UNIVERSIDAD NACIONAL DE LA PLATA
FACULTAD DE INFORMÁTICA

HTTPSniffer
MONITOREO DE TRÁFICO HTTP

HERRAMIENTA DE MONITOREO DE TRÁFICO HTTP BASADA EN LA WEB

ALUMNA:

Marisa Andrea Malvaso

DIRECTORES:

Prof. Javier Díaz

Prof. Miguel Luengo

Diciembre de 2000

A mis padres, que estuvieron a mi lado durante toda la carrera, siempre orgullosos de cada pequeño logro mío, que superaron mis pre-exámenes, que me alentaron, me apoyaron, me comprendieron, que estuvieron siempre a mi lado

A Fernando, mi compañero ideal, que siendo novios o esposos, siempre me acompañó y confió en que llegaría al final del camino, que superó mi ansiedad, que cada fin de semana que estudié estuvo conmigo, que gracias también a su esfuerzo estoy llegando a este momento

A Javier, mi director, que fue mi guía durante estos últimos años, que confió en mí, que me alentó siempre para salir adelante. Aún recuerdo cuando me acerqué a él en el año 96 y me hizo un lugar en el LINTI, y jamás voy a olvidar cuánto me costó decirle que empezaría a trabajar en Capital

A Miguel, que aceptó ser mi director, que me escuchó y me acompañó siempre, que siguió paso a paso cada avance de este trabajo, que es un amigo

A toda mi familia y mis amigos, que se alegraron de que haya llegado este momento y me acompañaron y comprendieron mi esfuerzo

---

ÍNDICE DE CONTENIDOS

Alumna: ... 1
Directores: ... 1

Capítulo I ... *5*

Introducción ... *5*

Conceptos preliminares ... **5**
El tipo de tráfico de interés ... 6

El papel de las redes de comunicaciones ... **7**
Capa de Transporte: Análisis de las conexiones TCP ... 7
Capa de Aplicación: HTTP ... 10
HTTP/1.0 vs. HTTP/1.1 ... 11
Tipos de Mensajes: solicitudes y respuestas ... 11
Campos del header del mensaje ... 12
Códigos de respuestas ... 13
Caching ... 15
Problemas de Performance ... 16

Capítulo II ... *17*

Logging vs. Monitereo ... *17*

Categorías de Logging ... **17**
Logs basados en un servidor de web ... 18
Logs basados en un servidor proxy ... 19
Logs basados en un cliente ... 20
Logs basados en la red ... 20

Aspectos de privacidad ... **23**

Capítulo III ... *24*

HTTPSniffer: la Herramienta de monitoreo ... *24*

Estructura de la herramienta de monitoreo ... **24**
Comunicación entre módulos y Plataforma de la herramienta ... 25

Implementación de cada capa de la herramienta ... **29**
Etapa de Captura y Filtrado de paquetes ... 29
Etapa de Decodificación y Registro ... 31
Etapa de Visualización ... 34

Instalación y Ejecucion de la herramienta de monitoreo ... **35**
Instalación de la herramienta ... 35
Forma de ejecución de la herramienta ... 36

Capítulo IV ... *37*

Otras herramientas de monitoreo ... *37*

Análisis de tcpdump ... **37**
BSD Packet Filter ... 37

HTTPDump: Script de parsing ... **38**

La novedad: Snort ... **39**

Demás herramientas analizadas ... **39**

TCPdpriv 39
TCP-reduce 39
Librería: Libcap 40
TCPtrace 40
Argus 40
Etherfind 40
IPtrace 40
NetSnoop 40
Snoop 41
Apéndice A *42*
Código de la herramienta implementada *42*
MAIN.C **42**
PARSER.C **43**
STORE.C **48**
HEADERS.H **51**
MYSQL_CREATE_DB.SQL **52**
Apéndice c *53*
Referencias *53*
Referencias a sitios de Internet **53**
Referencias a Libros **54**
Referencias a *Requests for comments* (RFCs) **54**
Referencias a Trabajos de Investigación **55**

HTTPSniffer
MONITOREO DE TRÁFICO HTTP

HERRAMIENTA DE MONITOREO DE TRÁFICO HTTP BASADA EN LA WEB

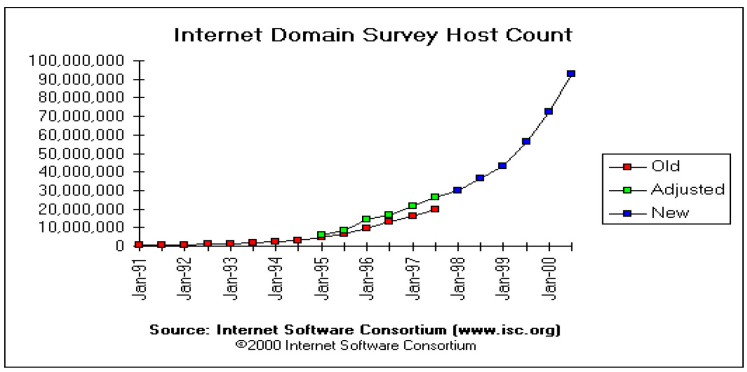
CAPÍTULO I
INTRODUCCIÓN

CONCEPTOS PRELIMINARES

En la actualidad, la necesidad de administrar un medio compartido, desencadena en la búsqueda de soluciones a la competencia por recursos; que se produce por aplicaciones que hacen un uso extensivo del ancho de banda, por ejemplo:

- clientes de bases de datos relacionales,
- transferencia de archivos extensos,
- navegación de páginas Web cargadas de imágenes, sonido y otro tipo de recursos multimediales (como video).

En principio, el servicio de navegación por páginas Web, si bien es un método amigable y simple de descubrir información, puede significar un uso importante de recursos, a punto tal de colapsar la red (más del 75% del tráfico de la red, proviene del acceso a páginas Web). Existen muchos estudios estadísticos respecto del crecimiento de la Web y de Internet. Uno de los más importantes es el proyecto de *Internet Domain Survey*[1], que intenta descubrir cada host en Internet realizando una búsqueda completa en el DNS (*Domain Name System*). La última estadística se completó en Julio del 2000, y los resultados fueron los siguientes:



[1] *Internet Software Consortium* (ISC) es el sponsor de este proyecto, y la operación técnica del mismo fue subcontratada a *Network Wizards*. Para obtener más información se puede acceder al sitio http://www.isc.org/ds

FECHA	CANTIDAD DE HOSTS	CANTIDAD AJUSTADA DE HOSTS[2]	RESPUESTA AL PING[3]
Julio 2000	93,047,785		
enero 2000	72,398,092		
julio 1999	56,218,000		
enero 1999	43,230,000		8,426,000
julio 1998	36,739,000		6,529,000
enero 1998	29,670,000		5,331,640
julio 1997	19,540,000	26,053,000	4,314,410
enero 1997	16,146,000	21,819,000	3,392,000
julio 1996	12,881,000	16,729,000	2,569,000
enero 1996	9,472,000	14,352,000	1,682,000
julio 1995	6,642,000	8,200,000	1,149,000
enero 1995	4,852,000	5,846,000	970,000
julio 1994	3,212,000		707,000
enero 1994	2,217,000		576,000
julio 1993	1,776,000		464,000
enero 1993	1,313,000		

■ *Tabla 1: Número de hosts advertidos en el DNS.*

EL TIPO DE TRÁFICO DE INTERÉS

Con el explosivo crecimiento de la World Wide Web [HTTP-IDS] surge la necesidad de caracterizar a quienes utilizan el servicio de Web y qué tipo de accesos se realizan. Por otra parte, para asistir al planeamiento del crecimiento de las redes de computadoras, quienes realizan network management[HTTP-TSW] [HTTP-SV3] requieren información del tráfico actual generado por la Web y una proyección del tráfico futuro. En términos más específicos, la World Wide Web es en la actualidad una fuente dominante del tráfico en el backbone de Internet, y muy pronto también lo será en redes de área local[4], y considerando que HTTP (*Hypertext Transfer Protocol)*[RFC-1945][RFC-2068] es la base para el Web, se selecciona este protocolo para el estudio de tráfico en el presente proyecto.

HTTP es utilizado por la iniciativa de información global de la World Wide Web desde 1990. Es un protocolo de nivel de aplicación con la facilidad y velocidad necesarias como para ser utilizado por sistemas de información distribuídos, colaborativos e hypermediales. Es un protocolo genérico, que no mantiene estado, orientado a objetos. A través de extensiones de su método de solicitud puede ser utilizado para diversas tareas. Una característica de importancia de HTTP es el tipado de la representación de los datos, permitiendo que los sistemas se construyan en forma independiente de la información que es transmitida[RFC-1945][RFC-2068].

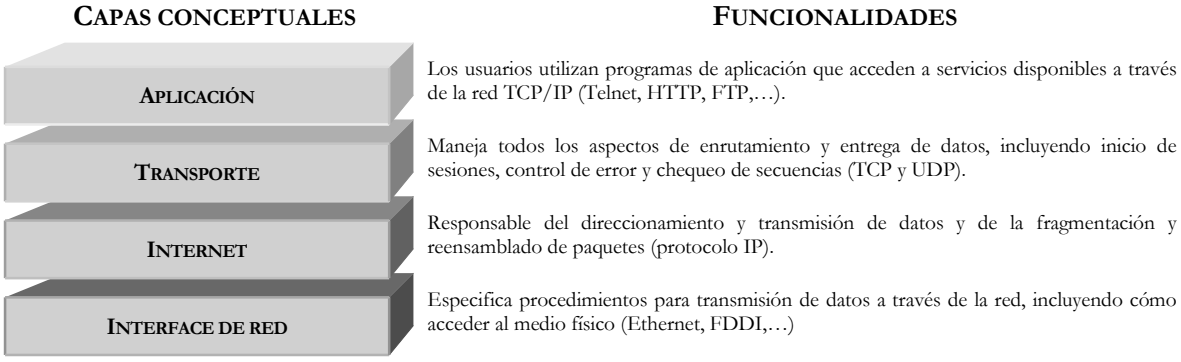
[2] La cantidad de hosts ajustada fue calculada incrementando las estadísticas anteriores conel porcentaje de dominios que no respondieron en su momento.

[3] Respuesta estimada ejecutando ping a una muestra de todos los hosts.

[4]En la actualidad, las aplicaciones basadas en Web (*Web-enabled applications*); incrementan notablemente el tráfico HTTP en LANs.

Para realizar actividades de análisis de tráfico (en especial el tráfico generado por la Web), y poder alcanzar una correcta interpretación de la información obtenida, se requiere un entendimiento razonable de las *redes de comunicaciones*.

Las redes de comunicaciones están organizadas en una serie de capas lógicas (con la excepción de aquella que se refiere a la transmisión sobre el medio físico). El modelo en capas mayormente utilizado en la actualidad es la *pila de protocolos TCP/IP*[LIB-TIV1TP]. Cuando se inicia una operación de red por un programa o comando de usuario, ésta viaja lógicamente hacia abajo a través de la pila de protocolos en la máquina local (via software), atraviesa el medio físico hacia la máquina destino y finalmente sube a través de la pila de protocolos remota hasta el recipiente de procesamiento apropiado. Cada capa es independiente de las demás y posee funcionalidades bien definidas:



Previo a comprender el funcionamiento de algunos de los protocolos pertenecientes a cada una de las capas, es importante considerar la forma en que los datos de las aplicaciones se relacionan con la interface física.

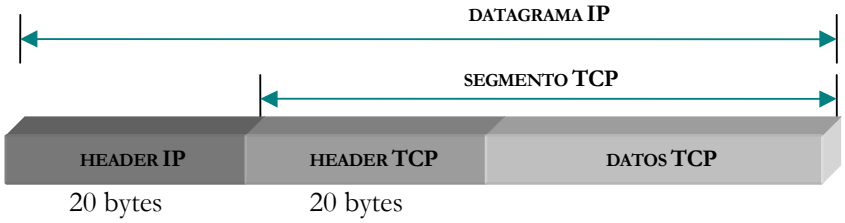
Cuando una aplicación envía datos utilizando protocolos de la pila TCP/IP, los datos son enviados hacia abajo por la pila de protocolos, a través de cada capa, hasta que son transmitidos como una cadena de bits a través de la red. Cada capa agrega información adelante de los datos que recibe, información llamada *header* (y algunas veces también agrega información al final, información llamada *trailer*). Este proceso se denomina *encapsulamiento de datos*.

Cada capa está equipada para manejar datos en unidades predefinidas. La unidad de datos que la Capa de Transporte envía a la Capa de Internet se denomina *segmento TCP*. La unidad de datos que la Capa de Internet envía a la interface de red se denomina *datagrama IP*. Finalmente, la cadena de bits que fluye a través del medio se denomina *frame* (o paquete).

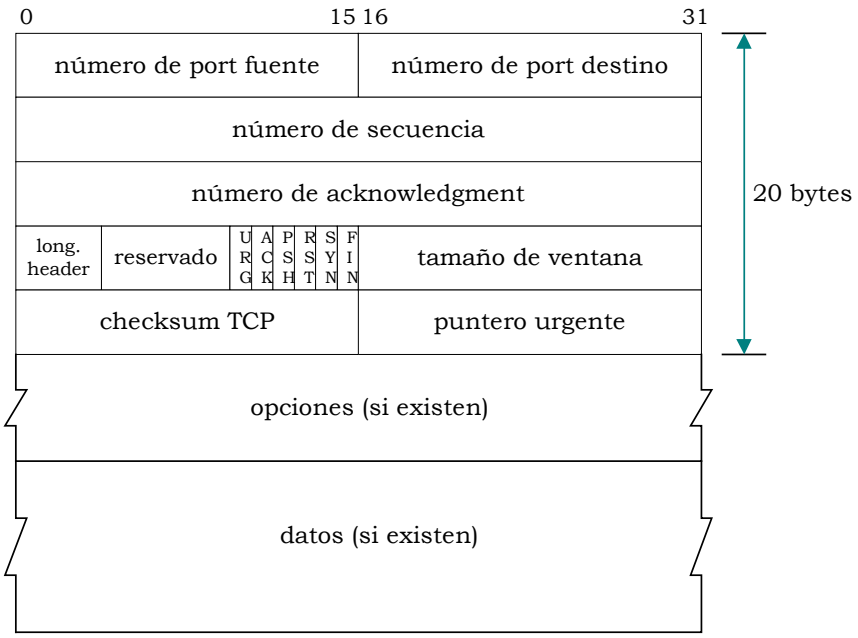
CAPA DE TRANSPORTE: ANÁLISIS DE LAS CONEXIONES TCP

Como se mencionó anteriormente, HTTP es el protocolo de interés en el presente proyecto. HTTP es un protocolo simple, perteneciente a la capa de aplicación, que utiliza TCP (*Transport Control Protocol*)[RFC-793] como capa de transporte. Esto conduce a establecer determinadas relaciones con algunas características del mundo real de TCP/IP.

TCP es un protocolo *orientado a conexión*, lo cual significa que dos aplicaciones que utilizan este protocolo deben establecer una conexión una con otra antes de iniciar el intercambio de datos. TCP ve al stream de datos como una secuencia de bytes, que para realizar la transmisión, divide en segmentos. Los datos TCP son encapsulados en un datagrama IP:



En el siguiente gráfico se muestra el formato del header TCP. Su tamaño habitual es de 20 bytes, a menos que se encuentren presentes opciones:



Cada segmento TCP contiene los *números de puerto* fuente y destino, para identificar las aplicaciones emisoras y receptoras. Esos dos valores, junto con las direcciones IP de origen y destino, identifican unívocamente cada conexión. La combinación de una dirección IP y un número de puerto, normalmente se denomina *socket*. El *par socket*[5] es el encargado de especificar los dos puntos finales que identifican en forma única cada conexión TCP en Internet.

Una de las características más importantes de TCP es que provee transmisión *confiable*. Esta característica se provee mediante el uso de los números de *secuencia* y de los *acknowledgments*. Conceptualmente, a cada byte de datos está asignado un número de

[5] Se llama *par socket* a la 4-tupla consistente de la dirección IP del cliente, el número de port del cliente, la dirección IP del servidor y el número de port del servidor.

secuencia. Por tanto TCP provee un servicio de conexión *full-duplex* a la capa de aplicación: esto significa que los datos pueden fluir en ambas direcciones en forma independiente. De esta manera, cada extremo de una conexión debe mantener un número de secuencia de los datos que fluyen en cada dirección. El número de secuencia del primer byte de datos en un segmento es transmitido en el mismo y se llama *número de secuencia del segmento*. Los segmentos también transportan un número de acknowledgment (confirmación) que es el número de secuencia del siguiente byte de datos de transmisión esperado en dirección inversa.

La *longitud del header* expresa la dimensión del header en una palabra de 32 bits. Este campo es necesario debido a que la longitud del campo de opciones posee un tamaño variable.

Existen seis *flags* en el header TCP, cada uno representado por un bit. Uno o más de los mismos pueden ser activados al mismo tiempo. El significado que poseen es el siguiente:

`URG`	Indica que el puntero urgente es válido.
`ACK`	Indica que el número de acknowledgment es válido.
`PSH`	Indica que el receptor debería pasar estos datos a la aplicación tan pronto como sea posible.
`RST`	Resetea la conexión.
`SYN`	Sincroniza los números de secuencia para iniciar la conexión.
`FIN`	Indica que el emisor terminó de enviar datos.

El control de flujo es la técnica mediante la cual se brinda la seguridad de que un transmisor no sobrecarga con datos a un receptor. Es provisto por cada extremo advirtiendo el *tamaño de la ventana*. La ventana es el número de bytes que el receptor está preparado para aceptar. El tamaño de la misma limita esta cantidad de bytes máxima a 65535.

El *checksum* cubre todo el segmento TCP: el header y los datos. Este es un campo obligatorio que debe ser calculado y almacenado por el emisor, y luego verificado por el receptor.

El *puntero urgente* solamente es válido si el flag URG está activo. Este puntero es un desplazamiento positivo, que debe ser sumado al campo de número de secuencia del segmento; para obtener el número de secuencia del último byte de datos urgentes. El modo urgente de TCP es una manera de que el emisor pueda transmitir datos de emergencia al otro extremo.

El campo de *opción* más común es la opción de máximo tamaño de segmento, llamada MSS. Normalmente, cada extremo de una conexión especifica esta opción en el primer segmento intercambiado (aquel segmento que posee el flag de SYN seteado para el establecimiento de la conexión). Esto especifica el máximo tamaño de segmento que el emisor quiere recibir.

Finalmente, la porción de *datos* del segmento es opcional. Esto se debe a que cuando se establece o se termina una conexión TCP, los segmentos intercambiados pueden contener únicamente el header TCP con posibles opciones. Además, un segmento sin la porción de datos puede ser utilizado para confirmar la recepción de datos (acknowledgment), si no existe información a ser transmitida en esa dirección. Debido a que el segmento TCP se transmite encapsulado en un datagrama IP, la porción de datos tiene un tamaño limitado por el MTU (*Maximun Transfer Unit*) de la red. Recordar que el MTU es la mayor cantidad de datos que pueden ser transferidos a través de una red física dada, y está determinado por el hardware de red[6].

CAPA DE APLICACIÓN: HTTP

Como se mencionó anteriormente, HTTP (*Hypertext Transfer Protocol*) es el protocolo que domina el intercambio de información a través de Internet, utilizado por clientes y servidores para enviar mensajes. HTTP es un protocolo simple de la capa de aplicación.

Los clientes de Web (habitualmente llamados *browsers*) establecen una conexión TCP con el servidor, envían una solicitud y esperan la respuesta del mismo. El servidor denota el final de su respuesta cerrando la conexión. Los tipos de documentos retornados por un servidor de Web pueden ser documentos HTML (*Hypertext Markup Language*)[LIB-H&C] y otros tipos de documentos; como por ejemplo imágenes, archivos PostScript, archivos en texto plano,…. Normalmente, los archivos retornados por los servidores de Web contienen punteros (links de hypertextos) a otros archivos, que además pueden residir en un servidor de Web diferente.

El cliente de Web es quien provee la simplicidad para el usuario, a través de agradables interfaces gráficas. Los servidores de HTTP, simplemente retornan los documentos solicitados por los clientes. De esta manera, los servidores de Web son más simples que los clientes de Web[7].

La primera versión de HTTP, referida como HTTP/0.9, fue un protocolo muy simple para transferencia a través de Internet de archivos planos. HTTP/1.0[RFC-1945] mejoró el protocolo permitiendo los mensajes en el formato MIME-like[HTTP-MIME] [RFC-2076], conteniendo metainformación acerca de los datos transferidos y de varios modificadores en la semántica solicitud/respuesta.

A pesar de las modificaciones realizadas, HTTP/1.0 no consideró lo suficiente los efectos de proxies jerárquicos, estrategias de caching, la necesidad de conexiones persistentes y la existencia de máquinas virtuales. Por esto se requirieron modificaciones que se refieren como HTTP/1.1[HTTP-2068] y que es la versión del protocolo que se utiliza en la actualidad.

En la práctica, los sistemas de información requieren mayor funcionalidad que simplemente el retorno de documentos, incluyendo búsquedas, actualizaciones front-end,…. HTTP permite un conjunto de métodos que indican el propósito de una solicitud, los cuales; para señalar el recurso sobre el cual se aplicará el método utilizan la

[6] Una red Ethernet posee generalmente un MTU de 1500 bytes.

[7] Por ejemplo: el servidor NCSA versión 1.3 para Unix tiene alrededor de 6500 líneas de código C, mientras el cliente Mosaic para Unix, que corre bajo el sistema X Windows tiene alrededor de 80.000 líneas de código C[LIB-TIIV3].

disciplina de referencia provista por la URI (*Uniform Resource Identifier*)[RFC-1630], como ubicación URL (*Uniform Resource Locator*)[RFC-1738] o nombre URN (*Uniform Resource Name*)[RFC-1737].

HTTP/1.0 VS. HTTP/1.1

En la actualidad suelen utilizarse tanto aplicaciones soportadas por la versión 1.0 de HTTP, como por la versión 1.1 del mismo. Muchas veces no suele repararse en esta condición, y sin embargo; existen varias características adicionales que se pueden obtener solamente con la versión 1.1 de este protocolo. Es por esa razón que se realizó un estudio comparativo entre ambos, con el objetivo de poder analizar cuáles son las ventajas y desventajas de cada una de las mencionadas versiones. A continuación en este documento se describen las diferencias encontradas.

Tipos de Mensajes: solicitudes y respuestas

De acuerdo "al lado" desde donde se inicia la conexión (del lado del cliente o del lado del servidor), existen dos tipos de mensajes HTTP: solicitudes y respuestas.

El formato de una *solicitud* HTTP es el siguiente:

```
linea-de-solicitud
campos del header      (0 o más)
<linea en blanco>
Cuerpo                 (sólo para una solicitud POST)
```

El formato de la *línea-de-solicitud* es:

```
Método     URI-solicitada     versión-de-HTTP
```

El formato de una *respuesta* HTTP es el siguiente:

```
linea-de-estado
campos del header      (0 o más)
<linea en blanco>
Cuerpo
```

El formato de una *línea-de-estado* es:

```
versión-de-HTTP   código-de-respuesta    frase-de-respuesta
```

El protocolo HTTP/1.0 soporta tres métodos de solicitudes diferentes:

- El método GET, que retorna la información que está identificada por la *URI-solicitada*.

- El método HEAD, similar al GET, pero solamente se retorna la información del header del servidor, no el actual contenido (cuerpo) del documento especificado. Esta solicitud normalmente es utilizada para testear la validez, accesibilidad y reciente modificación de un link de hypertexto.

- El método POST se utiliza para la entrega de correo electrónico, news o envío de formularios que son completados en forma interactiva por un usuario. Esta es la única solicitud que envía un cuerpo en el mensaje.

La implementación de HTTP/1.1 soporta algunos métodos adicionales que son de interés:

- El método OPTIONS representa una solicitud de información respecto de las opciones de comunicación disponibles sobre la cadena solicitudes/respuestas identificada por la URI solicitada. Este método permite a un cliente determinar las opciones y/o requerimientos asociados con un recurso, o las capacidades de un servidor, sin la necesidad de realizar acciones sobre un recurso o iniciar el retorno de un documento. Por ejemplo, un servidor proxy [8] puede verificar que un servidor cumpla con una determinada versión del protocolo.

- El método PUT se utiliza para solicitar que la entidad contenida en el requerimiento sea almacenada bajo la URI solicitada. Si la URI solicitada ya existe, el requerimiento se interpreta como una modificación del documento.

- El método DELETE solicita que el servidor elimine el recurso indicado por la URI solicitada. El cliente no puede asegurarse de que la operación sea llevada a cabo, salvo que el código de estado retornado por el servidor original indique que la acción se completó exitosamente.

Existen estadísticas que indican datos como los siguientes: de una muestra de 500.000 solicitudes de clientes sobre un servidor de Web, el 99.68% fueron GET, el 0.25% fueron HEAD y el 0.07% fueron POST [LIB-TIIV3].

Campos del header del mensaje

A partir de HTTP/1.0, tanto las solicitudes como las respuestas pueden contener un número variable de campos de header. Un campo de header consiste de un nombre de campo, seguido por dos puntos (":"), un espacio simple y el valor del campo. Dichos campos están separados del cuerpo del mensaje por una línea en blanco. Como se verá más adelante, es importante destacar este detalle, debido a que influencia fuertemente la determinación del tamaño del header de HTTP.

[8] Más adelante en este documento se ecplica el concepto de *servidor proxy*.

Los campos del header están divididos en cuatro categorías: los generales, aquellos que se aplican a solicitudes, los que se aplican a respuestas y los que describen el cuerpo del mensaje (también llamados campos de header de entidad). Los campos que describen el cuerpo del mensaje pueden aparecer en una solicitud POST o en cualquier respuesta. En la tabla (2) siguiente se muestran los 16 campos del header diferentes que corresponden a la versión HTTP/1.0. En la tabla (3) se muestran los campos del header adicionales existentes en la versión HTTP/1.1, que se identifican de la misma manera que en la versión HTTP/1.0.

- *Tabla 2: Campos del header HTTP/1.0 divididos por categorías.*

NOMBRE DEL CAMPO	TIPO DE CAMPO
Date Pragma	GENERALES
Authorization From If-Modified-Since Referer User-Agent	DE SOLICITUD
Location Server WWW-Authenticate	DE RESPUESTA
Allow Content-Encoding Content-Length Content-Type Expires Last-Modified	DE ENTIDAD

- *Tabla 3: Campos del header HTTP/1.1 divididos por categorías.*

NOMBRE DEL CAMPO	TIPO DE CAMPO
Cache-Control Connection Transfer-Encoding Upgrade Via	GENERALES
Accept Accept-Charset Accept-Encoding Accept-Language Host If-Match If-None-Match If-Range If-Unmodified-Since Max-Forwards Proxy-Authorization Range	DE SOLICITUD
Age Proxy-Authenticate Public Retry-After Vary Warning	DE RESPUESTA
Content-Base Content-Language Content-Location Content-MD5 Content-Range Etag	DE ENTIDAD

Códigos de respuestas

La primera línea de la respuesta del servidor se llama *línea de estado*. Esta línea comienza con la versión de HTTP, seguida por un código de respuesta numérico de tres dígitos y finalmente por una frase de respuesta legible. En la siguiente tabla se muestra el significado de cada uno de estos códigos numéricos de tres dígitos que se aplican en la versión de HTTP/1.0. Nuevamente, la implementación de HTTP/1.1 soporta algunos códigos más.

RESPUESTA	DESCRIPCIÓN
1yz	Informacional. Actualmente no utilizado.
	Éxito.
200	OK, solicitud exitosa.
201	OK, nuevo recurso creado (comando POST).
202	Solicitud aceptada pero procesamiento no finalizado.
203	OK, pero sin contenido a retornar.
	Redirección, se requiere que el agente de usuario realice acciones adicionales.
301	El recurso solicitado ha sido asignado a una URL nueva en forma permanente.
302	El recurso solicitado reside en forma temporal bajo una URL diferente.
304	El documento no ha sido modificado (GET condicional).
	Error de Cliente
400	Mala solicitud.
401	No autorizado, la solicitud requiere autenticación.
403	Prohibido por razones no especificadas.
404	No encontrado.
	Error de Servidor.
500	Error interno del servidor.
501	No implementado.
502	Gateway incorrecto, respuesta inválida del gateway.
503	Servicio no disponible en forma temporal.

▪ *Tabla 4: Códigos de respuestas de HTTP/1.0.*

Los códigos adicionales soportados por la versión 1.1 de HTTP son los siguientes:

RESPUESTA	DESCRIPCIÓN
	Éxito.
204	No Contenido: no hay nueva información para enviar por el servidor.
205	Resetear contenido.
206	Contenido parcial.
	Redirección, se requiere que el agente de usuario realice acciones adicionales.
305	Usar proxy.
	Error de Cliente
405	Método no permitido.
406	No aceptable.
407	Se requiere autenticación de proxy.
408	Solicitud vencida (request timeout).
409	Conflicto.
410	Gone: el recurso ya no está disponible en el servidor.
411	Se requiere longitud.
412	Falla de precondición.
413	Entidad solicitada demasiado larga.
414	URI solicitada demasiado larga.
415	Tipo de medio no soportado.
	Error de Servidor.
504	Gateway vencido (timeout).

505	Versión de HTTP no soportada.

▪ *Tabla 5: Códigos de respuestas de HTTP/1.1.*

Caching

HTTP/1.0 provee un simple mecanismo de caching[TRAB-ACOAB]. Utilizando el campo de header `Expires`, un servidor origen puede marcar una respuesta con el tiempo durante el cual un cache puede retornar un recurso sin violar la transparencia de la semántica. Así, un cache puede chequear la validez de una respuesta utilizando el método conocido como *requerimiento condicional*: incluye un campo `If-Modified-Since` en la solicitud del recurso; especificando el valor dado en el campo `Last-Modified` de la respuesta cacheada. HTTP/1.0 incluye además el mecanismo del campo `Pragma: no-cache`, para que el cliente se entere de que la respuesta no debe ser satisfecha por un cache. El mecanismo de caching utilizado por HTTP/1.0 trabaja moderadamente bien, pero no permite que los clientes y servidores den un conjunto completo y explícito de instrucciones al cache, dejando a este último la decisión de implementación.

En HTTP/1.0, un cache revalida una entrada utilizando el campo `If-Modified-Since`. Este campo utiliza timestamps absolutos con resolución de un segundo, lo que podría conducir a errores de cache, ya sea por sincronizaciones de relojes como por pérdida de resolución. De otro modo, HTTP/1.1 introduce un concepto más general de un string de validación, conocido como *tag de entidad*. Si dos respuestas del mismo recurso poseen el mismo tag de entidad, entonces las mismas deben ser idénticas (por especificación). Los clientes pueden comparar el tag de entidad por igualdad, pero no pueden modificarlo. Los servidores agregan un tag de entidad a las respuestas mediante el campo de header `Etag`.

HTTP/1.1 incluye el campo de header `If-None-Match`, que permite a los clientes presentar uno o más tags de entidad desde sus entradas de cache por un mismo recurso. Si ninguno de esos tags de entidad coincide con el actual valor del tag de entidad del recurso, el servidor retorna una respuesta normal, sino, retorna una respuesta 304 (No modificado) con el campo `Etag` que indica cuál de las entradas del cache es válida.

Para hacer los requerimientos de caching más explícitos, HTTP/1.1 agrega un nuevo campo `Cache-Control`, permitiendo que se transmitan un extenso conjunto de directivas de control de cache tanto en solicitudes como en respuestas. Las principales son:

- `max-age`: permite utilizar tiempos de expiración relativos.
- `private` y `no-store`: permiten a los clientes y servidores prevenir el almacenamiento en cache de algunas respuestas (como requerimiento de privacidad).
- `no-transform`: impide que un proxy transforme las respuestas (por ejemplo, para reducir la complejidad de una imagen que se va a transmitir sobre un medio lento).

Problemas de Performance

Dado el incremento en la utilización de HTTP, su impacto en Internet es de amplio interés. Como ya se mencionó, HTTP utiliza TCP como capa de transporte. Ciertas características de diseño de HTTP interactúan en forma ineficiente con TCP (al menos en la versión HTTP/1.0), provocando problemas con la performance y con la escalabilidad de servidores.

Existen principalmente dos métricas que permiten hacer un análisis de rendimiento de un servicio de HTTP: cantidad de conexiones establecidas con el servidor y tipo de las mismas.

El mayor factor que afecta el tiempo de respuesta percibido por los usuarios interactivos es la utilización de las conexiones TCP. Normalmente las páginas web contienen un documento HTML (*HyperText Markup Language*) y varias imágenes embebidas. Suele ser común que contengan 20 o más imágenes. Cada una de esas imágenes es un objeto independiente en la Web, retornado (o validado para cambio) en forma separada. De esta manera, el comportamiento común de un cliente web es buscar el documento HTML base e inmediatamente después buscar los objetos embebidos, los cuales, generalmente, se encuentran en el mismo servidor. Un gran número de objetos embebidos representa un cambio en el entorno para el cual fue diseñado HTTP. Como resultado: HTTP/1.0 maneja múltiples requerimientos al mismo servidor en forma ineficiente, creando una conexión TCP separada por cada objeto.

Una solución a esta situación es la utilización del campo de header HTTP `Pragma: hold-connection`. El campo general `Pragma` del header es utilizado para incluir directivas específicas de la implementación; que pueden ser aplicadas a cualquier receptor a través de una cadena de solicitudes/respuestas. Todas las directivas "pragma" especifican un comportamiento opcional desde el punto de vista del protocolo; de este modo, algunos sistemas pueden requerir que este comportamiento sea consistente con las directivas. Este campo permite que nuevos clientes se comuniquen con nuevos servidores para mantener la conexión abierta el tiempo que sea posible. Un servidor que no comprende este pragma lo ignora y cierra la conexión después de enviar cada documento.

Sin embargo, existe una diferencia muy significativa entre HTTP/1.1 y versiones anteriores de este protocolo. HTTP/1.1 permite *conexiones persistentes* como comportamiento por defecto de cualquier conexión HTTP. Es decir, a menos que se indique lo contrario, el cliente puede asumir que el servidor mantendrá una conexión persistente. Las conexiones persistentes proveen un mecanismo por el cual un cliente y un servidor pueden señalizar el cierre de una conexión TCP. Esta señalización toma lugar utilizando el campo `Connection` del header. El campo general `Connection` permite al emisor especificar opciones deseadas para la conexión corriente y que no deben ser comunicadas por proxies sobre el resto de las conexiones. Una vez que el cierre ha sido señalizado, el cliente *no debe* enviar más requerimientos sobre dicha conexión. Los clientes y servidores no deberían asumir que se mantienen conexiones persistentes para versiones de HTTP menores que 1.1 a menos que esto se indique explícitamente.

CAPÍTULO II
LOGGING VS. MONITEREO

CATEGORÍAS DE LOGGING

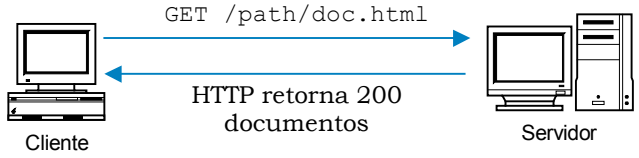
Previo a la investigación del monitoreo de tráfico, se realizó un estudio de diferentes técnicas utilizadas en el análisis del comportamiento de una red.

Una técnica comúnmente utilizada en el análisis del comportamiento de una red es el proceso de registro de logs, *logging*, que provee información exhaustiva respecto de las actividades que realizan determinada población de usuarios[TRAB-CSMRA].

El logging es un proceso de registro que consiste en almacenar en un archivo, en forma automática, una secuencia de eventos de máquina observables. Normalmente, esos eventos son *solicitudes* (por ejemplo: el retorno de un documento, la ejecución de un script, el download de un applet,…) o *respuestas* (por ejemplo: documentos contenidos en una solicitud, salidas desde los scripts, código de applets,…). Técnicamente, el logging puede realizarse sin conocimiento de los usuarios, a pesar de que eventualmente esto podría estar prohibido por normas éticas o legales.

El *logging* puede representar un registro casi perfecto de las solicitudes alcanzadas por un servidor de Web o por un proxy, siempre que hayan sido enviadas a través de la red o tipeadas en un cliente de Web (comúnmente llamado *browser*)[9]. Existen distintos métodos para realizar logging, y los mismos se pueden caracterizar teniendo en cuenta el lugar de la red desde donde se recolecta la información.

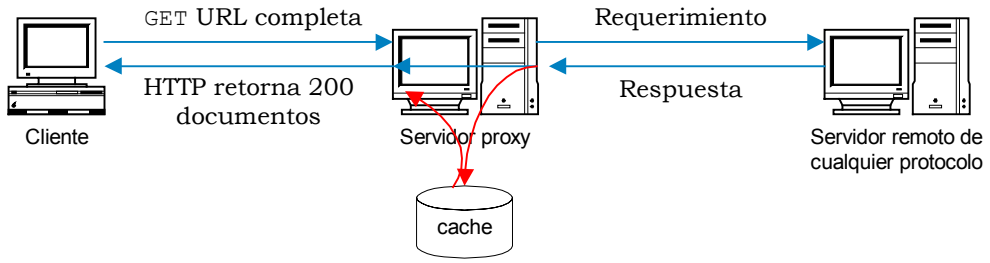
En la arquitectura de Web más sencilla, cada vez que un cliente envía una solicitud, se envía sobre la red un mensaje HTTP[RFC-1945][RFC-2068], desde el cliente hacia el servidor que se nombra en el campo URL (*Universal Resource Locator*)[RFC-1738] de la solicitud. Luego el servidor retorna uno o más mensajes conteniendo o bien la respuesta, o bien un código de error[10].



[9] Se dice "casi perfecto" porque el dispositivo que recolecta los logs puede fallar, el dispositivo de recolección (por ejemplo el software) puede contener bugs, o los logs recolectados pueden ser maliciosamente modificados o fabricados.

[10] Para conocer más del *diálogo* HTTP, ver el "Capítulo I" de este informe.

La arquitectura anterior es algo más compleja cuando se utilizan servidores proxies. Un *servidor proxy* es un programa intermediario que actúa como servidor y como cliente, con el propósito de realizar solicitudes en nombre de otros clientes. Las solicitudes que arriban al proxy pueden ser atendidas internamente o simplemente pasar a través del mismo. Un proxy debe interpretar y, de ser necesario, rescribir un mensaje de solicitud antes de reenviarlo. Los proxies pueden desempeñar el papel de firewall, de cache, o de ambos



Teniendo en cuenta las arquitecturas de Web recién presentadas, existen cuatro sitios posibles desde donde registrar logs, correspondiendo a las cuatro categorías de logging: en el servidor, en el proxy, en el cliente y sobre la red.

LOGS BASADOS EN UN SERVIDOR DE WEB

El logging realizado en un servidor es la forma más ampliamente utilizada. En esta estrategia se captura información acerca de las solicitudes de los clientes a un simple sitio Web, en cualquier parte del mundo donde el mismo se encuentre. Los servidores de Web pueden recolectar un log de solicitudes de documentos y pueden registrar cada método que se encuentre en los mensajes de HTTP que reciben (GET, POST o HEAD)[11].

Existen límites en la información que proveen los logs de un servidor de Web:

- Un gran número de solicitudes puede provenir desde servidores proxy. De esta manera, el nombre de la máquina cliente tal vez sea anónimo, debido a que el log del servidor contiene el nombre del servidor proxy.

- El administrador de Web tal vez deshabilite la posibilidad de recolectar los nombres de las máquinas clientes, debido a que dicha recolección puede no ser ética o legal[HTTP-CDT].

- El log del servidor solamente contiene aquellas solicitudes que actualmente alcanzan al servidor, y excluye aquellos casos en los que el usuario examina un documento que está cacheado por un servidor proxy o por el explorador de Web del cliente. De esta manera, debido a que la utilización de caches, particularmente de proxy caches, crece día a día en la Web, declina la precisión de los logs de servidores como una medida de solicitudes de páginas Web.

[11] En el "Capítulo I" de este informe se realizó un análisis de la técnica de caching.

Finalmente, existen distintas consideraciones que, en ciertos casos, hacen más dificultosa la implementación de esta categoría de logging.

- Si existe más de un servidor que provee el mismo servicio, surge la necesidad de relacionar los logs entre los mismos y de planear una estrategia para que toda la información registrada conserve el mismo formato y no sea redundante.

- Si en un servidor se provee más de un servicio, se debe tener especial cuidado para que los procesos de registro de información no interfieran unos con otros.

Se podría pensar que la solución ideal para estas circunstancias es el agregado de un dispositivo entre los clientes y servidores, que sea quien realice el proceso de logging, funcionando como un *filtro* para el acceso a los servicios. Las principales desventajas están relacionadas con la inserción de un único punto de falla[12] y la posible degradación en la performance de un servicio como consecuencia de un alto número de solicitudes que no están dirigidas al mismo pero que son filtradas por el mencionado dispositivo.

LOGS BASADOS EN UN SERVIDOR PROXY

El logging realizado en un servidor proxy captura la información que llega a dicho servidor. De esta manera caracteriza el conjunto de solicitudes realizadas por una población de clientes, que realizan solicitudes a servidores Web en cualquier parte del mundo, y cuyos exploradores están configurados para utilizar el servidor proxy.

La realización de logging en servidores proxies también tiene limitaciones:

- La mayoría de los clientes de Web crean caches en la memoria en forma automática y siempre crean caches en los discos de las máquinas en donde corren los navegadores. De esta manera, aquellas solicitudes de los usuarios que son satisfechas por el cache de memoria o de disco en el cliente no son enviadas al servidor proxy, y así el log de dicho servidor no las puede capturar.

- A diferencia de los caches de los clientes que son creados en forma automática, en la actualidad los navegadores de Web solamente utilizan un servidor proxy cuando el usuario explícitamente lo configura para tal finalidad. Como consecuencia de ello, los logs de un servidor proxy podrían representar sólo una muestra parcial de clientes, aquellos que poseen usuarios lo suficientemente sofisticados como para conocer la existencia de un servidor proxy y la manera de configurar el navegador para su utilización.

En consecuencia, los costos ocultos de esta categoría de logging involucran, entre otros, el proceso de configuración de los clientes y servidores para la utilización del servidor proxy.

[12] Si este dispositivo falla, además de impedir que se continúe con el logging, se esta quebrando la conexión con los servidores, con lo cual disminuye la disponibilidad del servicio.

LOGS BASADOS EN UN CLIENTE

En este caso el logging es realizado en una máquina cliente, ya sea por el mismo navegador de Web o por un proceso separado de monitoreo que ejecuta en dicha máquina. Los logs pueden ser una caracterización precisa de cada documento que examina el usuario[HTTP-CLT].

En la actualidad, normalmente los clientes no tienen facilidades adicionales para generar logs. De esta manera, no existe un formato o un conjunto de información a listar "estándar" para los logs de clientes, de donde se deduce la dificultad que provoca unificar la información obtenida en cada uno de ellos. De todos modos, técnicamente es posible registrar este tipo de logs. Por ejemplo, los clientes de Web NCSA Mosaic[HTTP-NCSAM] fueron modificados por diversas universidades (por ejemplo por la Universidad de Boston) para recolectar logs de clientes. Entre otra información permiten registrar los siguientes items: nombre de la máquina, hora, URL, tiempo de retorno y si la referencia URL fue retornada directamente o desde una referencia en otra URL. Otro ejemplo del potencial para el logging basado en el cliente es HindSite[HTTP-IHNN], un plug-in para Netscape Navigator. Este plug-in almacena en una base de datos la URL, título, fecha y hora de acceso, fecha y hora de la última modificación del documento y tamaño del documento para cada URL solicitada por el usuario.

La ventaja del logging basado en el cliente es que éste se sobrepone a algunos problemas anteriormente mencionados que ocurren utilizando las otras dos estrategias, debido a que también registra información de aquellas solicitudes que fueron satisfechas por el cache local del cliente (ya sea en memoria volátil o en disco). Esto se debe a que el logging basado en el cliente puede almacenar todos los accesos generados por un usuario, antes de que los mismos sean dirigidos al servidor de Web o al proxy cache.

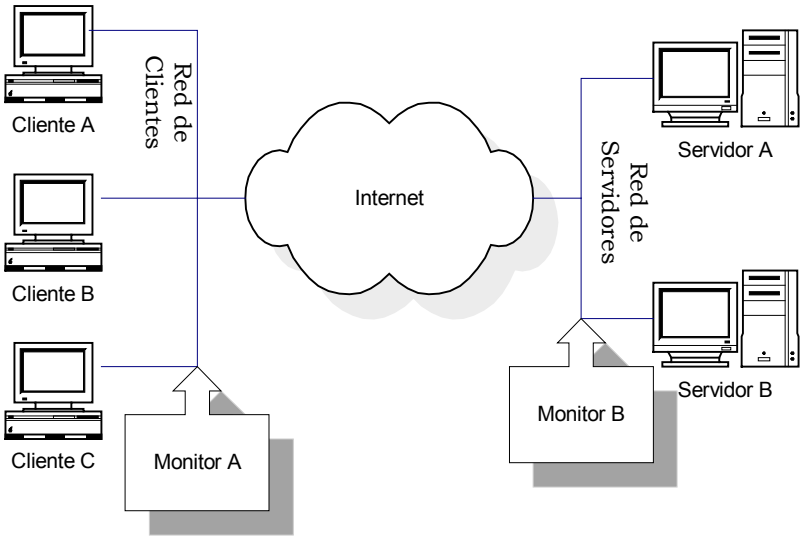
La limitación obvia del logging basado en el cliente es que la información puede ser recolectada solamente desde aquellos clientes que estén dispuestos a ejecutar un navegador de Web que registre logs. Además se debe disponer de un método adicional que permita combinar todos los registros de logs para obtener información de los clientes de la red (proceso que tal vez tenga la necesidad de ser realizado en forma manual), y este proceso, sin duda, requiere un gran esfuerzo adicional de programación y mantenimiento para que la información sea completa y no redundante. Finalmente, frente a usuarios expertos, se corre el riesgo de que los registros sean alterados.

LOGS BASADOS EN LA RED

En esta alternativa, una máquina es conectada a la red para que realice el monitoreo de la misma[13]. El monitor escucha pasivamente todo el tráfico que viaja en la red, identifica paquetes que contienen partes de mensajes HTTP y construye un log, que entre otra información, mantiene las URLs solicitadas en dichos paquetes. La población de clientes que el logging basado en la red puede caracterizar depende del lugar de la red en donde se conecta el monitor. Por ejemplo, un monitor ligado a una red de máquinas clientes que ejecutan exploradores de Web, como es el caso del "monitor A" en el siguiente gráfico, registra información respecto de dicho conjunto de clientes.

[13] Una máquina que realiza monitoreo de red, *network monitor*, también suele llamarse *network sniffer*.

Alternativamente, un monitor ligado a una red que conecta uno o más servidores de Web, como es el caso del "monitor B" en el gráfico anterior, registra información respecto de solicitudes destinadas a cualquiera de dichos servidores.



Finalmente, los monitores de red pueden ser útiles para mejorar el logging basado en un servidor proxy. Por ejemplo: el monitor puede registrar solicitudes de clientes que no estén configurados para utilizar un servidor proxy y de clientes que no estén conectados a redes con servidores proxies.

El dispositivo que monitorea la red puede ser tanto un instrumento de monitoreo (como por ejemplo el LANalyzer para Ethernet[14]) como un computador de propósito general ejecutando el software apropiado (como por ejemplo una herramienta como la que se implementó en este proyecto).

Esta estrategia de monitoreo de red, que no es nueva[15], posee varias ventajas:

- Es transparente: no se requieren cambios en el cliente, proxy o servidor de Web, y no tiene impacto sobre la performance de los clientes, proxies, servidores de Web o sobre la red.

- Es segura: nadie puede acceder a los datos monitoreados, excepto quien controle el dispositivo de monitoreo.

- Puede utilizarse para verificar información registrada por otros mecanismos de logging.

- El monitor de red puede estar programado para registrar información de otros protocolos además del HTTP. Además de capturar información de otros protocolos populares en la Web (FTP, Real audio,…) podría

[14] En http://www.caida.org/Caidants/meastools.html se pueden encontrar herramientas adicionales de análisis de paquetes.

[15] Los monitores han sido utilizados por mucho tiempo como herramientas para análisis de problemas en redes de comunicaciones.

capturar eventos tales como conexiones no concluidas o erróneas, así como también ampliarse para trabajar sobre otros protocolos y capturar tráfico excesivo en el que no suele repararse (ICMP, POP,…) y puede degradar la performance de la red cuando se utiliza en forma indiscriminada.

- No existe problema de muestreos o auto-selección: el monitor recolecta todas las solicitudes de documentos, y si el mismo es lo suficientemente lento como para no poder registrar todos los paquetes de la red, la muestra capturada es aleatoria.

- Un monitor de red puede utilizar el header de los mensajes HTTP capturados para realizar cálculos basados en múltiples paquetes, y de esta manera producir más información respecto de la utilización del Web de la que se puede obtener con el formato de logs estándar de los servidores proxy o de Web. Realizar este tipo de cálculos en un servidor que tiene una funcionalidad tal como cachear, filtrar tráfico (firewall) o proveer un servicio (en este caso Web), puede ser crítico para su óptimo funcionamiento.

- No se introduce un único punto de falla, si falla el monitor de red; lo único que falla es el registro de información, los demás servicios continúan su actividad en forma independiente.

Sin embargo, a pesar de todas las características favorables que posee, cuando se intenta aplicar una estrategia de monitoreo de red se deben superar las siguientes dificultades:

- La necesidad de o bien una red que permita broadcast (tal como una Ethernet, token ring o FDDI ring) o bien configurar el monitor como un gateway sobre un enlace de red punto-a-punto.

- La creciente tendencia en las redes de área local de reemplazar los hubs por switches; cuando se trata de dispositivos de interconexión, genera la necesidad de realizar configuraciones adicionales para poder capturar los paquetes que circulan por la red. Eso se debe al mecanismo que utiliza un switch para enviar paquetes de datos entre los distintos puertos que posee: almacena una tabla con las direcciones MAC de cada dispositivo conectado a sus puertos, y eso le permite establecer una conexión cuando resulte necesario y terminarla cuando ya no hay sesión alguna que soportar. De esta manera evita que se propaguen señales por tramos del medio que no conectan el origen y destino de una transmisión en particular. La única forma de utilizar la herramienta de monitoreo con este tipo de dispositivos, es conectarla a un puerto del mismo que se declare como "puerto de monitoreo", y agregar el resto de los puertos como candidatos a monitorear. Como resultado, todo el tráfico que circule por estos puertos a monitorear podrá ser capturado sin inconvenientes.

- Finalmente, se debe tener especial cuidado con el número de clientes y servidores que posee la red, debido a que existe un límite en el porcentaje

de paquetes que puede registrar el monitor. Una solución a esta restricción puede ser conectar más de un monitor a la red, configurando cada uno de ellos para que registre un conjunto de paquetes provenientes de diferentes direcciones de clientes o servidores.

ASPECTOS DE PRIVACIDAD

Cualquier estrategia de logging (ya sea basada en el servidor, en el cliente, en el proxy o en la red) promueve aspectos de privacidad. Estos aspectos suelen discutirse debido a que en algunos países existen guías de ética o reglas legales que describen cómo pueden utilizarse los registros de logs sin atentar contra la privacidad de los usuarios[HTTP-CDT]. Por ejemplo: actualmente en la Web, los servidores y proxies recolectan logs en forma rutinaria; que al almacenar los nombres de las máquinas clientes; identifican usuarios en forma indirecta[16].

En particular, el logging basado en la red acentúa este aspecto, debido a que se realiza en forma transparente, sin modificaciones en los clientes o servidores. Sin embargo, el objetivo de realizar monitoreo es, generalmente, identificar un problema de performance en la red o analizar aspectos de diseño y utilización de la misma, y para poder llevar a cabo esta tarea se debe comprender el entorno de los usuarios, las solicitudes de servicios que los mismos realizan y qué clase de respuestas obtienen.

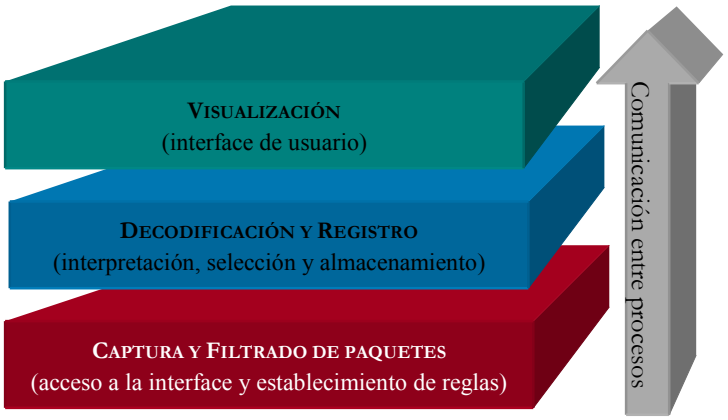
[16] Además, algunos de estos servidores utilizan herramientas adicionales para recolectar información de los usuarios conectados a los clientes y combinan esa información con bases de datos que reúnen características personales de los mismos.

CAPÍTULO III

HTTPSniffer: LA HERRAMIENTA DE MONITOREO

ESTRUCTURA DE LA HERRAMIENTA DE MONITOREO

Para comprender más fácilmente la estructura de una herramienta de monitoreo de tráfico (o como se lo llamó anteriormente *monitor de red*), se plantea un *modelo en capas*. Cada una de dichas capas es independiente de las demás en cuanto a su implementación. Sin embargo, esta división del software; introduce la necesidad del "pipeline" de procesamiento (se utilizan buffers intermedios que permiten que la salida de un programa se transforme en los datos de entrada para el programa que se encuentra en la capa superior).



La *captura de paquetes* se refiere a la implementación de los métodos que permiten acceder a la información que circula por la red. El *filtrado* de paquetes se refiere al establecimiento de reglas que permiten seleccionar cual es la información que se va a recolectar de la red (por ejemplo: aquellos mensajes pertenecientes al protocolo HTTP).

La etapa de *decodificación y registro* implica la interpretación de los paquetes capturados para poder seleccionarlos y clasificarlos de acuerdo al objetivo de la herramienta implementada. En esta etapa, también es necesario realizar un proceso de traducción ("parsing") que permita expresar la información recolectada de una manera más legible y amigable. Por otra parte requiere el establecimiento de un mecanismo de registro que permita almacenar toda la información capturada para su posterior utilización en el proceso de visualización.

Finalmente, la etapa de *visualización* es la que implementa la interface de usuario de los datos capturados, accediendo a la base de datos de información y realizando cálculos estadísticos que se visualizan en forma gráfica.

COMUNICACIÓN ENTRE MÓDULOS Y PLATAFORMA DE LA HERRAMIENTA

Este proyecto se concentró especialmente en los métodos utilizados para la captura y filtrado de paquetes, y para ello, el primer paso fue la especificación de la plataforma seleccionada: sistemas Unix o basados en Unix (sistemas *Linux*). Esta selección se fundamentó, principalmente, por su performance, confiabilidad y alta disponibilidad de herramientas que dichas plataformas soportan. La flexibilidad de estos sistemas es aún mayor si se considera que el código fuente de algunos de ellos es abierto (como es el caso de Linux). Por otra parte, las herramientas de monitoreo, requieren determinadas características que estos sistemas pueden brindar (manejo de threads, manejo de tareas en forma automática mediante scripts,…).

El control de procesos provisto por los sistemas basados en Unix incluye la creación de nuevos procesos, la ejecución de programas y la terminación de procesos. Contando solamente con lo recién mencionado, la única manera de que los procesos intercambien información es transfiriéndola a través de archivos abiertos mediante las *llamadas al sistema* fork o exec, o a través del *sistema de archivos* (file system). En esta implementación se utilizó la segunda forma de comunicación (accediendo a los archivos mediante funciones del sistema de archivos) [LIB-ESA] [LIB-ATUPFS] [LIB-TDUS] [LIB-APUE].

La forma de comunicación entre las capas de la herramienta se observa en el gráfico de la siguiente página, cuyo significado es el siguiente:

- Etapa de Captura y Filtrado:

  1) Se accede al medio a través de la interface de red, para realizar la captura de paquetes.

  2) Esa captura se registra en un archivo con formato de caracteres hexadecimales.

- Etapa de Decodificación y Registro:

  3) Se obtiene la información del archivo de caracteres hexadecimales para traducirlos.

  4) Se traduce el contenido hexadecimal a un archivo de texto plano.

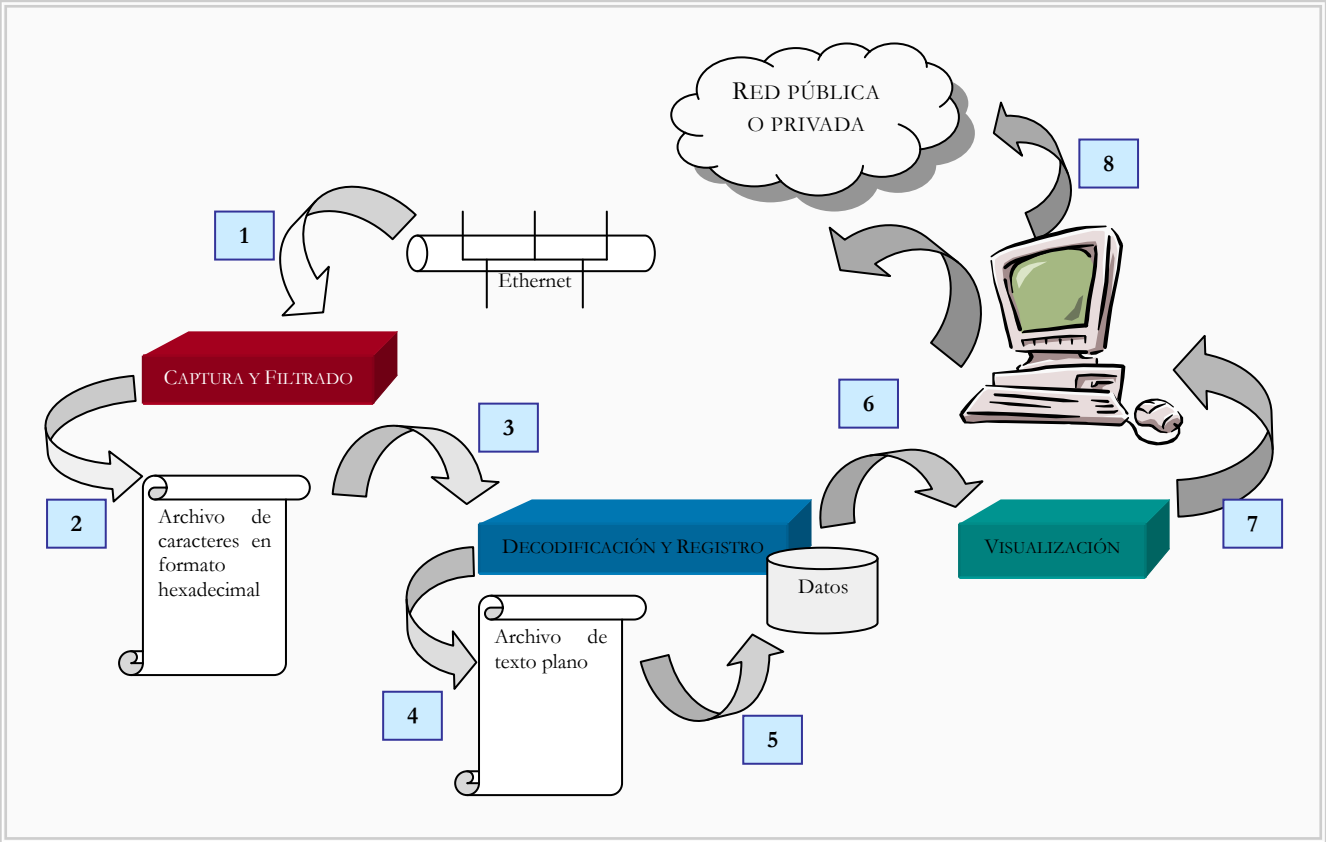
  5) Se toman los datos del archivo de texto plano y se formatean para ser almacenados en la fuente de datos.

- Etapa de Visualización:

  6) Se accede a los datos de la fuente para visualizar la información en forma gráfica en un sitio Web.

  7) Se publica el sitio que contiene la información de la captura de datos.

8) Este sitio puede ser accedido desde cualquier punto de la red, ya sea la pública o la privada (en el primer caso se debe tener especial cuidado con detalles de seguridad[17]).



La otra forma de transferencia de datos mencionada, involucra las llamadas al sistema *fork* y *exec*.

> *Con fork se pueden* ***crear*** *nuevos programas, y con las funciones exec se pueden* ***iniciar*** *nuevos programas.*

Cuando se dispara un *fork*, no hay garantía de quién se ejecuta primero (si el proceso "padre" o el proceso "hijo"), esto depende del algoritmo de planificación del kernel. Una alternativa puede involucrar la función *sleep* en el proceso padre, que permite detenerlo en el tiempo y darle la posibilidad de que se ejecute el hijo, pero tampoco asegura que así suceda.

[17] Para obtener mayor información al respecto, ver más adelante en este informe "Etapa de Visualización".

Existen dos maneras de utilizar la función *fork*:

- Cuando un proceso desea duplicarse a sí mismo: entonces cada uno (el padre y el hijo) pueden ejecutar diferentes secciones del código al mismo tiempo.

- Cuando un proceso quiere ejecutar un programa diferente: entonces el hijo hace un *exec* justo después del llamado a la función *fork*.

La función *vfork* tiene la misma secuencia de invocación y el mismo valor de retorno que fork. La intensión de *vfork* es crear un nuevo proceso que tenga como propósito ejecutar un *exec* a otro programa. *vfork* crea un nuevo proceso, de la misma manera que fork, sin copiar completamente el espacio de direcciones del padre en el hijo, debido a que el hijo no desea referenciar el espacio de direcciones del padre. Esta optimización provee una ganancia en la eficiencia sobre algunas implementaciones de memoria virtual programada en UNIX.

A pesar de que *fork* se utiliza para crear nuevos procesos, no es suficiente por si mismo para crear un nuevo proceso que ejecute un programa diferente. Para realizar esta tarea se necesita la función *exec*, con la cual el programa es completamente reemplazado por el nuevo, y este nuevo programa comienza su ejecución en su función "main". El identificador de proceso no cambia a través de un llamado a *exec*, debido a que no se crea un nuevo proceso (es por esa razón que se debe ejecutar una llamada a *fork*), se reemplaza el proceso actual (texto, datos, heap y pila) con datos de un nuevo programa del disco.

Existen seis funciones diferentes para la ejecución de *exec*, que se distinguen por lo siguiente:

- Recepción de un camino completo o de un nombre de archivo como parámetro.

- Pasaje de la lista de argumentos.

- Pasaje de la lista de entorno al siguiente programa.

La rutina de librería `system` es un ejemplo de la utilización de *fork*+*exec*:

`system(`*`comando`*`)`

que toma un string como parámetro y le pasa el string al shell como entrada (por ejemplo, al programa `/bin/sh`) para su ejecución, como si ese string hubiese sido un comando tipeado desde una terminal.

El siguiente es un ejemplo testeado para ser implementado con los diferentes procesos que involucra la herramienta de monitoreo de tráfico desarrollada:

PROCESO PADRE

```
#include <stdio.h>
#include <sys/types.h>

int main ()
{pid_t pid;

 if ( (pid = vfork()) == 0)
   execl ("./hijo.o","" ,"" );

 if ( (pid = vfork()) == 0)
   execl ("./hermano.o","" ,"" );

 /* padre */
 for (;;)
 printf ("Estoy en el padre\n");
 exit(0);
}
```

PROCESO HIJO

```
#include <stdio.h>

main ()
{for (;;)
 printf ("Estoy en el hijo\n");
 exit(0);
}
```

PROCESO HERMANO

```
#include <stdio.h>

main ()
{for (;;)
 printf ("Estoy en el hermano\n");
 exit(0);
}
```

La ejecución del programa generado con el *proceso padre*, desencadena la ejecución de los tres programas en forma simultánea y aleatoria, con diferentes códigos. Se imprimirá en forma alternada (dentro de un bucle infinito):

```
Estoy en el padre
Estoy en el hermano
Estoy en el hijo
```

Como ya fue mencionado, en esta herramienta se utilizó el mecanismo de archivos provisto por el sistema operativo para la comunicación entre procesos. Sin embargo, la

vinculación de herramientas que conforman la etapa de captura y decodificación de trafico; para ser visualizado en la Web, podría ser optimizada mediante una forma más eficiente de comunicación entre procesos[18], como por ejemplo *pipes*, la forma más antigua de IPC en Unix, provista por casi todas las versiones de estos sistemas. Un pipe permite la transferencia de datos entre procesos de forma FIFO (first-in-first-out) y además permite la sincronización de procesos en ejecución. Su utilización es muy importante, porque su implementación permite que los procesos se comuniquen sin tener necesidad de conocer cual es el proceso que se encuentra en el otro extremo del pipe. Esta forma de comunicación aún no fue implementada en este trabajo, pero podría considerarse como una tarea futura, para mejorar el rendimiento de la aplicación.

El *pipe* permite la lectura de un extremo y la escritura del otro. Se invoca por la función `pipe`, cuyo prototipo es:

```
int pipe (int FileDes[2]);
```

Cuando esta función tiene éxito retorna `0` y dos descriptores de archivos: `FileDes[0]` y `FileDes[1]`; que representan los extremos de lectura y escritura del pipe. En otro caso retorna `1`.

Debido a que es el mecanismo más antiguo de comunicación entre procesos en Unix y es muy utilizado, se imponen límites a la cantidad de memoria que puede ocupar el pipe (la cantidad de caracteres que el mismo puede almacenar). Ese límite se setea por la constante `PIPE_BUF` en `<sys/config.h>`, el cual normalmente es 8192 bytes. Si un proceso intenta escribir en un pipe que está completamente lleno, dicho proceso es detenido por el sistema operativo hasta que el pipe esté lo suficientemente vacío. Algo similar sucede con la lectura de un pipe vacío.

IMPLEMENTACIÓN DE CADA CAPA DE LA HERRAMIENTA

ETAPA DE CAPTURA Y FILTRADO DE PAQUETES

Para la implementación de la captura de información, en esta etapa se seleccionó la herramienta TCPDump como herramienta de captura de paquetes. Luego se establecieron las reglas necesarias para lograr que la mencionada herramienta realice una captura adecuada, considerando únicamente aquellos mensajes que contienen información referente al protocolo de nivel de aplicación de interés: HTTP. Para ello se utilizó un mecanismo de filtrado que seleccione únicamente aquellos paquetes que contienen el puerto bien conocido de HTTP (puerto 80), en algunos mensajes en la parte origen y en otros en la parte destino del socket.

Luego de haber investigado el código fuente de la herramienta TCPDump y las técnicas involucradas para su funcionamiento, y de haber realizado una comparación entre muestras de tráfico capturadas por la misma y muestras capturadas por otras herramientas comerciales importantes; tales como: LANAnalyzer y el software de Fluke[HTTP-FNTT] (ambas corriendo en una PC con un sistema operativo Windows), se seleccionó TCPDump como herramienta de captura de tráfico, en gran parte motivada por su eficiencia, flexibilidad y aplicación en diferentes plataformas.

[18] Actividad normalmente llamada *IPC* (Interprocess Communication).

A modo de ejemplo, se muestra a continuación uno de los paquetes capturados por la herramienta, y que es almacenado en un archivo para que pueda ser leído por el proceso que implementa la segunda etapa (*Decodificación y Registro*):

```
954943083.486126 163.10.20.15.33491 > 63.211.145.10.80: P
     1981418331:1981418657(326) ack 2365851654 win 8760 (DF)
```

Header IP
Header TCP
Header HTTP

```
4500 016e ba3f 4000 ff06 3853 a30a 140f
3fd3 910a 82d3 0050 761a 0b5b 8d04 0806
5018 2238 a787 0000 4745 5420 2f64 6f63
732f 2048 5454 502f 312e 300d 0a52 6566
6572 6572 3a20 6874 7470 3a2f 2f77 7777
2e61 7061 6368 652e 6f72 672f 6874 7470
642e 6874 6d6c 0d0a 436f 6e6e 6563 7469
6f6e 3a20 4b65 6570 2d41 6c69 7665 0d0a
5573 6572 2d41 6765 6e74 3a20 4d6f 7a69
6c6c 612f 342e 3531 205b 656e 5d20 2858
3131 3b20 493b 2053 756e 4f53 2035 2e37
2073 756e 3475 290d 0a48 6f73 743a 2077
7777 2e61 7061 6368 652e 6f72 670d 0a41
6363 6570 743a 2069 6d61 6765 2f67 6966
2c20 696d 6167 652f 782d 7862 6974 6d61
702c 2069 6d61 6765 2f6a 7065 672c 2069
6d61 6765 2f70 6a70 6567 2c20 696d 6167
652f 706e 672c 202a 2f2a 0d0a 4163 6365
7074 2d45 6e63 6f64 696e 673a 2067 7a69
700d 0a41 6363 6570 742d 4c61 6e67 7561
6765 3a20 656e 0d0a 4163 6365 7074 2d43
6861 7273 6574 3a20 6973 6f2d 3838 3539
2d31 2c2a 2c75 7466 2d38 0d0a 0d0a
```

La primera línea del paquete almacenado son datos generados por el TCPDump. El significado de cada uno de los campos es el siguiente:

- `954943083.486126` es el timestamp, que refleja el primer momento en el cual el kernel del sistema operativo observó el paquete. En este caso, se utilizaron opciones en la invocación del TCPDump que permitieron que este campo se imprima no formateado. La función del sistema `strftime`[19], permite formatear la fecha y hora. Por ejemplo: el siguiente script realizado en AWK

  ```
  gawk ´{print 954943083.486126
        strftime(" %Y-%m-%d %H:%M:%S", 954943083.486126)}´
  ```

  permite expresar el timestamp con el siguiente formato:

  ```
  aaaa-mm-dd hh:mm:ss
  2000-04-05 16:58:03
  ```

  Sin embargo, simplemente codificando la cadena de impresión, se puede expresar en forma diferente:

  ```
  gawk ´{print 954943083.486126
        strftime(" %A %d %B %Y %H:%M:%S", 954943083.486126)}´
  ```

[19] También disponible en PHP, que fue el lenguaje seleccionado para programar la interface de usuario, lo que permite la flexibilidad de almacenar ese instante de tiempo sin formato y visualizarlo de la manera que se desee.

```
Wednesday 05 April 2000 16:58:03
```

- `163.10.20.15.33491` y `63.211.145.10.80` son las direcciones IP y los puertos origen y destino, respectivamente.

- La letra `P` es un flag, que en este caso significa PUSH, pero podría ser cualquier combinación de S (SYN), F (FIN), P (PUSH) o R (RST), o un simple ".", que significa ausencia de flags.

- El siguiente campo significa el número de secuencia de datos, respetando la notación: *primero:ultimo(nbytes)*, lo cual significa: "desde el número de secuencia *primero* en adelante, sin incluir *ultimo*, que son *nbytes* de datos de usuario".

- `ack` es el número de secuencia del siguiente dato esperado en la otra dirección de esta conexión. Indica que se está utilizando la técnica de *piggy-backing*.

- `win` (windows) es el espacio en el buffer del receptor, expresado en número de bytes, que está disponible en la otra dirección de la conexión.

- `(DF)` significa que el paquete fue marcado con el flag de IP *no fragmentar*.

El resto de las líneas forman un paquete capturado, expresado en códigos hexadecimales.

Además de los datos de interés (campos de los headers IP, TCP y HTTP), al momento de realizar la captura de información, se debe tener especial cuidado con las consecuencias que puede provocar la pérdida de paquetes durante el mencionado proceso. Por ejemplo entre algunos de los inconvenientes que pueden surgir se detallan los siguientes:

- La pérdida puede ser de cualquier tipo de paquetes, con lo cual; si se pierden paquetes conteniendo apertura o cierre de conexiones TCP (flags SYN o FIN), si luego se desea identificar conexiones se puede tornar verdaderamente problemático.

- Si la pérdida se produce en paquetes que contienen información de solicitudes o respuestas HTTP, se corre el riesgo de ignorar datos de un mensaje.

Como conclusión: la implementación del proceso de captura de información es crítica.

ETAPA DE DECODIFICACIÓN Y REGISTRO

A partir de los resultados obtenidos por varias muestras de mensajes generadas por el proceso que realiza la correcta invocación a TCPDump, se realizó un exhaustivo análisis de la información presentada y se establecieron reglas para determinar, entre otros, cuáles son los paquetes de utilidad y cuáles serían candidatos a ser descartados. Por ejemplo: se considera que no es de interés para el objetivo de esta herramienta almacenar el contenido de los recursos direccionados (código HTML de páginas Web, representación

binaria de una imagen,…). Con esta justificación, y debido al límite impuesto para el tamaño máximo de las unidades de transferencia en una red (MTU o *Maximun Transfer Unit*), los recursos deben ser transmitidos en más de un datagrama IP, con lo cual, aquellos paquetes capturados que no aporten información de header adicional (en cualquiera de los tres niveles de protocolos en estudio: IP, TCP o HTTP), serán descartados y no serán considerados por las otras capas de la herramienta de monitoreo.

Una vez que fue aislada toda la información de interés, se comenzó con el proceso de parsing, con el objetivo de representar la información de manera más legible y amigable. Para la implementación de este proceso se seleccionó C [LIB-LPC][LIB-CPCC++] como lenguaje de programación[20], fundamentado por las siguientes razones:

- eficiencia y potente manejo de memoria y archivos,
- facilidad de integración con el proceso de captura de paquetes
- facilidades para el registro de información, y
- disponibilidad de una amplia variedad de fuentes de datos accesibles desde el mismo[21].

Previo a la selección de C como lenguaje de programación de esta etapa, y a la decisión de implementar un nuevo parser (sin la reutilización de código existente) se realizó un estudio de aquellas herramientas disponibles que podrían satisfacer el objetivo planteado, seleccionando, por popularidad y disponibilidad de código fuente, una de ellas llamada HTTPDump[22], codificada en el lenguaje Perl[LIB-LP]. Luego de haber realizado una profunda interpretación del código fuente de dicha herramienta (para lo que se necesitó estudiar el lenguaje en cuestión), y de haber testeado su funcionamiento, se llegó a la conclusión de que la misma no cumplía satisfactoriamente los objetivos planteados en este proyecto, más aún, tenía falencias en la traducción que descartaban paquetes de interés (por ejemplo: sólo consideraba paquetes relacionados con la versión 1.0 del protocolo HTTP).

Finalizado el proceso de parsing de los datos, se continúa con el proceso de registro de información, y requiere de los siguientes pasos: discriminar el final del header HTTP y descartar la información restante en cada uno de los paquetes capturados, traducir los paquetes y registrar los datos en formato de texto plano y obtener los datos del archivo de texto plano y registrarlos en una fuente de datos.

- *Discriminar el final del header HTTP y descartar la información restante en cada uno de los paquetes capturados*: se refiere a considerar sólo los metadatos de HTTP y no los datos reales. Cabe destacar la dificultad que presenta el

[20] Para el desarrollo del código se utilizó *Kdevelop (1998, 1999, 2000) Versión 1.1*, una herramienta de libre distribución, bajo los términos de GNU General Public License. Permite la creación de proyectos en C y C++ bajo UNIX, con interfaces específicas para la plataforma mencionada.

[21] Pensando en la etapa siguiente: donde se requiere almacenar la información para su posterior visualización.

[22] Ver en el "Capítulo IV" el funcionamiento de esta herramienta en detalle.

hecho de que el header HTTP no posea una longitud fija[23]. Esto implica que se deben recorrer todos los paquetes para "recortarlos" en la finalización del header HTTP, indicado por la cadena hexadecimal *0d0a0d0a*, que representa el salto de línea. Luego se descarta el resto de la información hasta el próximo inicio de paquete.

- *Traducir los paquetes y registrar los datos en formato de texto plano*: para realizar esta tarea no se vuelve a recorrer el archivo de la salida generada por el TCPDump, sino que, a medida que se van encontrando todos los datos de interés, se parsean y almacenan en otro archivo de salida, que será utilizado luego como entrada para el proceso que genera la fuente de datos. Un fragmento de ese archivo, a modo de ejemplo, es el siguiente:

```
TIMESTAMP:
962440433.335492
HEADER IP:
20 163.10.5.159 163.10.10.6
HEADER TCP:
1060 80 20 PSH ACK
HEADER HTTP:
GET / HTTP/1.0 Referer: http://ada.info.unlp.edu.ar/academica/calenacad.html
Host: www.linti.unlp.edu.ar Accept: application/msword, */* Accept-Language: es-
ar Accept-Encoding: gzip, deflate User-Agent: Mozilla/4.0 (compatible; MSIE 4.01;
Windows 98) Connection: Keep-Alive
```

A continuación se muestra el significado de cada unos de estos campos:

TIMESTAMP: común para los tres headers, debido a que se obtiene de la línea generada por el TCPDump. Es el instante en el que se capturó el paquete. Se almacena no formateado, para brindar la flexibilidad de darle el formato deseado en la interface de usuario.
HEADER IP: los campos de este header que se registran son los siguientes: longitud (habitualmente 20 bytes), dirección IP origen y dirección IP destino.
HEADER TCP: los campos de este header que se registran son los siguientes: puerto origen, puerto destino, longitud (habitualmente 20 bytes) y el conjunto de flags que participan del paquete de conexión TCP.
HEADER HTTP: se registra todo el header HTTP.

- *Obtener los datos del archivo de texto plano y registrarlos en una fuente de datos*: el objetivo de este proceso es el almacenamiento en una fuente de datos, para la fácil recuperación de la información desde la aplicación final. Para realizar esta tarea se compararon los productos Postgres SQL y mySQL, se seleccionó este último por rendimiento y simplicidad de utilización[24].

[23] Ver en la parte "*Campos del header del mensaje*" de este informe la especificación del header HTTP respecto de este tema.

[24] Estudios realizados en la Universidad de Rio Grande del Sur, Brasil, demostraron que frente a un elevado número de conexiones el funcionamiento de *PostgresSQL* no es suficientemente eficiente.

ETAPA DE VISUALIZACIÓN

La etapa de visualización se refiere al proceso que recolecta los datos de la base y los presenta al usuario. Es una interface basada en Web; lo cual presenta la ventaja de que puede ser direccionada desde cualquier sitio de la red. Cabe destacar que si la herramienta se accederá desde la red de dominio público (Internet), se deben tener en cuenta diferentes aspectos de seguridad, como por ejemplo un login y password de usuario y un servidor de Web seguro, por ejemplo: brindando el servicio de SSL (*Secure Socket Layer*) [HTTP-SSL]. Esto permitirá (además de la flexibilidad de estar monitoreando el tráfico HTTP desde cualquier punto de Internet) el acceso restringido y seguro[25] a información tan delicada como la que maneja la herramienta.

PHP [HTTP-PHP] y HotMetal Pro 6.0 [HTTP-HMP] fueron, respectivamente, el lenguaje de programación y la herramienta de diseño seleccionados para implementar el conjunto de páginas Web de la interface. La selección de PHP se fundamentó, principalmente, por la necesidad de un lenguaje que permita la creación de páginas dinámicas, y se complementó por su flexibilidad, el soporte para múltiples fuentes de datos, la posibilidad de incluir librerías gráficas y la sólida combinación que forma en conjunción con Apache [HTTP-APACHE] (el servidor de Web seleccionado[26]) y Linux [HTTP-LINUX] (la plataforma de la herramienta).

Respecto de la herramienta de diseño, a pesar de disponer de diferentes productos que permiten trabajar con el código PHP (PHPed, Script Worx 5,…), características como: aspectos WYSIWYG de la herramienta, contribución a la legibilidad del código HTML, soporte para lenguajes de script, visualización del diseño y las ayudas brindadas, provocaron que la elección se enfocara en HotMetal Pro 6.0 [TRAB-THCPWIU].

Como último aspecto a detallar, se encuentra la organización de la información en la presentación al usuario. La característica que más se destaca, es la discriminación de los datos de acuerdo a tres de las capas de la pila de protocolos TCP/IP: se presenta información del protocolo IP (capa de Internet), del protocolo TCP (capa de transporte) y del protocolo HTTP (capa de aplicación). De cada uno de los niveles se extracta información relevante y se presentan gráficos estadísticos, acompañados de una explicación y un fundamento teórico.

[25] Por medio de la encriptación implementada por el protocolo *SSL*.

[26] Se seleccionó *Apache* como servidor de Web por ser el más robusto y confiable servidor que corra sobre plataformas UNIX. Por otro lado, el mismo permite el manejo de una amplia variedad de técnicas incluídas en esta herramienta en desarrollo, o que posiblemente hubiesen sido candidatas a ser utilizadas en algún momento del proceso de diseño de la herramienta: PHP, ASP, SSL, mySQL,… 60% del mercado de Internet lo utiliza. Por el período de 3 años fue considerado como el servidor de Web del año por Datamation (http://www.soci.niu.edu/~huguelet/TLOFCRP/).

INSTALACIÓN DE LA HERRAMIENTA

Para la instalación de la herramienta se debe disponer de los siguientes requerimientos:

- Una PC con una plataforma Linux (cualquiera sea la distribución): la herramienta fue compilada en forma satisfactoria en este tipo de sistemas. Se debe tener instalado el compilador gcc.

- Herramienta de Captura de tráfico: TCPDump instalado.

- Motor de base de datos: mySQL instalado. No se debe olvidar setear una password para el usuario root de mySQL. También se debe recordar setear la siguiente variable de entorno (suponiendo una instalación con los parámetros por defecto, como por ejemplo: directorio de instalación `/usr/local/mysql`):

  ```
  # LD_LIBRARY_PATH=/usr/local/mysql/lib/mysql
  ```

  Una vez que el motor de datos se encuentra instalado, se debe crear la base, para lo cual se necesita ejecutar el script de creación:

  ```
  # mysql < crear_base_datos.sql
  ```

- Servidor de Web: Apache instalado con el módulo de PHP. Eventualmente, para implementar una solución segura, también se requeriría la instalación del SSL.

Se deben copiar en un directorio el proceso que representa la etapa de "Captura y Filtrado de paquetes" (llamado *llamada_tcpdump*) y el proceso que representa la etapa de "Decodificación y Registro" (llamado *decodif_reg*). Si se desea compilar el código fuente de ambos fragmentos de código, se debe tener en cuenta lo siguiente:

- Suministrar las siguientes opciones adicionales al compilador:

  ```
  -I/usr/local/mysql/include/mysql
  ```

  ```
  -L/usr/local/mysql/lib/mysql
  ```

- Enlazar con las siguientes bibliotecas adicionales:

  ```
  -lmysqlclient
  ```

  ```
  -lm
  ```

Finalmente, se debe publicar el sitio Web que permite visualizar los datos, en el directorio de publicación de páginas de Apache (en la mayoría de las instalaciones es `/usr/local/apache/htdocs`).

FORMA DE EJECUCIÓN DE LA HERRAMIENTA

Para que la herramienta genere la información necesaria para ser visualizada a través de la interface basada en web, solamente basta la ejecución ambos procesos que la integran (*llamada_tcpdump* y *decodif_reg*), el primero para que realice la captura de tráfico necesaria y el segundo para obtener los datos finales. Sin embargo, es de interés lograr una ejecución de la misma cada determinado intervalo de tiempo, de manera de capturar nuevos datos y actualizar las estadísticas. Para ello se debe ejecutar en forma periódica un script, que permita rotar los archivos de registro para no perder la información histórica, limpiar el contenido de la base de datos (que puede ser restaurado en base a la información anterior) para que contenga solamente datos actuales y ejecutar ambos procesos.

Una ejecución más "avanzada" de la herramienta; podría radicar en su ejecución en segundo plano y compartir recursos (ya sea a través de pipes o del sistema de archivos) que sean accesibles en forma sincronizada, mediante algún mecanismo de programación concurrente; como por ejemplo semáforos, monitores,…. Esto debería permitir la captura constante de información y el refresco de la interface en forma automática. Pero sin embargo, no se debe perder de vista el exponencial crecimiento del tamaño de los recursos almacenados, que no solamente podrían dominar el rendimiento de la herramienta, sino que también podrían acabar con recursos del sistema tales como disco y memoria.

CAPÍTULO IV
OTRAS HERRAMIENTAS DE MONITOREO

En este proyecto se evaluaron diferentes herramientas que desempeñan funcionalidades específicas, pero todas ellas capaces de intervenir en alguna de las etapas involucradas en el procedimiento de monitoreo del tráfico de Web.

Algunas de las herramientas analizadas únicamente realizan recolección de paquetes o análisis de tráfico, pero en el momento en que fueron seleccionadas se consideró especialmente que se pueda disponer de su código fuente (herramientas *open-source*), para poder planificar una extensión o modificación de las mismas, o interactuar con su código.

ANÁLISIS DE TCPDump

El programa `tcpdump`[HTTP-TPR] fue escrito por Van Jacobson, Craig Leres y Steven McCanne, todos ellos miembros del Laboratorio Lawrence Berkeley, Universidad de California, Berkeley. La versión que se utilizó para este proyecto fue la 3.4[27].

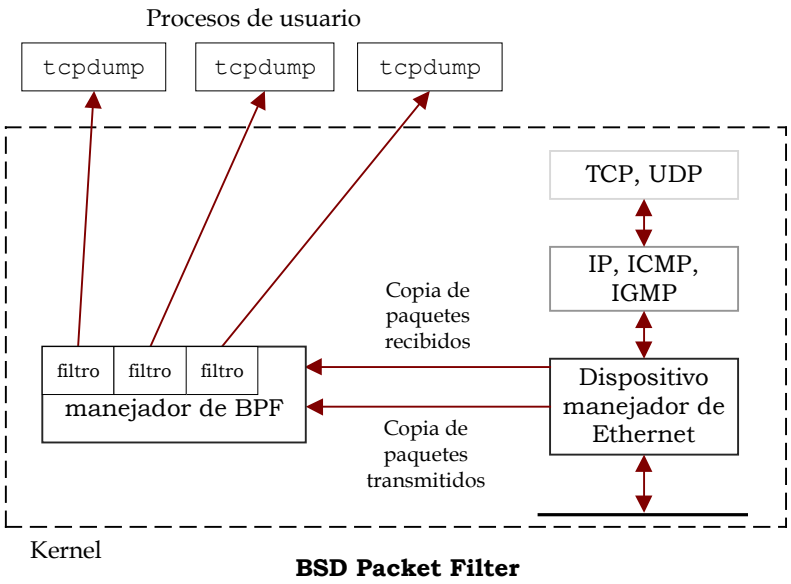
El sistema operativo de base debe permitir que la interface pueda colocarse en modo promiscuo para que el usuario pueda realizar la captura de tramas. El soporte para `tcpdump` es provisto (o puede ser agregado) por los siguientes sistemas UNIX y basados en UNIX: 4.4BSD, BSD/386, SunOS, Ultrix, HP-UX, Solaris y Linux.

BSD PACKET FILTER

Los actuales kernels basados en BSD, proveen el *BSD Packet Filter* (BPF)[TRAB-TBPF], que es un método utilizado por `tcpdump` para capturar y filtrar paquetes desde una interface de red que se encuentre en modo promiscuo.

BPF pone el manejador del dispositivo Ethernet en modo promiscuo y luego recibe de dicho manejador una copia de cada paquete recibido y transmitido. Esos paquetes se pasan por un filtro especificado por el usuario, para que sólo sean derivados para procesamiento aquellos paquetes que el usuario considera de interés. Diferentes procesos pueden estar monitoreando una interface dada, y cada uno de ellos puede establecer sus propios filtros. En la siguiente figura se muestran las características de BPF cuando se utiliza una red Ethernet. Se pueden observar diferentes instancias de `tcpdump` monitoreando la misma red Ethernet.

[27] Actualmente se encuentra disponible la versión 3.5 pero aún no está completamente testeado su funcionamiento.



BSD Packet Filter

HTTPDump: SCRIPT DE PARSING

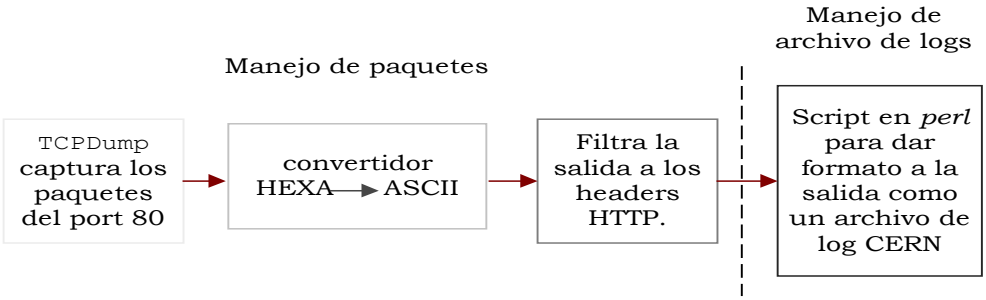
`HTTPdump` es una herramienta que permite aislar el tráfico de HTTP del tráfico de otras aplicaciones de red. Para recolectar los paquetes se basa en el programa `tcpdump`. Para maximizar su utilidad, la salida se produce en el formato de archivos de log *common format*[HTTP-WHCLF]. Esto permite compatibilidad con herramientas existentes de análisis de logs.

tcpdump debe ser ejecutado de manera que solamente entregue paquetes con sockets que contengan el port 80 (el port estándar para el tráfico HTTP). Luego se utilizan una serie de scripts codificados en perl que realizan las siguientes tareas:

1) convierten la salida obtenida de tcpdump en texto ASCII,

2) filtran los paquetes de salida a aquellos que solamente contengan headers HTTP, y finalmente

3) generan el registro de salida en un archivo de log.



La salida de un programa es almacenada en un buffer intermedio para que pueda ser utilizada como entrada del programa siguiente, tal que todos puedan ejecutar como procesos concurrentes separados en la misma máquina.

La implementación de este software que se puede obtener posee dos piezas de código: http-filter.pl y a_t_tcp2cern.pl[HTTP-WTAT].

LA NOVEDAD: SNORT

Snort [HTTP-SNORT] es una de las más nuevas herramientas relacionadas con el tema de monitoreo de tráfico, que tiene además aplicaciones de seguridad.

Snort es un sistema de detección de intrusiones, capaz de realizar; en tiempo real, análisis de tráfico y logging de paquetes sobre redes IP. Puede realizar análisis de protocolos, búsqueda y coincidencias de contenidos y ser utilizada para detectar una variedad de ataques y pruebas, tales como escaneos de puertos, ataques CGI, pruebas de SMB,… Es en realidad, una versión completa de una herramienta de monitoreo de tráfico, que tiene como objetivo colaborar con la seguridad de la red.

También posee capacidades de alertas en tiempo real, incorporando mecanismos de alertas para syslog, un archivo de usuario específico o ventanas de mensajes para clientes de Windows utilizando clientes SMB de Samba.

DEMÁS HERRAMIENTAS ANALIZADAS

TCPDPRIV

`tcpdpriv` es un programa que permite eliminar información confidencial de los paquetes recolectados por una interface de red (por ejemplo de archivos creados mediante la opción `-w` del programa `tcpdump`).

La distribución encontrada puede ser instalada bajo los sistemas SunOS, Solaris y FreeBSD (por ejemplo Linux)[HTTP-PECIT]. Posee requerimientos modestos, únicamente la librería de captura de paquetes `libcap`[HTTP-LNRG].

TCP-REDUCE

tcp-reduce es un conjunto de scripts realizados en el shell Bourne para reducir las salidas de tcpdump sólo a líneas que presenten las conexiones TCP. Únicamente se tienen en cuenta los paquetes que contienen los flags TCP SYN, FIN y RST.

La distribución encontrada está compuesta por las siguientes piezas de código[HTTP-SSTC]:

- tcp-reduce: toma como argumento un archivo resultado de tcpdump e imprime un resumen ordenado.

- tcp-conn: un script realizado en awk que realiza la mayor parte del trabajo.

- tcp-summary: un script realizado en awk que genera un resumen de todas las conexiones TCP producidas por tcp-reduce.

Los requerimientos que posee son simplemente algunas utilidades de UNIX como sed[28], el comando sort y el pseudolenguaje awk[LIB-TAPL].

LIBRERÍA: LIBCAP

Es una librería portable para la captura de paquetes. Provee gran portabilidad a los programas de captura que la utilizan, y facilita la codificación de los mismos[HTTP-LNRG].

TCPTRACE

Es una herramienta que se puede clasificar como de recolección de paquetes. Permite realizar el análisis de sesiones TCP, con el agregado de un programa graficador de las salidas.

ARGUS

Esta herramienta permite realizar la captura y el análisis de paquetes. Es muy poderosa para el monitoreo de redes IP. Está complementada con programas de análisis de paquetes basados en el paradigma de packetfilter[HTTP-ARESD].

ETHERFIND

Es un recolector de paquetes destinado para la plataforma SunOS. Está disponible únicamente en código binario[29].

IPTRACE

Este software es la alternativa a tcpdump implementada para AIX, y posee características similares al mismo. Está disponible únicamente en código binario[30].

NETSNOOP

Es el programa recolector de paquetes que está contenido en la distribución de Irix. Está disponible únicamente en código binario[31].

---

28 Para una información más detallada de *sed*, ver la página manual en los sistemas UNIX.

29 El contacto para este producto es la empresa Sun.

30 El contacto para este producto es la empresa IBM.

31 El contacto para este producto es la empresa SGI.

SNOOP

Es el programa recolector de paquetes que está contenido en la distribución de Solaris. Está disponible únicamente en código binario[32].

32 El contacto para este producto es la empresa Sun.

APÉNDICE A
CÓDIGO DE LA HERRAMIENTA IMPLEMENTADA

MAIN.C

```
/***************************************************************************
                          main.c  -  description
                             -------------------
    copyright            : (C) 2000 by Marisa Andrea Malvaso
    email                : mmalvaso@info.unlp.edu.ar

***************************************************************************/

#include "headers.h"
#include <stdio.h>

int main(int argc, char *argv[])
{
      FILE *registro;
      char *arch_texto;

      /*Invoco a la funcion que transforma los datos capturados por
      el TCPDump y los almacena en un archivo de texto plano. Toma el
      archivo resultado en la variable "registro".*/
      registro=(FILE *)parser();

      /*Invoco al proceso que toma los datos del archivo plano y los
      almacena en la base de datos.*/
      store(registro);
}
```

PARSER.C

```
/***************************************************************************
                          parser.c  -  description
                         -------------------------
    copyright            : (C) 2000 by Marisa Andrea Malvaso
    email                : mmalvaso@info.unlp.edu.ar

***************************************************************************/


#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "headers.h"

#define LINEA_LENGTH 200

/* Declaracion de encabezados de funciones. */
void parsear(char *paq_parseado, char *paq_crudo);
void parsear_headerip(HeaderIP *hp_parseado, char *hp);
void parsear_headertcp(HeaderTCP *hp_parseado, char *hp);
Boolean esValido(char *paquete_sinheader);
Void generar_salida(HeaderIP head_ip, HeaderTCP head_tcp, char
      *head_http, FILE *archivo, char timestamp[16]);

FILE *parser()
{
  FILE* entrada;
  FILE* salida;

  char *linea;
  char *paquete="";
  char *paquete_sin_header="";
  HeaderIP header_ip_parseado;
  HeaderTCP header_tcp_parseado;
  char *header_ip="";
  char *header_tcp="";
  char *paquete_parseado="";
  char timestamp[16];

  char cadenaFin[]="0d0a0d0a";

  Boolean finPaquete = FALSE;
  int i, j, long_header_ip, long_header_tcp;

  entrada = fopen("./archivos/archivo_hexadecimal","r");
  salida = fopen("./archivos/archivo_texto_plano","w");

  paquete=malloc(4000 * sizeof(char));
  paquete_parseado=malloc(4000 * sizeof(char));
  paquete_sin_header=malloc(4000 * sizeof(char));
  header_tcp=malloc(41 * sizeof(char));
```

```
header_ip=malloc(41 * sizeof(char));

linea = malloc(LINEA_LENGTH);

fgets(linea, LINEA_LENGTH, entrada);
strncpy(timestamp,&linea[0],16);

fgets(linea, LINEA_LENGTH, entrada);

while (!feof(entrada))
{
     while ((!feof(entrada)) && (finPaquete==FALSE))
     {
          if (strchr(linea, '>') != NULL) finPaquete=TRUE;
          else
          {
               for (i=j= 0; linea[i]!='\0'; i++)
                 if (!isspace(linea[i])) linea[j++]=linea[i];
               linea[j]='\0';
               strcat(paquete, linea);
               if ( strstr(linea, cadenaFin) != NULL)
                 while ((!feof(entrada)) && (finPaquete==FALSE))
                 {
                    fgets(linea, LINEA_LENGTH, entrada);
                    if (strchr(linea, '>') != NULL) finPaquete=TRUE;
                  }
                else
                  fgets(linea, LINEA_LENGTH, entrada);
           }
      }
      if (finPaquete==TRUE)
      {
          strncpy(header_ip,&paquete[0],40);
          parsear_headerip(&header_ip_parseado,header_ip);
          long_header_ip=(header_ip_parseado.longitud)*2;
          strncpy(header_tcp,&paquete[long_header_ip],40);
          parsear_headertcp(&header_tcp_parseado,header_tcp);
          long_header_tcp=(header_tcp_parseado.longitud)*2;
          strcpy(paquete_sin_header,&paquete[long_header_ip+long_hea
                 der_tcp]);

          if (esValido(paquete_sin_header)==TRUE)
          {
              parsear(paquete_parseado, paquete_sin_header);
              generar_salida(header_ip_parseado,header_tcp_parseado,
                       paquete_parseado, salida, timestamp);
           };

           paquete[0]='\0';
           paquete_parseado[0]='\0';
           paquete_sin_header[0]='\0';
           header_ip[0]='\0';
           header_tcp[0]='\0';

           finPaquete=FALSE;
           strncpy(timestamp,&linea[0],16);
       };
      if (!feof(entrada)) fgets(linea, LINEA_LENGTH, entrada);
};
```

```
  fclose(entrada);
  fclose(salida);

  free(paquete);
  free(linea);
  free(paquete_parseado);
  free(paquete_sin_header);
  free(header_tcp);
  free(header_ip);

  return (salida);

}; //fin parser


void parsear(char *paq_parseado, char *paq_crudo)
{
  int i,j;
  char codigo[3]="00";
  char letra;

  i=j=0;
  while (paq_crudo[i]!='\0')
  {
        codigo[0]=paq_crudo[i++];
        codigo[1]=paq_crudo[i++];

        letra=strtol(codigo, NULL, 16);
        if (letra > 127 || letra < 33) letra=' ';
        paq_parseado[j++]=letra;
   };
  paq_parseado[j++]='\0';
 };


void parsear_headerip(HeaderIP *hp_parseado, char *hp)
{
  char codigo[4]="0x00";
  char longi[2];
  int i,posicion;
  longi[0]=hp[1];
  longi[1]='\0';

  (*hp_parseado).longitud=(atoi(longi)) * 4;
  posicion=24;
  for (i=0;i<=3;i++)
  {
     codigo[2]=hp[posicion++];
     codigo[3]=hp[posicion++];
     (*hp_parseado).dir_origen[i]=strtol(codigo, NULL, 16);
   };
  for (i=0;i<=3;i++)
  {
     codigo[2]=hp[posicion++];
     codigo[3]=hp[posicion++];
     (*hp_parseado).dir_destino[i]=strtol(codigo, NULL, 16);
   };
 };
```

```
void parsear_headertcp(HeaderTCP *hp_parseado, char *hp)
{
  char codigo[6]="0x0000";
  char longi[2];
  char flags[4]="0x00";
  int i, posicion, flag_hexa;

  posicion=0;
  for (i=2;i<=5;i++) codigo[i]=hp[posicion++];
  (*hp_parseado).puerto_origen=strtol(codigo, NULL, 16);
  for (i=2;i<=5;i++) codigo[i]=hp[posicion++];
  (*hp_parseado).puerto_destino=strtol(codigo, NULL, 16);

  longi[0]=hp[24];
  longi[1]='\0';
  (*hp_parseado).longitud=(atoi(longi)) * 4;

  flags[2]=hp[26];
  flags[3]=hp[27];
  flag_hexa=strtol(flags, NULL, 16);
  for (i=0;i<=5;i++) (*hp_parseado).flag[i]="   ";
  if ((flag_hexa | 0xfe)==0xff) (*hp_parseado).flag[0]="FIN";
  if ((flag_hexa | 0xfd)==0xff) (*hp_parseado).flag[1]="SYN";
  if ((flag_hexa | 0xfb)==0xff) (*hp_parseado).flag[2]="RST";
  if ((flag_hexa | 0xf7)==0xff) (*hp_parseado).flag[3]="PSH";
  if ((flag_hexa | 0xef)==0xff) (*hp_parseado).flag[4]="ACK";
  if ((flag_hexa | 0xdf)==0xff) (*hp_parseado).flag[5]="URG";
 };


Boolean esValido(char *paquete_sinheader)
{
  char *header[]=
      {"4F5054494F4E53","474554","48454144","504F5354","505554","4445
      4C455445","5452414345","48545450"};
  Boolean resultado=FALSE;
  char *inicio_paquete="";
  int i;

  inicio_paquete=malloc(16 * sizeof(char));
  strncpy(inicio_paquete,&paquete_sinheader[0],15);
  for (i = 0; i<=7; i++)
      if (strstr(inicio_paquete,header[i])!=NULL) resultado=TRUE;

  free(inicio_paquete);

  return resultado;
};


void generar_salida(HeaderIP head_ip, HeaderTCP head_tcp,
                    char *head_http, FILE *archivo, char timestamp[16])
{
  int count, i;
  char *tmp="";

  tmp=malloc(15 * sizeof(char));
  strncpy(tmp,&timestamp[0],16);
  count=fprintf(archivo,"%s","TIMESTAMP: ");
```

```
 count=fputc('\n', archivo);
 count=fprintf(archivo,"%s",tmp);
 free(tmp);
 count=fputc('\n', archivo);
 count=fprintf(archivo,"%s","HEADER IP:");
 count=fputc('\n', archivo);
 count=fprintf(archivo,"%d",head_ip.longitud);
 count=fputc(' ', archivo);
 count=fprintf(archivo,"%d",head_ip.dir_origen[0]);
 count=fputc('.', archivo);
 count=fprintf(archivo,"%d",head_ip.dir_origen[1]);
 count=fputc('.', archivo);
 count=fprintf(archivo,"%d",head_ip.dir_origen[2]);
 count=fputc('.', archivo);
 count=fprintf(archivo,"%d",head_ip.dir_origen[3]);
 count=fputc(' ', archivo);

 count=fprintf(archivo,"%d",head_ip.dir_destino[0]);
 count=fputc('.', archivo);
 count=fprintf(archivo,"%d",head_ip.dir_destino[1]);
 count=fputc('.', archivo);
 count=fprintf(archivo,"%d",head_ip.dir_destino[2]);
 count=fputc('.', archivo);
 count=fprintf(archivo,"%d",head_ip.dir_destino[3]);
 count=fputc('\n', archivo);
 count=fprintf(archivo,"%s","HEADER TCP:");
 count=fputc('\n', archivo);
 count=fprintf(archivo,"%d",head_tcp.puerto_origen);
 count=fputc(' ', archivo);
 count=fprintf(archivo,"%d",head_tcp.puerto_destino);
 count=fputc(' ', archivo);
 count=fprintf(archivo,"%d",head_tcp.longitud);
 count=fputc(' ', archivo);
 for (i=0;i<=5;i++)
    if ((head_tcp.flag[i]) != "   ")
    {
        count=fprintf(archivo,"%s",head_tcp.flag[i]);
        count=fputc(' ', archivo);
     };
 count=fputc('\n', archivo);
 count=fprintf(archivo,"%s","HEADER HTTP:");
 count=fputc('\n', archivo);
 count=fputs(head_http, archivo);
 count=fputc('\n', archivo);
 count=fputc('\n', archivo);
}
```

```
/***************************************************************************
                          store.c  -  description
                             -------------------
    begin                : Sun Sep 24 2000
    copyright            : (C) 2000 by Marisa
    email                : mmalvaso@info.unlp.edu.ar

***************************************************************************/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include </usr/local/mysql/include/mysql/mysql.h>
#include "headers.h"

#define LINEA_LENGTH 4000

int store(FILE *entrada)
{
  MYSQL *connection, mysql;
  int state, i;
  char *linea="";
  char *cadena="";
  char *consulta="";
  char *campo;
  char *parte_dir;
  char *direccion_o;
  char *direccion_d;
  char *flags;

  HeaderIP hip;
  HeaderTCP htcp;
  char *timestamp;

  mysql_init(&mysql);
  connection=mysql_real_connect(&mysql,"localhost","root","",
                                "registro",0,0,0);
  if (connection == NULL)
  {
    printf(mysql_error(&mysql));
    return 1;
   };

  entrada = fopen("./archivos/archivo_texto_plano","r");

  linea=malloc(4000 * sizeof(char));
  cadena=malloc(4000 * sizeof(char));
  consulta=malloc(4500 * sizeof(char));
  campo=malloc(20 * sizeof(char));
  timestamp=malloc(20 * sizeof(char));
  parte_dir=malloc(3 * sizeof(char));
```

```
direccion_o=malloc(20 * sizeof(char));
direccion_d=malloc(20 * sizeof(char));
flags=malloc(24 * sizeof(char));

while (!feof(entrada))
{
   fgets(linea, LINEA_LENGTH, entrada);
   linea[strlen(linea)-1]='\0';

   if (strstr(linea,"TIMESTAMP:") != NULL)
   {
       fgets(linea, LINEA_LENGTH, entrada);
       linea[strlen(linea)-1]='\0';

       strcpy(timestamp,linea);
    }
    else
    if (strstr(linea,"HEADER IP:") != NULL)
    {
       fgets(linea, LINEA_LENGTH, entrada);
       linea[strlen(linea)-1]='\0';
       campo=strtok(linea, " ");

       hip.longitud=strtol(campo,NULL,10);
       campo=strtok(NULL, " ");
       strcpy(direccion_o,campo);

       campo=strtok(NULL, " ");
       strcpy(direccion_d,campo);

       sprintf(consulta,
                        "INSERT INTO headerip
                        (timestamp,longitud,dir_origen,dir_destino)
                        VALUES(%s','%u','%s','%s')",
               timestamp,hip.longitud,direccion_o,direccion_d);
       state=mysql_query(connection, consulta);
       if (state != 0)
       {
           printf(mysql_error(&mysql));
           return 1;
        };
     }
    else
    if (strstr(linea,"HEADER TCP:") != NULL)
    {
       fgets(linea, LINEA_LENGTH, entrada);
       linea[strlen(linea)-1]='\0';
       campo=strtok(linea, " ");
       htcp.puerto_origen=strtol(campo,NULL,10);
       campo=strtok(NULL, " ");
       htcp.puerto_destino=strtol(campo,NULL,10);
       campo=strtok(NULL, " ");
       htcp.longitud=strtol(campo,NULL,10);
       campo=strtok(NULL, " ");
       for (i=0;i<=5;i++) htcp.flag[i]="   ";
       i=0;
       strcpy(flags," ");
```

```
        while (campo != NULL)
        {
              htcp.flag[i]=campo;
              strcat(flags,htcp.flag[i++]);
              strcat(flags," ");
              campo=strtok(NULL, " ");
         };
        sprintf(consulta,
            "INSERT INTO headertcp
            (timestamp,puerto_origen,puerto_destino,longitud,flags)
            VALUES('%s','%u','%u','%u','%s')",
                 timestamp,htcp.puerto_origen,htcp.puerto_destino,
                 htcp.longitud,flags);
        state=mysql_query(connection, consulta);
        if (state != 0)
        {
            printf(mysql_error(&mysql));
            return 1;
         };
    }
    else
    if (strstr(linea,"HEADER HTTP:") != NULL)
    {
       fgets(linea, LINEA_LENGTH, entrada);
       campo=strtok(linea, "'");
       strcpy(cadena,campo);
       campo=strtok(NULL, "'");
       while (campo != NULL)
       {
           strcat(cadena,campo);
           campo=strtok(NULL, "'");
        };
       sprintf(consulta,
                      "INSERT INTO headerhttp
                      (timestamp,contenido)
                      VALUES ('%s','%s')",timestamp,cadena);
       state=mysql_query(connection, consulta);
       if (state != 0)
       {
           printf(mysql_error(&mysql));
           return 1;
        };
     };
  };

 free(linea);
 free(cadena);
 free(consulta);
 free(campo);
 free(timestamp);
 free(parte_dir);
 free(direccion_o);
 free(direccion_d);
 free(flags);
 fclose(entrada);

 mysql_close(connection);
 return EXIT_SUCCESS;
}
```

HEADERS.H

```
/***************************************************************************
                          headers.h  -  description
                             -------------------
    begin                : Sun Oct 22 2000
    copyright            : (C) 2000 by Marisa
    email                : mmalvaso@info.unlp.edu.ar

***************************************************************************/

#include <stdio.h>

typedef enum {FALSE=0, TRUE=1} Boolean;

typedef struct {
                int longitud;
                int tiposervicio;
                int protocolo;
                int dir_origen[4];
                int dir_destino[4];
                } HeaderIP;

typedef struct {
                int puerto_origen;
                int puerto_destino;
                int longitud;
                char *flag[6];
                } HeaderTCP;

FILE *parser();

int store(FILE *archivo);
```

MYSQL_CREATE_DB.SQL

```
# mysql_create_db.sql

# Crea la Base de Datos "registro"
CREATE DATABASE registro;

# Se conecta a la Base de Datos recien creada
CONNECT registro;

# Crea la tabla correspondiente al header IP
CREATE TABLE headerip (
                      timestamp VARCHAR(16),
                      longitud TINYINT UNSIGNED,
                      dir_origen VARCHAR(15),
                      dir_destino VARCHAR(15)
                      );

# Crea la tabla correspondiente al header TCP
CREATE TABLE headertcp (
                      timestamp VARCHAR(16),
                      puerto_origen SMALLINT UNSIGNED,
                      puerto_destino SMALLINT UNSIGNED,
                      longitud TINYINT UNSIGNED,
                      flags VARCHAR(24)
                      );

# Crea la tabla correspondiente al header HTTP
CREATE TABLE headerhttp (
                      timestamp VARCHAR(16),
                      contenido MEDIUMBLOB
                      );

# Crea una tabla que se utiliza para estadisticas en general
CREATE TABLE estadistica (
                      nombre CHAR(50)
                      );

# finalizado!
```

APÉNDICE C
REFERENCIAS

REFERENCIAS A SITIOS DE INTERNET

[HTTP-APACHE] "The Apache Software Fundation", http://www.apache.org/

[HTTP-ARESD] Para obtener el paquete `Argus`, "Argus Real Estate Software Download": http://www.argussoftware.com/download/download.htm

[HTTP-CDT] Center for Democracy and Technology, "CDT Privacy Issues Page",

http://www.cdt.org/previousheads/dataprivacy.shtml

"Privacytimes.com", http://www.privacytimes.com

[HTTP-CLT] En la siguiente dirección están disponibles ejemplos de logs en clientes:

ftp://cs-ftp.bu.edu/techreports/, C. R. Cunha, A. Bestavros y M. E. Crovella, "Client Log Traces"

[HTTP-FNTT] "Fluke Network Test Tools, Multimeters, Electronic Test Equipment, Instruments and Calibration", http://www.fluke.com/

[HTTP-HMP] "SoftQuad: Welcome to hotmetalpro.com", http://www.hotmetalpro.com/

[HTTP-IDS] "Internet Domain Survey", http://www.isc.org/dsview.cgi?domainsurvey/index.html

[HTTP-IHNN] ISVS HindSite para Netscape Navigator: http://www.isysdev.com/products/hindsite.htm

[HTTP-LINUX] "The Linux Home Page at Linux Online", http://www.linux.org/

[HTTP-LNRG] Para obtener la librería `libcap`, "LBNL's Network Research Group": http://ee.lbl.gov/

[HTTP-MIME] "MIME", http://webdesk.com/definitions/mime.html

[HTTP-NCSAM] "NCSA Mosaic", http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html

[HTTP-PECIT] Para obtener el paquete `TCPdpriv`, "Program for Eliminating Confidential Information from Traces": http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html

[HTTP-PHP] "PHP: Hypertext Preprosessor", http://www.php.net/

[HTTP-SNORT] "The Lightweight Network Intrusion Detection System", http://www.snort.org

[HTTP-SSL] "SSL Definition", http://www.mcps.k12.md.us/departments/pupilservices/studentservicelearning/definition.htm

"Internet Security Workshop", http://as400.rochester.ibm.com/tstudio/secure1/workshop/ssldef.htm

[HTTP-SSTC] Para obtener el paquete `tcp-reduce`, "Scripts for Summarizing TCP Connections": http://ita.ee.lbl.gov/html/contrib/tcp-reduce.html

[HTTP-SV3] "SNMP Version 3 (SNMPv3)", http://www.ibr.cs.tu-bs.de/ietf/snmpv3/

[HTTP-TPR] "TCPDUMP public repository", http://www.tcpdump.org

[HTTP-TSW] "The SimpleWeb", http://wwwsnmp.cs.utwente.nl/

[HTTP-WHCLF] W.W.W. Consortium, "W3c httpd common log format", http://www.w3.org/pub/WWW/Daemon/User/Config/Logging.html

[HTTP-WTAT] Para obtenerel paquete de `HTTPDump`, "World Wide Web (WWW) Traffic Analysis Tools": http://www.cs.vt.edu/~nrg/WWWTrafficTools.html

REFERENCIAS A LIBROS

[LIB-APUE] W. Richard Stevens, "Advanced Programming in the UNIX Environment", 1993.

[LIB-ATUPFS] Ronald J. Leach, "Advanced Topics in UNIX. Processes, files, and systems", 1994.

[LIB-CPCC++] H.M. Deitel y P.J. Deitel, "Cómo programar en C/C++", 1995.

[LIB-ESA] Æleen Frisch, "Essential System Administration", 1991

[LIB-H&C] J. December, "HTML & CGI", edit. UNLEASHED, 1995

[LIB-LP] Randal L. Schwartz y Tom Christiansen, "Learning Perl", 2nd edition, 1997.

[LIB-LPC] Brian W. Kernighan y Dennis M. Ritchie, "El Lenguaje de Programación C", 1992.

[LIB-TAPL] A. Aho, B. Kernighan, P. Weinberger, "The AWK Programming Language", 1998

[LIB-TDUS] Maurice J. Bach, "The Design of the UNIX Operating System", 1990.

[LIB-TIIV3] W. Richard Stevens, "TCP/IP Illustrated, Volume 3. TCP for transactions, HTTP, NNTP, and the UNIX Domain Protocols", 1996.

[LIB-TIV1TP] Douglas E. Comer, "Internetworking with TCP/IP, Volume 1. Principles, Protocols, and Architecture", 1995.

REFERENCIAS A *REQUESTS FOR COMMENTS* (RFCs)

[RFC-1630] T. Berners-Lee, "Universal Resource Identifiers in WWW, A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", RFC 1630, CERN, Junio de 1994.

[RFC-1737] K. Sollins, L. Masinter, "Functional Requirements for Uniform Resource Names", RFC 1737, MIT/LCS, Xerox Corporation, Diciembre de 1994.

[RFC-1738] T. Berners-Lee, L. Masinter, M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, CERN, Xerox PARC, University of Minnesota, Diciembre de 1994

[RFC-1945] T. Berners-Lee, R. Fielding, H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0", RFC 1945, Mayo de 1996

[RFC-2068] R. Fielding, J. Gettys, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol –HTTP/1.1", RFC 2068, Enero de 1997

[RFC-2076] J. Palme, "Common Internet Message Headers", RFC 2076, Febrero de 1997

[RFC-793] Information Sciences Institute, University of Southern California, "Transmission Control Protocol, TCP", Darpa Internet Program, Protocol Specification, RFC 793, Septiembre de 1981

REFERENCIAS A TRABAJOS DE INVESTIGACIÓN

[TRAB-ACOAB] A.C. Marisa A. Malvaso y Lic. Miguel A. Luengo, "Alternativas de Caching para optimizar el ancho de banda". Trabajo aprobado y presentado en la 5ta Jornada de Investigación, Asociación de Universidades del Grupo Montevideo, septiembre de 1997, San Bernardino, Paraguay.

[TRAB-CSMRA] A.C. Marisa A. Malvaso y Lic. Javier Díaz, "Complementando las solicitudes con un Monitoreo de Red Automatizado". Trabajo aprobado en WICC'99 (Workshop de Investigadores en Ciencias de la Computación), Universidad Nacional de San Juan, abril de 1999, San Juan, Argentina.

[TRAB-TBPF] S. McCanne, Van Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture", Diciembre de 1992.

[TRAB-THCPWIU] A.C. Marisa A. Malvaso, Lic. Claudia Banchoff, Lic. Javier Díaz, "¿Todas las herramientas de creación de páginas Web son igualmente útiles". Trabajo aprobado en ICIEY2K (VI Congreso Internacional de Ingeniería Informática), Universidad Nacional de Buenos Aires, abril de 2000, Buenos Aires, Argentina. http://www.linti.unlp.edu.ar