



ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

“ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA PARA EVITAR ATAQUES AL PROTOCOLO ARP EN REDES DE ÁREA LOCAL”

TESIS DE GRADO

Previa a la obtención del Título de:

**INGENIERO EN COMPUTACIÓN ESPECIALIZACIÓN
SISTEMAS TECNOLÓGICOS**

Presentada por:

**ANDRE PAOLA ORTEGA ALBÁN
XAVIER ENRIQUE MARCOS RODRÍGUEZ**

Guayaquil - Ecuador

2008

AGRADECIMIENTO

*A Dios, todopoderoso y eterno.
A nuestras familias, y especialmente
a la ingeniera Cristina Abad,
por su apoyo, confianza y orientación,
en todo momento.*

DEDICATORIA

*A nuestros padres,
A Titov y a Carla.*

TRIBUNAL DE GRADO

PRESIDENTE




Ing. Holger Cevallos Ulloa

DIRECTOR DE TESIS



M. Sc. Cristina Abad

MIEMBROS PRINCIPALES



M. Sc. Guido Caicedo



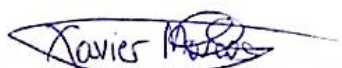
M. Sc. Rebeca Estrada

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado, nos corresponde exclusivamente; y el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral"

(Reglamento de exámenes y títulos profesionales de la ESPOL)


Andre Paola Ortega Albán


Xavier Enrique Marcos Rodríguez

RESUMEN

En este trabajo se presenta la propuesta de un nuevo mecanismo para evitar ataques al protocolo ARP en redes de área local. El documento está dividido en 7 capítulos que comprenden la identificación del problema, la fase de análisis, diseño, implementación y los resultados obtenidos al probar nuestro sistema contra distintos ataques en una red de área local conformada por cuatro computadoras.

En el Capítulo 1 se describe el protocolo ARP y su funcionalidad. Además se identifica el problema al cual es susceptible y se plantean los objetivos del presente trabajo. También se explica el funcionamiento del protocolo que sustituye a ARP en IPv6.

En el Capítulo 2 se realiza una breve descripción acerca de las técnicas existentes de detección, mitigación y prevención contra los ataques de envenenamiento a la caché ARP, y se profundiza más acerca del mismo; se describen los métodos de envenenamiento y se detallan las características que hacen del protocolo, un protocolo inseguro.

Para que una solución ante los ataques contra el protocolo ARP sea la adecuada, debe cumplir con un conjunto de requisitos, los cuales se detallan en el Capítulo 3, además se lista lo necesario para que nuestra solución

pueda llevarse a cabo. Se detallan las razones por las cuales trabajamos con el hardware y firmware del ruteador y el software de nuestro servidor.

En el Capítulo 4 se explica todo lo referente al diseño de nuestra solución. Se describen los dos programas fundamentales que ejecuta el servidor para mantener su caché ARP actualizada y para responder las consultas ARP que realizan las computadoras de la red local. Se dan a conocer conceptos básicos sobre el protocolo DHCP, pues a través de estos mensajes se modifica la caché ARP del servidor. Además se presentan las reglas de ebttables del ruteador, configuradas para bloquear las respuestas ARP y cambiar el destino de las consultas ARP.

Una descripción detallada de los pasos seguidos para la ejecución de los programas que corren en el servidor y lo realizado en el ruteador para la ejecución de las reglas de ebttables, se muestra en el Capítulo 5. También se detalla lo realizado para la creación e instalación en el ruteador del paquete que reenvía los mensajes DHCP hacia el servidor.

En el Capítulo 6 se realiza una descripción de los escenarios de ataque con las pruebas de funcionamiento a los que nuestro esquema fue expuesto. Se detallan los resultados obtenidos al probar nuestra solución en los distintos escenarios de ataque ARP y además se muestran gráficos comparativos

entre el rendimiento del ruteador con las configuraciones realizadas y sin éstas. También se explica el trabajo que se debería realizar para que la solución funcione en redes más grandes.

Finalmente se detallan las conclusiones del trabajo de investigación así como también las recomendaciones para la exitosa ejecución de nuestro sistema.

ÍNDICE GENERAL

<i>AGRADECIMIENTO</i>	<i>ii</i>
<i>DEDICATORIA</i>	<i>iii</i>
<i>TRIBUNAL DE GRADO</i>	<i>iv</i>
<i>DECLARACIÓN EXPRESA</i>	<i>v</i>
<i>RESUMEN</i>	<i>vi</i>
<i>ÍNDICE GENERAL</i>	<i>ix</i>
<i>ABREVIATURAS</i>	<i>xii</i>
<i>ÍNDICE DE GRÁFICOS</i>	<i>xiii</i>
<i>ÍNDICE DE TABLAS</i>	<i>xiv</i>
<i>INTRODUCCIÓN</i>	<i>2</i>
<i>CAPÍTULO I</i>	<i>4</i>
<i>1 PLANTEAMIENTO DEL PROBLEMA Y ANÁLISIS CONTEXTUAL</i>	<i>4</i>
<i>1.1 Definición del problema</i>	<i>4</i>
<i>1.2 Objetivos del proyecto de tesis</i>	<i>5</i>
<i>1.3 Definición del protocolo ARP</i>	<i>6</i>
1.3.1 Formato de una trama ARP	<i>6</i>
1.3.2 Generación de una trama de consulta ARP	<i>7</i>
1.3.3 Recepción de una trama ARP	<i>8</i>
<i>1.4 Protocolo NDP en IPV6</i>	<i>10</i>
<i>CAPÍTULO II</i>	<i>12</i>
<i>2 TÉCNICAS DE DETECCIÓN Y PREVENCIÓN ACTUALES CONTRA LOS ATAQUES DE ENVENENAMIENTO A LA CACHÉ ARP</i>	<i>12</i>
<i>2.1 Definición de envenenamiento a la caché ARP</i>	<i>12</i>
<i>2.2 Métodos de envenenamiento a la caché ARP</i>	<i>14</i>
<i>2.3 Detección de ataques ARP</i>	<i>18</i>
<i>2.4 Mitigación de ataques ARP</i>	<i>20</i>
<i>2.5 Prevención contra ataques ARP</i>	<i>22</i>
<i>CAPÍTULO III</i>	<i>26</i>
<i>3 ANÁLISIS</i>	<i>26</i>
<i>3.1 Requerimientos de la solución</i>	<i>26</i>

3.2	Estudio de las alternativas y selección de la más apropiada.....	28
3.3	Análisis del software elegido para el servidor.....	29
3.4	Análisis del hardware escogido	30
3.5	Análisis del firmware escogido	31
CAPÍTULO IV.....		34
4	DISEÑO.....	34
4.1	Diseño general.....	34
4.2	Diseño del servidor	36
4.2.1	Diseño lógico.....	36
4.2.2	Programa que mantiene actualizada la caché ARP del sistema.....	38
4.2.3	Programa que escucha el tráfico ARP y responde las consultas ARP.....	51
4.3	Diseño de reglas y programa del switch	56
4.3.1	Envío de tráfico DHCP de la red hacia el servidor.....	56
4.3.2	Redireccionamiento y bloqueo de las tramas ARP.....	59
CAPÍTULO V.....		63
5	IMPLEMENTACIÓN.....	63
5.1	Implementación del servidor	63
5.1.1	Configuración del servidor.....	64
5.2	Configuraciones en el switch WRTSL54GS.....	66
5.2.1	Creación, instalación y ejecución del programa que reenvía las tramas DHCP del servidor DHCP al servidor de nuestro esquema	67
5.2.2	Configuración de reglas de ebttables para el bloqueo de respuestas ARP y redireccionamiento de consultas ARP.....	74
5.3	Limitaciones de la implementación.....	77
CAPÍTULO VI.....		79
6	PRUEBAS Y ANÁLISIS DE RESULTADOS OBTENIDOS.....	79
6.1	Escenarios de ataque	79
6.2	Pruebas de funcionamiento.....	83
6.3	Impacto del rendimiento	93
6.4	Trabajo futuro	109
CONCLUSIONES Y RECOMENDACIONES		111
APÉNDICES.....		116
A APÉNDICE A: ARCHIVOS DE CÓDIGO FUENTE DE LOS PROGRAMAS DEL SERVIDOR.....		117
A.1	Programa update_arp_cache.....	117
A.2	Programa send_arp_reply	127

<i>B</i>	<i>APÉNDICE B: ARCHIVO DE CÓDIGO FUENTE DEL PAQUETE DEL RUTEADOR.....</i>	<i>136</i>
	<i>B.1 Programa ForwardDhcpFrames</i>	<i>136</i>
<i>C</i>	<i>APÉNDICE C: ARCHIVOS MAKEFILE</i>	<i>139</i>
	<i>C.1 Makefile para la compilación de ForwardDhcpFrames en el entorno de compilación cruzada.....</i>	<i>139</i>
	<i>C.2 Makefile para la compilación de ForwardDhcpFrames en el ruteador.....</i>	<i>139</i>
<i>D</i>	<i>APÉNDICE D: ARCHIVOS DE FIREWALL.....</i>	<i>141</i>
	<i>D.1 Archivo firewall del ruteador con reglas para guardar tramas DHCP (/etc/init.d/S35firewall)</i>	<i>141</i>
	<i>D.2 Archivo firewall.user del ruteador con reglas de ebttables para redireccionar consultas y bloquear respuestas ARP (/etc/firewall.user)</i>	<i>143</i>
<i>E</i>	<i>APÉNDICE E: VALORES OBTENIDOS DURANTE PRUEBAS DE RENDIMIENTO</i>	<i>143</i>
	<i>E.1 Número de paquetes perdidos.....</i>	<i>143</i>
	<i>E.2 Valores de tiempo de recepción de respuestas ARP</i>	<i>144</i>
	<i>E.3 Valores de tiempo de recepción de respuestas eco ICMP</i>	<i>145</i>

ABREVIATURAS

ARP Address Resolution Protocol (Protocolo de Resolución de Direcciones)

IP Internet Protocol (Protocolo de Internet)

LAN Local Area Network (Red de Área Local)

MAC Medium Access Control address (dirección de Control de Acceso al Medio)

NDP Neighbor Discovery Protocol (Protocolo de Descubrimiento de Vecinos)

SNDP Protocolo Seguro de Descubrimiento de Vecinos

VLAN Virtual LAN (Red de Área Local Virtual)

BPF Berkeley Packet Filter (Filtro de paquetes de Berkeley)

ÍNDICE DE GRÁFICOS

<i>Figura 1.3.1-1. Formato de trama ARP con cabecera Ethernet.</i>	7
<i>Figura 1.3.3-1 Funcionamiento del protocolo ARP.</i>	10
<i>Figura 2.1-1. Envenenamiento de la caché ARP.</i>	14
<i>Figura 3.4-1 Linksys WRTSL54GS.</i>	31
<i>Figura 4.1-1. Ejemplo de funcionamiento de nuestro mecanismo para evitar ataques al protocolo ARP.</i>	36
<i>Figura 4.2.2.1-1. Diagrama de tiempo del intercambio de mensajes entre el cliente y el servidor DHCP durante la asignación de la configuración de red.</i>	42
<i>Figura 4.2.2.1-2. Diagrama de tiempo del intercambio de mensajes entre el cliente y el servidor DHCP durante la reasignación de la dirección de red usada.</i>	45
<i>Previamente.</i>	45
<i>Figura 4.2.2.3-1. Formato de un mensaje DHCP.</i>	51
<i>Figura 4.2.3.2-1. Formato de consulta ARP y respuesta ARP que recibe y envía el servidor.</i>	56
<i>Figura 4.3.2-1. Diagrama del ruteador WRTSL54GS con OpenWrt.</i>	60
<i>Figura 6.2-1 Cain & Abel escaneando los computadores de la red.</i>	85
<i>Figura 6.2-2 Tramas ARP vistas en el servidor.</i>	87
<i>Figura 6.2-3 Ejecución del programa que responde las consultas ARP.</i>	88
<i>Figura 6.2-4 Caché ARP del servidor.</i>	89
<i>Figura 6.2-5 Selección de los computadores a atacar.</i>	90
<i>Figura 6.2-6 Cain envenenando, pero sin resultados.</i>	91
<i>Figura 6.2-7 Cain cuando logra realizar el ataque de hombre en el medio.</i>	92
<i>Figura 6.2-8 Caché ARP del servidor.</i>	93
<i>Figura 6.3-1 Porcentaje de paquetes perdidos.</i>	95
<i>Figura 6.3-2 Medición de tiempo de recepción de respuestas ARP.</i>	97
<i>Figura 6.3-3 Distribución de tráfico total – Día 1.</i>	103
<i>Figura 6.3-4 Distribución de tráfico total – Día 2.</i>	104

<i>Figura 6.3-5 Distribución de tráfico total – Día 3.</i>	<i>105</i>
<i>Figura 6.3-6 Distribución de tráfico total – Día 4.</i>	<i>106</i>
<i>Figura 6.3-7 Distribución de tráfico total – Día 5.</i>	<i>107</i>
<i>Figura 6.3-8 Medición de tiempo de transmisión de paquetes.</i>	<i>108</i>

ÍNDICE DE TABLAS

<i>Tabla 4.2.2.1-1 Mensajes DHCP [27].</i>	43
<i>Tabla 6.3-1 Tiempos de respuesta ARP.</i>	99

INTRODUCCIÓN

En una red de computadoras, la información se envía en paquetes de datos. El computador debe conocer la dirección física de la máquina destino en la red de área local, para que la capa de enlace pueda proceder con la transmisión de tramas (nombre que reciben los paquetes de capa de enlace de datos). Para ello se hace uso del Protocolo de Resolución de Direcciones (ARP). La función de ARP es averiguar la dirección física (MAC) o de capa 2, asociada a una dirección de protocolo (generalmente IP), o de capa 3. El equipo que desea enviar los datos difunde un pedido ARP a la dirección de broadcast para que la consulta llegue a todas las computadoras que conforman la red de área local. El computador que hace la consulta ARP espera que la máquina con la dirección IP correspondiente devuelva su respuesta con su dirección MAC. Adicionalmente, se optimiza el protocolo con la introducción de cachés de ARP. En estas cachés se almacena la correspondencia entre direcciones IP y las direcciones MAC del segmento de red, de forma que antes de enviar una consulta ARP se trata de resolver la dirección buscándola en las propias entradas de la caché.

El protocolo ARP trabaja bien en circunstancias regulares, pero no fue diseñado para lidiar con nodos maliciosos; de esta forma, durante el tiempo entre la transmisión de la consulta ARP y la respuesta, los datos son vulnerables a modificaciones, secuestros o redireccionamiento hacia un tercero no autorizado. Los distintos ataques que pueden ser realizados a este

protocolo, comprometen la seguridad de una red. Es así, que el objetivo de este trabajo de investigación es presentar un nuevo esquema para asegurar ARP y combatir los problemas a los que este protocolo es susceptible, a través de una propuesta económica, eficiente y sobretodo fácil de implementar.

Nuestro diseño es un esquema centralizado compuesto por un servidor, un switch y las computadoras que conforman la red local. La función del servidor consiste en recibir todas las consultas ARP que se transmitan en la red de área local, y emitir una respuesta ARP correcta (en base a una tabla de mapeos <IP, MAC> que mantiene el servidor). El switch está configurado de manera que bloquea las respuestas ARP, excepto aquellas que provengan de nuestro servidor; además, modifica el destino de las consultas ARP, haciendo que éstas sean enviadas únicamente hacia el servidor, el cual no permite que su caché ARP sea actualizada con consultas; de esta forma las respuestas maliciosas no son transmitidas ni tampoco las consultas maliciosas modifican cachés, logrando que los ataques ARP no sean exitosos.

Paralelamente a este trabajo de investigación se probaron de manera exhaustiva distintos ataques contra ARP, los cuales nos sirvieron para comprobar si nuestra solución combate todos los tipos de problemas que este protocolo enfrenta.

Los resultados fueron prometedores: ningún ataque fue exitoso mientras nuestra solución se ejecutaba.

CAPÍTULO I.

1 PLANTEAMIENTO DEL PROBLEMA Y ANÁLISIS CONTEXTUAL

El presente capítulo describe cuál es el problema a enfrentar y detalla el funcionamiento del protocolo ARP. Adicionalmente, se exponen las diferencias entre el protocolo NDP (Neighbor Discovery Protocol) de IPv6 y el protocolo ARP de IPv4.

1.1 Definición del problema

Para que se de la comunicación entre computadores en las redes de área local (LAN, por sus siglas en inglés) es de uso frecuente el protocolo ARP (Address Resolution Protocol). ARP permite que un computador pueda obtener la dirección física (dirección MAC) de otro nodo para completar los bytes del paquete que se desea enviar con la dirección MAC del destinatario y finalmente con esta información,

poder realizar el envío.

El protocolo ARP hace uso de cachés en las cuales se guardan los mapeos entre direcciones de red (IP) y las direcciones físicas de las computadoras de la red. Antes de enviar una consulta ARP, el sistema operativo trata de resolver la dirección buscándola en las entradas de la caché ARP del sistema.

ARP funciona correctamente en condiciones normales. El problema se da cuando se hace un uso mal intencionado de este protocolo, y se decide usarlo de tal manera que se pueda cambiar la dirección física de uno mismo, o hacerle creer a otro computador que la dirección física propia es otra diferente (envenenamiento a la caché ARP), con el fin de tomar la identidad de otro nodo en la red y llevar a cabo algún tipo de ataque. Los métodos de envenenamiento a la caché ARP son explicados con mayor profundidad en el Capítulo 2.

1.2 Objetivos del proyecto de tesis

Dado el problema que actualmente afecta a nuestras redes de área local, la presente tesis presenta una implementación, a manera de “prueba de concepto”, de una posible solución al problema de envenenamiento ARP o falsificación ARP, con la finalidad de demostrar que se puede llegar a tener bajo cierto control el problema del envenenamiento ARP haciendo uso de herramientas no tan

sofisticadas.

Se espera que nuestra solución quede como ejemplo o punto de partida para otras personas que deseen tener más control sobre el tráfico de sus redes y no sepan cómo afrontar estos problemas.

1.3 Definición del protocolo ARP

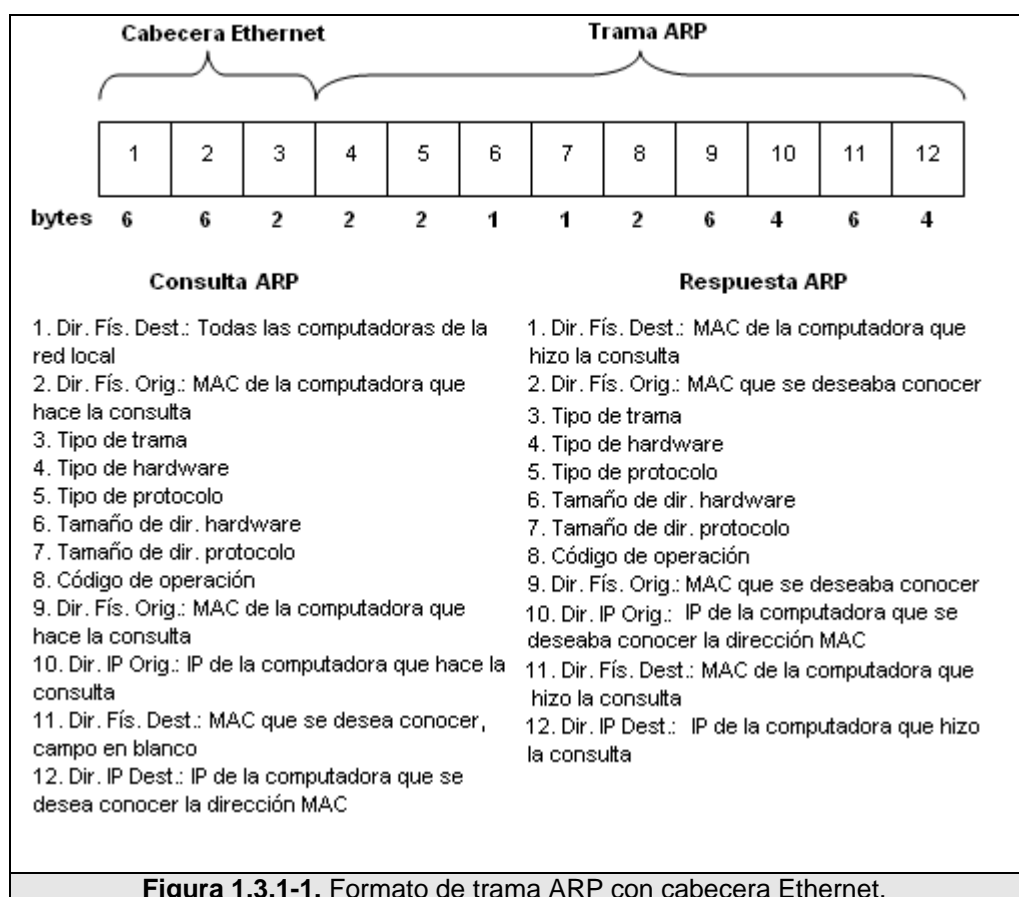
El propósito del diseño del protocolo ARP es poder obtener la dirección física (dirección MAC de 48 bits) de un computador, dada su dirección lógica (dirección IP de 32 bits). El funcionamiento de este protocolo puede ser explicado en los dos usos que tiene. El primero es cuando se desea hacer la consulta de una dirección MAC entonces se genera una trama de consulta ARP, y el segundo es cuando se recibe una trama ARP ya sea consulta o respuesta.

El protocolo se optimiza con la utilización de cachés ARP. En estas cachés se guarda la correspondencia entre direcciones IP y las direcciones físicas de los nodos de la red. Antes de enviar una consulta ARP, se trata de resolver la dirección buscándola en las entradas de la caché.

1.3.1 Formato de una trama ARP

El formato de la trama ARP consta de 2 partes [1]. La primera, que contiene información de la capa de enlace de datos (Ethernet),

mientras que la segunda parte es la parte de los datos ARP. La Figura 1.3.1-1 muestra el formato de una trama normal de consulta ARP y una trama normal de respuesta ARP con la cabecera Ethernet.



A continuación se detallan los dos usos del protocolo ARP de acuerdo al RFC 826 [1].

1.3.2 Generación de una trama de consulta ARP

Cuando se quiere enviar un paquete a un determinado computador

se necesita su dirección física. En este momento se realiza la consulta al módulo de conversión de direcciones del sistema (el cual nos retornará una dirección física dada una lógica). Si se conoce la dirección lógica del computador al cual se quiere enviar el paquete, el módulo de conversión buscará en su caché la dirección física correspondiente y si la tiene enviará el paquete sin problema a su destino. Si no la encuentra, entonces procederá a enviar la trama ARP tipo consulta a manera de broadcast a toda la red de área local, preguntando por la dirección física dada la dirección lógica.

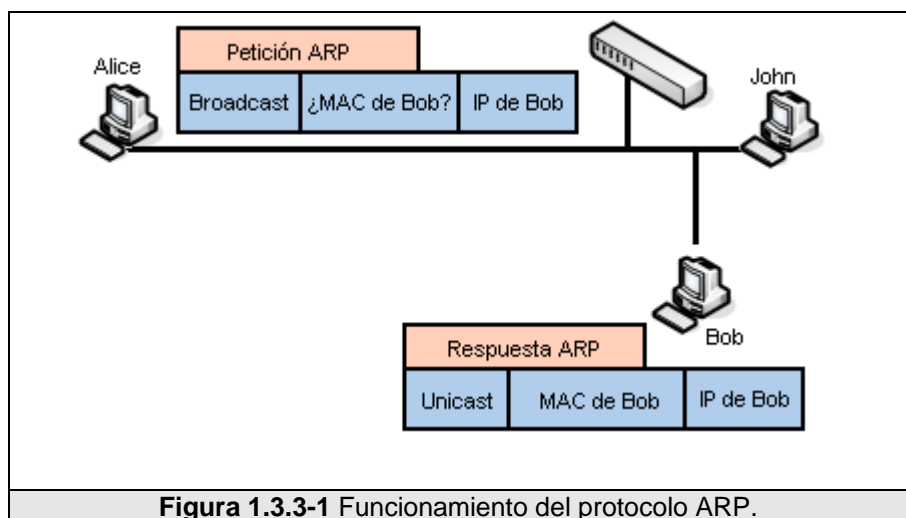
En dicha trama, se especifica si es de tipo IP, la longitud en bytes de la dirección IP, el tipo de operación de la trama que en este caso será de tipo Consulta, entre otros. También se coloca en la trama la dirección física y lógica del computador que hace la consulta, la dirección física que se desea conocer es un campo en blanco sin contenido y el campo final de la trama contiene la dirección IP o dirección lógica de la cual se desea conocer la dirección MAC.

1.3.3 Recepción de una trama ARP

Cuando una trama ARP es recibida, el módulo receptor Ethernet del sistema pasa esta trama al módulo de Resolución de Direcciones, cuyo funcionamiento se detalla a continuación.

Cuando recibe una trama ARP verifica que la computadora que la

recepta trabaje con el mismo protocolo, si es así se setea la bandera Merge_Flag en falso, una vez verificado esto se chequea si en la tabla se conoce ya la dirección lógica y física de quién envía esa trama. Si las conoce se procede a actualizar dichos datos con los nuevos recibidos y se modifica la bandera Merge_Flag a verdadero. Luego se pregunta si la computadora que recibe la trama tiene la dirección IP del destinatario de la trama, de no ser así se descarta la trama, caso contrario se pregunta por el Merge_Flag. Si este Merge_Flag es falso se procede a actualizar las direcciones de destino tanto física como lógica y el tipo de protocolo en la tabla. Si no, se continúa con el algoritmo sin realizar esta actualización. Finalmente se pregunta recién si el campo de operación es Consulta, de ser así entonces se hace el intercambio y se llenan los campos del destinatario con las direcciones que se encontraban como fuente y en los campos de dirección tanto lógica como física de la fuente con las direcciones del computador que está recibiendo la trama. Se cambia también el campo de operación con el número 2, que significa operación de respuesta y se reenvía la trama a su nuevo destino.



1.4 Protocolo NDP en IPV6

El protocolo NDP (Neighbor Discovery Protocol) usado en IPv6 y el protocolo ARP usado en IPv4 cumplen con la misma finalidad. La diferencia está en los términos que se emplean para referirse a las diferentes acciones que se llevan a cabo.

Aquí en lugar de hablar de una consulta ARP (ARP request) tenemos una solicitud de vecino (Neighbor Solicitation), y en vez de hablar de una respuesta ARP (ARP reply) tenemos lo que se conoce como anuncio de vecino (Neighbor Advertisement) que aunque suenen a cosas diferentes en realidad el comportamiento que tienen es casi el mismo. Es decir, el primero es usado para llegar a conocer alguna dirección de hardware como fue descrito en la sección anterior de las consultas ARP y el segundo es usado para responder a una solicitud de manera similar como también ya fue explicado. Otra diferencia, es

el nombre que recibe la caché ARP de los nodos; la cual en NDP se la conoce como caché de vecinos (Neighbor Cache).

Dadas las similitudes de ambos protocolos, en el caso de NDP se presentan problemas similares a los que actualmente presenta el protocolo ARP. Por esta razón, ya se ha planteado una alternativa segura denominada S NDP (Secure Neighbor Discovery Protocol) [2], la cual es una extensión al protocolo NDP que usa criptografía para asegurar las comunicaciones.

CAPÍTULO II.

2 TÉCNICAS DE DETECCIÓN Y PREVENCIÓN ACTUALES CONTRA LOS ATAQUES DE ENVENENAMIENTO A LA CACHÉ ARP

En este capítulo se presenta el problema de envenenamiento a la caché ARP que se desea combatir con nuestra solución.

Además se estudian los diferentes métodos a través de los cuales se pueden realizar ataques contra el protocolo.

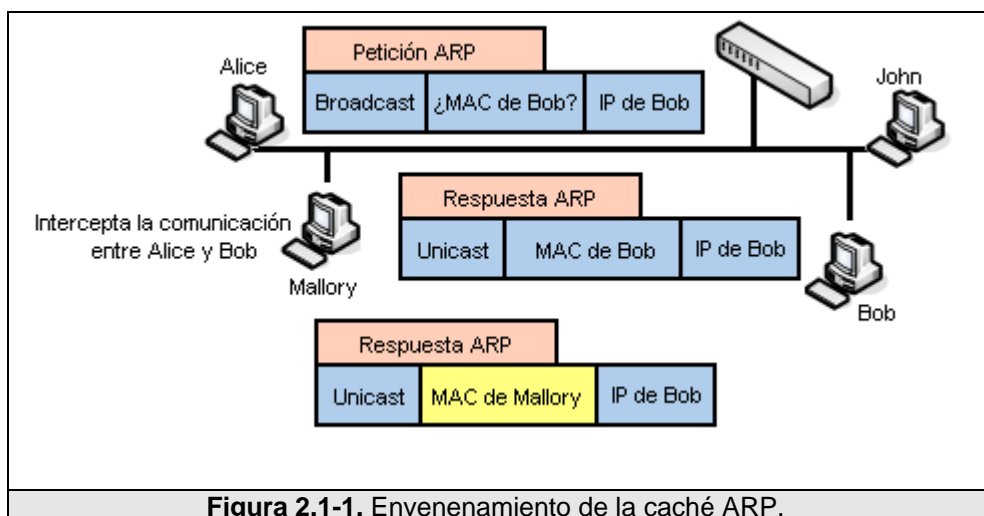
Posteriormente se abarcan las distintas técnicas existentes ya sea para detectar, prevenir o mitigar los ataques de envenenamiento a la caché ARP, las cuales han sido estudiadas y analizadas previamente [3]. Además se mencionan las características del protocolo que hacen que éste sea susceptible a los ataques mencionados.

2.1 Definición de envenenamiento a la caché ARP

El envenenamiento ARP, también conocido como falsificación ARP, es una técnica usada para infiltrarse en una red Ethernet conmutada, a través de la cual un atacante maliciosamente modifica el mapeo correcto entre una dirección de protocolo (dirección IP) y la dirección física (dirección MAC) correspondiente en la caché ARP de otro u otros nodos en la red de área local. La Figura 2.1-1 muestra un ejemplo de envenenamiento a la caché ARP.

El principio del envenenamiento o falsificación ARP es enviar mensajes ARP falsos a la red. Su finalidad consiste en asociar la dirección MAC del atacante con la dirección IP de otro nodo (el nodo atacado) de la red, así cualquier tráfico dirigido a la dirección IP de ese nodo, será erróneamente enviado al atacante, en lugar de a su destino real.

En la Figura 2.1-1, el envenenamiento ARP se ejecuta en el transcurso de transacciones ARP, creando una condición de carrera, pero el envenenamiento más común se da con la distribución de respuestas ARP no solicitadas, que son almacenadas por los nodos en sus cachés ARP, generando de esta manera el escenario de cachés ARP envenenadas [4].



2.2 Métodos de envenenamiento a la caché ARP

La implementación del protocolo ARP es sencilla y tiene ciertas carencias que facilitan el uso ilegítimo del mismo. Las siguientes características son claves para que los métodos de envenenamiento a la caché ARP se lleven a cabo [5]:

- **Protocolo sin estado.** Cuando una respuesta ARP es recibida por una máquina en la red local, ésta actualiza su caché ARP a pesar de no haber enviado una consulta ARP anteriormente.
- **Ausencia absoluta de autenticación en el protocolo.** Un computador modificará su comportamiento acorde con las tramas ARP recibidas, sin poder determinar de ningún modo la autenticidad de las mismas.
- **Cachés sujetas a alteraciones externas.** Es posible modificar

los contenidos de una caché ARP tan sólo con construir y enviar una consulta o respuesta adecuada.

A continuación se describen los métodos de envenenamiento a la caché ARP [6]:

Respuesta no solicitada

- Una respuesta ARP falsificada podría ser enviada a cualquier nodo y con esta respuesta, el nodo actualizará su caché ARP.
- Una respuesta ARP falsificada también podría ser difundida a todas las computadoras que forman parte de la red local, envenenando de esta forma a la caché ARP de todas las computadoras con un solo mensaje.

Consulta

- Cuando un computador recibe una consulta ARP, la capa ARP del mismo actualizará su caché ARP con el mapeo de los campos IP fuente y MAC fuente de la trama de consulta ARP, aún si la consulta no fue para ese computador. Es así que un atacante, solo necesita enviar una consulta ARP falsificada para envenenar la caché ARP de todas las máquinas en la red de área local.

Respuesta a una consulta

- Un nodo malicioso en la red local, al recibir una consulta ARP legítima, puede enviar una respuesta ARP falsificada. Podría haber una condición de carrera entre la respuesta falsa y la legítima para alcanzar al computador solicitante; la caché ARP será actualizada con la última respuesta ARP recibida.

Los ataques mencionados anteriormente son a menudo usados como parte de otros serios ataques: denegación del servicio, suplantación de identidad, ataque de hombre en el medio.

A continuación se describe un escenario en el cual se produce un ataque de envenenamiento ARP:

1. La máquina atacante, conociendo las direcciones IP de los dos nodos cuyas comunicaciones se quieren intervenir, resuelve mediante ARP, si es necesario, las direcciones MAC que les corresponden.
2. A través de respuestas ARP o mediante la técnica de ARP gratuito¹, el atacante modifica el contenido de las cachés de las víctimas de forma que la dirección IP de uno de los nodos

¹ Consiste en el envío de peticiones ARP preguntando por la dirección IP propia, una máquina puede detectar conflictos con otros nodos con su misma dirección IP (en caso de recibir respuesta), e informar al resto de su presencia durante el arranque, induciendo la actualización de sus cachés.

en la comunicación corresponda con la dirección MAC del atacante.

3. Cada vez que alguno de los nodos quiera enviar información al otro, resolverá la dirección MAC del mismo mediante su caché ARP previamente envenenada, enviando así el tráfico al atacante en lugar de al destinatario real.
4. El switch enviará las tramas hacia el destinatario, que en este caso es el atacante.
5. El atacante reenviará el contenido de las tramas al destinatario real. La única diferencia entre la trama original y la modificada es la dirección física del destinatario, que varía de la del atacante a la de una de las víctimas.
6. El nodo correspondiente recibirá el tráfico como si nada hubiese ocurrido. El atacante, haciendo uso del envenenamiento ARP y la técnica del hombre en el medio ha interceptado el tráfico sin que ninguno de los interlocutores se percate.

Los distintos ataques que pueden ser realizados a este protocolo, comprometen la seguridad de una red. Es así que existen diversos esquemas para añadir protección al protocolo ARP ya sea previniendo, detectando o mitigando el problema, pero no existe una

solución perfecta pues cada una tiene sus desventajas como el no haber sido probadas con todos los tipos de ataques ARP, ser costosas de implementar, y además complejas de administrar.

2.3 Detección de ataques ARP

Hay herramientas especializadas como arpswatch [7], que pueden ser usadas para detectar tráfico ARP sospechoso. Arpswatch monitorea la actividad Ethernet y mantiene una base de datos de pares <IP, MAC>. Cuando un par cambia, el administrador de la red es alertado vía correo electrónico. Esta herramienta es ligera y está altamente disponible, pero depende del administrador el poder diferenciar entre eventos no maliciosos y ataques de envenenamiento a la caché ARP, y además de su capacidad para tomar medidas apropiadas cuando un ataque ocurra.

Los sistemas de detección de intrusos (IDSs) como Snort [8], también son utilizados para detectar ataques al protocolo ARP e informan al administrador de red de ellos a través de la generación de alarmas. El principal problema con los IDSs es que tienden a generar un alto número de falsos positivos, que para ser eficaces, es un deber de la empresa poner a una persona a trabajar con esos eventos y para muchas, tener que pagar a alguien por la realización de este trabajo, no es una opción. Además la habilidad de los IDSs

para detectar ataques al protocolo ARP es limitada, ya que no los detectan todos.

M. Carnut y J. Gondim [9] propusieron una arquitectura para la detección de ataques ARP en redes conmutadas. Ésta no requiere que software especial sea instalado en los nodos de la red, en su lugar, delega la tarea de detección a una o más estaciones. Sus experimentos demostraron que la solución es muy buena al detectar ataques ARP sin generar falsos positivos, sin embargo los atacantes podrían esconderse tras volúmenes de tráfico sin que pudieran detectarlos por largos periodos de tiempo.

ARP Guard [10] es un sistema que forma un escudo de protección activa contra ataques internos a la red, incluyendo ataques ARP. Los ataques de suplantación de MAC pueden ser detectados enviando una consulta ARP inversa para una dirección MAC. La respuesta puede ser usada para determinar si una computadora está realizando la clonación (si y solo si el computador siendo clonado no ha sufrido una denegación de servicio o ha sido apagado). Esta solución es muy limitada ya que solo detecta este tipo de ataque ARP.

El uso de herramientas para detectar ataques ARP muchas veces resulta no ser efectivo. Para el momento en el cual el administrador se da cuenta del problema y tome las medidas apropiadas, puede ser muy tarde pues el atacante pudo ya haber obtenido acceso a

información sensible.

2.4 Mitigación de ataques ARP

Como una forma para limitar la exposición a los ataques ARP, se sugiere que la red sea dividida en un gran número de subredes con una pequeña cantidad de nodos en cada una [11], la desventaja, son los altos costos administrativos que puede involucrar esta medida.

Ebtables [12] es una utilidad de Linux usada para crear dispositivos de puenteo programables, permite realizar filtrado a nivel de tramas Ethernet, entre otras cosas. Ebtables puede ser usado para implementar mecanismos de prevención contra ataques ARP, pero la eficacia de este método no ha sido estudiada. Con esta solución, se podrían filtrar los mensajes ARP que pasan a través del nodo en donde se configuran las reglas de filtrado, mientras que otras áreas de la red permanecerían desprotegidas. Esto representa una desventaja. Adicionalmente, las reglas de ebtables para prevenir ataques ARP no están ampliamente disponibles, y el administrador de la red tendría la tarea de realizarlas, el cual podría cometer errores al momento de implementar el puente.

Una de las variantes de los ataques ARP usa clonación de dirección MAC para suplantar la identidad de un nodo en la red. La mayoría de los switches implementan una característica llamada seguridad de

puerto (port security) [13] basada en direcciones físicas estáticas. Estos ataques pueden ser prevenidos por el mecanismo de seguridad de puerto. Cuando en el switch está habilitada esta característica, ésta se asegura de que un número limitado de direcciones MAC sea usado en un puerto físico del mismo. De esta forma, si el número límite es uno, un switch puede efectivamente prevenir que un atacante clone la dirección física de otro nodo en la red. Esta solución es muy eficiente, sin embargo no previene otros tipos de ataques ARP y además requiere de administración, aún más si los usuarios son móviles dentro de la red local.

Algunos sistemas operativos como Solaris, aceptan respuestas ARP solo después de que la entrada <IP, MAC> ha expirado [14]. Esta disposición, dificulta al atacante el poder envenenar una caché ARP, pero tampoco es imposible hacerlo. Cuando este tipo de mecanismo es usado, un atacante puede envenenar la caché haciendo que su respuesta ARP llegue antes que la respuesta del nodo legítimo o enviando peticiones echo ICMP forzadas que aparenten salir de una de las víctimas.

La propuesta de M. Tripunitara y P. Dutta [15] consiste en la implementación de una pila de protocolo basado en streams, para bloquear respuestas ARP no solicitadas y enviar alarmas cuando una respuesta no coincide con la entrada existente en la caché ARP. La

implementación de este esquema, requiere de su instalación en cada uno de los nodos de la red. La principal desventaja de esta solución es que al depender de duplicados para detectar ataques, no los previene ni detecta cuando el nodo siendo suplantado está apagado o enfrenta un ataque de denegación de servicio.

Los métodos nombrados anteriormente para mitigar ataques ARP también presentan limitaciones.

2.5 Prevención contra ataques ARP

Una forma simple para prevenir ataques ARP es usar entradas estáticas en la caché ARP [11]. Esta solución tiene 2 desventajas: no funciona en ambientes dinámicos y no es escalable. Además algunos sistemas operativos como Windows o la distribución Ubuntu de Linux, pueden aceptar respuestas ARP dinámicas y actualizar sus entradas estáticas.

Existen algunos parches para el núcleo del sistema operativo que tratan de defenderse en contra del envenenamiento ARP. Anticap [16] es un parche para varios sistemas UNIX que previene el envenenamiento rechazando actualizaciones a la caché ARP que contienen una dirección MAC diferente de la existente en la entrada ARP para la dirección IP. Esta solución funciona en ambientes estáticos, pero no en redes dinámicas, y está disponible para un

número limitado de sistemas operativos.

Antidote [17] es otro parche, cuando una respuesta ARP nueva anuncia un cambio en un par <IP, MAC>, Antidote trata de descubrir si la dirección MAC previa aún no expira. Si la consulta es contestada con la dirección MAC anterior, la actualización es rechazada y la nueva dirección MAC es añadida a la lista de direcciones prohibidas. Nuevamente una de las desventajas de este tipo de solución es el soporte para limitados sistemas operativos.

M- Gouda y C. T. Huang [18] propusieron una arquitectura para resolver direcciones IP en direcciones MAC en una red Ethernet. Ésta consiste de un servidor seguro conectado en la red y dos protocolos usados para comunicarse con el servidor: un protocolo de invito-acepta y un protocolo consulta-respuesta. El primero es usado por cada nodo en la red para registrar su mapeo <IP, MAC> con el servidor y el protocolo consulta-respuesta es usado por los nodos para obtener la dirección MAC de algún nodo, desde la base de datos del servidor seguro. Este mecanismo no es práctico puesto que requiere modificar la implementación del protocolo ARP de cada nodo en la red local.

Han sido propuestas algunas soluciones que involucran técnicas criptográficas para autenticar el origen de las tramas ARP. SARP [19] es una extensión compatible con ARP que consiste en la criptografía

de una clave pública para autenticar las respuestas ARP. Para que esta solución sea implementada en una red local, cada nodo debe ser modificado para que use SARP en lugar de ARP. Adicionalmente debe existir una autoridad de certificación, llamada AKD, la cual es contactada para obtener la llave pública de un nodo y así las respuestas puedan ser autenticadas verificando la firma anexada. Una de las desventajas de esta solución es que AKD constituye un solo punto de fallo en la red.

V. Goyal, V. Kumar y M. Singh [20] propusieron una nueva arquitectura para asegurar la resolución de direcciones. Su sistema está basado en un árbol hash, un nodo confiable de la red (NC) y un protocolo de autenticación broadcast. Esta solución tiene la ventaja de que no se requiere de operaciones criptográficas simétricas o asimétricas; en su lugar, funciones hash de una sola vía son usadas. Los nodos pueden continuar trabajando inclusive si el NC ha fallado. El NC solo es necesario cuando una nueva computadora es añadida a la red, o un par <IP, MAC> cambia. En estos casos, el árbol es recalculado y se distribuye al resto de nodos. La mayor desventaja de esta solución es que no es compatible con ARP y puede ser muy ineficiente en redes dinámicas.

Algunos switches Cisco tienen una característica llamada Inspección Dinámica de ARP (DAI) [21]. Esta función permite al switch rechazar

tramas ARP con mapeos <IP, MAC> inválidos, para poder detectarlos, el switch usa una tabla construida localmente usando una característica llamada DHCP snooping. Este esquema es una solución muy efectiva, pero la principal desventaja es el alto costo que tienen los switches con esa funcionalidad.

De las soluciones descritas anteriormente, se puede concluir que ningún esquema ofrece una solución efectiva para el problema de ataques ARP. Nuestro objetivo es presentar un nuevo sistema para asegurar ARP y combatir los problemas a los que este protocolo es susceptible, de manera que cumpla con un número de requerimientos que la constituyan como una solución ideal. En el siguiente capítulo se describirán los requerimientos con los que debe cumplir nuestra solución.

CAPÍTULO III.

3 ANÁLISIS

A diferencia del capítulo anterior en el que se describen algunas técnicas existentes para combatir ataques contra el protocolo ARP, en este capítulo se realiza un análisis general de las mismas.

Además se presentan los requerimientos de hardware para poner en marcha nuestra solución y se listan los puntos principales con los que debe cumplir nuestro esquema para que pueda definirse como una solución ideal. También se explica el por qué del uso de las herramientas utilizadas para llevarla a cabo.

3.1 Requerimientos de la solución

La solución desarrollada debe cumplir con los siguientes requerimientos para que pueda ser considerada una solución ideal:

- No requerir cambios en cada uno de los computadores de la red, por ejemplo, no se debe de realizar la instalación de un software específico en cada nodo, ya que esto incrementaría los costos administrativos.
- Se debe evitar el uso de técnicas criptográficas, pues éstas disminuirían la rapidez del protocolo.
- El esquema debe estar ampliamente disponible.
- La solución debe ser fácil de implementar.
- Un esquema que evita o detecta ataques ARP es mejor que una técnica que los previene o bloquea.
- Se deben minimizar en lo posible requerimientos costosos de hardware.
- Todos los tipos de ataques ARP deben ser combatidos.

La solución depende de los siguientes requisitos de hardware:

- Una red de computadores pequeña formada por medio de un ruteador/switch programable (en nuestro caso, un Linksys WRTSL54GS).
- El ruteador debe de hacer uso de un firmware programable, como OpenWRT.
- Un computador con Linux como sistema operativo para que

desempeñe el papel de servidor.

3.2 Estudio de las alternativas y selección de la más apropiada

En el área de la investigación de redes y en seguridad informática muchas veces se desean implementar nuevas soluciones o nuevas variantes a protocolos, que ayuden a nuestras redes a tener un mejor desempeño. Para llevar a cabo esta tarea se cuenta con algunas alternativas, las cuales fueron descritas en el Capítulo 2 y que ahora serán analizadas de manera general.

Anteriormente se mencionó a los switches Cisco que cuentan con una característica llamada Inspección Dinámica de ARP. Estos switches ya tienen embebida la tecnología o funcionalidad que se necesita, el problema de ellos, primero es su alto costo y segundo que muchas veces pueden tener funcionalidades extra que no son necesariamente las que buscamos.

Las soluciones que utilizan técnicas criptográficas no son ideales al momento de medir su rendimiento, pues hacen que el protocolo funcione a una velocidad menor a la adecuada y además no son escalables ya que en ocasiones requieren que cada nodo participe de la red, modifique su implementación del protocolo ARP.

Luego está el hardware desarrollado usando Linux pero el problema de esto es que usualmente este hardware suele ser un simple

computador con algunas, por no decir muchas tarjetas de red, y que tiene corriendo algún programa o sistema que hace una tarea específica.

Otra opción es hacer uso de switches que soporten un firmware de código abierto como los switches Linksys, los cuales son altamente personalizables y permiten desarrollar la solución específica que se desea implementar. Debido a su bajo costo y fácil acceso, este último enfoque es el que decidimos aplicar en nuestra solución.

3.3 Análisis del software elegido para el servidor

Para implementar el servidor de nuestra solución se hace uso de un computador con Ubuntu Desktop 7.04. La utilización de este sistema operativo se debe a que nos presenta la facilidad del uso de las librerías libpcap y dumbnet para poder enviar paquetes y fácilmente manipular la caché ARP del servidor respectivamente.

Libpcap [22] es una librería de código abierto escrita en lenguaje C que ofrece al programador una interfaz de fácil manejo desde la cual capturar paquetes en la capa de red a nivel de usuario. Además, es perfectamente portable entre un gran número de sistemas operativos.

Dumbnet es la librería que utilizamos para manipular la caché ARP del servidor, además posee la descripción de las cabeceras IP, UDP, ARP, Ethernet, y facilita la obtención de datos a través de sus

funciones de conversión.

Se eligió usar un computador con la distribución de Linux, ya que uno de sus paquetes¹ oficiales es arptables; útil ya que con la configuración de una regla se evita que la caché ARP del servidor sea actualizada a través de consultas ARP, de esta forma si la consulta es una consulta falsificada, se evita que se envenene la caché ARP del servidor.

Los programas que se encuentran corriendo en el servidor fueron programados en lenguaje C debido a que se deseaba procurar obtener la mayor eficiencia posible en la solución.

3.4 Análisis del hardware escogido

En cuanto al hardware escogido lo primero que se tuvo en mente fue simplemente optar por un switch que soportara el uso de algún firmware personalizable como OpenWRT. El ruteador/switch Linksys WRTSL54GS cuenta con 8 Mb de memoria flash y 32 Mb de memoria RAM. El CPU es el más rápido de los de la serie WRT54G y usa un chipset BCM4704 que tiene una frecuencia de 266Mhz. También cuenta con una entrada de puerto USB para poder conectar a él dispositivos de almacenamiento extra [23].

¹ Nombre que reciben los programas en los sistemas Linux.



Figura 3.4-1 Linksys WRTSL54GS.

3.5 Análisis del firmware escogido

Existen diversos firmwares para los ruteadores Linksys, dentro de los cuales los más reconocidos están OpenWrt, DD-Wrt, y EWrt.

OpenWrt [24] apareció en el 2004. Es un firmware basado en el núcleo de Linux pero con funcionalidad mínima, básicamente especializado para que lo soporte el procesador de un ruteador. Presenta un manejador de paquetes similar al de Debian, lo cual permite que sea un firmware altamente personalizable y también cuenta con un ambiente de desarrollo para dispositivos embebidos llamado OpenWRT Software Development Kit (OpenWrt SDK), lo cual permite que se desarrollen aplicaciones propias para el ruteador. Existen dos versiones de firmware que derivan de OpenWrt, la primera es Whiterussian y la segunda es Kamikaze. La diferencia

principal entre éstas es que la versión Whiterussian es más estable y actualmente ya no se han desarrollado versiones nuevas del mismo. Mientras que por otro lado la versión Kamikaze continúa en desarrollo y por ende hay funcionalidades que a veces pueden tener ciertos errores, dependiendo de la versión, pues algunas son aún inestables. La funcionalidad que estos 2 firmwares ofrecen es muy extensa ya que tienen una gran cantidad de paquetes para instalar, que va desde la creación de un simple cortafuegos haciendo uso de iptables, hasta la creación y uso de redes virtuales (VLANS) en el ruteador.

El firmware DD-WRT [25] surgió para ofrecer una versión gratis del firmware Sveasoft, y se basó en la última versión de Sveasoft llamada Alchemy. Tiene el mismo núcleo base de OpenWrt y su estructura de paquetes ipkg. Este firmware presenta una excelente interfase Web, razón por la cual se ha convertido en una de las versiones más populares para muchos que no necesariamente están relacionados con la programación, pero les interesan mucho los asuntos asociados con las redes y computadoras.

En general este firmware es más orientado hacia las personas que están interesadas en obtener mayor funcionalidad de su ruteador, pero de una manera sencilla, que sea fácil de usar.

eWRT [26] en cambio es un firmware que provee las facilidades para hacer un “Captive Portal Page” similares a los que se encuentran en aeropuertos y cafés. Es decir, la utilización de este firmware se da cuando se desea ofrecer un servicio de Internet como parte de un negocio, y el objetivo no es crear toda la interfase desde cero.

Una vez analizados varios de los diferentes firmwares existentes, se pudo concluir que OpenWrt era uno de los que más se apegaba a nuestras necesidades. Por este motivo decidimos trabajar con él. Además, la documentación sobre OpenWrt es abundante.

Elegimos trabajar con la versión Whiterussian RC6 debido a que como se mencionó previamente, ésta ofrece una mayor estabilidad que la versión Kamikaze y además presenta soporte para ebttables.

CAPÍTULO IV.

4 DISEÑO

En el presente capítulo se explica todo lo referente al diseño de nuestro esquema para evitar ataques al protocolo ARP.

Debido a ciertas limitaciones en el ruteador Linksys WRTSL54GS también diseñamos un programa que se encargue de enviar los mensajes DHCP hacia el servidor. Éste y las reglas de iptables configuradas en el ruteador serán descritos a continuación.

En el transcurso de este capítulo se describen los programas que deben ejecutarse en el servidor y en el ruteador, para que nuestra solución funcione correctamente.

4.1 Diseño general

De manera general nuestra solución la integran 2 elementos

esenciales:

- Computador que responde las consultas ARP, el servidor.
- Ruteador/Switch.

Nuestro sistema en general evita los ataques ARP contra el Protocolo de Resolución de Direcciones (ARP) en redes de área local. El **ruteador**, tiene como una de sus funcionalidades, ser el servidor DHCP de la red; a través de un programa que ejecuta permanentemente, reenvía las tramas DHCP que recibe de y envía hacia los clientes DHCP, al **computador que responde las consultas ARP**, el mismo que se encarga de actualizar la caché ARP del sistema con la ayuda de estas tramas. El servidor escucha las consultas ARP en la red y las responde a través de los mapeos de la caché ARP que mantiene, siendo el único que puede hacerlo, pues en el ruteador a través de ebttables existen reglas de bloqueo que impiden que otros nodos en la red respondan las consultas ARP. Debido a que los mensajes de consulta ARP falsificados podrían también envenenar la caché ARP de todas las máquinas en la red de área local, el ruteador redirecciona esas tramas hacia el servidor, para que solo éste los reciba, y a su vez el servidor no permite que las consultas ARP actualicen su caché, sea o no que estos mensajes

estén falsificados, lo cual lo logra a través de arptables. La Figura 4.1-1 muestra el funcionamiento de nuestro mecanismo para evitar ataques ARP.

Físicamente, el switch interno del ruteador posee 4 puertos. Para las pruebas disponemos de una red de cuatro máquinas en donde una de ellas es el computador que responde las consultas ARP; todas se encuentran conectadas al ruteador que tiene como sistema operativo OpenWrt.

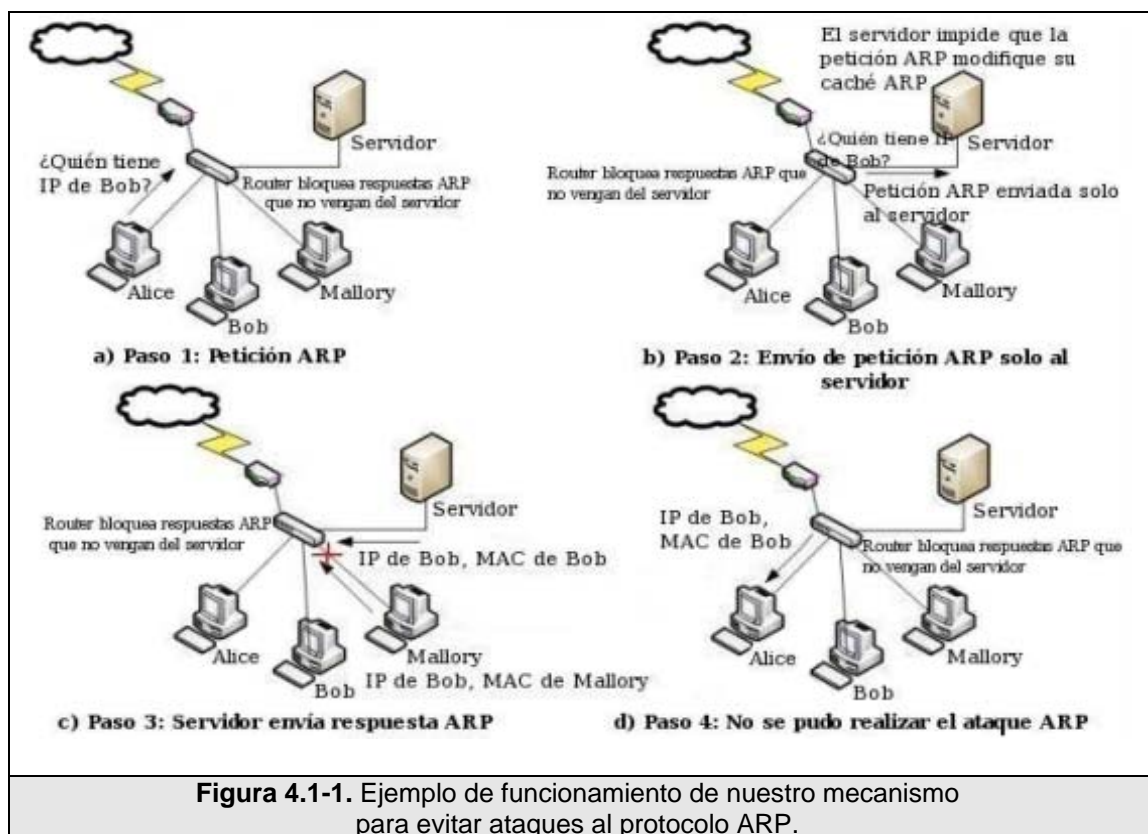


Figura 4.1-1. Ejemplo de funcionamiento de nuestro mecanismo para evitar ataques al protocolo ARP.

4.2 Diseño del servidor

4.2.1 Diseño lógico

Como se mencionó anteriormente, el servidor es el encargado de responder las consultas ARP que realizan las computadoras de la red local. Adicionalmente mantiene actualizada la caché ARP del sistema.

En el servidor se ejecutan permanentemente 2 programas:

- Programa que mantiene actualizada la caché ARP del sistema.
- Programa que escucha el tráfico ARP y responde las consultas ARP.

Además a través de una regla de arptables se impide que las consultas ARP modifiquen la caché ARP del servidor.

La ejecución automática de ambos programas se la realiza a través de un script que se ejecuta en la secuencia de arranque del sistema.

La prioridad del programa que envía las respuestas ARP para el sistema operativo debe ser alta. A modo de resumen, cada proceso en Linux tiene un nivel de prioridad que va de -20 (prioridad más alta) hasta 19 (prioridad más baja). Cuanto mayor sea el nivel de prioridad, más lentamente se ejecutará el proceso, por esta razón se asigna un valor negativo al programa que responde las consultas ARP.

4.2.2 Programa que mantiene actualizada la caché ARP del sistema

La llamada correcta para su ejecución es de la forma:
update_arp_cache interfase_lan.

Este programa escucha las tramas DHCP que circulan por la interfaz que ingresa el usuario. A través de los mensajes DHCPACK, DHCPRELEASE y DHCPDECLINE que recibe, actualiza la caché ARP del servidor.

Para el entendimiento de la elección de los mensajes DHCP para actualizar la caché ARP del sistema, a continuación se detalla lo que necesitamos comprender del funcionamiento del protocolo DHCP.

4.2.2.1 Funcionamiento del protocolo DHCP

DHCP [27] es un protocolo de red que permite a los nodos de una red IP obtener sus parámetros de configuración de red automáticamente. Se trata de un protocolo de tipo cliente/servidor en el que generalmente un servidor posee una lista de direcciones IP dinámicas y las va asignando a los clientes conforme éstas se liberan, sabiendo en todo momento quién ha estado en posesión de esa IP, cuánto tiempo la ha tenido (tiempo de arrendamiento) y a quién se la ha asignado después. El servidor DHCP y el cliente DHCP se comunican a través del puerto 67 y 68 respectivamente.

De acuerdo al escenario, para la obtención de los parámetros de red, la comunicación entre el servidor DHCP y el cliente fluye de la siguiente manera:

Escenario 1: Asignación de la configuración de red a un cliente nuevo

El diagrama de tiempo de la Figura 4.2.2.1-1 muestra las relaciones de tiempo de la típica interacción entre el cliente y el servidor DHCP. Si el cliente ya conoce su dirección, algunos pasos pueden ser omitidos, éste corresponde al segundo escenario.

De acuerdo al RFC 2131, que define las especificaciones del protocolo DHCP, los siguientes mensajes, también descritos en la tabla 4.2.2.1-1, son intercambiados entre el servidor y el cliente DHCP.

1. La comunicación inicia cuando el cliente propaga un mensaje DHCPDISCOVER a todas las máquinas de su red local. El mensaje DHCPDISCOVER puede incluir opciones que sugieren valores para la dirección de red y la duración de arrendamiento.
2. Cada servidor DHCP puede responder con un mensaje DHCPPOFFER que incluye una dirección de red disponible en

el campo 'tu dirección IP' (y otros parámetros de configuración en el campo 'opciones' de la trama DHCP).

3. El cliente recibe uno o más mensajes DHCP OFFER de uno o más servidores. El cliente elige un servidor para solicitarle los parámetros de configuración, basado en los parámetros de configuración ofrecidos en el mensaje DHCP OFFER. El cliente difunde un mensaje DHCP REQUEST a todas las computadoras que conforman su red local, el cual debe incluir entre su campo de 'opciones' el 'identificador del servidor DHCP' para indicar cual servidor ha sido seleccionado y puede incluir otras opciones que demuestren los valores de configuración deseables.
4. Los servidores reciben el mensaje DHCP REQUEST del cliente. Aquellos servidores que no fueron seleccionados, usarán este mensaje como notificación de que el cliente ha rechazado los parámetros ofrecidos por él. El servidor elegido confirma la aceptación del cliente y le responde con un mensaje DHCP ACK, el mismo que contiene los parámetros de configuración solicitados por el cliente. El campo 'tu dirección IP' contiene la dirección de red asignada al cliente y el campo 'dirección de hardware del cliente', su dirección física o dirección MAC. Si el servidor seleccionado no puede

satisfacer el mensaje DHCPREQUEST del cliente, entonces el servidor contestará con un mensaje DHCPNAK. En este momento, el cliente posee los parámetros de configuración de red. Si el cliente detecta que la dirección está actualmente en uso, el cliente debe enviar un mensaje DHCPDECLINE al servidor y reiniciar el proceso de configuración.

5. El cliente puede optar por renunciar a su contrato de arrendamiento de la dirección IP asignada enviando un mensaje DHCPRELEASE al servidor.

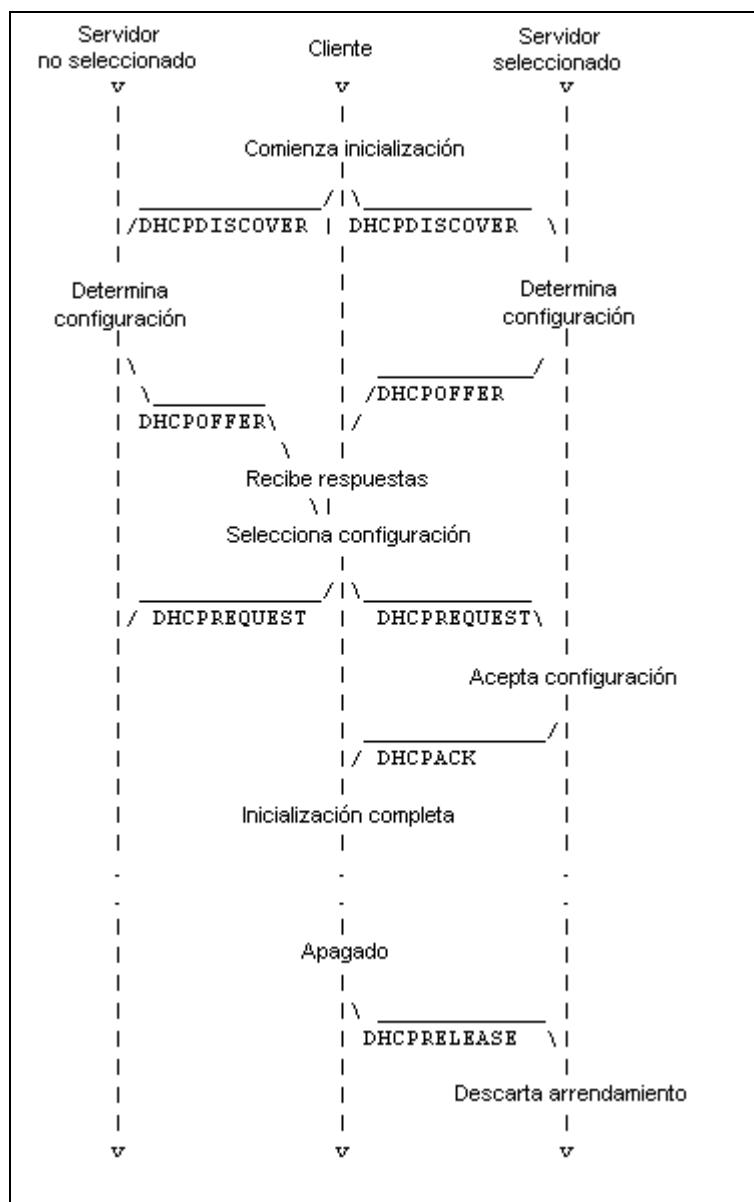


Figura 4.2.2.1-1. Diagrama de tiempo del intercambio de mensajes entre el cliente y el servidor DHCP durante la asignación de la configuración de red [F1].

Mensaje	Uso
DHCPDISCOVER	Cliente lo difunde a las computadoras de la red local para localizar servidores DHCP disponibles.
DHCPOFFER	Servidor a cliente en respuesta a DHCPDISCOVER, ofreciendo parámetros de configuración.
DHCPREQUEST	Mensaje del cliente a los servidores, ya sea a) solicitando los parámetros ofrecidos de un servidor e implícitamente rechazando la oferta del resto, b) confirmación de la correcta asignación de dirección IP previamente asignada, o c) extendiendo el arrendamiento de una dirección IP.
DHCPACK	Servidor al cliente, contiene los parámetros de configuración.
DHCPNAK	Servidor al cliente, indicando que noción de red del cliente es incorrecta o que el tiempo de arrendamiento de la dirección IP ha finalizado.
DHCPDECLINE	Cliente al servidor, indicando que la dirección de red ya está en uso.
DHCPRELEASE	Cliente al servidor, renunciando a su dirección de red y cancelando el tiempo de arrendamiento restante.
DHCPINFORM	Cliente al servidor, preguntando solo por los parámetros locales de configuración.

Tabla 4.2.2.1-1 Mensajes DHCP [27].

Escenario 2: Reutilización de una dirección de red asignada previamente

Si un cliente recuerda y desea reusar una dirección IP asignada previamente, puede elegir omitir algunos pasos descritos en el escenario 1. La Figura 4.2.2.1-2 muestra las relaciones de tiempo en una interacción típica entre cliente y servidor para un cliente que

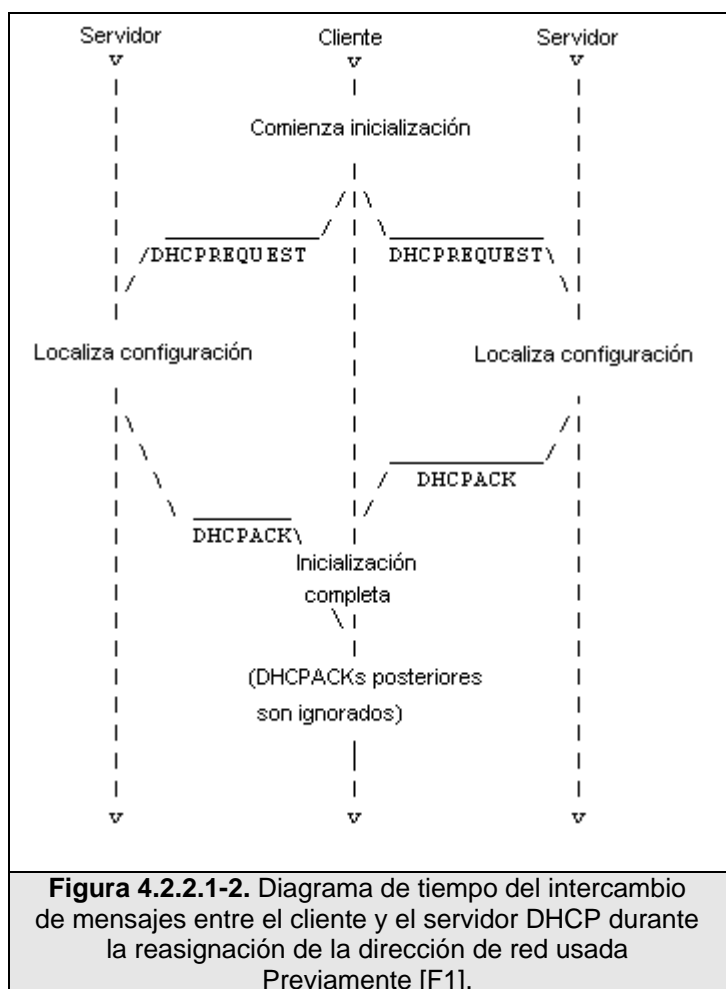
desea reusar una dirección de red asignada anteriormente.

Los mensajes en este escenario durante la obtención de la configuración de red, son los siguientes:

1. El cliente difunde un mensaje DHCPREQUEST a todas las máquinas de la red local. El mensaje incluye la dirección de red del cliente en el parámetro 'dirección IP solicitada' del campo opción.
2. Los servidores con conocimiento de los parámetros de configuración solicitados por el cliente responden con un mensaje DHCPACK al cliente. De la trama DHCPACK, el campo 'tu dirección IP' contiene la dirección de red asignada al cliente y el campo 'dirección de hardware del cliente', su dirección física o dirección MAC. Si la consulta del cliente es inválida (por ejemplo, el cliente se ha movido a una nueva subred), los servidores deberán contestar con un mensaje DHCPNAK al cliente. El cliente recibe el mensaje DHCPACK con los parámetros de configuración de red. En este momento, el cliente está configurado. Si el cliente detecta que la dirección está actualmente en uso, el cliente debe enviar un mensaje DHCPDECLINE al servidor y reiniciar el proceso de configuración. Si el cliente recibe un mensaje DHCPNAK, éste

no podrá reusar su dirección IP recordada. Debe en su lugar, solicitar una nueva dirección, reiniciando el proceso de configuración.

3. El cliente puede optar por renunciar a su contrato de arrendamiento de la dirección IP asignada enviando un mensaje DHCPRELEASE al servidor.



4.2.2.2 Algoritmo del programa update_arp_cache

El algoritmo del programa que actualiza la caché ARP de nuestro servidor es el siguiente:

Cuando una trama DHCP es recibida:

```
Si el campo 'tipo de mensaje DHCP' no es DHCPACK o DHCPRELEASE
o DHCPDECLINE,
```

```
    No se hace nada
```

```
Caso contrario,
```

```
    Si el campo 'tipo de mensaje DHCP' es DHCPACK,
```

```
        Obtengo la dirección IP del campo 'tu dirección
        IP' y la comparo con la dirección IP del servidor
        Si la dirección IP corresponde a la del servidor,
```

```
            No se hace nada
```

```
        Caso contrario,
```

```
            Se obtiene el campo 'dirección de hardware
            del cliente' y se añade la entrada 'tu
            dirección IP' y 'dirección de hardware del
            cliente' a la caché ARP del servidor,
            además se actualiza el archivo de mapeos
            llamado cache_file.txt
```

```
Caso contrario,
```

```
    Si el campo 'tipo de mensaje DHCP' es DHCPRELEASE,
    Obtengo la dirección IP del campo
    'dirección IP del cliente' y la comparo con
    la dirección IP del servidor
```

```
    Si la dirección IP corresponde a la del
    servidor,
```

```
        No se hace nada
```

```
    Caso contrario,
```

```
        Elimino el par correspondiente a
        'dirección IP del cliente' y su MAC
        respectiva de la caché ARP del
        servidor, además se actualiza el
        archivo de mapeos llamado
        cache_file.txt
```

```
Caso contrario,
```

```
    Obtengo el parámetro 'dirección IP
    solicitada' del campo 'opciones' y la
    comparo con la dirección IP del servidor
    Si la dirección IP corresponde a la del
    servidor,
```

```
        No se hace nada
```

```
    Caso contrario,
```

```
        Elimino el par correspondiente a
        dirección IP solicitada' y su MAC
        respectiva de la caché ARP del
        servidor, además se actualiza el
        archivo de mapeos llamado
        cache_file.txt
```

4.2.2.3 Descripción detallada del programa

La ejecución correcta del programa es de la forma:

```
update_arp_cache interfase_lan.
```

El programa que mantiene actualizada la caché ARP del servidor, al momento de su ejecución, valida la sintaxis de la interfaz ingresada, interfaz a través de la cual se escuchan los mensajes DHCP que recibe y envía el servidor DHCP. El programa solo procesa tramas DHCP por la expresión filtro definida de la siguiente forma: `udp and (dst port 67 or dst port 68)`, la cual identifica a las tramas UDP cuyo puerto destino sea el 67 o el 68. Las tramas DHCP viajan como datagramas UDP, todas las tramas enviadas desde el cliente DHCP tienen como puerto fuente el 68 y como puerto destino el 67. Las tramas como DHCPACK, DHCPRELEASE y DHCPDECLINE no son difundidas hacia todas las máquinas de la red local (broadcast), por consiguiente, una trama DHCP enviada desde el ruteador (servidor DHCP de la red local) hacia otra máquina de la red que no sea el servidor, no podría ser vista por éste; la manera de lograr que esos mensajes puedan ser leídos por el servidor se detalla en el punto 4.3.

Inicialmente, si existe el archivo, se realiza su carga. Este archivo contiene la información que el comando `arp1 -n` retorna después de

¹ Programa que permite la manipulación de la caché ARP del sistema.

su ejecución. El archivo se modifica cada vez que la caché ARP del sistema se actualiza. La carga del mismo al inicio del programa es útil en el caso de que por algún motivo el servidor haya sido apagado, por esta razón, su caché ARP pierde las entradas que contiene, pero el programa mantiene el archivo con las relaciones IP – MAC de la caché, es así que esas entradas son recuperadas cuando se carga el archivo; es decir, esta es una característica de tolerancia a fallos. Seguidamente, se obtiene la dirección IP del servidor que corresponde a la interfaz ingresada, esta IP será utilizada más adelante. A través de las funciones de la librería libpcap se inicia con la captura de las tramas DHCP. El formato de un mensaje DHCP se muestra en la Figura 4.2.2.3-1. Por cada trama DHCP obtenida, se verifica si es de tipo DHCPACK, DHCPRELEASE o DHCPDECLINE, y se chequea con la dirección IP obtenida anteriormente que corresponde al servidor, que estas tramas no vayan hacia o salgan de él, puesto que en la caché ARP no se puede ingresar el mapeo IP – MAC del sistema que es su propietario. Si el mensaje es de tipo DHCPACK, entonces se toman los bytes necesarios que corresponden a los campos 'tu dirección IP' y 'dirección de hardware del cliente' y con la ayuda del comando `arp -s` se añade la entrada 'tu dirección IP' – 'dirección de hardware del cliente' de manera estática a la caché del sistema, así mismo se

actualiza el archivo que guarda los mapeos. Una entrada estática es válida hasta que sea eliminada manualmente de la caché ARP. Además las entradas estáticas solo se pierden con el reinicio del computador o de la interfaz de red. Si el mensaje es de tipo DHCPRELEASE, se utiliza el campo 'dirección IP del cliente' y a través del comando `arp -d` se elimina el mapeo correspondiente a esa dirección IP, y de la misma forma se actualiza el archivo en el que se guardan las entradas. Cuando el mensaje es de tipo DHCPDECLINE, hacemos uso del parámetro 'dirección IP solicitada' del campo 'opciones' y nuevamente, con el comando `arp -d` se elimina la entrada correspondiente a esa IP, y se actualiza el archivo de mapeos. De esta forma, a través de los mensajes DHCP se mantiene la caché ARP del servidor, en donde se guardan todos los mapeos IP – MAC correspondientes a las computadoras que conforman la red local, mapeos que serán utilizados por el programa que responde las consultas ARP.

A continuación se presenta la descripción de las funciones que hacen que lo detallado anteriormente sea posible:

- `validate_device(char * device)`, valida que la interfaz ingresada: `device`, sea del tipo `ethx` o `ethxx`.
- `load_cache_from_file()`, carga el archivo donde se guardan los

mapeos IP – MAC pertenecientes a las computadoras de la red local.

- `get_IP_from_device(char *device)`, retorna la dirección IP correspondiente a la interfaz ingresada: `device`.
- `get_DHCP_msg_type(u_char *dhcp_message, int dm_lgth)`, identifica si el mensaje DHCP: `dhcp_message` es del tipo `DHCPACK`, `DHCPRELEASE` o `DHCPDECLINE`.
- `write_cache_file()`, redirecciona la salida del comando `arp -n` hacia el archivo `cache_file.txt`, esta salida contiene los mapeos IP – MAC de las máquinas de la red local.
- `update_IP_MAC_table (u_char *user, const struct pcap_pkthdr* pkthdr, const u_char* packet)`, actualiza la caché ARP del sistema y el archivo donde se guardan los mapeos, a través de los mensajes DHCP.

Los archivos de código fuente que forman parte del programa `update_arp_cache` pueden ser revisados en el Apéndice A.1.



4.2.3 Programa que escucha el tráfico ARP y responde las consultas ARP

La llamada correcta para su ejecución es de la forma:
`send_arp_reply interfase_lan.`

Este programa escucha las tramas ARP que circulan por la interfaz indicada por el usuario. Su función consiste en responder las consultas ARP que realizan las computadoras de la red local, en base a la caché ARP del sistema que mantiene el programa `update_arp_cache.`

4.2.3.1 Algoritmo del programa send_arp_reply

El algoritmo para el programa que responde las consultas ARP de las computadoras de la red local es el siguiente:

Cuando una trama ARP es recibida:

```

Si el campo 'operación' indica que la trama es una consulta
ARP,
    Si la consulta ARP pregunta por la MAC del servidor,
        El campo MAC fuente de la respuesta ARP es llenado
        con la MAC del servidor
    Caso Contrario,
        Consulto a la caché ARP del sistema
        Si no existe el mapeo,
            No se hace nada,
        Caso Contrario,
            El campo MAC fuente de la respuesta ARP es
            llenado con la MAC obtenida
    Completo la trama de respuesta ARP, la cual comenzó a
    ser formada previamente con los campos constantes en la
    función principal del programa y se la envió a la
    computadora que realizó la consulta ARP
Caso contrario,
    No se hace nada

```

4.2.3.2 Descripción detallada del programa

La ejecución correcta del programa es de la forma:

```
send_arp_reply interfase_lan.
```

El programa que responde las consultas ARP de las computadoras de la red local, al momento de su ejecución, valida la sintaxis de la interfaz ingresada, interfaz a través de la cual se escuchan las tramas ARP que son reenviadas por el ruteador solo hacia el servidor. Las tramas de consulta ARP, a través de una regla de ebttables colocada en el ruteador, no son difundidas a todas las

máquinas de la red local, y son reenviadas únicamente hacia el servidor, el cual impide que estas consultas modifiquen su caché ARP con la siguiente regla de arptables:

```
arptables -A INPUT --opcode 1 -j DROP
```

Las configuraciones realizadas en el ruteador, serán explicadas con más detalle en el punto 4.3.

Seguidamente, se obtiene la dirección IP del servidor que corresponde a la interfaz ingresada, y además la dirección física correspondiente; ambas serán utilizadas más adelante.

Antes de iniciar con el olfateo de las tramas ARP, se llenan los campos constantes de una trama de respuesta ARP, para que este proceso sea ejecutado una sola vez y no cada vez que se recepta una trama. De esta forma, los campos cuyos valores son constantes en nuestra red son los siguientes: en la cabecera Ethernet el campo de dirección física origen, llenado con la dirección MAC del servidor y el campo que especifica el tipo de protocolo, el cual es ARP. En la cabecera ARP los campos constantes son el formato de la dirección física el cual es Ethernet, el tipo de protocolo de transporte, la longitud en bytes de la dirección física, la longitud en bytes que tendrá la dirección de protocolo, y el campo del código de operación, cuyo valor es 2 ya que la trama es de tipo respuesta.

Luego de este preformado de la trama ARP, a través de las funciones de la librería libpcap se inicia con la captura de las tramas ARP. Por

cada trama recibida se realiza lo siguiente: Se verifica si la trama es una consulta ARP, y si en la consulta se pregunta por la MAC del servidor, entonces se contesta con la MAC obtenida anteriormente. La verificación se la realiza comparando la dirección IP del servidor con la dirección IP por la que se pregunta en la trama de consulta ARP. El sistema operativo, por la regla de arptables, no puede contestar las consultas ARP que preguntan por su propia dirección física, por esta razón, el programa también se encarga de contestarlas.

Si la consulta ARP pregunta por la MAC de otra dirección IP de la red local, entonces se obtiene la dirección física correspondiente de la caché ARP del servidor.

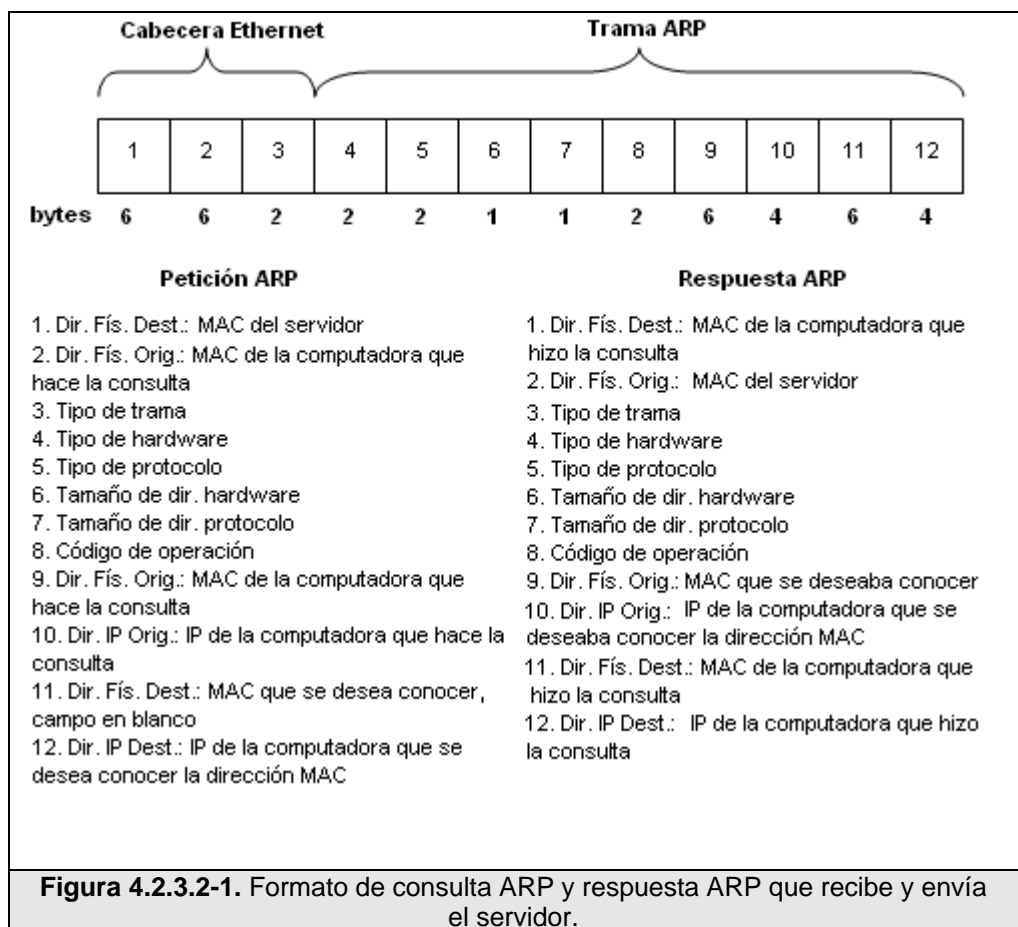
Finalmente se crea la trama de respuesta ARP y se la envía hacia el computador que realizó la consulta. La trama de respuesta ARP formada por el servidor tiene como dirección física origen en la cabecera Ethernet, la MAC del servidor. La Figura 4.2.3.2-1 muestra el formato de las tramas de consulta ARP que recibe el servidor, y una trama de respuesta ARP generada por el servidor, con su respectiva cabecera Ethernet. Como se puede observar en la imagen, la diferencia con las tramas ARP normales, se da en la dirección física destino de la cabecera Ethernet de las consultas ARP y la dirección física origen de la cabecera Ethernet de las respuestas

ARP.

A continuación se presenta la descripción de las funciones que hacen que lo detallado anteriormente sea posible:

- `validate_device(char * device)`, valida que la interfaz ingresada: `device`, sea del tipo `ethx` o `ethxx`.
- `get_IP_from_device(char *device)`, retorna la dirección IP correspondiente a la interfaz ingresada: `device`.
- `get_local_MAC_addr(char *device, u_char *addr)`, obtiene la dirección física de la interfaz deseada.
- `get_mac_from_IP(char *source_ip, struct arp_entry *arp_entry)`, obtiene la MAC correspondiente a la dirección IP especificada por `source_ip`.
- `complete_and_send_ARP_reply(u_char *user, const struct pcap_pkthdr* pkthdr, const u_char *packet)`, responde las consultas ARP.

Los archivos de código fuente que forman parte del programa `send_arp_reply` pueden ser revisados en el Apéndice A.2.



4.3 Diseño de reglas y programa del switch

Para nuestra solución, en el ruteador/switch Linksys WRTSL54GS es fundamental la configuración de las reglas de iptables que redireccionan y bloquean las tramas ARP, y la ejecución del programa que con la ayuda de iptables envía el tráfico DHCP de la red local hacia el servidor.

4.3.1 Envío de tráfico DHCP de la red hacia el servidor

Debido a la limitación de no poder configurar un puerto como 'puerto monitor'¹ o también conocido como 'puerto espejo' en el ruteador, realizamos un programa que se encargue de enviar hacia el servidor, las tramas DHCP recibidas y enviadas por el servidor DHCP de la red local. La utilización de iptables fue esencial, ya que a través de su alcance ULOG se guardaban en un archivo las tramas que serían utilizadas por nuestro programa.

El módulo ULOG de iptables guarda la trama completa de lo que se indique a través de las reglas al archivo indicado en el archivo de configuración de ULOG.

Las reglas de iptables para la realización del logeo fueron las siguientes:

- Para guardar las tramas DHCP recibidas por el ruteador:

```
iptables -t mangle -A INPUT -s IP_RED_LOCAL -j MARK --set-mark 100
iptables -A INPUT -m mark --mark 100 -p udp --sport 68 -j ULOG
```

- Para guardar las tramas DHCP enviadas por el ruteador:

```
iptables -t mangle -A OUTPUT -d IP_RED_LOCAL -j MARK --set-mark 200
iptables -A OUTPUT -m mark --mark 200 -p udp --dport 68 -j ULOG
```

A través de la tabla mangle de iptables, se permite el marcado de paquetes; es así que haciendo uso de esta funcionalidad, marcamos los paquetes tanto de entrada como de salida, y solo aquellos que

¹ A través de la configuración de un puerto monitor, el tráfico de cualquier puerto en el switch puede ser copiado hacia otro puerto.

vienen del puerto 68 y cuyo protocolo es UDP con marca número 100, son guardados en un archivo a través de ULOG, de la misma forma se guardan solo aquellos cuyo protocolo es UDP y se dirigen hacia el puerto 68, con marca 200.

A continuación se presenta la descripción del programa que utiliza el archivo de tramas DHCP formado por iptables.

4.3.1.1 Descripción detallada del programa

Su ejecución correcta es de la forma: `ForwardDhcpFrames MAC_servidor.`

El argumento que acompaña al nombre del programa indica la MAC del computador hacia el cual se debe enviar el tráfico DHCP, que sería la dirección física del servidor de nuestra solución.

Inicialmente se obtiene la dirección física de la interfaz `br0`, correspondiente a la interfaz de red local del ruteador.

El archivo del ruteador en el que se guardan las tramas, cuya ruta es `/var/log` se denomina `ulogd.pcap`. Por cada trama DHCP del archivo `ulogd.pcap`, se obtiene su tamaño. Cabe mencionar, que las tramas logeadas por ULOG, no poseen cabecera Ethernet, así es que la trama DHCP creada en el programa tiene como tamaño la longitud de la cabecera Ethernet, más la de la cabecera IP, más el tamaño contenido en el campo longitud de la cabecera UDP. En la cabecera

Ethernet de la nueva trama DHCP, se coloca como dirección física destino, la MAC que se envía como argumento en la llamada de ejecución del programa, y como dirección física de origen, la MAC de br0. El resto de la trama DHCP nueva es igual a la trama leída en el archivo ulogd.pcap. Una vez que la trama DHCP está formada, es enviada hacia el servidor.

A continuación se presenta la descripción de las funciones que hacen que lo detallado anteriormente sea posible:

- `eth_pton(const char *p, u_char *data)`, convierte la cadena `p` a arreglo hexadecimal de `u_char`.
- `get_MAC_addr(u_char *addr)`, función que obtiene la dirección física de la interfaz `br0`.
- `forward_dhcp_message(u_char *user, const struct pcap_pkthdr* pkt_hdr, const u_char* packet)`, reenvía los mensajes DHCP recibidos y enviados por el ruteador hacia el servidor.

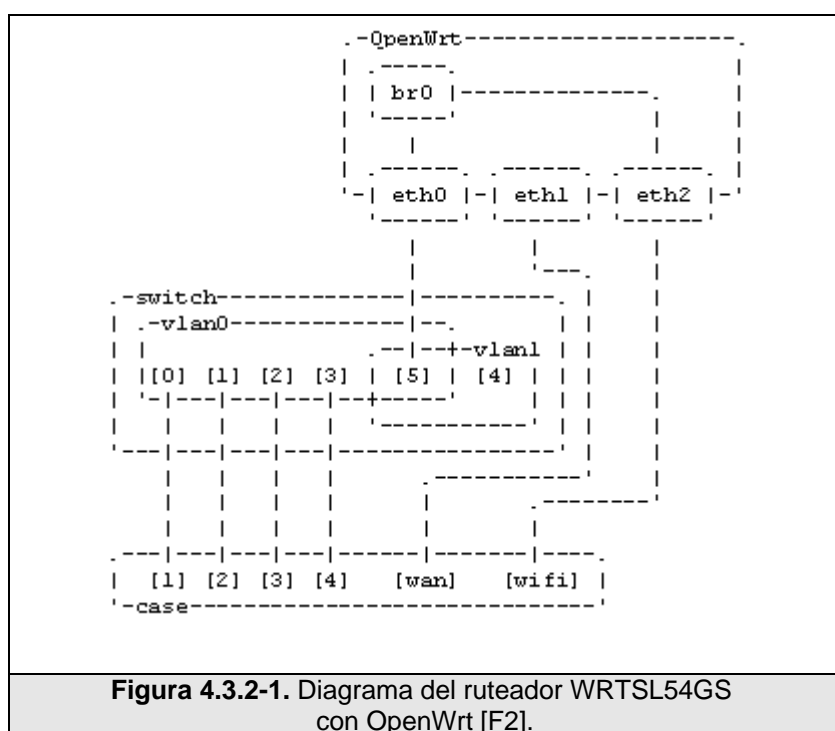
El archivo de código fuente del programa `ForwardDhcpFrames` puede ser revisado en el Apéndice B.1.

4.3.2 Redireccionamiento y bloqueo de las tramas ARP

Originalmente, para el sistema operativo OpenWrt del ruteador

Linksys, y debido al hardware de éste, existen solo 4 interfases: br0, eth0, eth1, eth2 y la interfaz de loopback lo.

La representación de una interfaz por cada puerto del switch interno del WRTSL54GS, no existe, pues br0 corresponde al puente entre la interfaz eth0 y la interfaz eth2, eth0 corresponde a la red local, eth2 a la red inalámbrica y la interfaz eth1 es la WAN. Un diagrama del ruteador/switch WRTSL54GS se muestra en la Figura. 4.3.2-1.



Para poder realizar el bloqueo de respuestas ARP que no sean originadas por el servidor, lo que se realizó, fue la creación de una red virtual (VLAN) por cada puerto del switch del Linksys WRTSL54GS. De esta forma, se podría identificar el puerto al que

está conectado el servidor. Así es, que con la ayuda del comando `nvrnm`, y gracias a que en el switch interno del ruteador se pueden configurar redes virtuales, se identificó a cada puerto del ruteador como una VLAN diferente, y nuevamente para que todas formen parte de una sola red, se creó un puente entre ellas.

El switch interno del WRTSL54GS se conforma por 6 puertos, 4 de los cuales pueden ser vistos externamente. Luego de la creación de las VLANs, cada puerto del 1 al 4, correspondía a una red diferente, a las cuales se las nombró como `vlan1`, `vlan2`, `vlan3` y `vlan4`. Una vez creadas las redes virtuales y el puente entre ellas, el sistema operativo OpenWrt reconocía cada puerto como una interfaz nueva, así es que luego de esta configuración, que será explicada detalladamente en el Capítulo 5 de implementación, el sistema operativo OpenWrt, en lugar de ver solo las interfases `br0`, `eth0`, `eth1`, `eth2`, también reconocía a las interfases `vlan1`, `vlan2`, `vlan3` y `vlan4`. Suponiendo que el servidor que responde las consultas ARP de la red local, estuviese conectado al puerto 3 del ruteador, la regla que bloquea las respuestas ARP que no proceden de él, es la siguiente:

```
ebtables -A FORWARD -p ARP --arp-opcode 2 -i ! vlan3 -j DROP
```

Como se mencionó anteriormente, las consultas ARP también podrían envenenar las cachés ARP de no solo una, sino todas las

máquinas que conforman la red local, ya que este mensaje es de tipo broadcast. Para evitar que las consultas sean difundidas a todas las computadoras, y sean enviadas únicamente hacia el servidor, las reglas utilizadas fueron las siguientes:

```
ebtables -t nat -A PREROUTING -p ARP --arp-opcode 1 -s !
MAC_SERVIDOR -d ff:ff:ff:ff:ff:ff -j dnat --to-destination
MAC_SERVIDOR
ebtables -t nat -A PREROUTING -p ARP --arp-opcode 1 -s !
MAC_SERVIDOR -d ! ff:ff:ff:ff:ff:ff -j dnat --to-destination
MAC_SERVIDOR
```

Ya que las tramas de consulta ARP pueden ser también dirigidas hacia un solo computador, y no ser broadcast, como ocurre comúnmente, colocamos la segunda regla que indica que aquellas consultas que son realizadas de manera unicast, también sean enviadas solo hacia el servidor.

CAPÍTULO V.

5 IMPLEMENTACIÓN

En el presente capítulo se hace una descripción detallada de los pasos seguidos para la instalación y ejecución del programa del ruteador que reenvía las tramas DHCP al servidor. Además se describe la configuración de las reglas de iptables que bloquean y redireccionan las tramas ARP y lo realizado para ejecutar los programas que deben correr en el servidor para que éste responda las consultas ARP y actualice su caché ARP.

5.1 Implementación del servidor

El servidor de nuestro esquema es una computadora portátil HP COMPAQ con las siguientes características:

- Procesador Pentium Dual Core 1.7 Ghz.
- Memoria RAM de 1GB.
- Disco duro de 120 GB.
- Tarjeta de red 10/100 Mbps.

Las características de almacenamiento, procesador y memoria RAM se basan en las necesidades de hardware que el sistema operativo Ubuntu Desktop 7.04, requiere.

5.1.1 Configuración del servidor

Los programas `update_arp_cache` y `send_arp_reply` fueron desarrollados usando Anjuta¹. Las herramientas de programación y sus dependencias² necesarias para la manipulación del código fueron instaladas con las siguientes versiones: automake 1.10, gcc 4.1.2, libglib 2.12.11, binutils 2.17, libpcap 0.9.5 y libdumbnet 1.8-1.5. Su instalación se llevó acabo a través del gestor de paquetes³ Synaptic.

Para la ejecución de `update_arp_cache` y `send_arp_reply` en consola, se realizó lo siguiente:

¹ Es un Entorno de Desarrollo Integrado (IDE) para programar en C y C++ en sistemas GNU/Linux.

² Librerías necesarias para la instalación de un programa.

³ Administrador de paquetes del sistema. Instala, lista, actualiza y desinstala programas.

1. En el intérprete de comandos, como usuario administrador del sistema, se editó el archivo: `/etc/profile`, para esto utilizamos el editor de texto `vi`:

```
vi /etc/profile
```

2. Se modificó la variable de ambiente `PATH`, añadiendo al final del archivo la ruta de los binarios de ambos programas de la siguiente forma:

```
export  
PATH=/ruta_binario_update_arp_cache:/ruta_binario_send_arp_reply:  
$PATH
```

Así para la ejecución de `update_arp_cache` y `send_arp_reply` solo es necesario llamar a ambos desde el intérprete de comandos de la siguiente forma:

```
root@nombre_maquina:~# update_arp_cache interfase_lan &  
root@nombre_maquina:~# nice -n -20 send_arp_reply interfase_lan &
```

Donde el comando `nice` lanza el proceso `send_arp_reply` con prioridad de `-20`, indicándole así al procesador del servidor que su prioridad de ejecución es alta.

Estos programas se ejecutan de manera permanente, escuchando los mensajes DHCP y ARP que llegan y salen por la interfaz que acompaña a su llamada. El símbolo `&` utilizado, indica que las tareas

serán ejecutadas en segundo plano, así se podrá retomar el control de la consola y ejecutar nuevas órdenes.

Para evitar que las consultas ARP actualicen la caché ARP del sistema, debe colocarse la regla de arptables descrita en el Capítulo 4. Para este propósito los pasos realizados fueron los siguientes:

1. En el intérprete de comandos, siendo usuario administrador del sistema, se editó el archivo: `/etc/rc.local`¹, para esto utilizamos el editor de texto `vi`:

```
vi /etc/rc.local
```

2. Al final del archivo `rc.local` antes de la línea que dice `exit 0`, se añadió la regla de arptables:

```
arptables -A INPUT --opcode 1 -j DROP
```

De esta forma, cada vez que el servidor inicia, la regla de arptables es ejecutada en el sistema.

A continuación se describen las configuraciones realizadas en el ruteador, las cuales complementan los pasos detallados anteriormente y hacen que nuestro esquema funcione.

5.2 Configuraciones en el switch WRTSL54GS

¹ Uno de los scripts que se ejecutan de manera automática durante la secuencia de arranque de Linux.

Para el reenvío de las tramas DHCP hacia el servidor fue necesaria la creación de un programa que cumpla con esta función y el uso de reglas de iptables. Debido al limitado recurso de memoria RAM con el que cuenta el ruteador, no es posible utilizar herramientas de compilación como gcc, es así que para la creación del programa necesario, hicimos uso de la **compilación cruzada**¹.

El bloqueo de las respuestas ARP que no provengan del servidor, y la modificación del destino de las consultas ARP, las realizamos con reglas de iptables.

La implementación de estas configuraciones se detalla a continuación.

5.2.1 Creación, instalación y ejecución del programa que reenvía las tramas DHCP del servidor DHCP al servidor de nuestro esquema

El paquete generado por la compilación cruzada debe ser instalado en el ruteador. Para realizar su compilación fue necesario contar con una serie de programas y librerías que establezcan un ambiente propicio para lograrlo. Este ambiente se denomina **entorno de compilación cruzada**, el cual nos brinda la posibilidad de aprovechar los recursos que disponemos en una computadora tipo

¹ La compilación de código fuente que, realizada bajo una determinada arquitectura genera código ejecutable para una arquitectura diferente.

PC.

La tarea se llevó a cabo sobre la distribución Centos 5.0, en una computadora con las mismas características de hardware que posee el servidor de nuestro esquema.

Los pasos realizados para la creación e instalación del paquete fueron los siguientes [28]:

1. Primeramente se realizó la descarga del kit de ambiente de desarrollo (OpenWrt SDK) para dispositivos embebidos que posee OpenWrt [29]. El SDK depende de la arquitectura de la computadora en la que se realice la compilación cruzada y de la versión de OpenWrt instalada en el ruteador. Nuestro ruteador cuenta con Whiterussian RC6 y la caja de desarrollo fue i686, por lo tanto de la página de descargas de OpenWrt bajamos el SDK correspondiente a esas especificaciones.
2. El siguiente paso fue descomprimir el SDK descargado. El conjunto de subdirectorios de la carpeta descomprimida son los siguientes: dl, docs, examples, include, package, scripts y staging_dir_mipsel. Para la creación del paquete, la única carpeta cuyo contenido fue modificado es package.
3. Dentro del subdirectorio package creamos una carpeta con el nombre ForwardDhcpFrames, y a su vez dentro de

ForwardDhcpFrames otra carpeta con el nombre src, cuyo contenido fue el código fuente del programa que reenvía las tramas DHCP al servidor y el archivo Makefile necesario para su compilación, el cual se muestra en el Apéndice C.1. Este Makefile es útil para compilar el código fuente en el entorno de compilación cruzada y a su vez es necesario para generar el paquete para el ruteador.

4. La arquitectura del ruteador es diferente a la del entorno de compilación cruzada, es así que para realizar la compilación del código para el ruteador, creamos un archivo Makefile diferente al anterior, el cual debe colocarse dentro de la carpeta ForwardDhcpFrames. El contenido de este archivo se muestra en el Apéndice C.2.
5. Una vez realizados los pasos anteriores nos colocamos en el directorio raíz, es decir dentro de la carpeta descomprimida y ejecutamos `make V=99` para proceder con la compilación del código.
6. Luego de la compilación exitosa se creó automáticamente una carpeta llamada bin, cuyo contenido era el paquete ipkg listo para ser instalado en el ruteador.
7. Para la instalación del paquete, lo trasladamos hacia el ruteador con el comando `scp`, y dentro del mismo, en una

consola ejecutamos:

```
root@OpenWrt:~# ipkg install ForwardDhcpFrames_1_mipsel.ipk
```

Luego de la instalación del paquete de reenvío de tramas DHCP en el ruteador lo que faltaba para ejecutar el programa, era la configuración de las reglas de iptables para guardar las tramas DHCP en un archivo (el cual es utilizado por el programa para obtener las tramas DHCP), la creación del archivo, y unos cambios en el archivo de configuración de ulogd, lo cual se logró de la siguiente forma:

1. Por defecto OpenWrt RC 6 no carga el módulo de ULOG de la herramienta iptables, para ésto se editó el archivo `/etc/modules` y añadimos: `ipt_ULOG`. De esta forma cada vez que inicia el ruteador, se carga el módulo `ipt_ULOG`. Para cargarlo en el instante, se ejecuta: `insmod ipt_ULOG`.
2. Las reglas mencionadas en la parte de diseño de nuestra solución:

```
iptables -t mangle -A INPUT -s IP_RED_LOCAL -j MARK --set-mark 100
iptables -A INPUT -m mark --mark 100 -p udp --sport 68 -j ULOG

iptables -t mangle -A OUTPUT -d IP_RED_LOCAL -j MARK --set-mark 200
iptables -A OUTPUT -m mark --mark 200 -p udp --dport 68 -j ULOG
```

Fueron colocadas en el archivo de firewall del ruteador (`/etc/init.d/S35firewall`), así cada vez que éste inicia, las reglas

toman efecto. El archivo de firewall con las reglas de ULOG incluidas se muestra en el Apéndice D.1.

Una vez colocadas las reglas de iptables, el siguiente paso es la creación del archivo en el cual el demonio ulogd guardará las tramas DHCP.

Por defecto, el módulo ULOG de iptables a través del plugin ulogd-mod-pcap, que es un paquete que incluye OpenWrt White Russian RC6, guarda los paquetes en un archivo normal, pero por motivos citados a continuación, nosotros modificamos ULOG para que cree una **tubería nombrada (named pipe)** en lugar de ese archivo de texto común.

Esta clase de archivo, también conocido como archivo FIFO o named pipe, es una extensión del concepto tradicional de tubería utilizado en los sistemas operativos POSIX, y es uno de los métodos de comunicación entre procesos (IPC) [30], es así que una de las razones de su utilización es que los 2 procesos: programa que envía los mensajes leídos de él, y demonio ulogd que escribe las tramas en él, se comuniquen entre sí leyendo de y escribiendo en la tubería nombrada. Además otra razón para su uso es que este archivo nunca crece, el mismo orden de bytes que entran al archivo, sale cuando el otro proceso que forma parte de la comunicación los lee, siendo de

esta forma siempre el tamaño de la tubería nombrada igual a 0 bytes. En una consola del ruteador, para crear el archivo FIFO, ejecutamos en la ruta /var/log lo siguiente:

```
root@OpenWrt:~# rm ulogd.pcap
root@OpenWrt:~# mkfifo ulogd.pcap
```

Primeramente borramos el archivo de texto normal y luego creamos la tubería nombrada.

El nombre ulogd.pcap y su ruta son valores por defecto que se encuentran configurados en el archivo /etc/ulogd.conf de ULOG. El plugin ulogd-mod-pcap instalado en el ruteador, perteneciente al módulo ULOG, guarda los paquetes en formato tcpdump, lo cual fue de mucha ayuda ya que en el programa que reenvía las tramas, pudimos usar funciones de la librería libpcap para procesar los paquetes ahí guardados.

Para que el archivo ulogd.pcap se cree como tubería nombrada de manera automática cada vez que se encienda el ruteador se debe editar el script de ejecución del demonio ulogd: /etc/init.d/ulogd y añadir lo siguiente:

```
rm /var/log/ulogd.pcap
mkfifo /var/log/ulogd.pcap
```

El archivo ulogd deberá quedar de la siguiente forma:

```
#!/bin/sh

BIN=ulogd
DEFAULT=/etc/default/$BIN
LOG_D=/var/log
```

```
[ -f $DEFAULT ] && . $DEFAULT

case $1 in
  start)
    $BIN $OPTIONS
    ;;
  *)
    echo "usage: $0 (start)"
    exit 1
esac

rm /var/log/ulogd.pcap
mkfifo /var/log/ulogd.pcap

exit $?
```

Es así que ya teniendo el paquete instalado, la tubería nombrada creada y las reglas de iptables configuradas; solo falta descomentar el plugin que necesitamos: `plugin="/usr/lib/ulogd/ulogd_PCAP.so"` en el archivo de configuración de ULOG (`/etc/ulogd.conf`) y comentar el que no se usará: `plugin="/usr/lib/ulogd/ulogd_LOGEMU.so"`, pues por defecto es lo contrario.

Ahora para ejecutar el programa que reenvía las tramas DHCP del ruteador al servidor, solo basta con llamarlo desde una consola del ruteador e iniciar el demonio `ulogd` en otra, para que éste comience con el guardado de tramas DHCP:

1. En una consola del ruteador se ejecuta el programa de reenvío de tramas DHCP:

```
root@OpenWrt:~# ForwardDhcpFrames MAC_SERVIDOR
```

2. En otra consola dentro del ruteador, se debe iniciar el demonio `ulogd`:

```
root@OpenWrt:~# /etc/init.d/ulogd start
```

De esta forma, realizado todo lo detallado anteriormente, el programa ForwardDhcpFrames hace uso de las tramas guardadas por ulogd en el archivo FIFO /var/log/ulogd.pcap y las reenvía hacia el equipo con MAC_SERVIDOR, el cual es el servidor de nuestra solución.

5.2.2 Configuración de reglas de ebttables para el bloqueo de respuestas ARP y redireccionamiento de consultas ARP

Como se mencionó en el Capítulo 4, se realizó la configuración de redes virtuales en el ruteador para identificar a cada puerto del switch como una VLAN diferente y de esta forma poder diferenciar el puerto del switch al que se conecta el servidor del resto de puertos.

Por defecto el switch interno del ruteador tiene dos redes virtuales creadas, la VLAN0 que está formada por los puertos 0, 1, 2, 3 y 5; los cuales corresponden a los puertos externos 1, 2, 3, 4 y al puerto interno 5 del switch; y la VLAN1 formada por el puerto 4 y 5, los cuales son solo puertos internos. La vlan0 corresponde a la red local y es vista por OpenWrt como la interfaz eth0, la cual forma parte del puente br0 que mantiene con la interfaz de red inalámbrica eth2. Mientras que la VLAN1 es inutilizable. Esas son las configuraciones de redes virtuales por defecto del ruteador WRTSL54GS con OpenWrt.

Los pasos realizados para la configuración de redes virtuales en OpenWrt RC 6 fueron los siguientes:

1. Se realizó el borrado de las VLANs existentes, con los siguientes comandos:

```
root@OpenWrt:~# nvramp unset vlan0ports
root@OpenWrt:~# nvramp unset vlan0hwname
root@OpenWrt:~# nvramp unset vlan1ports
root@OpenWrt:~# nvramp unset vlan1hwname
```

2. Configuramos las nuevas redes virtuales de la siguiente forma:

```
root@OpenWrt:~# nvramp set vlan0ports="0 5"
root@OpenWrt:~# nvramp set vlan0hwname=et0
root@OpenWrt:~# nvramp set vlan1ports="1 5"
root@OpenWrt:~# nvramp set vlan1hwname=et0
root@OpenWrt:~# nvramp set vlan2ports="2 5"
root@OpenWrt:~# nvramp set vlan2hwname=et0
root@OpenWrt:~# nvramp set vlan3ports="3 5"
root@OpenWrt:~# nvramp set vlan3hwname=et0
```

3. Para conectar nuevamente los diferentes segmentos de red originados por la creación de las redes virtuales como uno solo y tener una red de área local configuramos un nuevo puente br0, el cual quedó conformado por las vlans creadas y la interfaz de red inalámbrica de la siguiente forma:

```
root@OpenWrt:~# nvramp unset lan_ifnames (antes esa variable
estaba definida como lan_ifnames=eth0 eth2)
root@OpenWrt:~# nvramp set lan_ifnames="vlan0 vlan1 vlan2 vlan3
eth2"
```

Donde vlan0, vlan1, vlan2, y vlan3 son las redes virtuales creadas en el paso 2 y corresponden a los puertos 1, 2, 3 y 4 del switch respectivamente y la interfaz eth2 es la interfaz de red

inalámbrica.

4. Finalmente para que los cambios queden guardados en la NVRAM del switch, ejecutamos el comando:

```
root@OpenWrt:~# nvram commit
```

5. Después de realizar los pasos listados anteriormente, ejecutamos el comando `ifconfig` y la salida del mismo presentaba las nuevas interfases creadas (`vlan0`, `vlan1`, `vlan2` y `vlan3`) correspondientes a cada uno de los puertos del switch, de la misma forma al ejecutar el comando `brctl show`:

```
root@OpenWrt:~# brctl show
bridge name      bridge id          STP enabled
interfaces
br0              8000.0010189020db  no           vlan0
                                                         vlan1
                                                         vlan2
                                                         vlan3
                                                         eth2
```

Pudimos observar la nueva configuración del puente `br0`, ahora conformado por las nuevas redes virtuales y la interfaz de red inalámbrica.

Con las redes virtuales una vez creadas, suponiendo que el servidor de nuestro esquema esté conectado al puerto 4 del ruteador, la regla para bloquear las respuestas ARP que no provengan de ese puerto sería:

```
etables -A FORWARD -p ARP --arp-opcode 2 -i ! vlan3 -j DROP
```

Para que ésta y las reglas que modifican el destino de las tramas de consulta ARP se ejecuten de manera automática cada vez que se

enciende el ruteador, se realizó lo siguiente:

1. Por defecto OpenWrt RC 6 no carga los módulos de ebtables necesarios para la ejecución de las reglas, por esta razón, se debe editar `/etc/modules` y añadir los módulos: `ebtables`, `ebtable_filter`, `ebtable_nat`, `ebt_arp`, `ebt_dnat`, uno en cada línea del archivo. Así cada vez que inicie el ruteador, los módulos serán cargados.
2. Editamos el archivo `/etc/firewall.user` con el comando vi:

```
root@OpenWrt:~# vi /etc/firewall.user
```

3. Y añadimos al final del mismo las reglas de ebtables:

```
ebtables -A FORWARD -p ARP --arp-opcode 2 -i !VLAN_servidor -j
DROP
ebtables -t nat -A PREROUTING -p ARP --arp-opcode 1 -s !
MAC_SERVIDOR -d ff:ff:ff:ff:ff:ff -j dnat --to-destination
MAC_SERVIDOR
ebtables -t nat -A PREROUTING -p ARP --arp-opcode 1 -s !
MAC_SERVIDOR -d ! ff:ff:ff:ff:ff:ff -j dnat --to-destination
MAC_SERVIDOR
```

Donde `VLAN_servidor` es el ID de la VLAN que corresponde al puerto al que está conectado el servidor y `MAC_SERVIDOR` es la dirección física del mismo. El archivo `firewall.user` con las reglas de ebtables incluidas se muestra en el Apéndice D.2.

5.3 Limitaciones de la implementación

Nuestra implementación tiene ciertas limitaciones que se listan a continuación:

- Al ser un esquema centralizado, si el servidor que responde las consultas ARP no está disponible, los nodos de la red de área local no podrán comunicarse.

Esta limitación es fácilmente contrarrestada si la solución completa es implementada en el ruteador, es decir si tanto el programa que actualiza la caché ARP del sistema como el que responde las peticiones ARP fueran ejecutados en él.

- Es necesario contar con un switch que permita la configuración de ebttables o que a través de listas de control de acceso (ACLs) bloquee las respuestas ARP no deseables y modifique el destino de las tramas de consulta ARP, enviándolas solo hacia el servidor.

CAPÍTULO VI.

6 PRUEBAS Y ANÁLISIS DE RESULTADOS OBTENIDOS

En este capítulo se realiza una descripción de las pruebas de funcionamiento y escenarios de ataque a los que nuestra solución fue expuesta. Se muestra a través de gráficos el impacto del rendimiento de la solución, comparando los tiempos y paquetes perdidos obtenidos entre el ruteador con las configuraciones realizadas y sin ellas.

Finalmente se describe el trabajo que debería realizarse para que la solución funcione en redes de área local más grandes.

6.1 Escenarios de ataque

Los escenarios de ataque descritos a continuación fueron desarrollados por Luis Chiang como parte de su proyecto de tesis. Estos ataques nos ayudaron con la realización de las pruebas de nuestra solución:

- **Escenario 1**

En este escenario se realizó el envío de 100 respuestas ARP envenenadas. La respuesta ARP formada por el atacante consistía en una trama ARP, cuyo campo dirección IP origen podía tener como contenido la dirección lógica de una de las computadoras de la red local o una dirección lógica cualquiera, pero su campo dirección física origen correspondía a la dirección MAC del atacante. El resultado de realizar este ataque contra nuestra solución no fue exitoso.

- **Escenario 2**

En este ataque se enviaron 100 consultas ARP envenenadas. Una consulta ARP envenenada consiste en que el campo dirección física origen y el campo dirección IP origen de la trama ARP no corresponden al mapeo verdadero. La dirección IP podía hacer referencia a una de las máquinas de la red local o a una dirección IP cualquiera, pero la dirección MAC pertenecía a la del

atacante. La ejecución de este ataque no fue exitosa cuando lo probamos contra nuestra solución.

- **Escenario 3**

Durante este ataque el nodo malintencionado espera que una computadora de la red local realice una consulta ARP, así la máquina atacante al recibir la consulta envía 100 respuestas ARP envenenadas. Este ataque no fue exitoso mientras nuestra solución se ejecutaba.

- **Escenario 4**

El cuarto escenario de ataque consistió en el envío de paquetes de petición Echo ICMP falsos, es decir emitidos desde el atacante ya sea con una dirección lógica perteneciente a una de las máquinas de la red local, o una dirección IP cualquiera. Al enviar la petición el atacante espera a que el nodo hacia el cual se dirige el ICMP, consulte por la dirección física correspondiente a la dirección lógica del remitente a través de la consulta ARP, e inmediatamente luego de esto envía 100 respuestas ARP envenenadas. Este ataque tampoco fue exitoso durante la ejecución de la solución.

- **Escenario 5**

Este escenario es muy parecido al anterior con la excepción de que luego de enviar la petición Eco ICMP (pings) falsa el atacante no espera por la consulta ARP, en su lugar realiza el envío de 100 respuestas ARP envenenadas inmediatamente luego de la emisión del ICMP. Igualmente el resultado de realizar este ataque contra nuestra solución no fue exitoso.

- **Escenario 6**

A diferencia de los dos escenarios anteriores, en éste se realiza el envío de mensajes de respuesta Eco ICMP (pings) falsos, pero en estos mensajes la consulta ARP realizada por el atacante preguntando por la dirección física hacia la cual se dirige su respuesta es enviada de manera unicast y es una consulta ARP envenenada. Gracias al redireccionamiento de las tramas de consulta ARP ya tengan éstas destino broadcast o unicast hacia el servidor, y que éste impide que su caché ARP sea actualizada a través de ellas, este ataque tampoco resultó exitoso mientras la solución se ejecutaba.

- **Escenario 7**

Este ataque se diferencia del anterior en que se realiza el envío

de peticiones Eco ICMP (pings) falsas, cuyas consultas ARP envenenadas son igualmente que en el escenario anterior enviadas de manera Unicast. Este ataque tampoco cumplió con su objetivo de envenenar la caché ARP del nodo atacado.

6.2 Pruebas de funcionamiento

Para las pruebas de funcionamiento se utilizaron cuatro computadoras: una de ellas el servidor que actualiza la caché ARP y responde las consultas ARP realizadas por las máquinas de la red local, otra, el computador atacante y las otras dos computadoras, los nodos cuyas cachés ARP debían ser envenenadas, una con Ubuntu 7.04 y otra con Windows XP.

Los pasos realizados para la ejecución de nuestra solución fueron los siguientes:

1. Inicialmente todas las máquinas excepto el servidor estaban desconectados del ruteador.
2. Se ejecutó en el ruteador el programa que reenvía las tramas DHCP hacía el servidor y se inició el demonio ulogd.
3. En el servidor se ejecutaron el programa que actualiza la caché ARP y el que responde las consultas ARP.
4. Realizado lo anterior se conecta el resto de los nodos que

forman parte de la red local.

Una vez iniciados los programas que conforman nuestro esquema, éste fue probado con los ataques descritos en el punto 6.1. Se obtuvo en cada uno de ellos un resultado exitoso por parte de la solución, en donde el sistema operativo de la computadora que debía ser atacada varió entre Windows y Ubuntu.

De esta forma a través de los distintos escenarios de ataque a los que se expuso nuestra solución e impidiendo con ella que los ataques cumplan su objetivo, concluimos que nuestro esquema contrarresta todos los tipos de ataques ARP existentes.

Adicionalmente como parte de las pruebas de funcionamiento, se intentó llevar a cabo un ataque de hombre en el medio entre el ruteador y el servidor, intentando de esta forma envenenar sus cachés ARP.

El ataque de hombre en el medio es una técnica en la que el enemigo adquiere la capacidad de leer, insertar y modificar a voluntad, los mensajes entre dos partes sin que ninguna de ellas conozca que el enlace entre ellos ha sido violado. El atacante debe ser capaz de observar e interceptar mensajes entre los dos nodos [31]. Esto lo realiza el atacante a través del envenenamiento ARP de ambas víctimas.

Para lograrlo, hicimos uso de la aplicación Cain & Abel [32], la cual es un olfateador que permite ejecutar este ataque de manera muy sencilla enviando una serie de tramas ARP envenenadas.

La Figura 6.2-1 muestra el momento en que Cain & Abel busca los computadores que forman parte de la red local, lo cual lo realiza a través de consultas ARP, obteniendo finalmente una tabla de correspondencias entre las direcciones IP y las direcciones físicas de los nodos de la red. Para la búsqueda se especificó el rango de direcciones IP:192.168.1.1 - 192.168.1.254.

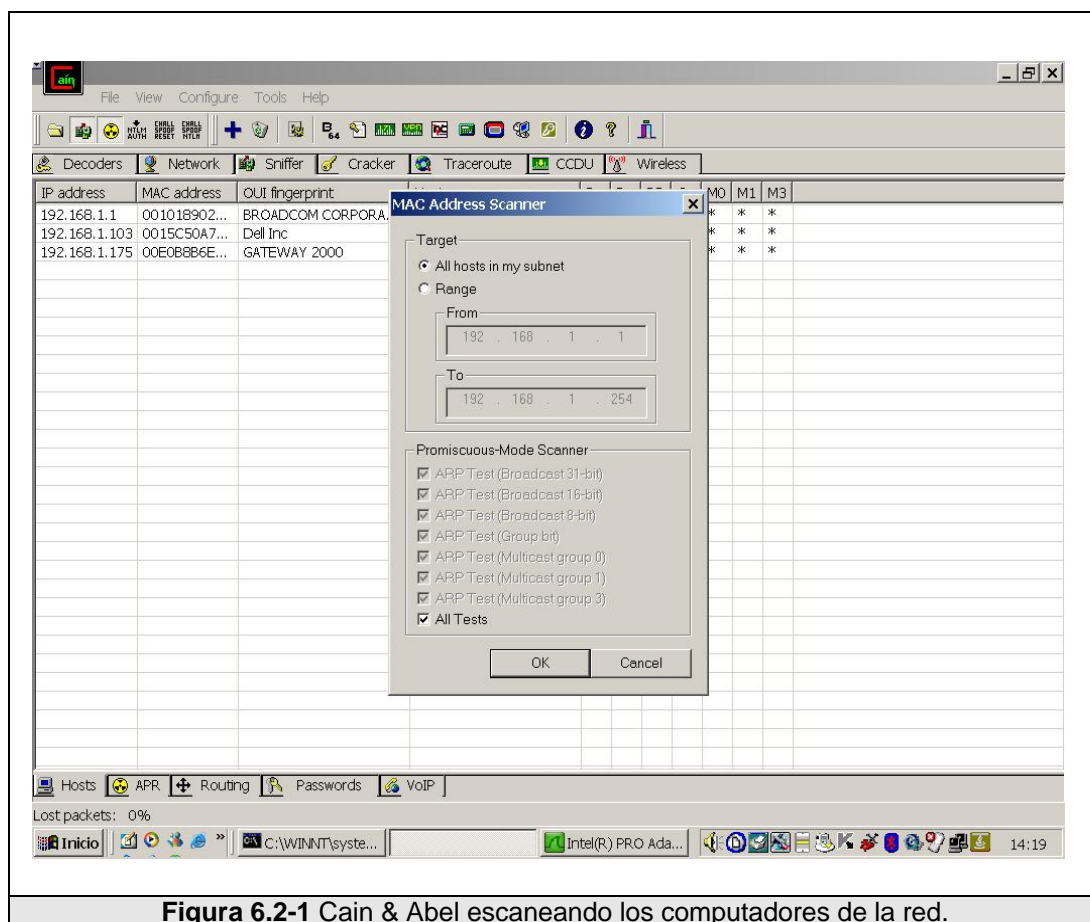


Figura 6.2-1 Cain & Abel escaneando los computadores de la red.

Hasta este punto con Cain & Abel se conocen cuales son los computadores que se encuentran conectados en la red de área local. En la Figura 6.2-2 podemos observar en el servidor todas las consultas ARP que generó Cain & Abel, las cuales son redireccionadas únicamente hacia él a través de las reglas de iptables colocadas en el ruteador. Como se puede apreciar, la dirección física de las peticiones no es broadcast; es la dirección MAC que corresponde al servidor.

Y en la Figura 6.2-3 se observa la salida de ejecución del programa `send_arp_reply`, que muestra que no encuentra las direcciones físicas de ciertas direcciones IP preguntadas por Cain & Abel ya que no forman parte de la red, es así que nunca se transmitieron las tramas DHCP correspondientes a esas direcciones y por esta razón la caché ARP del servidor no contiene esos mapeos y no puede responder.

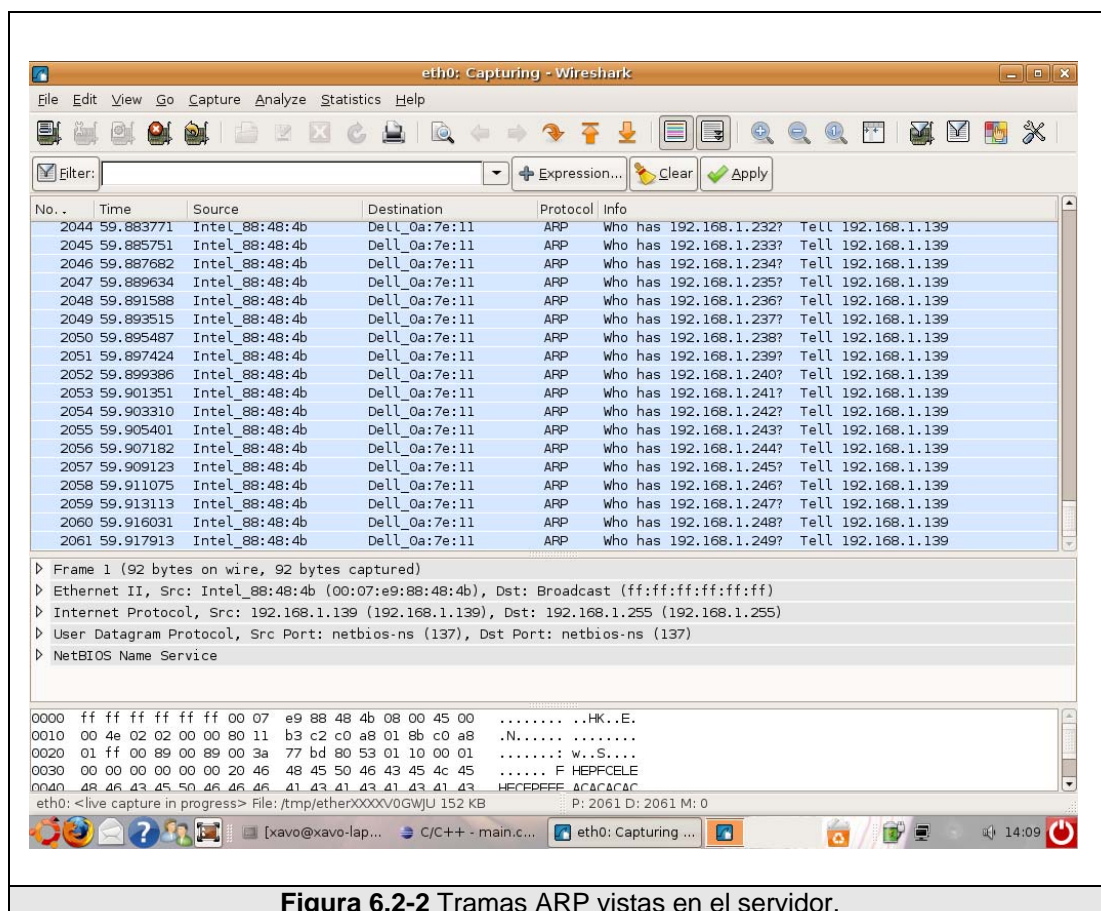


Figura 6.2-2 Tramas ARP vistas en el servidor.

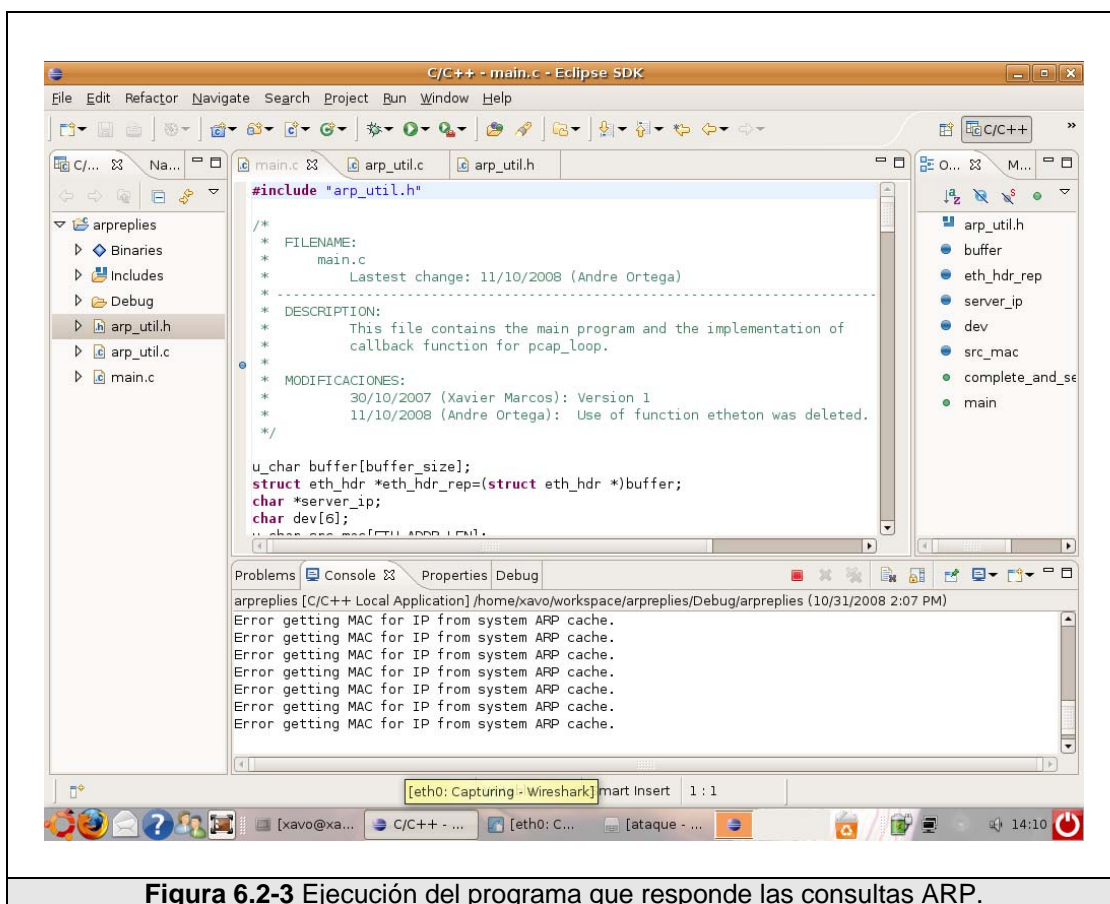
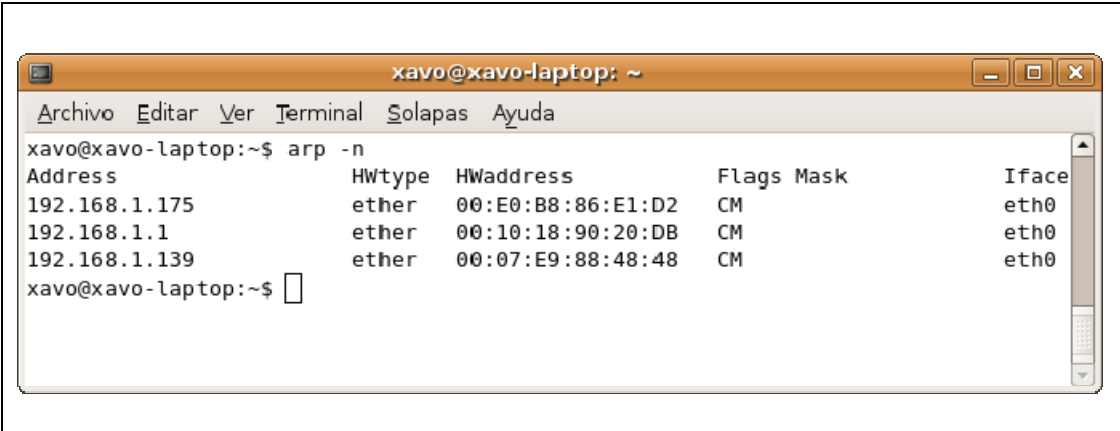


Figura 6.2-3 Ejecución del programa que responde las consultas ARP.

Antes de realizar el ataque de hombre en el medio, chequeamos la caché ARP de nuestro servidor como se puede observar en la Figura 6.2-4.



```
xavo@xavo-laptop: ~  
Archivo Editar Ver Terminal Solapas Ayuda  
xavo@xavo-laptop:~$ arp -n  
Address          HWtype  HWaddress      Flags Mask    Iface  
192.168.1.175    ether    00:E0:88:86:E1:D2  CM           eth0  
192.168.1.1      ether    00:10:18:90:20:DB  CM           eth0  
192.168.1.139    ether    00:07:E9:88:48:48  CM           eth0  
xavo@xavo-laptop:~$
```

Figura 6.2-4 Caché ARP del servidor.

En la caché se encuentran tres entradas que corresponden a dos computadores de la red: 192.168.1.175, un nodo cualquiera; 192.168.1.139, el atacante y 192.168.1.1, el ruteador.

Posteriormente en el computador atacante procedemos a seleccionar los computadores víctimas. En este caso queremos hacer un ataque entre el ruteador y el servidor como se muestra en la Figura 6.2-5, donde la dirección IP del ruteador corresponde a 192.168.1.1 y la del servidor a 192.168.1.103.

Con la selección de la dirección IP del ruteador se husmeará toda la red, ya que ésta contiene al resto de IPs y es el lugar por donde pasan todas las comunicaciones. De esta forma si se logra el ataque se suplantaría su identidad, es decir se tendría en la caché ARP del servidor, la dirección IP del ruteador pero con la dirección MAC del atacante como su correspondencia y de la misma forma en la caché ARP del servidor, se tendría la dirección IP del ruteador pero

mapeada con la dirección física del atacante.

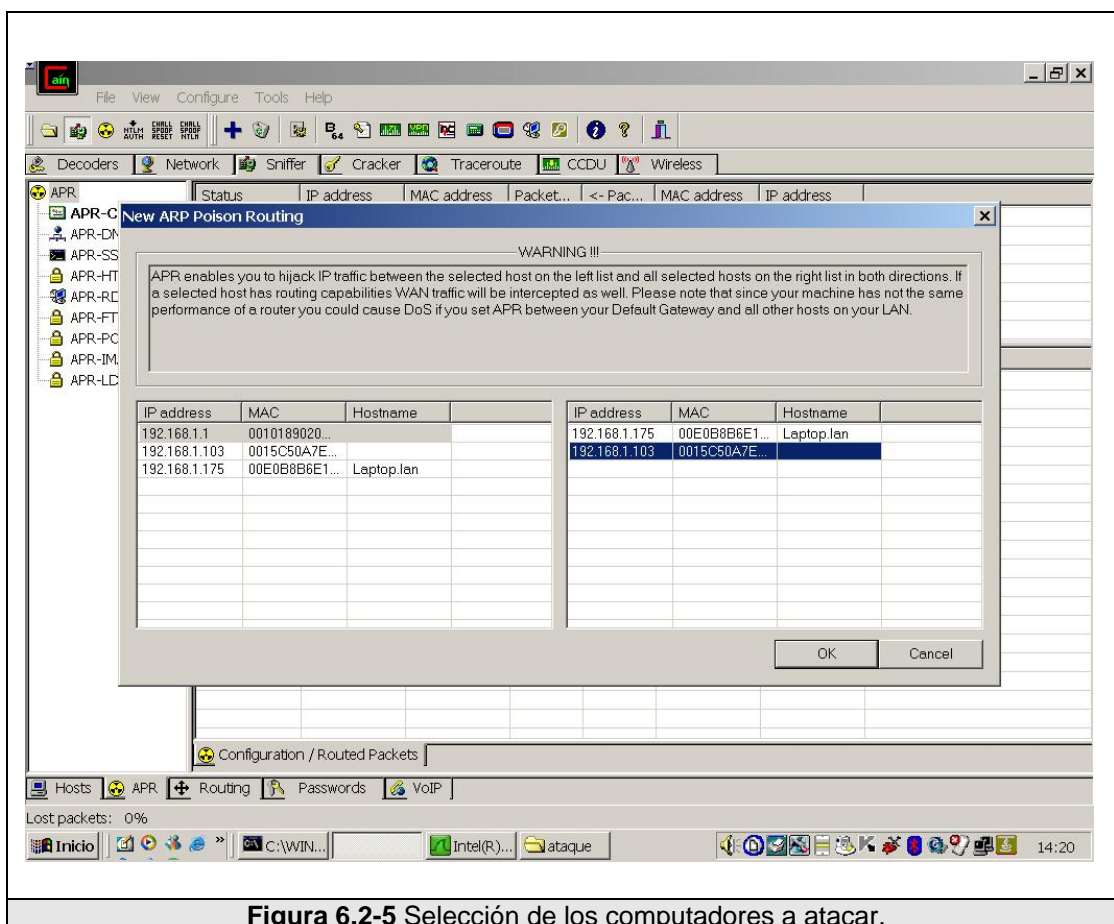


Figura 6.2-5 Selección de los computadores a atacar.

Una vez que seleccionamos los computadores entre los cuales la comunicación será interceptada por el hombre en el medio (computadora atacante), realizamos el ataque e intentamos envenenar sus cachés ARP como se ve en la Figura 6.2-6.

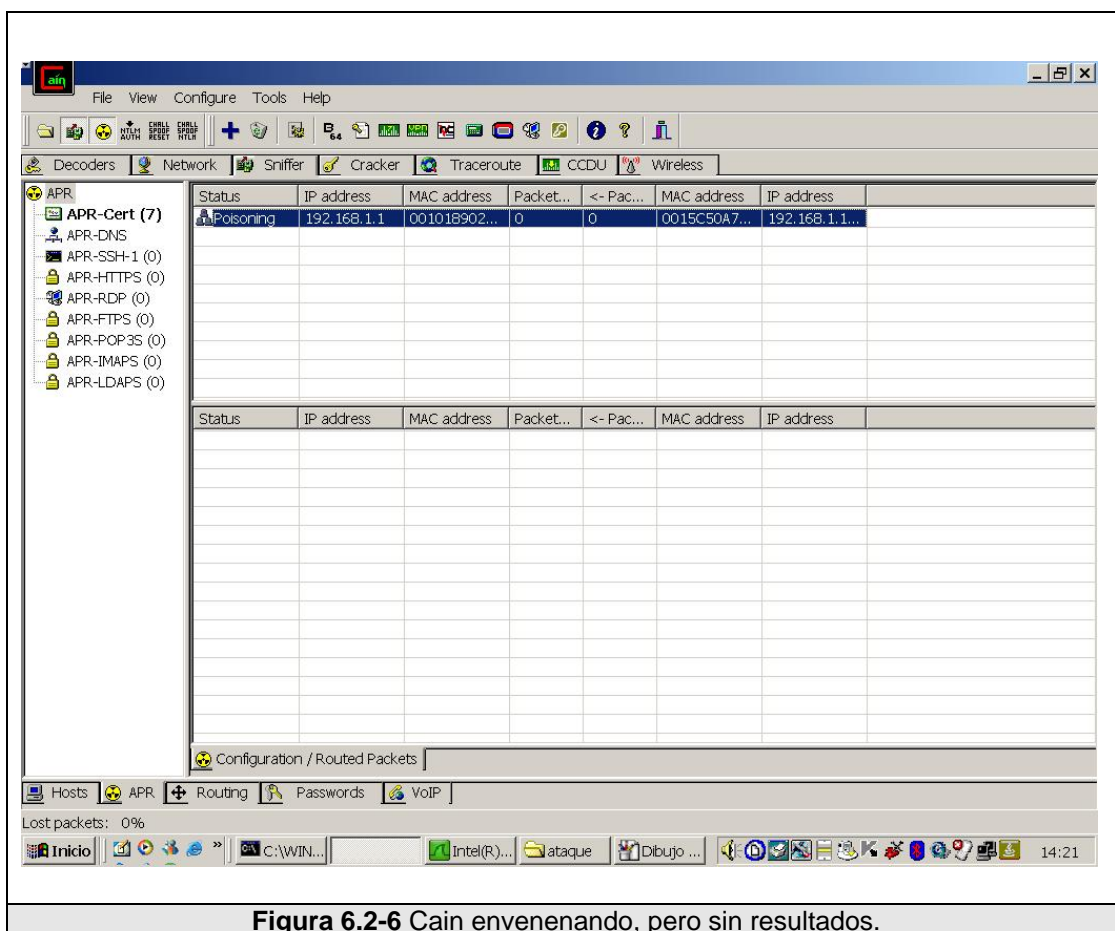


Figura 6.2-6 Cain envenenando, pero sin resultados.

En esta imagen se puede observar como la aplicación está intentando envenenar las cachés sin embargo no lo logra.

En la Figura 6.2-7 mostramos el comportamiento de Cain & Abel cuando el envenenamiento es exitoso, para compararlo con el envenenamiento fallido que obtenemos con nuestra solución.

Como se puede ver con la solución ejecutándose no se llega a enrutar ningún paquete como sucede cuando si se logra el envenenamiento.

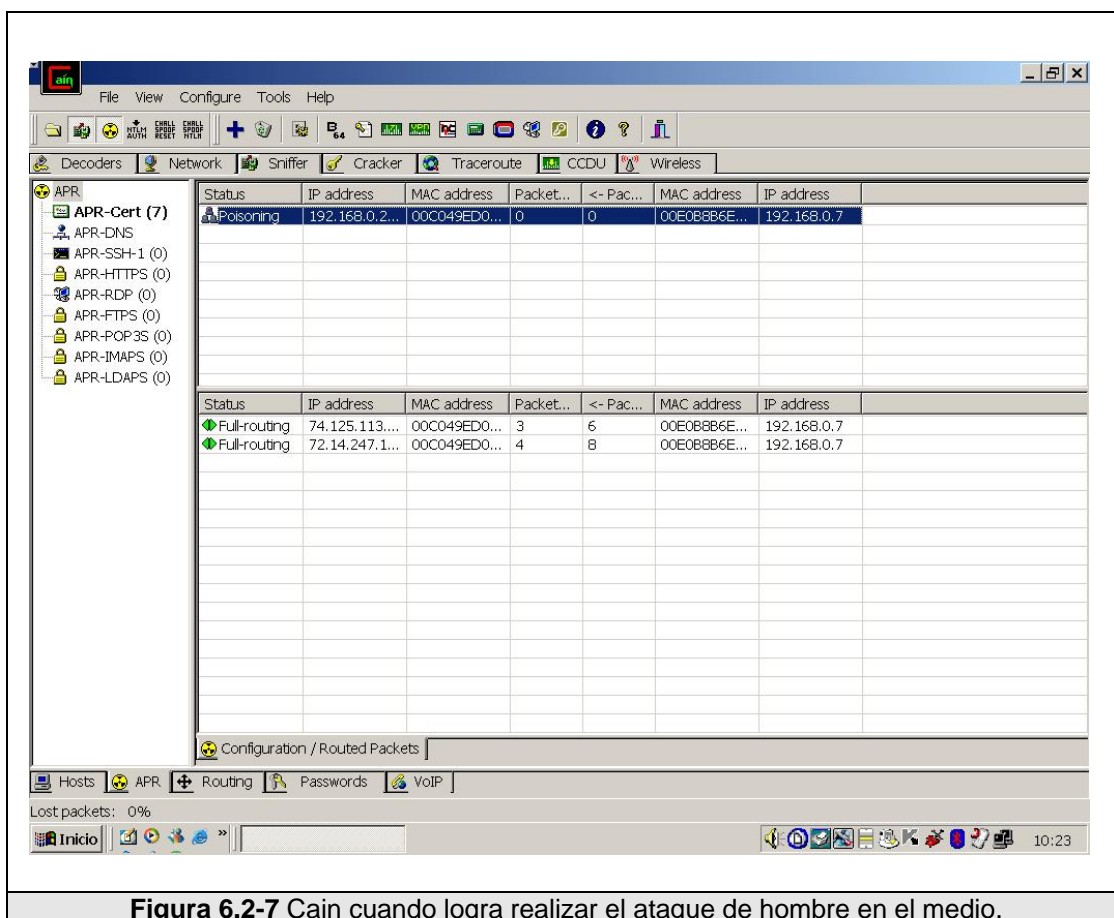
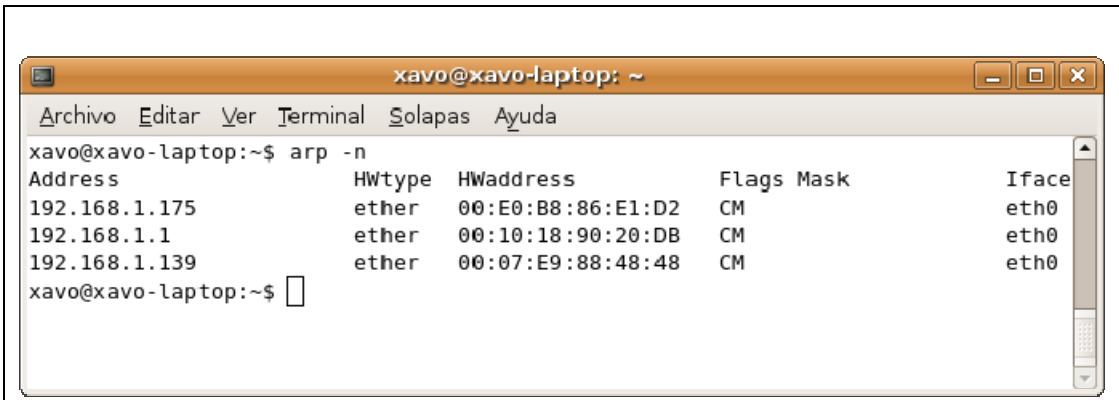


Figura 6.2-7 Cain cuando logra realizar el ataque de hombre en el medio.

Por último, una vez que intentamos envenenar con Cain & Abel, en la Figura 6.2-8 revisamos nuevamente la caché ARP del servidor para verificar que ésta no haya sido envenenada.



```
xavo@xavo-laptop: ~  
Archivo Editar Ver Terminal Solapas Ayuda  
xavo@xavo-laptop:~$ arp -n  
Address          HWtype  HWaddress      Flags Mask    Iface  
192.168.1.175    ether   00:E0:88:86:E1:D2  CM           eth0  
192.168.1.1      ether   00:10:18:90:20:DB  CM           eth0  
192.168.1.139   ether   00:07:E9:88:48:48  CM           eth0  
xavo@xavo-laptop:~$
```

Figura 6.2-8 Caché ARP del servidor.

Y como lo esperábamos, la caché de nuestro servidor se mantuvo igual que antes de realizar el ataque. Ninguna entrada se vio afectada por los intentos de envenenamiento que Cain & Abel realizó. Esta prueba puede ser realizada entre los distintos computadores que conforman la red local, para intentar interceptar la comunicación entre ellos y acceder a la información que transmiten, pero de igual forma con la solución ejecutándose, este intento será fallido.

6.3 Impacto del rendimiento

Para calcular el impacto en el rendimiento de la red de área local de nuestra solución, se utilizaron como parámetros de medición: el tiempo y el número de paquetes.

Las gráficas mostradas a continuación comparan en cada caso 3 escenarios:

1. Ruteador solo con OpenWrt, sin ninguna configuración realizada.
2. Ruteador con VLANs creadas.
3. Solución ejecutándose.

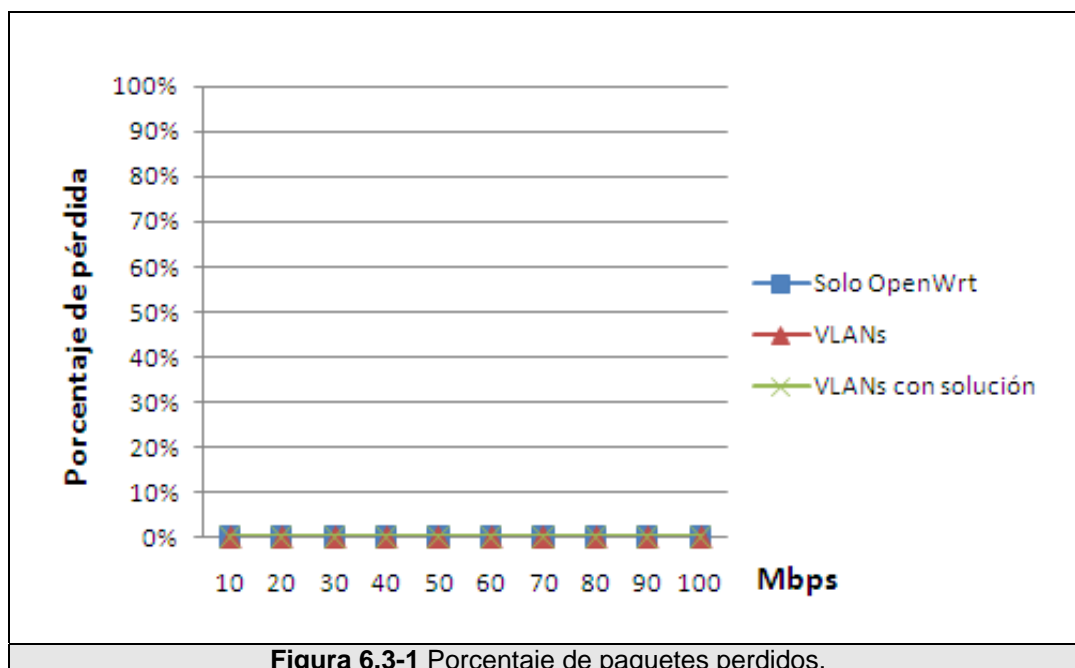
Las gráficas ilustran los siguientes resultados obtenidos:

1. Porcentaje de paquetes perdidos.
2. Medición de tiempo de recepción de respuestas ARP.
3. Medición de tiempo de transmisión de paquetes.

▪ **Porcentaje de paquetes perdidos**

Inicialmente se realizaron las pruebas para obtener el porcentaje de paquetes perdidos en la red para los tres escenarios.

Los resultados se exponen a continuación a través de la Figura 6.3-1.



Donde podemos observar que la ejecución de nuestra solución no produjo pérdida alguna de paquetes.

Los valores utilizados para la obtención de la gráfica fueron el resultado de la aplicación de la siguiente fórmula:

$$\frac{x \text{ Mbps} * \frac{1024 \text{ bits}}{1 \text{ Mbit}}}{\frac{n \text{ bytes} * 8 \text{ bits}}{1 \text{ byte}}} = \frac{\text{Número de paquetes}}{s}$$

Donde x Mbps corresponde al ancho de banda de la conexión, la cual varía desde 10 Mbps hasta 100 Mbps y n bytes es la cantidad de bytes que posee un paquete. Con la división de estos valores se obtuvo el número de paquetes que se debían enviar para cada ancho

de banda correspondiente. Los paquetes enviados fueron de tipo petición eco ICMP (pings). Se realizaron 10 pruebas de envío por cada Número de paquetes, así es que para 10Mbps, y 98 bytes que posee el paquete, el número de peticiones eco ICMP enviadas fueron 13 y se realizaron 10 pruebas de envío de 13 paquetes cada uno, es decir para 10Mbps se enviaron 130 paquetes. En cada una de las pruebas se comprobó si existían paquetes perdidos y se calculó su porcentaje.

Como la gráfica lo refleja, en ninguno de los casos se perdió paquetes eco ICMP.

El número de paquetes enviados por cada prueba realizada se encuentra en el Anexo E.1.

▪ **Medición de tiempo de recepción de respuestas ARP**

De la misma forma se realizó una gráfica en la que se compararon los 3 escenarios listados anteriormente. Los resultados se obtuvieron del envío de 40 paquetes de petición eco ICMP.

Cuando un computador desea enviar a través de la utilidad ping paquetes de petición eco ICMP a otro nodo en la red de área local o cuando desea transmitir cualquier otro tipo de paquete, inicialmente necesita conocer la dirección física destino, es así que si no posee el mapeo previamente en su caché, realiza una consulta ARP.

Por cada ping realizado de la siguiente forma desde un computador a otro con dirección IP_DESTINO:

```
nombre_usuario@nombre_maquina:~# ping IP_DESTINO -c 1
```

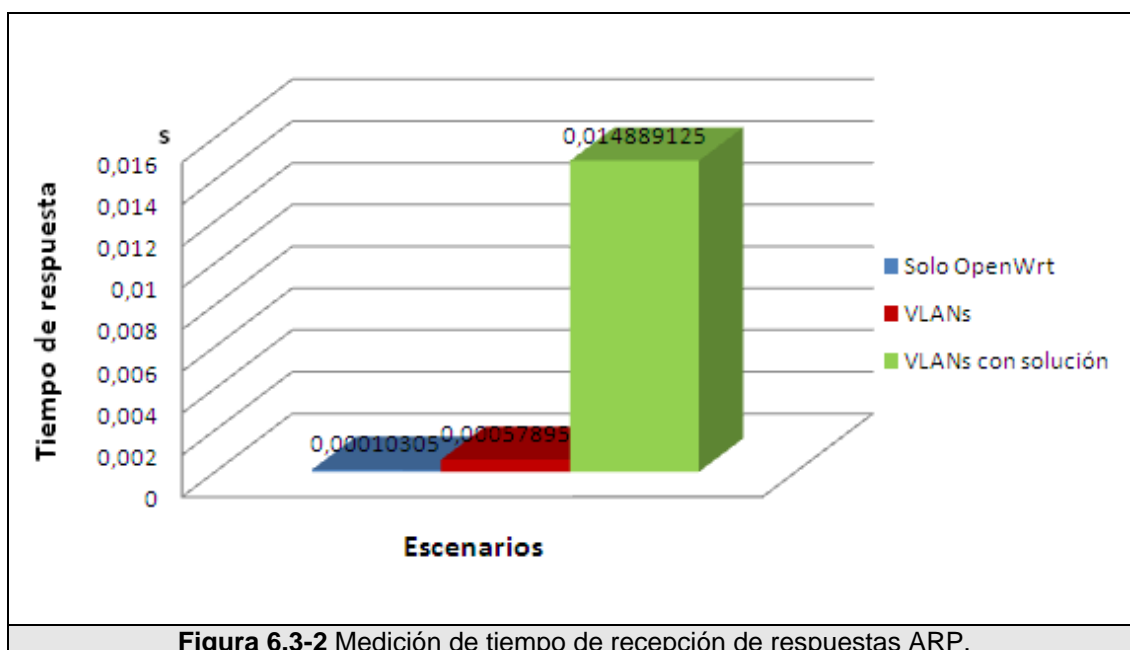
Se forzó al computador origen a realizar una consulta ARP a través del comando arp:

```
nombre_usuario@nombre_maquina:~# arp -d IP_DESTINO
```

El cual elimina el mapeo IP – MAC de la dirección IP_DESTINO.

El tiempo de respuesta de la consulta ARP realizada se obtenía a través del olfateador Wireshark. Los tiempos de recepción de cada trama de respuesta ARP pueden ser observados en el Apéndice E.2.

Los resultados se muestran en la Figura 6.3-2:



Se puede observar que la configuración de VLANs incrementa en un 462% el tiempo en que una respuesta ARP llega a los nodos con

referencia al resultado promedio de tiempo obtenido con OpenWrt sin VLANs creadas. Este porcentaje se refleja en el tiempo promedio de respuestas ARP de la solución, sin embargo este retardo solo representa el 3.89% del retardo total. El resto de porcentaje se divide entre distintos factores.

Para determinar las causas del elevado tiempo obtenido, se observaron los tiempos de procesamiento de las consultas ARP en el servidor a través de Wireshark y se compararon con los tiempos de respuesta que ve el nodo cliente que realiza la consulta. Los resultados obtenidos se muestran a continuación:

Tiempo cliente (s)	Tiempo servidor (s)	Diferencia (s)
0,012287	0,011736	0,000551
0,012	0,011598	0,000402
0,016068	0,015707	0,000361
0,011889	0,011486	0,000403
0,015736	0,015334	0,000402
0,015551	0,015144	0,000407
0,01937	0,01896	0,00041
0,015	0,014583	0,000417
0,01505	0,014643	0,000407
0,0229	0,022496	0,000404
0,022754	0,022361	0,000393
0,010412	0,010004	0,000408
0,014202	0,013806	0,000396
0,018069	0,017668	0,000401
0,017897	0,017506	0,000391
0,017705	0,017318	0,000387
0,013586	0,013195	0,000391
0,01343	0,013024	0,000406
0,017247	0,016851	0,000396
0,013089	0,012703	0,000386

0,012949	0,012552	0,000397
0,016798	0,016393	0,000405
0,020648	0,02024	0,000408
0,016464	0,016067	0,000397
0,020292	0,019938	0,000354
0,016126	0,01575	0,000376
0,019969	0,019581	0,000388
0,019813	0,019415	0,000398
0,015647	0,015249	0,000398
0,015436	0,015057	0,000379
0,019289	0,018916	0,000373
0,019123	0,01872	0,000403
0,014963	0,01457	0,000393
0,018799	0,018394	0,000405
0,014518	0,014121	0,000397
0,014343	0,013952	0,000391
0,014187	0,013798	0,000389
0,014028	0,013631	0,000397
0,01784	0,017455	0,000385
0,013659	0,013266	0,000393

Tabla 6.3-1 Tiempos de respuesta ARP.

Los cuales confirman que el mayor retardo se produce en el servidor, específicamente por el programa `send-arp_reply`, encargado de responder las consultas ARP, puesto que en estos resultados no interviene el programa `update_arp_cache`.

El programa `send_arp_reply` fue implementado con lenguaje C, haciendo uso principalmente de las librerías `libpcap` y `dumbnet`.

Con respecto al bajo rendimiento, teníamos las siguientes hipótesis:

- La búsqueda de direcciones físicas en la caché ARP a

través de dumbnet en el programa `send_arp_reply` es ineficiente.

- La librería `libcap` es ineficiente.

La primera hipótesis fue descartada cuando de nuestro código separamos la función `get_MAC_from_IP`, que a través de la función `arp_get` de dumbnet busca en la caché ARP del sistema la dirección física que corresponde a la dirección IP enviada como uno de sus argumentos. De reemplazo colocamos una dirección estática como respuesta a una consulta. A pesar de lo realizado, se obtuvieron los mismos tiempos.

La segunda hipótesis fue corroborada cuando encontramos la siguiente información [33]:

“Libpcap sufre de grandes problemas de eficiencia sobre implementaciones de colección de tráfico al nivel de usuario. Libpcap debe solicitar al sistema operativo ejecutar una copia del paquete requerido en la pila de red (por transparencia), lo cual puede duplicar el tiempo necesario para procesar un paquete. El método exacto utilizado por libpcap y otras herramientas varía en función del sistema operativo, pero siempre implica un cambio de contexto entre el modo de núcleo y modo usuario y una copia de memoria desde el núcleo hasta la librería de nivel de usuario. Esta “llamada-y-copia” es

repetida por cada paquete escogido.”

La ineficiencia de la librería libpcap no estuvo prevista al inicio de la implementación de la solución.

El bajo rendimiento obtenido en esta métrica podría ser evitado haciendo una implementación de nuestro sistema que trabaje en modo núcleo.

En un estudio realizado al tráfico de una red de 5 computadoras se obtuvo que ARP representa aproximadamente el 0.06% del tráfico total, es así que la demora amortizada a todo el tráfico no resulta muy grave.

El estudio del tráfico de una red de 5 computadores fue realizado durante 5 días, analizándolo a través de una herramienta libre llamada ntop¹. La red a la que se hace referencia fue creada en el departamento de Operaciones de Telmex, teniendo la posibilidad de sondear el tráfico común que generan 5 ingenieros a diario. El tráfico fue monitoreado desde las 10:00 hasta las 17:30 de cada día. Los gráficos mostrados a continuación representan el tráfico total de ese horario.

Las 5 computadoras fueron conectadas a un hub, el cual a su vez estaba conectado a un ruteador que proveía el Internet. Ntop permite

¹ Es una herramienta utilizada para sondear el uso de tráfico de una red, similar al popular comando UNIX top, usada a través de una interfaz Web.

el filtrado de paquetes a través de reglas cuya sintáxis usa las mismas expresiones BPF (Berkeley Packet Filter), comunes por otras herramientas como tcpdump. Es así que se definió la siguiente regla para sondear únicamente el tráfico generado por las 5 computadoras:

```
src IP1 or src IP2 or src IP3 or src IP4 or src IP5 or dst IP1 or dst IP2  
or dst IP3 or dst IP4 or dst IP5
```

Donde IP1, IP2, IP3, IP4 e IP5 corresponden a las direcciones de red de los 5 nodos.

Los resultados obtenidos a través de ntop durante 5 días de sondeo se representan en las siguientes gráficas:

Día 1: 17:30

Total de paquetes recibidos:	1,384,940
Total de paquetes procesados:	1,384,940

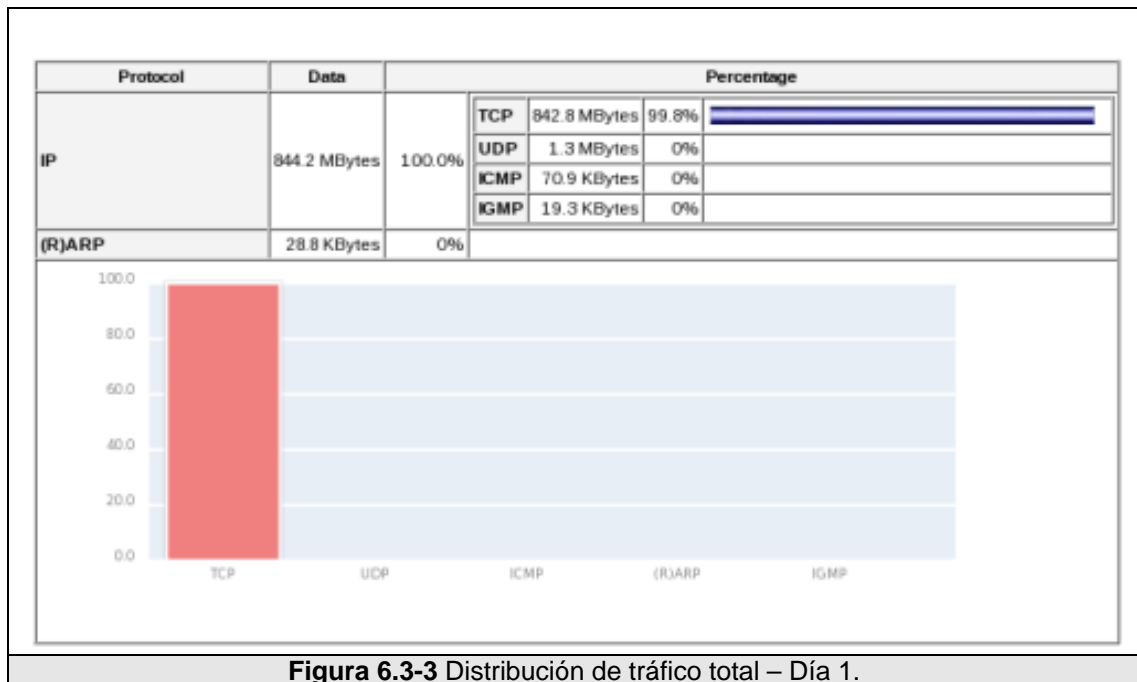


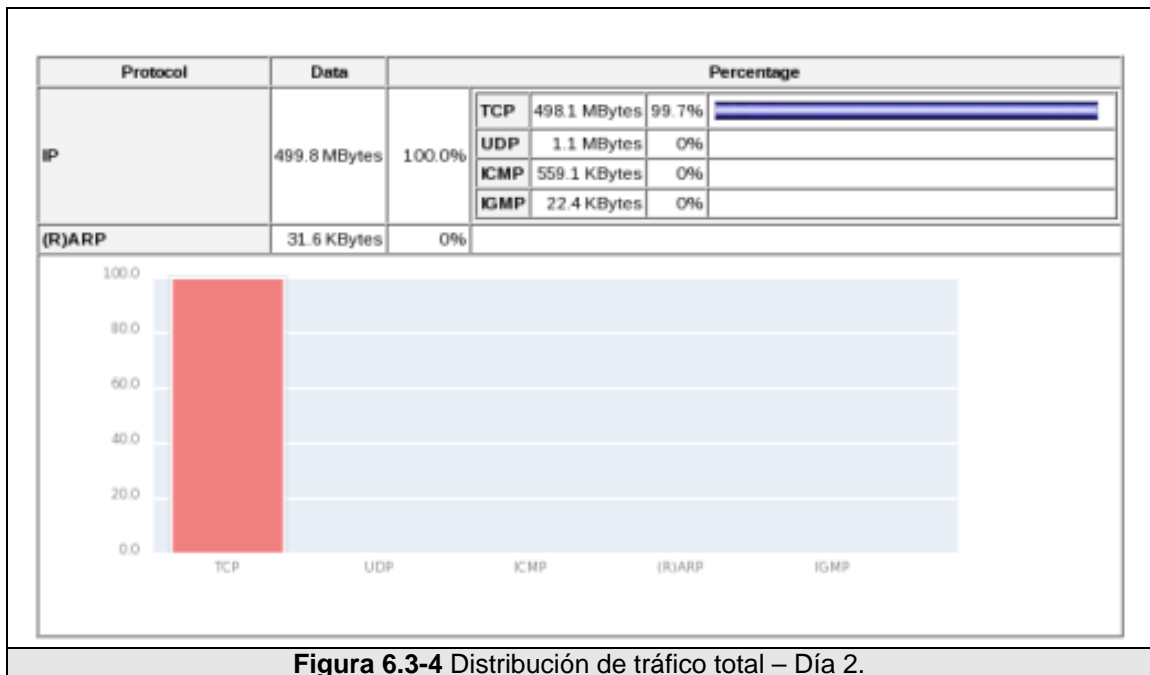
Figura 6.3-3 Distribución de tráfico total – Día 1.

Como los datos obtenidos por ntop lo indican, el número total de paquetes procesados fue de 1'384.940. De los cuales 844,2 MBytes correspondieron a paquetes IP y 28,8 KBytes se distribuyeron entre tramas ARP y RARP, es decir que del número total de tramas ARP si consideramos que el tamaño de una trama ARP es de 42 bytes, fue de 686 tramas R(ARP) de un total de 1'384.940 paquetes, lo que corresponde al 0.049% del tráfico.

El mismo análisis se realizó para el resultado de los demás días.

Día 2: 17:30

Total de paquetes recibidos:	956,075
Total de paquetes procesados:	956,075



Total de paquetes recibidos: **956,075**
Tráfico R(ARP): **31,6 KBytes**
Porcentaje de tramas ARP: **0,078%**

Día 3: 17:30

Total de paquetes recibidos: **1,058,426**
Total de paquetes procesados: **1,058,426**

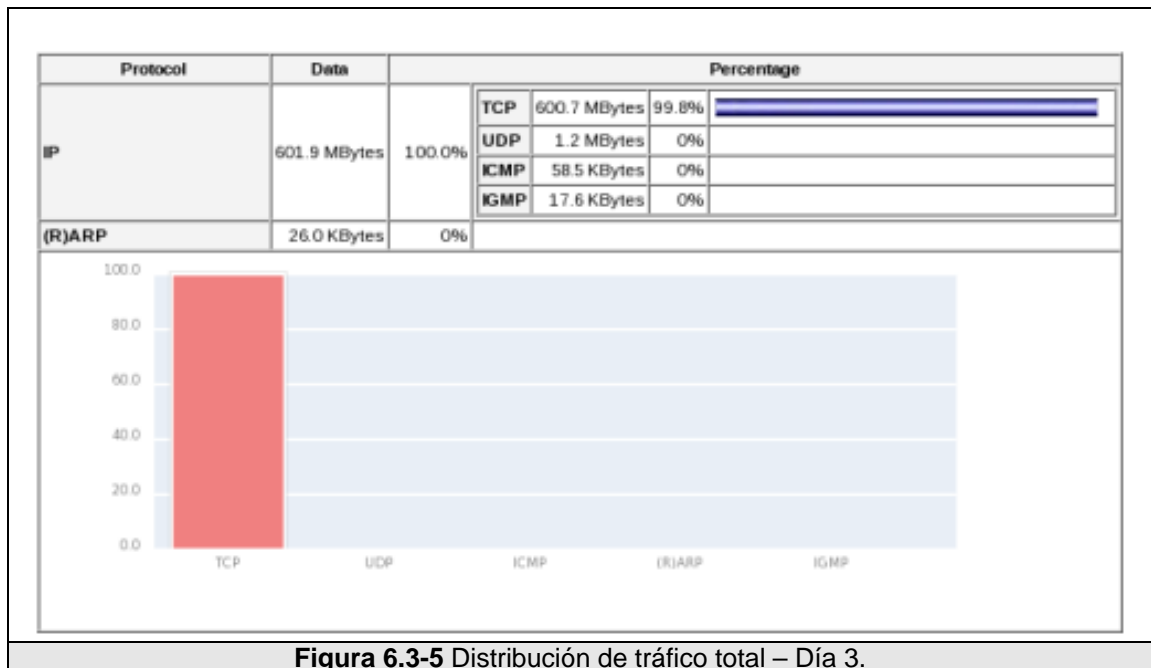


Figura 6.3-5 Distribución de tráfico total – Día 3.

Total de paquetes recibidos: **1,058,426**
 Tráfico R(ARP): **26,0 KBytes**
 Porcentaje de tramas ARP: **0,058%**

Día 4: 17:30

Total de paquetes recibidos: **1,308,394**
 Total de paquetes procesados: **1,308,394**

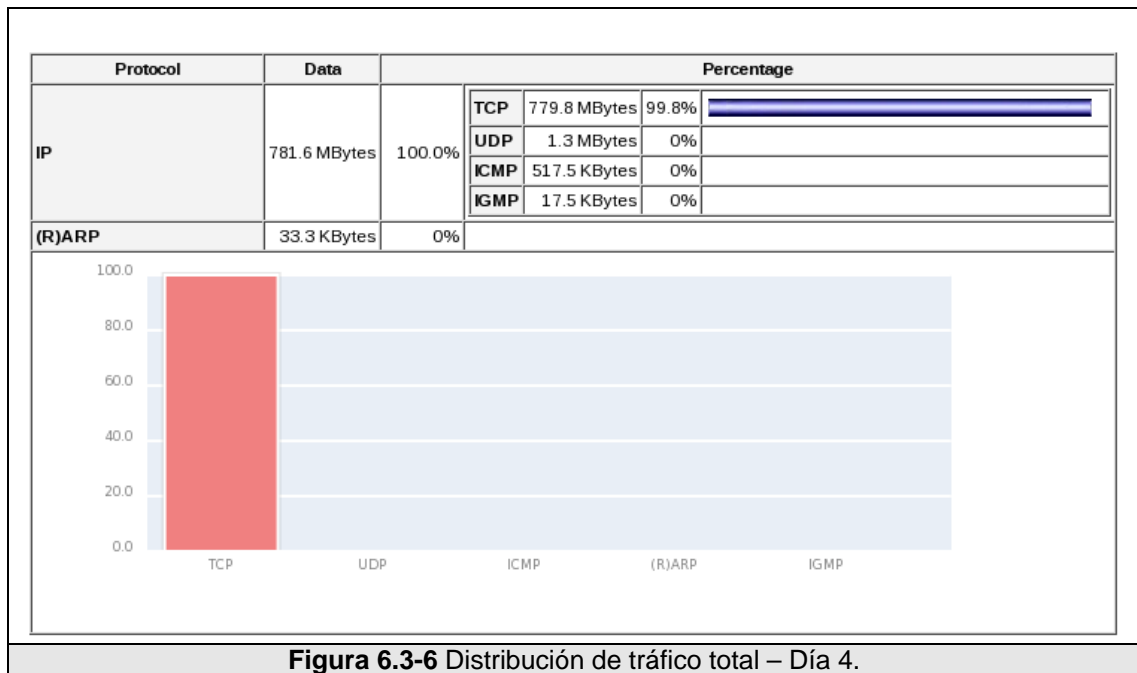


Figura 6.3-6 Distribución de tráfico total – Día 4.

Total de paquetes recibidos: **1,308,394**
 Tráfico R(ARP): **33,3 KBytes**
 Porcentaje de tramas ARP: **0,060%**

Día 5: 17:30

Total de paquetes recibidos: **1,233,874**
 Total de paquetes procesados: **1,233,874**

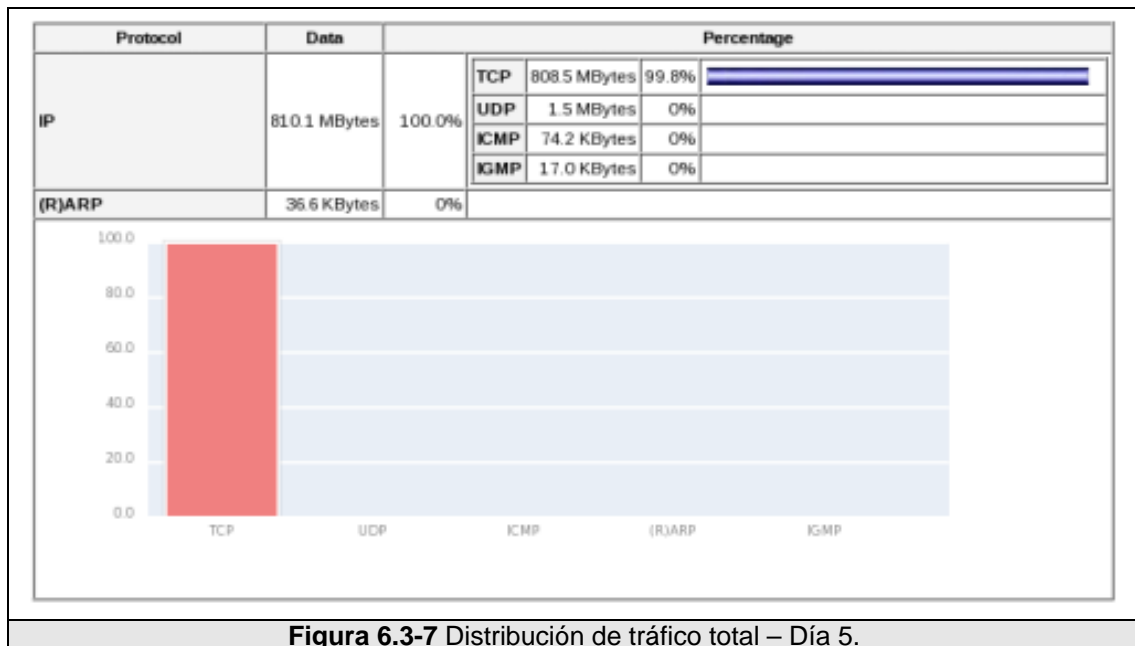


Figura 6.3-7 Distribución de tráfico total – Día 5.

Total de paquetes recibidos:	1,233,874
Tráfico R(ARP):	36,6 KBytes
Porcentaje de tramas ARP:	0,070%

Es así, que el resultado obtenido del tiempo en que una respuesta ARP llega a un nodo que realiza una consulta, no es considerado crítico, ya que únicamente las consultas ARP son afectadas y en una red normal, el porcentaje de tráfico ARP puede ser considerado despreciable.

- **Medición de tiempo de transmisión de paquetes**

Para la obtención de este tiempo, se realizó el envío de 40 paquetes de petición eco ICMP. El tiempo considerado para la realización de las gráficas, es el tiempo en el que la respuesta eco ICMP llega a la

computadora que realiza la petición.

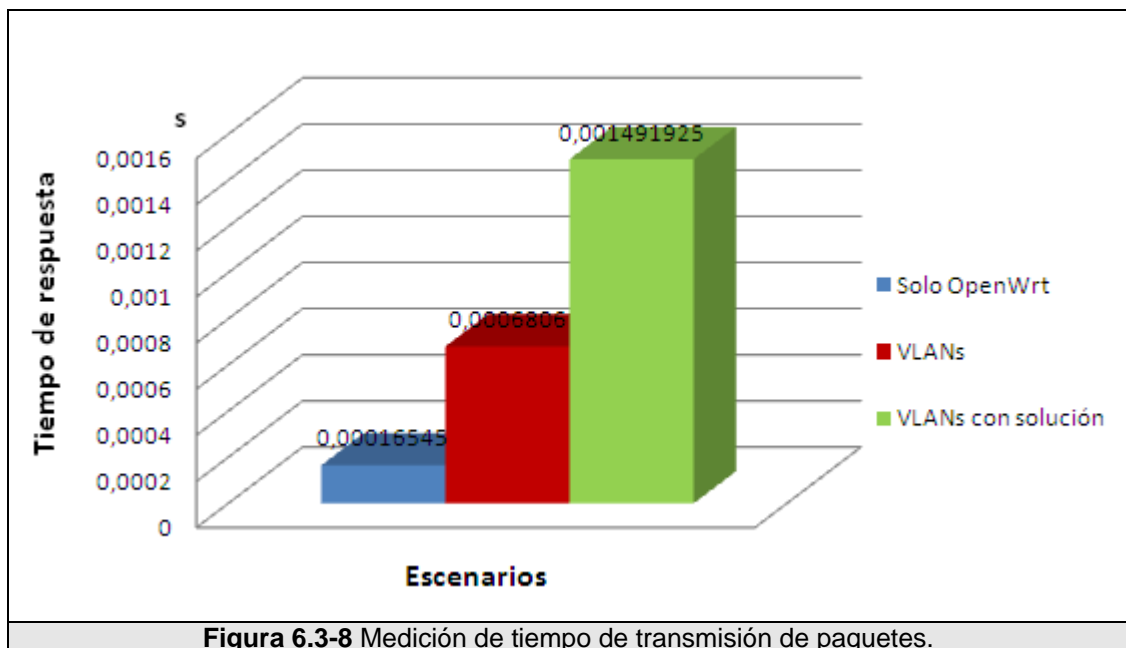
A través de la gráfica vemos como el tiempo de transmisión de los paquetes ICMP se ve afectado por el primer valor de tiempo obtenido durante las pruebas. Valor igual a 0,033203 segundos.

Los tiempos de recepción de cada paquete de respuesta eco ICMP pueden ser observados en el Apéndice E.3.

La primera vez que se envía el paquete ICMP, el computador origen realiza una consulta ARP para conocer la dirección física destino, debido a esta consulta realizada, el valor 0,033203 segundos es un tiempo grande en comparación al resto de valores obtenidos.

Nuevamente, si el tráfico medido hubiese sido mucho mayor, este valor no hubiera afectado tanto los resultados.

La gráfica es la siguiente:



6.4 Trabajo futuro

Como se planteó inicialmente, nuestra solución fue probada en una red pequeña que consta de cuatro computadoras, entre las cuales se incluye a nuestro servidor. Esta configuración es adecuada para una red casera o comercial tipo SOHO (Pequeña Oficina u Oficina Casera), pero no será útil en una red de área local de mayor escala, como por ejemplo en la red LAN de una universidad o alguna determinada gran empresa.

Si se quisiera tener una red de área local mucho más grande que funcione en conjunto con nuestra solución, habría que cerciorarse de que ocurra lo siguiente: Que todos los mensajes DHCP de la red sean redireccionados hacia el servidor, que se bloqueen las respuestas ARP que no provengan de él y se modifique el destino de todas las consultas ARP, para que éstas solo vayan hacia el servidor y por último, que a todos los segmentos de la red les llegue la respectiva respuesta ARP generada por nuestro servidor.

Para que todas las consultas ARP sean contestadas con una respuesta correcta, se debería formar la red con switches que filtren el tráfico ARP de toda la red y envíen las peticiones ARP hacia nuestro servidor para que éste pueda contestarlas.

Si se quisiera implementar una red como para una empresa, lo que se podría hacer es conectar a los tres puertos restantes de nuestro

ruteador, tres switches iguales al que usamos para nuestra solución y que tengan exactamente la misma funcionalidad por medio de OpenWRT. Es decir deberán de bloquear todo el tráfico ARP y reenviarlo por el puerto al que se encuentra conectado nuestro pequeño ruteador para que éste procese esas peticiones ARP y el servidor pueda contestarlas con la respuesta adecuada, y sea enviada por el puerto correcto de nuestro ruteador hacia el switch que se encargará de entregárselo al computador que la solicitó.

Así de esta manera obtendremos expandir más la red pero ya de una manera segura, y lo que actualmente empezó como una prueba de conceptos podrá ser visto ahora como una red confiable más grande libre de envenenamientos ARP.

También hay que tomar en cuenta que los ruteadores/switches que actualmente se encuentran en el mercado para uso personal o redes de áreas locales pequeñas por lo general vienen con solo 4 puertos para la red. Pero debido al uso más frecuente de aparatos de este tipo como laptops, celulares, teléfonos de VoIP, entre otros, es probable entonces que se empiecen a desarrollar switches de características similares pero con mayor número de puertos y así llegar a tener redes seguras con mayor capacidad.

CONCLUSIONES Y RECOMENDACIONES

- Después de desarrollar esta solución pudimos darnos cuenta de la magnitud del problema que estábamos enfrentando, aunque encontrar una solución para este envenenamiento ARP no es una cosa de todos los días, tampoco necesariamente tuvo que ser una solución de un gran sistema con muchos módulos, al contrario, realmente fue cuestión de aplicar de manera acertada una serie de restricciones a nuestra pequeña red hasta lograr el propósito esperado.
- Aprendimos el uso de algunas herramientas muy útiles con las que contaba el sistema operativo OpenWRT del ruteador, como son las reglas ebtables e iptables que suelen ser empleadas para crear firewalls o realizar filtros en redes, el uso de VLANs y de la herramienta de desarrollo SDK también fue muy importante.
- El llevar a cabo el desarrollo de una solución de este tipo nos hizo

tomar en cuenta el gran impacto que ha traído el desarrollo a lo largo de estos años de las herramientas de software libre, sin las cuales esta solución jamás hubiera sido posible.

- El nivel de rendimiento que obtendríamos por la utilización de la librería libpcap no fue considerado inicialmente, pero éste puede ser contrarrestado si la solución se implementara en modo núcleo.
- El esquema propuesto es válido en gran parte, mas no en su totalidad, ya que no logramos cumplir con uno de los objetivos específicos que lo constituyan como una solución ideal, el cual es no disminuir la rapidez del protocolo.
- El alto valor obtenido en la métrica de tiempo de las respuestas ARP no es grave. Existen mejoras que pueden ser aplicadas para incrementar la eficiencia de nuestro sistema, como lo es el implementar el programa `send_arp_reply` en modo kernel y no en modo usuario.
- El contínuo desarrollo del software libre es muy importante y su uso aplicado a alguna tarea específica puede llegar a tener resultados muy interesantes.

- Se espera que a partir de una pequeña prueba de conceptos como ésta se pueda desarrollar un mecanismo que sea aplicado para las redes futuras y así evitar que se sigan realizando los ataques que pueden ocurrir como consecuencia de un envenenamiento ARP.

- Antes de correr la solución se aconseja siempre verificar que en el ruteador se encuentren corriendo correctamente las reglas iptables que registran los paquetes DHCP en el archivo `ulogd.pcap`. Cabe recalcar que la utilización de iptables con el alcance ULOG fue a consecuencia de que en el ruteador, no se pudo configurar un puerto como puerto espejo. El uso de un ruteador con esta característica, evitaría el uso del programa `ForwardDhcpFrames` y de las reglas de ULOG con iptables.

- Revisar que las reglas ebttables estén bloqueando todo el tráfico ARP excepto el proveniente del servidor, para lo cual hay que verificar que se tenga escrita correctamente la dirección física del servidor en ellas.

- Es importante que el programa que se encarga de responder las

consultas ARP se ejecute con una prioridad muy alta, ya que debido al retardo que se produce en el tráfico por las VLANs y la librería libpcap, es útil para que la respuesta ARP llegue al nodo que consulta de la manera mas rápida posible.

- Implementar el programa que contesta las consultas ARP a nivel de núcleo. Recomendamos su desarrollo para que el programa sea ejecutado en modo núcleo como parte de otro trabajo de investigación, y así de esta forma, los tiempos de envío de respuestas ARP no sean tan diferentes a los obtenidos cuando el sistema operativo de la máquina real a la que se le realiza la petición ARP, contesta.

APÉNDICES

A APÉNDICE A: ARCHIVOS DE CÓDIGO FUENTE DE LOS PROGRAMAS DEL SERVIDOR

A.1 Programa update_arp_cache

- dhcp_util.h

```

#ifndef _DHCP_UTIL
#define _DHCP_UTIL

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <dumbnet.h>
#include <pcap.h>
#include <errno.h>
#include <libgen.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/if.h>

/*
 * FILENAME:
 *   dhcp_util.h
 *   Lastest change: 18/08/2008 (Andre Ortega)
 *   -----
 * DESCRIPTION:
 *   This file contains the functions declaration
 *   for dhcp_util.c.
 * MODIFICACIONES:
 *   07/04/2008 (Xavier Marcos):      Version 1
 *   30/07/2008 (Andre Ortega): Modifications to the function:
 *   get_DHCP_msg_type(u_char *dhcp_message, int dm_lgth).
 */

/*
 * Function:  usage
 *   -----
 * Usage:
 *   char *progname;
 *   usage(progname);
 *   This function shows the correct way to run the program
 *   send_arp_reply.
 */
void usage(char *progname);

/*
 * Function:  validate_device
 *   -----
 * Usage:
 *   char *dev;
 *   validate_device(dev);
 *   This function validates the device entered by user.
 */

```

```

int validate_device(char * device);

/*      Function:  load_cache_from_file
 *      -----
 *      Usage:
 *      int status;
 *      char *dev;
 *      status=load_cache_from_file();
 *      This function loads IP-ARP mappings from file to system ARP cache.
 */
int load_cache_from_file();

/*      Function:  get_IP_from_device
 *      -----
 *      Usage:
 *      char *server_ip;
 *      char *dev;
 *      server_ip=get_IP_from_device(dev);
 *      This function returns the IP address associated with the device dev.
 */
char* get_IP_from_device(char *device);

/*      Function:  get_DHCP_msg_type
 *      -----
 *      Usage:
 *      int dhcp_message_type;
 *      u_char *dhcp_message=NULL;
 *      int size_dhcp;
 *      dhcp_message_type=get_DHCP_msg_type(dhcp_message, size_dhcp);
 *      This function returns the DHCP message type.
 *      Returns 5 if the DHCP message is a DHCPACK, 7 if it is a DHCPRELEASE,
 *      and 4 if it is a DHCPDECLINE, otherwise it returns -1.
 */
int get_DHCP_msg_type(u_char *dhcp_message, int dm_lgth);

/*      Function:  itoa
 *      -----
 *      Usage:
 *      char *s;
 *      int i;
 *      s=itoa(i);
 *      This function takes the integer input value i and converts it to a
 *      char*.
 */
char *itoa (int i);

/*      Function:  htoa
 *      -----
 *      Usage:
 *      char *s;
 *      int i;
 *      s=htoa(i);
 *      This function takes the integer input value i and converts it to a
 *      char* (%X).
 */
char *htoa (int h);

/*      Function:  write_cache_file
 *      -----
 *      Usage:

```

```

*     int status;
*     status=write_cache_file();
*     This function sends the output from arp -s to a file named
cache_file.txt.
*/
int write_cache_file();

/*     Function:  trim
*     -----
*     Usage:
*     char *s;
*     trim(s);
*     This function eliminates the spaces from beginning and end of a
string.
*/
void trim(char *s);

#endif

```

• dhcp_util.c

```

#include "dhcp_util.h"

/*
*     FILENAME:
*     dhcp_util.c
*     Lastest change: 18/08/2008 (Andre Ortega)
*     -----
*     -----
*     DESCRIPTION:
*     This file contains the implementation of the
*     functions used to update system ARP cache.
*
*     MODIFICACIONES:
*     07/04/2008 (Xavier Marcos):      Version 1
*     30/07/2008 (Andre Ortega): Modifications to the function:
get_DHCP_msg_type(u_char *dhcp_message, int dm_lgth).
*/

void usage(char *programe)
{
    printf("Error calling function, the correct use is: %s
lan_interface\n", (char *)basename(programe));
}

int validate_device(char * device)
{
    int lgth= strlen(device);

    /*for example, device eth1*/
    if(lgth==4){
        if(device[0]!='e' || device[1]!='t' || device[2]!='h' ||
(device[3]!='0' && atoi(&device[3])==0))
            return -1;
        else
            return 1;
    }
    /*for example, device eth10*/
    else if(lgth==5){

```

```

        if(device[0]!='e' || device[1]!='t' || device[2]!='h' ||
device[3]!='0' || atoi(&device[3])==0 || (device[4]!='0' &&
atoi(&device[4])==0))
            return -1;
        else
            return 1;
    }
    else
        return -1;
}

int load_cache_from_file()
{
    FILE *fp;
    char line[BUFSIZ];
    char ip_addr[16]="";
    char mac_addr[18]="";
    char comando[51]="";
    int result=-1;

    fp = fopen("/tmp/cache_file.txt","r");

    if(fp!=NULL){
        fgets(line, sizeof(line),fp);
        while (fgets(line, sizeof(line),fp)!=NULL)//reading each line
        {
            if(line[33]!='('){
                memcpy(ip_addr,line,16);
                memcpy(mac_addr,line+33,17);
                strcpy(comando,"");
                strcat(comando,"arp -s ");
                strcat(comando,ip_addr);
                strcat(comando," ");
                strcat(comando,mac_addr);
                if(system(comando)==-1 || system(comando)==127){

                    return -1;
                }
                else
                    result=1;
            }
        }
        fclose(fp);
    }
    return result;
}

char* get_IP_from_device(char *device)
{
    int sock;
    struct ifreq ifr;
    struct sockaddr_in *ip_addr;
    char *ipa=NULL;

    /*Socket Creation*/
    if((sock = socket(AF_INET,SOCK_DGRAM,0))===-1){
        return NULL;
    }

    strncpy(ifr.ifr_name,device,6);

```

```

    ip_addr = (struct sockaddr_in *) & (ifr.ifr_addr);

    /*Getting ip address from device*/
    if(ioctl(sock, SIOCGIFADDR, &ifr) == 0){
        ipa = inet_ntoa(ip_addr->sin_addr);
        return ipa;
    }
    else
        return NULL;
}

int get_DHCP_msg_type(u_char *dhcp_message, int dm_lgth)
{
    int i=0;
    int ret=-1;
    for(i=0;i<dm_lgth;i++){
        /*dhcpack*/
        if(dhcp_message[i]==0x35    &&    dhcp_message[i+1]==0x01    &&
dhcp_message[i+2]==0x05){
            return 5;
        }
        else if(dhcp_message[i]==0x35    &&    dhcp_message[i+1]==0x01    &&
dhcp_message[i+2]==0x07){
            /*dhcprelease*/
            return 7;
        }
        else if(dhcp_message[i]==0x35    &&    dhcp_message[i+1]==0x01    &&
dhcp_message[i+2]==0x04){
            /*dhcpcdecline*/
            return 4;
        }else{
            ret=-1;
        }
    }
    return ret;
}

char *itoa (int i)
{
    char str_val [32] ;
    snprintf(str_val, sizeof (str_val), "%d", i) ;

    /*Use strdup to duplicate str_val which is currently on the stack.*/
    return strdup(str_val) ;
}

char *htoa (int h)
{
    char str_val [32] ;
    snprintf(str_val, sizeof (str_val), "%X", h) ;

    /*Use strdup to duplicate str_val which is currently on the stack.*/
    return strdup(str_val) ;
}

int write_cache_file()
{
    FILE *fp;

```

```

        fp = fopen("/proc/net/arp", "r");

        if(fp!=NULL){
            system("arp -n > /tmp/cache_file.txt");
            fclose(fp);
            return 1;
        }
        else
            return -1;
    }

void trim(char *s)
{
    /*Trim spaces from beginning*/
    int i=0, j;
    while(s[i]==' '){
        i++;
    }
    if(i>0){
        for(j=0; j<strlen(s); j++){
            s[j]=s[j+i];
        }
        s[j]='\0';
    }

    /*Trim spaces from end*/
    i=strlen(s)-1;
    while(s[i]==' '){
        i--;
    }
    if(i<(strlen(s)-1)){
        s[i+1]='\0';
    }
}

```

- **main.c**

```

#include "dhcp_util.h"

/*
 *   FILENAME:
 *       main.c
 *
 *       Lastest change: 11/10/2008 (Andre Ortega)
 * -----
 *
 *   DESCRIPTION:
 *       This file contains the main program and the
implementation of
 *       callback function for pcap_loop.
 *
 *   MODIFICACIONES:
 *       07/04/2007 (Xavier Marcos):      Version 1
 *       11/08/2008 (Andre Ortega): Use of function:
get_DHCP_msg_type.
 */

char *server_ip_address=NULL;

void update_IP_MAC_table(u_char *user, const struct pcap_pkthdr*

```

```

pkthdr, const u_char* packet)
{
    u_char *dhcp_message=NULL;
    char ip_addr[16]="";
    char mac_addr[18]="";
    char comando[51]="";
    int dhcp_size=0;
    struct udp_hdr *udp_hdr=NULL;
    int i=0;

    udp_hdr = (struct udp_hdr *) (packet + sizeof(struct eth_hdr) +
sizeof(struct ip_hdr));
    dhcp_size = ntohs(udp_hdr->uh_ulen) - sizeof(struct udp_hdr);

    dhcp_message=packet+sizeof(struct eth_hdr)+sizeof(struct
ip_hdr)+sizeof(struct udp_hdr);

    /*If the packet is not dhcpack, dhcprelease neither dhcpdecline, then
we dont process it.*/
    if(get_DHCP_msg_type(dhcp_message, dhcp_size)==-1){
        return;
    }
    else if(get_DHCP_msg_type(dhcp_message, dhcp_size)==5){
        /*dhcpack*/
        /*ip address*/
        strcpy(ip_addr, "");
        if(dhcp_message[16]!=0){
            /*Using "Your ip address" field.
            strcpy(ip_addr, itoa(dhcp_message[16]));
            strcat(ip_addr, ".");
            strcat(ip_addr, itoa(dhcp_message[17]));
            strcat(ip_addr, ".");
            strcat(ip_addr, itoa(dhcp_message[18]));
            strcat(ip_addr, ".");
            strcat(ip_addr, itoa(dhcp_message[19]));
        }
        else{
            /*If DHCPACK contains 0.0.0.0 in "Your ip address"
field, because DHCP client already has IP address, then we use "Client ip
address" field*/
            strcpy(ip_addr, itoa(dhcp_message[12]));
            strcat(ip_addr, ".");
            strcat(ip_addr, itoa(dhcp_message[13]));
            strcat(ip_addr, ".");
            strcat(ip_addr, itoa(dhcp_message[14]));
            strcat(ip_addr, ".");
            strcat(ip_addr, itoa(dhcp_message[15]));
        }

        /*If the dhcpack is for us, the server, then we dont process
it.*/

        /*Because we cant add that mapping to its kernel arp cache.*/
        trim(ip_addr);
        if(strcmp(ip_addr, server_ip_address)==0){
            return;
        }

        /*mac address*/
        strcpy(mac_addr, "");

```



```

strcpy(mac_addr, htoa(dhcp_message[28]));
strcat(mac_addr, ":");
strcpy(mac_addr, htoa(dhcp_message[29]));
strcat(mac_addr, ":");
strcpy(mac_addr, htoa(dhcp_message[30]));
strcat(mac_addr, ":");
strcpy(mac_addr, htoa(dhcp_message[31]));
strcat(mac_addr, ":");
strcpy(mac_addr, htoa(dhcp_message[32]));
strcat(mac_addr, ":");
strcpy(mac_addr, htoa(dhcp_message[33]));
strcpy(comando, "");
strcat(comando, "arp -s ");
strcat(comando, ip_addr);
strcat(comando, " ");
strcat(comando, mac_addr);
/*Adding IP MAC to kernel arp cache*/
if(system(comando)==-1 || system(comando)==127){

    fprintf(stderr, "Error executing command system.\n");
}
else
    fprintf(stdout, "Command system(arp -s) has been
successfully executed.\n");

    /*Saving mapping to arp file*/
    if(write_cache_file()==1){
        fprintf(stdout, "cache_file.txt has been updated
successfully.\n");
    }
    else
        fprintf(stderr, "cache_file.txt could not been
updated.\n");
}
else if(get_DHCP_msg_type(dhcp_message, dhcp_size)==7){
    /*dhcrelease*/
    strcpy(ip_addr, "");
    strcpy(ip_addr, itoa(dhcp_message[12]));
    strcat(ip_addr, ".");
    strcat(ip_addr, itoa(dhcp_message[13]));
    strcat(ip_addr, ".");
    strcat(ip_addr, itoa(dhcp_message[14]));
    strcat(ip_addr, ".");
    strcat(ip_addr, itoa(dhcp_message[15]));

    /*If the dhcrelease goes out from us, the server, then we dont
process it.*/
    /*Because we dont have that mapping in this kernel arp cache.*/
    trim(ip_addr);
    if(strcmp(ip_addr, server_ip_address)==0){
        return;
    }

    strcpy(comando, "");
    strcat(comando, "arp -d ");
    strcat(comando, ip_addr);
    /*Deleting IP MAC from kernel arp cache*/
    if(system(comando)==-1 || system(comando)==127){
        fprintf(stderr, "Error executing command system.\n");
    }
}

```

```

else
    fprintf(stdout,"Command system(arp -s) has been
successfully executed.\n");
    /*Updating cache file*/
    if(write_cache_file()==1){
        fprintf(stdout,"cache_file.txt has been updated
successfully.\n");
    }
    else
        fprintf(stderr,"cache_file.txt could not be
updated.\n");
    }
    else{
        /*dhcpdecline*/
        for(i=240;i<dhcp_size;i++){//From position 240 because there is
where the option fields begins.
            /*Requested ip address from DHCP options*/
            if(dhcp_message[i]==0x32 && dhcp_message[i+1]==0x04){
                strcpy(ip_addr,"");
                strcpy(ip_addr,itoa(dhcp_message[i+2]));
                strcat(ip_addr,".");
                strcat(ip_addr,itoa(dhcp_message[i+3]));
                strcat(ip_addr,".");
                strcat(ip_addr,itoa(dhcp_message[i+4]));
                strcat(ip_addr,".");
                strcat(ip_addr,itoa(dhcp_message[i+5]));
                i=dhcp_size;
            }
        }

        /*If the dhcpdecline goes out from us, the server, then we dont
process it.*/
        /*Because we dont have that mapping in this kernel arp cache.*/
        trim(ip_addr);
        if(strcmp(ip_addr,server_ip_address)==0){
            return;
        }

        strcpy(comando,"");
        strcat(comando,"arp -d ");
        strcat(comando,ip_addr);
        /*Deleting IP MAC from kernel arp cache*/
        if(system(comando)==-1 || system(comando)==127){
            fprintf(stderr,"Error executing command system.\n");
        }
        else
            fprintf(stdout,"Command system(arp -d) has been
successfully executed.\n");
            /*Updating cache file*/
            if(write_cache_file()==1){
                fprintf(stdout,"cache_file.txt has been updated
successfully.\n");
            }
            else
                fprintf(stderr,"cache_file.txt could not be
updated.\n");
        }
    }
}

```

```

int main(int argc, char **argv)
{
    pcap_t *descr=NULL;
    char errbuf[PCAP_ERRBUF_SIZE]="";
    u_char *user = NULL;
    char dev[5]="";
    bpf_u_int32 maskp;
    bpf_u_int32 netp;
    struct bpf_program fp;
    char filter[37]="";
    strncpy(filter, "udp and (dst port 67 or dst port 68)", 37);

    /*Validating program execution*/
    if(argc!=2){
        usage(argv[0]);
        return -1;
    }

    /*Validating interface entered by user*/
    if(validate_device(argv[1])<0){
        fprintf(stderr, "Error writing device, the correct format is
ethx or ethxx.\n");
        return -1;
    }

    strncpy(dev, argv[1], 6);

    if(load_cache_from_file()==1){
        fprintf(stdout, "IP-ARP mappings loaded sucesfully from
cache_file.txt.\n");
    }
    else
        fprintf(stderr, "Unable to load IP-ARP mappings to system ARP
cache, file does not exist.\n");

    /*Getting server IP address from device dev*/
    server_ip_address=get_IP_from_device(dev);
    if(server_ip_address==NULL){
        fprintf(stderr, "Error getting server ip address from device
%s.\n", dev);
        return -1;
    }

    /*Getting network address and subnet mask for device*/
    if(pcap_lookupnet(dev, &netp, &maskp, errbuf)<0){
        fprintf(stderr, "Error getting network address and mask for
device %s: %s.\n", dev, errbuf);
        return -1;
    }

    /*Creating and initializing a packet capture descriptor(descr) and
opening the specified device*/
    descr = pcap_open_live(dev, BUFSIZ, 1, -1, errbuf);
    if(descr==NULL){
        fprintf(stderr, "Error opening device %s: %s.\n", dev, errbuf);
        return -1;
    }

    /*Compiling a filter expression into a filter program*/
    /*The filter allow us to sniff only DHCP frames*/

```

```

        if(pcap_compile(descr,&fp,filter,0,maskp)<0){
            fprintf(stderr,"Error calling pcap_compile.\n");
            return -1;
        }

        /*Setting the compiled program as the filter*/
        if(pcap_setfilter(descr,&fp)<0){
            fprintf(stderr,"Error setting filter: %s.\n",
pcap_geterr(descr));
            return -1;
        }

        /*Reading and processing each DHCP frame*/
        if(pcap_loop(descr, -1, update_IP_MAC_table, user)<0){
            fprintf(stderr,"Error reading packets from device %s: %s.\n",
dev, pcap_geterr(descr));
            return -1;
        }

        /*Closing the packet capture device*/
        pcap_close(descr);
        return (0);
    }

```

A.2 Programa send_arp_reply

- arp_util.h

```

#ifndef _ARP_UTIL
#define _ARP_UTIL
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <pcap.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include <netinet/ether.h>
#include <net/if_arp.h>
#include <getopt.h>
#include <libgen.h>
#include <dumbnet.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <linux/if.h>

#define buffer_size sizeof(struct eth_hdr) + sizeof(struct arp_hdr) +
sizeof(struct arp_ethip)

/*
 * FILENAME:
 * arp_util.h
 * Lastest change: 11/10/2008 (Andre Ortega)
 * -----
-----

```

```

*      DESCRIPTION:
*      This file contains the functions declaration
*      for arp_util.c.
*
*      MODIFICACIONES:
*      30/10/2007 (Xavier Marcos):      Version 1
*      11/10/2008 (Andre Ortega): Function etheton: deleted.
*/

/*      Function:  usage
*      -----
*      Usage:
*      char *progname;
*      usage(progname);
*      This function shows the correct way to run the program
send_arp_reply.
*/
void usage(char *progname);

/*      Function:  validate_device
*      -----
*      Usage:
*      char *dev;
*      validate_device(dev);
*      This function validates the device entered by user.
*/
int validate_device(char * device);

/*      Function:  get_IP_from_device
*      -----
*      Usage:
*      char *server_ip;
*      char *dev;
*      server_ip=get_IP_from_device(dev);
*      This function returns the IP address associated with the device dev.
*/
char* get_IP_from_device(char *device);

/*      Function:  get_local_MAC_addr
*      -----
*      Usage:
*      int status;
*      char *dev;
*      char *mac_address;
*      status=get_local_MAC_addr(dev,mac_address);
*      This function returns the MAC address associated with the device dev.
*      Returns 1 if the MAC address was obtained and 0 if it fails.
*/
int get_local_MAC_addr(char *device,u_char *addr);

/*      Function:  itoa
*      -----
*      Usage:
*      char *s;
*      int i;
*      s=itoa(i);
*      This function takes the integer input value i and converts it to a
char*.
*/
char *itoa(int i);

```

```

/*      Function:  trim
*      -----
*      Usage:
*      char *s;
*      trim(s);
*      This function eliminates the spaces from beginning and end of a
string.
*/
void trim(char *s);

/*      Function:  get_MAC_from_IP
*      -----
*      Usage:
*      int status;
*      char *ip_address;
*      struct arp_entry arp_entry;
*      status=get_MAC_from_IP(ip_address, &arp_entry);
*      This function gets the MAC address related to ip_address from system
ARP cache.
*      Returns 1 if the MAC address was obtained and 0 if it fails.
*/
int get_MAC_from_IP(char *source_ip, struct arp_entry *arp_entry);

#endif

```

• arp_util.c

```

#include "arp_util.h"

/*
*      FILENAME:
*      arp_util.c
*      Lastest change: 11/10/2008 (Andre Ortega)
*      -----
*      DESCRIPTION:
*      This file contains the implementation of the
*      functions used to reply ARP requests.
*      MODIFICACIONES:
*      30/10/2007 (Xavier Marcos):      Version 1
*      11/10/2008 (Andre Ortega):Function etheton: deleted.
*/

void usage(char *programe)
{
    printf("Error calling program, the correct use is: %s
lan_interface\n", (char *)basename(programe));
}

int validate_device(char * device)
{
    int lgth= strlen(device);

    /*for example, device eth1*/
    if(lgth==4){
        if(device[0]!='e' || device[1]!='t' || device[2]!='h' ||
(device[3]!='0' && atoi(&device[3])==0))

```

```

        return -1;
    else
        return 1;
}
/*for example, device eth10*/
else{
    if(lgth==5){
        if(device[0]!='e' || device[1]!='t' || device[2]!='h' ||
device[3]!='0' || atoi(&device[3])==0 || (device[4]!='0' &&
atoi(&device[4])==0))
            return -1;
        else
            return 1;
    }
    else
        return -1;
}
}

char* get_IP_from_device(char *device)
{
    int sock;
    struct ifreq ifr;
    struct sockaddr_in *ip_addr;
    char *ipa;

    /*Socket Creation*/
    if((sock = socket(AF_INET,SOCK_DGRAM,0))==-1){
        return NULL;
    }

    strncpy(ifr.ifr_name,device,6);

    ip_addr = (struct sockaddr_in *) & (ifr.ifr_addr);

    /*Getting IP address from device*/
    if(ioctl(sock, SIOCGIFADDR, &ifr) == 0){
        ipa = inet_ntoa(ip_addr->sin_addr);
        return ipa;
    }
    else
        return NULL;
}

int get_local_MAC_addr(char *device,u_char *addr)
{
    int sock;
    struct ifreq ifr;
    int res = 0;

    /*Socket Creation*/
    if((sock = socket(AF_INET,SOCK_DGRAM,0))==-1){

        return -1;
    }

    strncpy(ifr.ifr_name,device,6);

    /*Getting the active flag word of the device.*/

```

```

if(ioctl(sock, SIOCGIFFLAGS, &ifr) == 0){
    /*If the interface is loopback then don't get MAC address*/
    if(!(ifr.ifr_flags & IFF_LOOPBACK)){
        /*Getting MAC address from device*/
        if(ioctl(sock, SIOCGIFHWADDR, &ifr) == 0){
            memcpy(addr,ifr.ifr_hwaddr.sa_data,6);

                res = 1;
            }
            else{
                res = -1;
            }
        }
    }
    return res;
}

char *itoa(int i)
{
    char str_val [32] ;
    snprintf(str_val, sizeof (str_val), "%d", i) ;

    /*Use strdup to duplicate str_val which is currently on the stack.*/
    return strdup(str_val) ;
}

void trim(char *s)
{
    /*Trim spaces from beginning*/
    int i=0,j;
    while(s[i]==' '){
        i++;
    }
    if(i>0){
        for(j=0;j<strlen(s);j++){
            s[j]=s[j+i];
        }
        s[j]='\0';
    }

    /*Trim spaces from end*/
    i=strlen(s)-1;
    while(s[i]==' '){
        i--;
    }
    if(i<(strlen(s)-1)){
        s[i+1]='\0';
    }
}

int get_MAC_from_IP(char *source_ip, struct arp_entry *arp_entry){

    arp_t *arp;

    /*Obtaining a handle to access the kernel ARP cache*/
    arp = arp_open();
    if(arp!=NULL){
        /*Converting an address from a string to network format*/

        if(addr_pton(source_ip,&arp_entry->arp_pa)==0){

```



```

/*Getting the ARP entry for the protocol address
specified by arp_pa.*/
if(arp_get(arp,arp_entry)==0){
    /*Closing the handle*/
    arp_close(arp);
    return 1;
}
else
    arp_close(arp);
    return -1;
}
else
    arp_close(arp);
    return -1;
}
else
    return -1;
}

```

- **main.c**

```

#include "arp_util.h"

/*
 * FILENAME:
 * main.c
 * Lastest change: 11/10/2008 (Andre Ortega)
 * -----
 * DESCRIPTION:
 * This file contains the main program and the
implementation of
 * callback function for pcap_loop.
 *
 * MODIFICACIONES:
 * 30/10/2007 (Xavier Marcos): Version 1
 * 11/10/2008 (Andre Ortega):Use of function etheton was
deleted.
 */

u_char buffer[buffer_size];
struct eth_hdr *eth_hdr_rep=(struct eth_hdr *)buffer;
char *server_ip;
char dev[6];
u_char src_mac[ETH_ADDR_LEN];

void complete_and_send_ARP_reply(u_char *user, const struct pcap_pkthdr*
pkthdr, const u_char *packet)
{
    struct arp_hdr *arp_hdr_in=NULL;
    arp_hdr_in=(struct arp_hdr *)(packet + sizeof(struct eth_hdr));

    /*Answering each ARP request*/
    if(ntohs(arp_hdr_in->ar_op)==ARP_OP_REQUEST)
    {
        pcap_t *descr=NULL;
        char ip_addr[16]="";
        char errbuf[PCAP_ERRBUF_SIZE]="";
        struct arp_entry entry;
    }
}

```

```

        u_char mac_requested[ETH_ADDR_LEN]="";
        struct arp_ethip *ip_arp_hdr_in=NULL;
        struct arp_ethip *ip_arp_hdr_rep=NULL;

        ip_arp_hdr_in=(struct arp_ethip *) (packet + sizeof(struct
eth_hdr) + sizeof(struct arp_hdr));
        ip_arp_hdr_rep=(struct arp_ethip *) (buffer + sizeof(struct
eth_hdr) + sizeof(struct arp_hdr));

        /*Ethernet header*/
        memcpy(eth_hdr_rep->eth_dst.data,ip_arp_hdr_in-
>ar_sha,ETH_ADDR_LEN); //destination MAC address

        /*IP ARP header*/
        /*ip_addr: IP address from which someone requested its MAC
address*/
        strcpy(ip_addr,"");
        strcpy(ip_addr,itoa(ip_arp_hdr_in->ar_tpa[0]));
        strcat(ip_addr,".");
        strcat(ip_addr,itoa(ip_arp_hdr_in->ar_tpa[1]));
        strcat(ip_addr,".");
        strcat(ip_addr,itoa(ip_arp_hdr_in->ar_tpa[2]));
        strcat(ip_addr,".");
        strcat(ip_addr,itoa(ip_arp_hdr_in->ar_tpa[3]));

        /*Getting MAC requested*/
        trim(ip_addr);
        /*If requested IP address is server IP then fill mac_requested
with server MAC address*/
        if(strcmp(ip_addr,server_ip)==0){
            memcpy(mac_requested,src_mac,ETH_ADDR_LEN);
        }
        /*Else if requested IP address is different from server IP,
then search MAC address in the system arp cache*/
        else{
            if(get_MAC_from_IP(ip_addr, &entry)==-1){
                printf("Could not find MAC for IP from system ARP
cache, entry does not exist.\n");
                return;
            }
            else
            {
                memcpy(mac_requested,(&entry.arp_ha.addr_eth.data),ETH_ADDR_LEN);
            }
        }

        memcpy(ip_arp_hdr_rep->ar_sha,mac_requested,ETH_ADDR_LEN);
//The MAC address that was requested
        memcpy(ip_arp_hdr_rep->ar_spa,ip_arp_hdr_in-
>ar_tpa,IP_ADDR_LEN); //The IP address that corresponds to the MAC requested
        memcpy(ip_arp_hdr_rep->ar_tha,ip_arp_hdr_in-
>ar_sha,ETH_ADDR_LEN); //Computer's MAC address that made the request
        memcpy(ip_arp_hdr_rep->ar_tpa,ip_arp_hdr_in-
>ar_spa,IP_ADDR_LEN); //Computer's IP that made the request

        /*Creating and initializing a packet capture descriptor(descr)
and opening the specified device*/
        descr = pcap_open_live(dev,BUFSIZ,1,-1,errbuf);
        if(descr == NULL){

```

```

        fprintf(stderr,"Error opening device %s when trying to
send ARP reply: %s.\n",dev, errbuf);
    }

    /*Sending ARP reply*/
    if(pcap_sendpacket(descr, buffer, buffer_size)!=0){
        fprintf(stderr,"Error sending ARP reply: %s.\n",
pcap_geterr(descr));
    }

    /*Closing the packet capture device*/
    pcap_close(descr);
}

int main(int argc, char **argv)
{
    pcap_t *descr=NULL;
    char errbuf[PCAP_ERRBUF_SIZE]="";
    u_char *user=NULL;
    bpf_u_int32 maskp;
    bpf_u_int32 netp;
    struct bpf_program fp;
    int stat;
    char filter[4]="";
    strncpy(filter,"arp",4);
    struct arp_hdr *arp_hdr_rep=NULL;

    /*Validating program execution*/
    if(argc!=2){
        usage(argv[0]);
        return -1;
    }

    /*Validating interface entered by user*/
    if(validate_device(argv[1])<0){
        fprintf(stderr,"Error writing device, the correct format is
ethx or ethxx.\n");
        return -1;
    }

    strncpy(dev,argv[1],6);

    /*Getting server IP address from device dev*/
    server_ip=get_IP_from_device(dev);
    if(server_ip==NULL){
        fprintf(stderr,"Error getting server IP address from device
%s.\n",dev);
        return -1;
    }

    /*Getting network address and subnet mask for device*/
    if(pcap_lookupnet(dev,&netp,&maskp,errbuf)<0){
        fprintf(stderr,"Error getting network address and mask for
device %s: %s.\n",dev, errbuf);
        return -1;
    }

    /*Creating and initializing a packet capture descriptor(descr) and
opening the specified device*/

```

```

descr = pcap_open_live(dev,BUFSIZ,1,-1,errbuf);
if(descr==NULL){
    fprintf(stderr,"Error opening device %s: %s.\n",dev, errbuf);
    return -1;
}

/*Compiling a filter expression into a filter program*/
/*The filter allow us to sniff only ARP frames*/
if(pcap_compile(descr,&fp,filter,0,maskp)<0){
    fprintf(stderr,"Error calling pcap_compile.\n");
    return -1;
}

/*Setting the compiled program as the filter*/
if(pcap_setfilter(descr,&fp)<0){
    fprintf(stderr,"Error setting filter: %s.\n",
pcap_geterr(descr));
    return -1;
}

/*Getting MAC address from lan device*/
stat = get_local_MAC_addr(dev,src_mac);
if(stat != 1){
    fprintf(stderr,"Error getting MAC address from device br0.\n");
    return -1;
}

/*Filling ARP reply constant fields*/
eth_hdr_rep=(struct eth_hdr *)buffer;
arp_hdr_rep=(struct arp_hdr *)(buffer + sizeof(struct eth_hdr));
memset(buffer,0,buffer_size);
/*Ethernet header*/
memcpy(eth_hdr_rep->eth_src.data,src_mac,ETH_ADDR_LEN); //source MAC
address
eth_hdr_rep->eth_type = htons(ETH_TYPE_ARP); //payload type
/*ARP header*/
arp_hdr_rep->ar_hrd = htons(ARP_HRD_ETH); //format of hardware
address
arp_hdr_rep->ar_pro = htons(ARP_PRO_IP); //format of protocol address
arp_hdr_rep->ar_hln = ETH_ADDR_LEN; //length of hardware address
arp_hdr_rep->ar_pln = IP_ADDR_LEN; //length of protocol address
arp_hdr_rep->ar_op = htons(ARP_OP_REPLY); //operation

/*Reading and processing each ARP frame*/
if(pcap_loop(descr, -1, complete_and_send_ARP_reply, user)<0){
    fprintf(stderr,"Error reading ARP frames from device %s:
%s.\n", dev, pcap_geterr(descr));
    return -1;
}

/*Closing the packet capture device*/
pcap_close(descr);
return (0);
}

```

B APÉNDICE B: ARCHIVO DE CÓDIGO FUENTE DEL PAQUETE DEL RUTEADOR

B.1 Programa ForwardDhcpFrames

- **ForwardDhcpFrames.c**

```

#include <stdio.h>
#include <pcap.h>
#include <string.h>
#include <net/ethernet.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <linux/if.h>
#include <stdlib.h>
#include <libgen.h>

/*
 *   FILENAME:
 *       main.c
 *
 *   Lastest change: 25/06/2008 (Xavier Marcos)
 *   -----
 *
 *   DESCRIPTION:
 *       This file contains the main program and functions
 *       used to send DHCP messages from /var/log/ulogd.pcap to
server.
 *
 *   MODIFICACIONES:
 *       05/05/2008 (Xavier Marcos):      Version 1
 *       25/06/2008 (Andre Ortega):The way to obtain buffer_size
was changed.
 */

#define PCAP_SAVEFILE "/var/log/ulogd.pcap" //File path where ulogd keeps
the dhcp packets

u_char src_mac[6]="";
u_char dst_mac[6]="";

/*Function that gets MAC address from device br0*/
int get_MAC_addr(u_char *addr)
{
    int sock;
    struct ifreq ifr;
    int res = 0;

    /*Socket Creation*/
    if((sock = socket(AF_INET,SOCK_DGRAM,0))== -1){

        return -1;
    }
}

```

```

strncpy(ifr.ifr_name,"br0",4);

/*Getting the active flag word of the device.*/
if(ioctl(sock, SIOCGIFFLAGS, &ifr) == 0){
    /*If the interface is loopback then don't get MAC address*/
    if(!(ifr.ifr_flags & IFF_LOOPBACK)){
        /*Getting MAC address from device*/
        if(ioctl(sock, SIOCGIFHWADDR, &ifr) == 0){
            memcpy(addr,ifr.ifr_hwaddr.sa_data,6);

                res = 1;
            }
        else{
            res = -1;
        }
    }
}
return res;
}

//eth_pton - Source: libdumbnet
int eth_pton(const char *p, u_char *data)
{
    char *ep;
    long l;
    int i;

    for(i = 0; i < ETHER_ADDR_LEN; i++){
        l = strtoul(p, &ep, 16);
        if(ep == p || l < 0 || l > 0xff ||
            (i < ETHER_ADDR_LEN - 1 && *ep != ':'))
            break;
        data[i] = (u_char)l;
        p = ep + 1;
    }
    return((i == ETHER_ADDR_LEN && *ep == '\0') ? 0 : -1);
}

/*Function that will forward DHCP messages to our server*/
void inline forward_DHCP_message(u_char *user, const struct pcap_pkthdr*
pkt_hdr, const u_char* packet)
{
    pcap_t *descr=NULL;
    char dev[4]="";
    char errbuf[PCAP_ERRBUF_SIZE]="";
    u_int buffer_size2=0;

    struct udphdr *udp_hdr = (struct udphdr *) (packet + sizeof(struct
ip));

    unsigned int buffer_size = sizeof(struct ether_header) +
sizeof(struct ip) + ntohs(udp_hdr->len);

    u_char buffer[buffer_size];
    memset(buffer, 0, buffer_size);

    struct ether_header *eth_hdr = (struct ether_header *)buffer;

```

```

/*Router device*/
strncpy(dev,"br0", 4);

/*Filling ethernet header*/
memcpy(eth_hdr->ether_dhost,dst_mac,ETHER_ADDR_LEN); //Dst.  MAC   -
Server MAC
memcpy(eth_hdr->ether_shost,src_mac,ETHER_ADDR_LEN); //Src.  MAC   -
Router mac
eth_hdr->ether_type = htons(ETHERTYPE_IP);

buffer_size2 = buffer_size - 14;
memcpy(buffer+14,packet,buffer_size2);

/*Creating and initializing a packet capture descriptor(descr) and
opening the specified device*/
descr = pcap_open_live(dev,BUFSIZ,1,-1,errbuf);
if(descr == NULL){
    fprintf(stderr,"Error opening device %s when trying to send
DHCP frame: %s.\n",dev, errbuf);
    return;
}

/*Sending DHCP frame*/
if(pcap_sendpacket(descr, buffer, buffer_size)!=0){
    fprintf(stderr,"Error sending DHCP frame: %s.\n",
pcap_geterr(descr));
    return;
}

/*Closing the packet capture device*/
pcap_close(descr);
}

void usage(char *progname)
{
    printf("Error calling program, the correct use is: %s
xx:xx:xx:xx:xx:xx [infile]\n", (char*)basename(progname));
}

int main(int argc, char **argv)
{
    pcap_t *descr=NULL;
    char errbuf[PCAP_ERRBUF_SIZE]="";
    u_char *user= NULL;
    char filename[30]="";
    int stat;

    /*Validating program execution*/
    if(argc > 3 || argc < 2){
        usage(argv[0]);
        return -1;
    }

    /*Validating MAC address entered by user*/
    if(eth_pton(argv[1],dst_mac)!=0){
        fprintf(stderr,"Error writing destination MAC address, the
correct format is xx:xx:xx:xx:xx:xx\n");
        return -1;
    }
}

```

```

        /*If the user didnt enter a filename, the default file will be
/var/log/ulogd.pcap*/
        if(argv[2])
            strcpy(filename,argv[2]);
        else
            strcpy(filename,PCAP_SAVEFILE);

        /*Creating and initializing a packet capture descriptor(descr) and
opening the specified savefile*/
        descr = pcap_open_offline(filename, errbuf);
        if(descr==NULL){
            fprintf(stderr,"Error opening file %s in reading mode:
%s.\n",filename, errbuf);
            return -1;
        }

        /*Getting MAC address from device br0*/
        stat = get_MAC_addr(src_mac);
        if(stat != 1){
            fprintf(stderr,"Error getting MAC address from device br0.\n");
            return -1;
        }

        /*Reading and processing each DHCP frame from file ulogd.pcap*/
        if(pcap_loop(descr, -1, forward_DHCP_message, user) < 0){
            fprintf(stderr,"Error reading DHCP frames from file %s: %s.\n",
filename, pcap_geterr(descr));
            return -1;
        }

        /*Closing the packet capture device*/
        pcap_close(descr);
        return (0);
}

```

C APÉNDICE C: ARCHIVOS MAKEFILE

C.1 Makefile para la compilación de ForwardDhcpFrames en el entorno de compilación cruzada

```

ForwardDhcpFrames: ForwardDhcpFrames.o
    $(CC) $(LDFLAGS) -lpcap -I. -I.. -I/usr/include/ ForwardDhcpFrames.o -
o ForwardDhcpFrames
ForwardDhcpFrames.o: ForwardDhcpFrames.c
    $(CC) $(CFLAGS) -I. -I.. -I/usr/include/ -c ForwardDhcpFrames.c

clean:
    rm *.o ForwardDhcpFrames

```

C.2 Makefile para la compilación de ForwardDhcpFrames en el ruteador


```

include $(TOPDIR)/rules.mk

# Name and release number of this package
PKG_NAME:=ForwardDhcpFrames
PKG_RELEASE:=1

# This specifies the directory where we're going to build the program.
# The root build directory, $(BUILD_DIR), is by default the build_mipsel
# directory in your OpenWrt SDK directory
PKG_BUILD_DIR:=$(BUILD_DIR)/$(PKG_NAME)

include $(INCLUDE_DIR)/package.mk

# Specify package information for this program.
# The variables defined here should be self explanatory.
# If you are running Kamikaze, delete the DESCRIPTION
# variable below and uncomment the Kamikaze define
# directive for the description below
define Package/ForwardDhcpFrames
    SECTION:=net
    CATEGORY:=Network
    TITLE:=Forward DHCP frames
    DESCRIPTION:=\
        Program to forward DHCP frames
endef

# Specify what needs to be done to prepare for building the package.
define Build/Prepare
    mkdir -p $(PKG_BUILD_DIR)
    $(CP) ./src/* $(PKG_BUILD_DIR)/
endef

define Build/Compile
    $(MAKE) -C $(PKG_BUILD_DIR) \
        LIBS="-nodefaultlibs -lgcc -lc -lpcap" \
        LDFLAGS="$(EXTRA_LDFLAGS)" \
        CXXFLAGS="$(TARGET_CFLAGS) \
$(EXTRA_CPPFLAGS) -nostdinc++" \
        $(TARGET_CONFIGURE_OPTS) \
        CROSS="$(TARGET_CROSS)" \
        ARCH="$(ARCH)" \
        $(1);
endef

# Specify where and how to install the program.
define Package/ForwardDhcpFrames/install
    $(INSTALL_DIR) $(1)/bin
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/
    ForwardDhcpFrames $(1)/bin/
endef

# This line executes the necessary commands to compile our program.
# The above define directives specify all the information needed, but this
# line calls BuildPackage which in turn actually uses this information to
# build a package.
$(eval $(call BuildPackage, ForwardDhcpFrames))

```

D APÉNDICE D: ARCHIVOS DE FIREWALL

D.1 Archivo firewall del ruteador con reglas para guardar tramas

DHCP (/etc/init.d/S35firewall)

```
#!/bin/sh

## Please make changes in /etc/firewall.user

. /etc/functions.sh
WAN="$(nvram get wan_ifname)"
WANDEV="$(nvram get wan_device)"
LAN="$(nvram get lan_ifname)"

## CLEAR TABLES
for T in filter nat; do
    iptables -t $T -F
    iptables -t $T -X
done

iptables -N input_rule
iptables -N output_rule
iptables -N forwarding_rule

iptables -t nat -N prerouting_rule
iptables -t nat -N postrouting_rule

iptables -N LAN_ACCEPT
[ -z "$WAN" ] || iptables -A LAN_ACCEPT -i "$WAN" -j RETURN
[ -z "$WANDEV" -o "$WANDEV" = "$WAN" ] || iptables -A LAN_ACCEPT -i
"$WANDEV" -j RETURN

iptables -A LAN_ACCEPT -j ACCEPT

iptables -t mangle -A INPUT -s IP_RED_LOCAL -j MARK --set-mark 100
iptables -A INPUT -m mark --mark 100 -p udp --sport 68 -j ULOG

### INPUT
### (connections with the router as destination)

# base case
iptables -P INPUT DROP
iptables -A INPUT -m state --state INVALID -j DROP
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A INPUT -p tcp --tcp-flags SYN SYN --tcp-option \! 2 -j DROP

#
# insert accept rule or to jump to new accept-check table here
#
iptables -A INPUT -j input_rule

# allow
iptables -A INPUT -j LAN_ACCEPT # allow from lan/wifi interfaces
iptables -A INPUT -p icmp -j ACCEPT # allow ICMP
iptables -A INPUT -p gre -j ACCEPT # allow GRE
```

```

# reject (what to do with anything not allowed earlier)
iptables -A INPUT -p tcp -j REJECT --reject-with tcp-reset
iptables -A INPUT -j REJECT --reject-with icmp-port-unreachable

### OUTPUT
### (connections with the router as source)

iptables -t mangle -A OUTPUT -d IP_RED_LOCAL -j MARK --set-mark 200
iptables -A OUTPUT -m mark --mark 200 -p udp --dport 68 -j ULOG

# base case
iptables -P OUTPUT DROP
iptables -A OUTPUT -m state --state INVALID -j DROP
iptables -A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

#
# insert accept rule or to jump to new accept-check table here
#
iptables -A OUTPUT -j output_rule

# allow
iptables -A OUTPUT -j ACCEPT          #allow everything out

# reject (what to do with anything not allowed earlier)
iptables -A OUTPUT -p tcp -j REJECT --reject-with tcp-reset
iptables -A OUTPUT -j REJECT --reject-with icmp-port-unreachable

### FORWARDING
### (connections routed through the router)

# base case
iptables -P FORWARD DROP
iptables -A FORWARD -m state --state INVALID -j DROP
iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-
to-pmtu
iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT

#
# insert accept rule or to jump to new accept-check table here
#
iptables -A FORWARD -j forwarding_rule

# allow
iptables -A FORWARD -i br0 -o br0 -j ACCEPT
iptables -A FORWARD -i $LAN -o $WAN -j ACCEPT

# reject (what to do with anything not allowed earlier)
# uses the default -P DROP

### MASQ
iptables -t nat -A PREROUTING -j prerouting_rule
iptables -t nat -A POSTROUTING -j postrouting_rule
iptables -t nat -A POSTROUTING -o $WAN -j MASQUERADE

## USER RULES
[ -f /etc/firewall.user ] && . /etc/firewall.user
[ -e /etc/config/firewall ] && {
    awk -f /usr/lib/common.awk -f /usr/lib/firewall.awk
    /etc/config/firewall | ash
}

```

D.2 Archivo firewall.user del ruteador con reglas de ebttables para redireccionar consultas y bloquear respuestas ARP (/etc/firewall.user)

```
#!/bin/sh
. /etc/functions.sh

WAN=$(nvram get wan_ifname)
LAN=$(nvram get lan_ifname)

iptables -F input_rule
iptables -F output_rule
iptables -F forwarding_rule
iptables -t nat -F prerouting_rule
iptables -t nat -F postrouting_rule

### BIG FAT DISCLAIMER
## The "-i $WAN" is used to match packets that come in via the $WAN
interface.
## it WILL NOT MATCH packets sent from the $WAN ip address -- you won't be
able
## to see the effects from within the LAN.

### Open port to WAN
## -- This allows port 22 to be answered by (dropbear on) the router
# iptables -t nat -A prerouting_rule -i $WAN -p tcp --dport 22 -j ACCEPT
# iptables -A input_rule -i $WAN -p tcp --dport 22 -j ACCEPT

### Port forwarding
## -- This forwards port 8080 on the WAN to port 80 on 192.168.1.2
# iptables -t nat -A prerouting_rule -i $WAN -p tcp --dport 8080 -j DNAT --
to 192.168.1.2:80
# iptables -A forwarding_rule -i $WAN -p tcp --dport 80 -d
192.168.1.2 -j ACCEPT

### DMZ
## -- Connections to ports not handled above will be forwarded to
192.168.1.2
# iptables -t nat -A prerouting_rule -i $WAN -j DNAT --to 192.168.1.2
# iptables -A forwarding_rule -i $WAN -d 192.168.1.2 -j ACCEPT

#ebtables
ebtables -A FORWARD -p ARP --arp-opcode 2 -i !vlan3 -j DROP
ebtables -t nat -A PREROUTING -p ARP --arp-opcode 1 -s ! MAC_SERVIDOR -d
ff:ff:ff:ff:ff:ff -j dnat --to-destination MAC_SERVIDOR
ebtables -t nat -A PREROUTING -p ARP --arp-opcode 1 -s ! MAC_SERVIDOR -d !
ff:ff:ff:ff:ff:ff -j dnat --to-destination MAC_SERVIDOR
```

E APÉNDICE E: VALORES OBTENIDOS DURANTE PRUEBAS DE RENDIMIENTO

E.1 Número de paquetes perdidos

Mbps	No. Peticiones Eco ICMP	No. Total de paquetes enviados	Solo OpenWrt	VLANs	VLANs con solución
10	13	130	0	0	0
20	26	520	0	0	0
30	39	1170	0	0	0
40	52	2080	0	0	0
50	65	3250	0	0	0
60	78	4680	0	0	0
70	91	6370	0	0	0
80	104	8320	0	0	0
90	117	10530	0	0	0
100	131	13100	0	0	0

E.2 Valores de tiempo de recepción de respuestas ARP

No. Trama	Solo OpenWrt (s)	VLANs (s)	VLANs con solución (s)
1	0,00011	0,000596	0,012287
2	0,000096	0,000567	0,012
3	0,0001	0,000577	0,016068
4	0,000099	0,000586	0,011889
5	0,000111	0,000565	0,015736
6	0,000099	0,000567	0,015551
7	0,000097	0,00058	0,01537
8	0,000099	0,000597	0,0152
9	0,000097	0,000581	0,01505
10	0,000107	0,000565	0,01429
11	0,000095	0,000564	0,0143
12	0,00009	0,000583	0,010412
13	0,000116	0,000577	0,014202
14	0,000097	0,00057	0,018069
15	0,000103	0,000586	0,017897
16	0,000097	0,000574	0,017705
17	0,000113	0,000569	0,013586
18	0,00011	0,000585	0,01343
19	0,000097	0,000577	0,017247
20	0,000098	0,000589	0,013089
21	0,000115	0,000612	0,012949

22	0,000101	0,000592	0,016798
23	0,000105	0,000581	0,014648
24	0,000111	0,000576	0,016464
25	0,000096	0,000563	0,01521
26	0,000096	0,000572	0,016126
27	0,000096	0,000577	0,01535
28	0,000114	0,000567	0,015813
29	0,000094	0,000576	0,015647
30	0,000113	0,000573	0,015436
31	0,000095	0,000621	0,014289
32	0,000102	0,000575	0,01512
33	0,000115	0,000564	0,014963
34	0,000102	0,000576	0,014799
35	0,000113	0,000568	0,014518
36	0,000094	0,000578	0,014343
37	0,00012	0,000571	0,014187
38	0,000108	0,000574	0,014028
39	0,000098	0,000581	0,01784
40	0,000103	0,000606	0,013659
Promedio	0,00010305	0,00057895	0,014889125

E.3 Valores de tiempo de recepción de respuestas eco ICMP

No. Paquete	Solo OpenWrt (s)	VLANs (s)	VLANs con solución (s)
1	0,001038	0,000624	0,033203
2	0,000144	0,000683	0,000685
3	0,000143	0,00068	0,000623
4	0,000135	0,000676	0,000693
5	0,000146	0,000686	0,000665
6	0,000149	0,000627	0,000686
7	0,000141	0,000735	0,000685
8	0,000143	0,000673	0,000701
9	0,000142	0,000678	0,000683
10	0,000136	0,000674	0,000716
11	0,000142	0,000676	0,000688
12	0,000143	0,000698	0,000718
13	0,000143	0,000678	0,000689

14	0,000144	0,000674	0,000715
15	0,000143	0,000676	0,000688
16	0,000146	0,000674	0,0006
17	0,000141	0,000686	0,000684
18	0,000144	0,000697	0,000686
19	0,000137	0,000674	0,000687
20	0,000145	0,00067	0,000683
21	0,000143	0,000684	0,000688
22	0,000142	0,00069	0,000671
23	0,000147	0,000678	0,000685
24	0,000144	0,000709	0,000691
25	0,000141	0,000685	0,000672
26	0,000146	0,000706	0,000684
27	0,000146	0,000681	0,000681
28	0,000148	0,000711	0,000711
29	0,000135	0,000681	0,000681
30	0,000144	0,000693	0,000676
31	0,000144	0,000676	0,000654
32	0,000149	0,000679	0,000682
33	0,000145	0,000689	0,000603
34	0,000133	0,000678	0,000605
35	0,000145	0,000676	0,000687
36	0,00015	0,0007	0,000687
37	0,000144	0,000679	0,000674
38	0,000143	0,000681	0,00068
39	0,000143	0,000631	0,000683
40	0,000141	0,000678	0,000704
Promedio	0,00016545	0,0006806	0,001491925

REFERENCIAS DE GRÁFICOS

[F1] “RFC 2131”, 1997, <<http://www.ietf.org/rfc/rfc2131.txt>> [Consulta: Sábado, 19 de enero de 2008]

[F2] “OpenWrtDocs/Hardware/Linksys/WRTSL54GS”, 2007, <<http://wiki.openwrt.org/OpenWrtDocs/Hardware/Linksys/WRTSL54GS>> [Consulta: Sábado, 8 de diciembre de 2007]

REFERENCIAS BIBLIOGRÁFICAS

- [1] “RFC 826”, <<http://www.ietf.org/rfc/rfc826.txt>> [Consulta: Sábado, 6 de octubre de 2007].
- [2] “Secure Neighbor Discovery Protocol”, <http://en.wikipedia.org/wiki/Secure_Neighbor_Discovery> [Consulta: Lunes, 2 de junio de 2008].
- [3] **Abad, Cristina, Bonilla, Rafael**. “An Analysis of the Schemes for Detecting and Preventing ARP Cache Poisoning Attacks”. En *Proceedings of the IEEE International Conference on Distributed Computing Systems Workshops (ICDCS 2007 Workshops)*, Toronto, Canadá, Junio 2007, ISBN: 0-7695-2838-4.
- [4] “ARP Spoofing”, <http://es.wikipedia.org/wiki/ARP_Spoofing> [Consulta: Viernes, 1 de agosto de 2008].
- [5] **Pérez Crespo, Jaime**, “Envenenamiento ARP”, <http://blackspiral.org/docs/arp_spoofing.html> [Consulta: Viernes, 10 de octubre de 2007].
- [6] **Philip, Roney**, “Securing Wireless Networks from ARP Cache Poisoning”. *In partial Fulfillment of the Requirements for the Degree Master of Computer Science*, San Jose State University, 2007.
- [7] “Arpwatch”, <<http://en.wikipedia.org/wiki/Arpwatch>> [Consulta: Jueves, 10 de julio de 2008].
- [8] “Snort.org”, <<http://www.snort.org/>> [Consulta: Jueves, 10 de julio

de 2008].

[9] **M. Carnut, J. Gondim**. “ARP Spoofing Detection on Switched Ethernet Networks: A Feasibility Study”. En *Proceedings of the 5th Symposium on Security in Informatics*, São Paulo, Brasil, Noviembre 2003.

[10] “Arp Guard”, <<https://www.arp-guard.com/>> [Consulta: Jueves, 10 de julio de 2008].

[11] **T. Demuth, A. Leitner**. “ARP Spoofing an Poisoning: Traffic Tricks”. En *Linux Magazine*, 56:26-31, Julio 2005.

[12] “Ebttables”, <<http://ebtables.sourceforge.net/>> [Consulta: Jueves, 10 de julio de 2008].

[13] “Configure your Catalyst for a more secure layer 2”, <<http://www.enterprisenetworkingplanet.com/netsecur/article.php/3462211>> [Consulta: Jueves, 10 de julio de 2008].

[14] “An introduction to ARP spoofing 2600: The Hacker Quarterly”, <<http://www.node99.org/projects/arpspoof/arpspoof.pdf>> [Consulta: Jueves, 10 de julio de 2008].

[15] **M. Tripunitara, P. Dutta**. “A Middleware Approach to Asynchronous and Backward Compatible Detection and Prevention of ARP Cache Poisoning”. En *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC '99)*, Diciembre 1999.

[16] “Anticap”, <<http://www.antifork.org/viewcvs/trunk/anticap>>

[Consulta: Jueves, 10 de julio de 2008].

[17] “Antidote”, <<http://antidote.sourceforge.net/>> [Consulta: Jueves, 10 de julio de 2008].

[18] **M. Gouda, C. T. Huang**. “A secure Address Resolution Protocol”. En *Computer Networks*, 41(1):57-71, Enero 2003.

[19] **D. Bruschi, A. Ornaghi, E. Rosti**. “S-ARP: A Secure Address Resolution Protocol”. En *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC '03)*, Diciembre 2003.

[20] **V. Goyal, V. Kumar, M. Singh**. “A new architecture for address resolution”, 2005. Sin publicar.

[21] Cisco Systems. Configuring Dynamic ARP Inspection, Cap. 39 pág. 39:1–39:22. 2006. Catalyst 6500 Series Switch Cisco IOS Software Configuration Guide, Release 12.2SX.

[22] “Aprendiendo a Programar con Libpcap”, < <http://www.e-ghost.deusto.es/docs/2005/conferencias/pcap.pdf>> [Consulta: Martes, 30 de octubre de 2007].

[23] **Asadoorian, Paul y Pesce, Larry**. “Linksys WRT54G Ultimate Hacking”, Burlington, MA, 2007, p. 68 [Consulta: Sábado, 12 de julio de 2008].

[24] “OpenWrt”, <<http://openwrt.org/>> [Consulta: Martes, 30 de octubre de 2007].

[25] “DD-WRT”, <<http://www.dd-wrt.com/dd-wrtv3/index.php>>

[Consulta: Martes, 30 de octubre de 2007].

[26] “eWRT”,
<<http://www.linksysinfo.org/forums/forumdisplay.php?f=141>>

[Consulta: Martes, 30 de octubre de 2007].

[27] “RFC 2131”, <<http://www.ietf.org/rfc/rfc2131.txt>> [Consulta: Sábado, 19 de enero de 2008].

[28] **Bishop, Eric**, “Writing and Compiling A Simple Program For OpenWrt”, <<http://gargoyle-router.com/openwrt-coding.html>>

[Consulta: Viernes, 13 de junio de 2008].

[29] “OpenWrt Wireless Freedom”, <<http://downloads.openwrt.org>>

[Consulta: Viernes, 13 de junio de 2008].

[30] “Tubería nombrada”, <http://es.wikipedia.org/wiki/Named_pipe>

[Consulta: Lunes, 9 de junio de 2008].

[31] “Ataque Man-in-the-middle”,
<http://es.wikipedia.org/wiki/Ataque_Man-in-the-middle> [Consulta:

Lunes, 27 de octubre de 2008].

[32] “Cain & Abel”, <<http://www.oxid.it/cain.html>> [Consulta: Sábado, 4 de octubre de 2008].

[33] **Weigle, Eric, Feng, Wu-chun**. “TICKETing High-Speed Traffic with Commodity Hardware and Software”. En *Passive & Active Measurement Workshop (PAM2002)*, Fort Collins, Colorado, Marzo 2002.