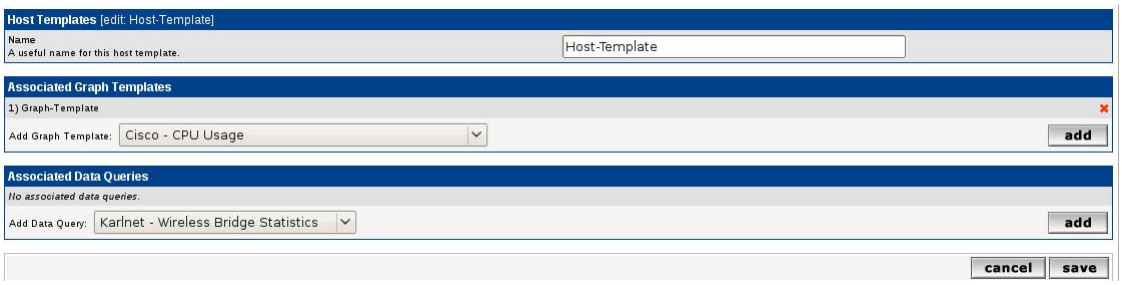
Item #4:
Data Source: Data Template creado
Color: seleccionar none
Grah Item Type: GPRINT
Consolidation Function: MAX
CDEF Function: ares-av
Gprint Type: nomal
Text Format: Máx

Para los demás protocolos que se quieran visualizar en la misma gráfica se deben crear, como se mencionó antes, 4 ítems para cada uno de ellos. La diferencia es que para el primer ítem asociado a cada uno de ello, se debe seleccionar STACK en lugar de ÁREA para que se grafique en stack. La configuración de los demás ítems es la misma mostrada para ares, cambiando solamente la CDEF seleccionada.
Con la configuración presentada, la gráfica incluyendo todos los protocolos p2p, se visualizará de la siguiente manera:



Crear Host Template

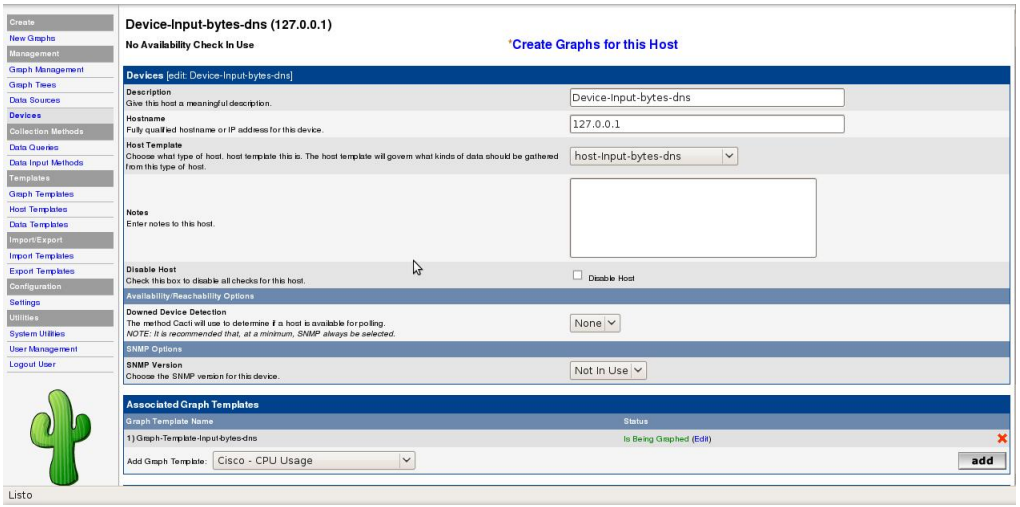
Ir a la solapa Host Template y hacer click en Add



Campos a completar:
Host Templates:
Name: Nombre del nuevo Host Template.
Associated Graph Templates: Agregar el Graph Template creado anteriormente.
Associated Data Queries: no asociar ningún 'data querie'.

Crear Crear device

Ir a la solapa Device y agregar uno nuevo. En este ítem es donde se especifica la IP del device, el Host Tempate utilizado y qué gráficas se van a ver en conjunto.



Campos a completar:
Desciption: descripción del nuevo dispositivo.
Hostname: seleccionar la IP o el nombre del host. En nuestro caso, ultizamos el localhost (127.0.0.1)

Host Template: seleccionamos el host template creado anteriormente.
Availability/Reachability Options: None
SNMP Options: Not In Use
Associated Gragh Templates: seleccionamos el Graph Template creado anterioremente.
Associated Graph Templates: Agregar el Graph Template creado.
Associated Data Queries: no asociar nada.

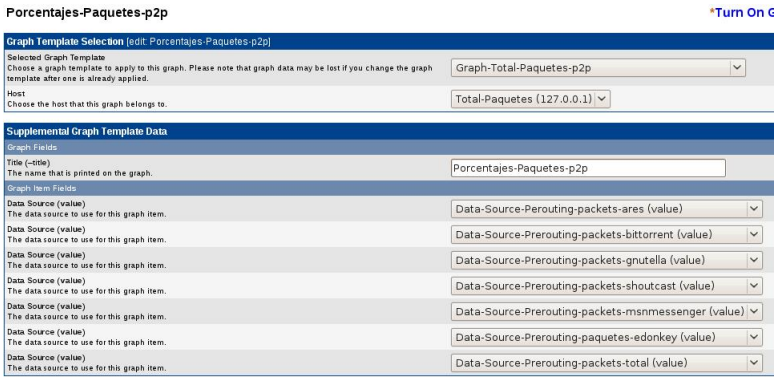
Crear Data Source

Se debe crear una nueva Data Source para cada fuente de datos a graficar.

Campos a completar:
Data Template Selection:
Selected Data Template: seleccionar el Data Template creado.
Host: seleccionar el Device creado.
Supplemental Data Template Data:
Name: escribir nombre para la nueva Data Source
Data Source Path: especificar la ruta completa en donde se encuentra la base de datos RRD a graficar, incluyendo el nombre de dicha base. Por ejemplo: /var/lib/collectd/rrd/vero-laptop/Iptables-mangle-PREROUTING/ipt_packets-ares-PREROUTING.rrd

Crear Graph Managment

En este ítem es donde se especifica que protocolos se van a visualizar utilizando el Graph Template creado. Para el ejemplo que tomamos, se había creado el Graph Template de forma tal que se pudieran ver los porcentajes del los protocolos p2p (ares, bittorrent, edonkey, msnmessenger, gnutella). Se puede observar a continuación que se seleccionaron esas fuentes de datos. Además se selecciona la fuente de datos que contiene el tráfico en su totalidad para poder realizar el cálculo necesario para hallar el porcentaje de cada uno de los protocolos.



Campos a completar:
Graph Template Selection:
Selected Graph Template: seleccionar el Graph Template creado.
Host: seleccionar el Device creado para el nuevo Host Template.
Supplemental Graph Template Data:
Title: escribir titulo de la gráfica.
Data Source [value]: seleccionar el Data Source creada.

Para ver cómo se debe crear la función CDEF correspondiente a cada protocolo, en la misma pantalla mostrada arriba, al hacer click en *Turn On Graph Debug Mode y se despliega lo siguiente:

```
/usr/bin/rrdtool graph - \
-imgformat=PNG \
-start=-86710 \
-end=-10 \
-title="Porcentajes-Paquetes-p2p"\
-rigid \
-base=1000 \
-height=120 \
-width=500 \
-upper-limit=100 \
-lower-limit=0 \
-vertical-label="Porcentajes"\
-slope-mode \
-font TITLE:12: \
-font AXIS:8: \
-font LEGEND:10: \
-font UNIT:8: \
```

DEF:a=/var/lib/collectd/rrd/vero-laptop/Iptables-mangle-PREROUTING/ipt_packets-ares-PREROUTING.rrd":value:AVERAGE

DEF:b=/var/lib/collectd/rrd/vero-laptop/Iptables-mangle-PREROUTING/ipt_packets-ares-PREROUTING.rrd":value:MIN

DEF:c=/var/lib/collectd/rrd/vero-laptop/Iptables-mangle-PREROUTING/ipt_packets-ares-PREROUTING.rrd":value:MAX

DEF:d=/var/lib/collectd/rrd/vero-laptop/Iptables-mangle-PREROUTING/ipt_packets-bittorrent-PREROUTING.rrd":value:AVERAGE

DEF:e=/var/lib/collectd/rrd/vero-laptop/Iptables-mangle-PREROUTING/ipt_packets-bittorrent-PREROUTING.rrd":value:MIN

DEF:f=/var/lib/collectd/rrd/vero-laptop/Iptables-mangle-PREROUTING/ipt_packets-bittorrent-PREROUTING.rrd":value:MAX

....

DEF:bi=/var/lib/collectd/rrd/vero-laptop/Iptables-mangle-REROUTING/ipt_packets-total-PREROUTING.rrd":value:AVERAGE

DEF:bj=/var/lib/collectd/rrd/vero-laptop/Iptables-mangle-REROUTING/ipt_packets-total-PREROUTING.rrd":value:MIN

DEF:ca=/var/lib/collectd/rrd/vero-laptop/Iptables-mangle-REROUTING/ipt_packets-total-PREROUTING.rrd":value:MAX

CDEF:cdefa=a,bi,/,100, *
CDEF:cdefe=d,bi,/,100, *
CDEF:cdefi=g,bi,/,100, *
....

AREA:cdefa#FF0000FF:.ªres\:"
GPRINT:cdefa:MIN:"Min\: %8.2lf%s"
GPRINT:cdefa:AVERAGE:"Promedio\: %8.2lf%s"
GPRINT:cdefa:MAX:"Máx\: %8.2lf%s \n"

*AREA:cdefe#7EE600FF:"bittorent\:":STACK *
GPRINT:cdefe:MIN:"Min\: %8.2lf%s"
GPRINT:cdefe:AVERAGE:"Promedio\: %8.2lf%s"
GPRINT:cdefe:MAX:"Máx\: %8.2lf%s\n"

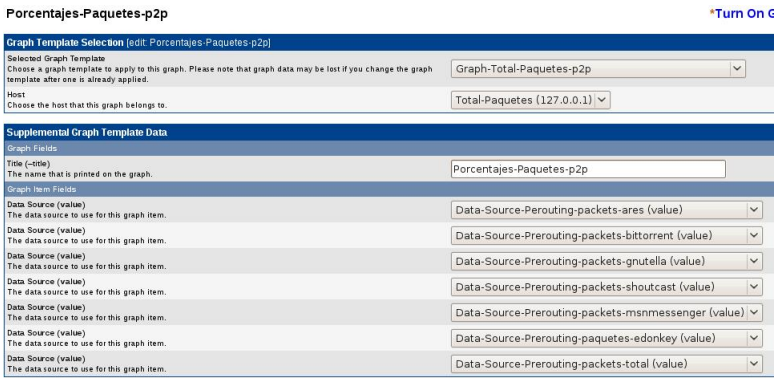
*AREA:cdefi#00234BFF:"gnutella\:":STACK *
GPRINT:cdefi:MIN:"Min\: % 8.2lf%s"
GPRINT:cdefi:AVERAGE:"Promedio\: %8.2lf%s"
GPRINT:cdefi:MAX:"Máx\: %8.2lf%s\n"

*AREA:cdefbc#FFF200FF:"shoutcast\:":STACK *
GPRINT:cdefbc:MIN:"Min\: %8.2lf%s"
GPRINT:cdefbc:AVERAGE:"Promedio\: %8.2lf%s"
GPRINT:cdefbc:MAX:"Máx\: %8.2lf%s\n"

....
RRDTool Says:
OK

En lo desplegado arriba, se puede ver algunas de las CDEFs creadas. Para el caso de ares, dicha función debe ser: CDEF:cdefa=a,bi,/,100,* \
Esta función toma el promedio de los datos obtenidos del protocolo ares (DEF:a=)y calcula el porcentaje sobre el total del tráfico (DEF:=bi).
Esto último hace además de agregar la cantidad de ítems correspondientes a los diferentes protocolos en Graph Template Item, sea necesario agregar también para el tráfco total y así poder realizar la cuenta correspondiente para dicho calculo.

Graph Tree -default tree



Campos a completar:
Graph Trees:
Name: nombre del árbol de gráficas.
Sorting Type: forma en que se ordenan los gráficos.
Tree Items: agregar la(s) gráfica(s) a visualizar. Al clickear en 'Add' aparece una nueva solapa, con los siguientes campos a completar:
Tree Items:
Parent Item:seleccionar 'root'.
Tree Item Type: seleccionar 'graph'.
Graph: seleccionar la gráfica a visualizar.
Round Robin Archive: elegir método 'round robin' para la gráfica.

Configuración herramienta en modo pasivo

Bridge

Es necesario contar con el paquete bridge-utils, por lo tanto se debe ejecutar:

apt-get install bridge-utils

Una vez instalado se debe configurar como se indica continuacion:

```
# agregamos un bridge con el nombre L7bridge
brctl addbr L7bridge
# asociamos eth0 al bridge
brctl addif L7bridge eth0
# se levanta la interfaz eth1
ifconfig eth1 up
# asociamos eth1 al bridge
brctl addif L7bridge eth1
# se levanta la interfaz nombrada L7bridge
ifconfig L7bridge up
# seteamos que las entradas al bridge no se actualicen
brctl setageing L7bridge 0
# se desahabilita la funcionalidad de spanning tree
brctl stp L7bridge off
```

Evitar contabilizar duplicado

Como se ha utilizado un loop ethernet en la interfaz de salida es necesario evitar que estos paquetes vuelvan a ingresar y sean contabilizados nuevamente, esto se puede evitar de dos maneras.

HALF DUPLEX

Se necesita contar con una herramienta que permita cambiar la manera en como trabaja la tarjeta de red donde se conecte el loop ethernet, esto se puede realizar por ejemplo utilizando ethtool o mii-tool. Si se utiliza el primero, se debe ejecutar:

```
ethtool -s interfaz autoneg off duplex half
```

PHYSDEV

Este módulo de Iptables se utiliza para definir acciones sobre paquetes que atraviesan un bridge, ejecutando el siguiente comando se descartan los paquetes que vuelven a ingresar por la interfaz donde se encuentre conectado el loop ethernet.

```
Iptables -t mangle -I PREROUTING -m physdev -physdev-in interfaz
-j DROP
```