Trabajo - Instalación, Administración y Uso de un servicio en Linux

<Servidor proxy SOCKS con cache HTTP: SQUID>

Servicios Telemáticos

Departamento de Ingeniería Telemática

IMPORTANTE: **Antes** de rellenar esta plantilla **lea detenidamente** los documentos "**Enunciado** del Trabajo" y "**FAQ** del Trabajo".

El contenido esperado para cada uno de los apartados de esta memoria se indica en el **Anexo A** del documento "**Enunciado** del Trabajo".

<Carlos Carballo Ortiz>

ÍNDICE

1. Objetivos y Alcance

1.1. Introducción

En un mundo tan globalizado en el que cada vez hay más restricciones y vulnerabilidades de los servicios, se opta por la opción, sobre todo en el ámbito empresarial, de la mayor protección y seguridad posible de los datos. También es un hecho el aumento de control e intrusión de las agencias gubernamentales. El concepto proxy ha comenzado a volverse cada vez más popular, y se presenta como una nueva alternativa para esa seguridad por la que cada vez más se lucha.

1.2. Motivación y funcionalidad del servicio

Nos despierta nuestro interés saber sobre el funcionamiento de un proxy, debido a que se ha vuelto cada vez más una necesidad para muchas personas debido a múltiples vulnerabilidades relacionadas con la privacidad de los datos. En esta memoria nos adentraremos a conocer lo que es un proxy, los modos de funcionamiento que puede tener y la inmensa cantidad de funcionalidades que ofrece y las que van a llegar.

Las funcionalidades más comunes y por las que más se usan los *proxies* se pueden enumerar en las siguientes:

- Otorga un mayor grado de anonimato,
- Permite acceso a contenido restringido o bloqueado geográficamente,
- Balanceo de carga en múltiples servidores,
- Caché de recursos,
- Seguridad
  - ✓ Protección de ataques DDOS
  - ✓ Limitación de ancho de banda de ciertos equipos
  - ✓ Bloqueo de direcciones IP
  - ✓ Uso de blacklists
  - ✓ Autenticación para el acceso
  - ✓ Registro de logs y monitorización del tráfico.

1.3. Documentación bibliográfica

Referencias:

[1] https://developer.mozilla.org/es/docs/Web/HTTP/Overview

[2] https://developer.mozilla.org/es/docs/Web/HTTP/Messages

[3] RFC 1928

[4] http://www.squid-cache.org/

[5] Saini K.: *Squid Proxy Server 3.1: Beginner's Guide*, Open Source community, February 2011.

[6] http://www.squid-cache.org/Intro/who.html

[7] http://www.delegate.org/delegate/

[8] https://www.inet.no/dante/index.html

[9] http://www.squid-cache.org/Doc/config/

2. Base Teórica

El servicio proxy tiene como función principal la actuación de un equipo como intermediario entre dos partes remotas de la red (cliente-servidor), permitiendo una conexión indirecta entre ellos. El tipo de servicio que ofrece se puede diferenciar según el protocolo para el que se realiza la intermediación (proxy ARP, FTP, SSL…), y en este trabajo vamos a centrarnos en dos tipos de proxy concretos: HTTP y SOCKS, que definiremos a continuación.

2.1. Descripción del servicio y conceptos implicados

En un servicio proxy, la función de intermediario la hace el **servidor proxy**, que no es más que un software que corre en un sistema operativo de cierto equipo informático, y permite a los usuarios conectarse a él, con el fin de generar una comunicación indirecta entre remitente y destinatario. Un servidor proxy puede tener varios modos de funcionamiento, como es el *modo trasparente*, que por defecto recibe el tráfico cliente y este lo suministra al destino sin modificar nada de la petición (solo cumple con la función de reenvío); el *modo directo*, que

requiere que en los clientes se configure la dirección del servidor para poder ser usado, y ofrece cierto grado de anonimato al realizar las peticiones con la dirección IP del propio servidor y no del equipo que la genera; o el *modo inverso*, que realiza el proceso contrario al anterior, ya que actúa en el lado del servidor (con uno o varios servidores) y todo el tráfico procedente de internet con destino a esa red de servidores es gestionado y reenviado/atendido por ese proxy. Este modo proporciona distintas utilidades desde el lado servidor como el balanceo de carga entre varios servidores, el uso de caché de contenido (normalmente estático) que permite al propio servidor proxy atender peticiones de clientes actuando como "servidor medio", proporcionando ahorro de ancho de banda y reducción de tráfico a los servidores internos, o aumento de la seguridad de la red de servidores con opciones como acceso restringido, autenticación, monitorización de tráfico o la protección y gestión más sencilla de los ataques DDOS (ya que atacaría al servidor proxy y los servidores internos podrían continuar con su función mientras se mitiga desde ese equipo).

Destacaremos dos tipos de servidores proxy para este trabajo:

- **Proxy HTTP:** utilizado para atender peticiones con el protocolo HTTP/HTTPS y para el cacheo de páginas web.

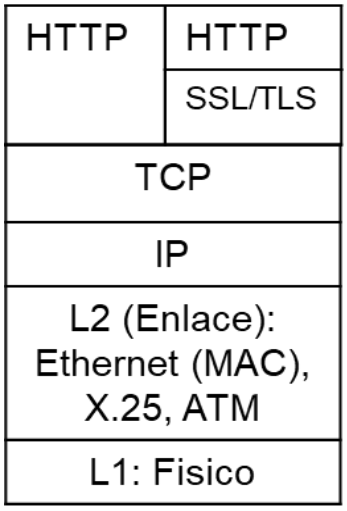
- **Proxy SOCKS:** se apoya en el protocolo SOCKS de la capa de sesión del modelo OSI, y atiende peticiones de cualquier protocolo de la capa de aplicación siempre y cuando vayan sobre TCP y UDP (este último desde SOCKS5, última actualización del protocolo).

2.2. Protocolos utilizados por el servicio

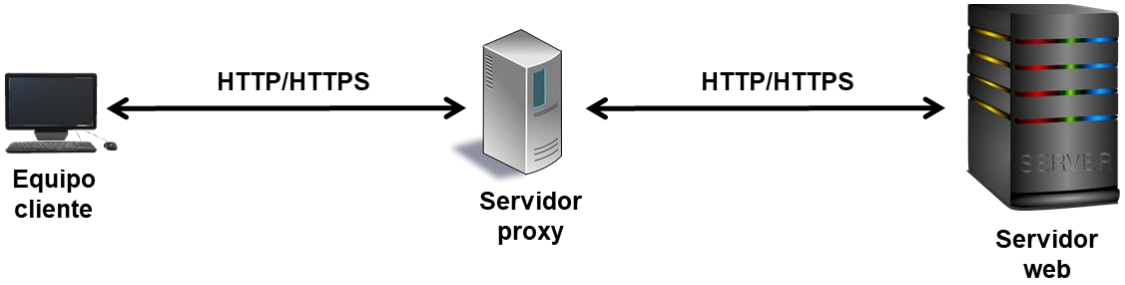
Para los tipos de proxy que abarcaremos en el trabajo, existen distinciones en los protocolos que soportan cada uno:

Proxy HTTP

La pila de protocolos de este proxy es:

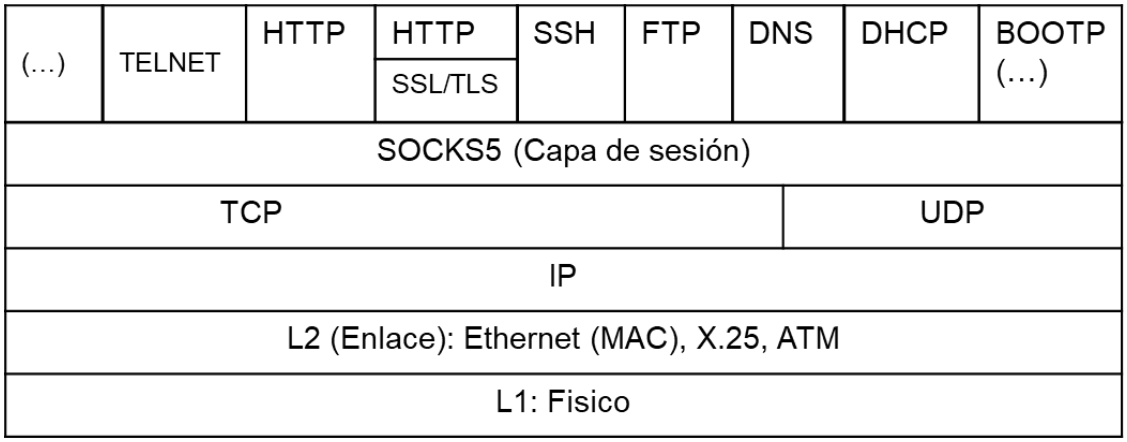


Como antes hemos mencionado, este tipo de proxy actuaría como intermediario entre un equipo cliente y un servidor web, ya que únicamente atiende/reenvía peticiones del protocolo HTTP/HTTPS. Gráficamente:

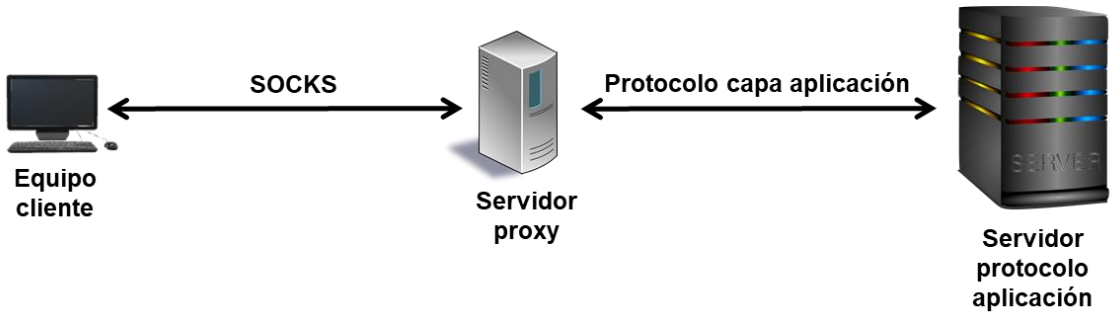


Proxy SOCKS

La pila de protocolos de este proxy es:



Este tipo de proxy atiende peticiones de cualquier protocolo de aplicación que funcione sobre los protocolos de TCP o UDP. Para ello, el equipo cliente interesado en establecer conexión indirecta mediante un proxy SOCKS, debe autenticarse con el servidor proxy mediante el protocolo SOCKS5. Tras este proceso, el proxy se comunicará con el servidor final mediante el protocolo de la capa de aplicación que se haya establecido por el cliente. Gráficamente:



Se dará un funcionamiento más detallado de los principales protocolos de cada proxy en los siguientes puntos.

2.2.1. Protocolos Comunes

Para el Proxy HTTP únicamente podemos destacar el protocolo HTTP (HTTP/1.1 – RFC 7230-5 y HTTP/2.0 – RFC 7540), del que analizaremos su funcionamiento en el siguiente punto.

Para el Proxy SOCKS, podemos destacar algunos protocolos de aplicación que se apoyen más comúnmente en SOCKS5 para la comunicación indirecta mediante proxy, de tipo cliente-servidor, como pueden ser:

- HTTP/HTTPS (previamente mencionado)
- SSH (RFC 4253)
- FTP (RFC 959), SFTP (913), TFTP (RFC 1350)
- TELNET (RFC 854-5)
- DNS (RFC 1034-5)
- RPC (RFC 5531)

2.2.2. Protocolos Específicos del servicio

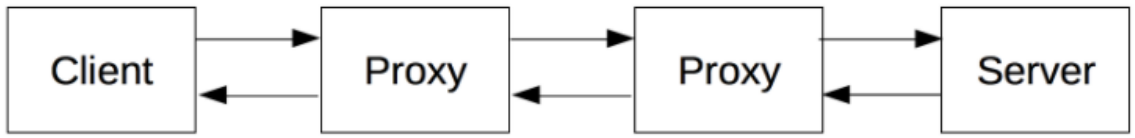
Hablaremos de los dos protocolos principales por los que se define cada tipo de proxy.

HTTP (*Hypertext Transfer Protocol*)

Fijandonos en la referencia de [1], se trata de un protocolo de modelo cliente-servidor, el cual nos permite realizar peticiones de datos y recursos (de tipo HTML, XML, JavaScript…) a través de agentes clientes (como navegadores) a los distintos servidores de la web. Es, por tanto, la base de cualquier intercambio de datos en la Web. Fue desarrollado por el ***World Wide Web Consortium*** y la ***Internet Engineering Task Force (IETF)***, colaboración que culminó en 1999 con la publicación de una serie de RFC, siendo el más importante de ellos el **RFC 2616** que especifica la versión 1.1. HTTP.

Aparte de transmitir documentos de hipertexto (HTML), también se usa para la transmisión de videos, imágenes o partes de documentos, para actualizar páginas, o para el envío de datos y contenido a los servidores (como es el caso de los formularios).

Cada petición individual se envía a un servidor, el cual la gestiona y responde. Entre cada petición y respuesta, puede haber varios intermediarios, como los *proxies* que abarcamos en este trabajo, que pueden realizar distintas funciones como *gateway* o caché.



FLUJO DE MENSAJES

Siguiendo la información de [2], cuando un cliente quiere comunicarse con un servidor, se produce una conexión TCP entre ambos, que será la que se use para realizar el envío y recepción de los mensajes HTTP. Hay dos tipos de mensajes: peticiones (*HTTP Request*), enviadas por el cliente al servidor para pedir el inicio de una acción, y respuestas (*HTTP Response*) del servidor. Ambos mensajes comparten una estructura similar, que puede ser visualizada en texto plano para los mensajes previos a HTTP/2.0. Se estructuran con:

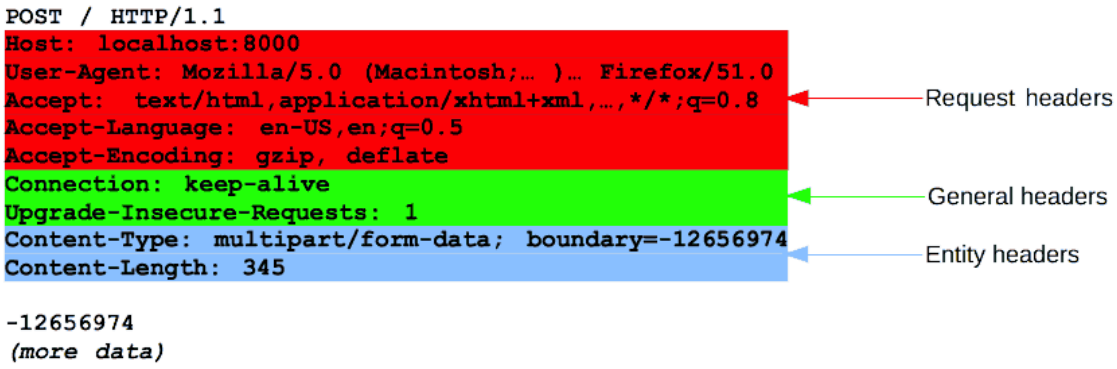
1. Una línea de inicio (única) que describe la petición a implementar o el mensaje de éxito o fracaso de la respuesta.

2. Un grupo opcional de cabeceras HTTP, que son cadenas de caracteres con el modelo "*parámetro: valor"* formadas en una única línea.

3. Una línea vacía para indicar la finalización de los metadatos del mensaje.

4. Un campo de cuerpo de mensaje que lleva los datos asociados a la petición, o los archivos y documentos de la respuesta (el tamaño del cuerpo se indica en la cabecera).

Las peticiones HTTP tienen este formato:



Esta estructura de mensaje la podemos dividir en 3 partes,

- Una línea de inicio formada por 3 elementos, donde:

    1. El primero indica el método HTTP, que puede ser **GET** (solicita una representación del recurso especificado en la URL, con lo cual indica que un archivo ha de ser enviado hacia el cliente), **POST** (envía datos para que los procese el recurso identificado por la URI de la petición, con el objetivo de crear un nuevo recurso cuya naturaleza vendrá especificada por la cabecera Content-Type) o **PUT** (similar a POST, pero se diferencia en que llamar una o varias veces al mismo recurso en PUT tiene el mismo efecto, en POST puede haber efectos adicionales). También puede llevar las acciones de **HEAD, DELETE** y **OPTIONS**.

    2. El segundo elemento indica el destino de la petición, normalmente una URL.

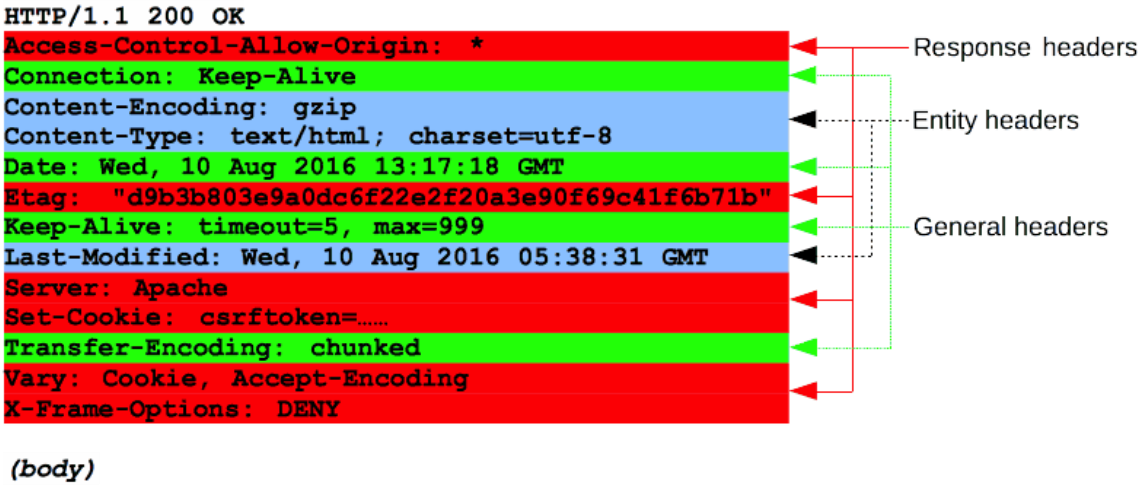
    3. El último elemento indica la versión HTTP que define la estructura del mensaje, que actúa como indicador de la versión que buscamos en la respuesta.

- Las cabeceras HTTP, entre las que destacan:

- *User-Agent*: tipo de aplicación (navegador) y espera la respuesta.
- *Content-Type:* Tipo MIME del cuerpo del mensaje.
- *If-Modified-Since*: fecha de la última actualización del recurso en la caché.
- *Host*: extraído de la URL, nombre de dominio del servidor.

- Cuerpo. No todas las peticiones lo llevan, depende del método usado indicado en la línea inicial de la petición (es el caso del método POST). Puede llevar un único archivo definido por las cabeceras *Content-Type* y *Content-Length*, o varios y distintos contenidos.

Las respuestas HTTP tienen este formato:

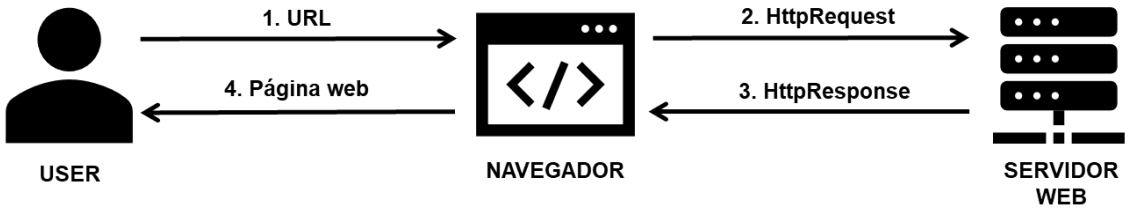


- La línea de inicio se conoce como línea de estado , y contiene:
  1. La versión del protocolo.
  2. El código de estado indicando éxito o fracaso. Los códigos de estado se definen en la sección 10 de RFC 2616.
  3. El texto de estado que acompaña al código, a modo informativo.
- Cabeceras HTTP, entre las que podemos destacar:
  - *Server*: servidor que responde

- *Last-Modified*: última modificación del recurso.
- *Vary*: determina como hacer coincidir los encabezados de las solicitudes futuras para decidir si se puede utilizar una respuesta almacenada en caché en lugar de solicitar una nueva desde el servidor de origen.
- *Location*: en caso de redirección a otra URL porque la que haya sido accedida esté obsoleta.

- Cuerpo. Archivos y documentos de respuesta. No todas las respuestas tienen uno, depende del código de estado (*201-Created* y *204- No Content* normalmente prescinden de él).

La secuencia de mensajes de red mediante el protocolo HTTP, en un escenario muy simplificado, sería así:



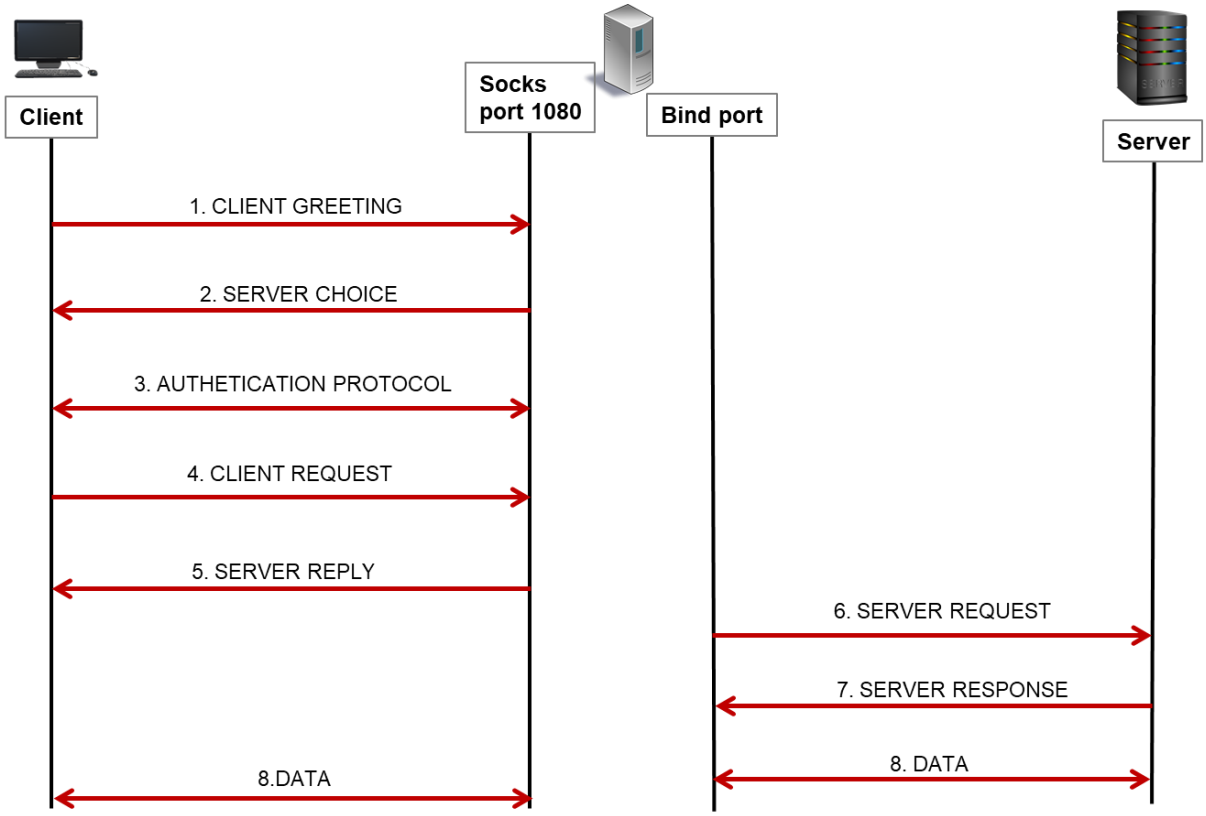
SOCKS5

Protocolo definido en el RFC 1928, y se sitúa en la capa de sesión según el modelo OSI (capa 5), entre la capa de aplicación y la capa de transporte. SOCKS aparece para utilizarse junto con sistemas firewall para ofrecer un acceso seguro de las redes tanto internas como externas que comunican este firewall. El uso de cortafuegos de red cada vez es más popular, y debido a la aparición de protocolos de aplicación cada vez más sofisticados diseñados para el descubrimiento de información global, existe la necesidad de proporcionar un marco general para que estos protocolos sigan atravesando los firewalls de forma transparente y segura. Se añade la necesidad de una autenticación fuerte, por la necesidad de controlar las

relaciones cliente-servidor entre redes de varias organizaciones. Por lo tanto, el protocolo descrito va a proporcionar un marco para aplicaciones cliente-servidor en los dominios TCP y UDP (este último es implementado en la última actualización del protocolo, SOCKS5) para permitir el correcto uso de los servicios de cortafuegos. El protocolo no proporciona servicios de red, como el reenvío de mensajes ICMP.

Previamente a SOCKS5, ya existía un protocolo, SOCKS versión 4, que permite atravesar el cortafuegos de forma segura para las aplicaciones cliente-servidor basadas en TCP, incluyendo TELNET, FTP y los protocolos populares de búsqueda de información como HTTP, WAIS y GOPHER. Este nuevo protocolo amplía el modelo SOCKS versión 4 para incluir UDP, y amplía el marco para incluir esquemas de autenticación fuerte, además de ampliar el esquema de direccionamiento para nombres de dominio y direcciones IPv6.

A continuación, apoyándonos en [3], vamos a analizar el funcionamiento y flujo de mensajes del protocolo SOCKS5, desde un escenario típico en el que un cliente (que soporta SOCKS) solicita conexión a un servidor, accediendo mediante un servidor proxy SOCKS que actuará de intermediario en la conexión indirecta, y desde una aplicación compatible con SOCKS. El esquema del flujo de mensajes sería el siguiente:



Todos los mensajes que se envían en el protocolo van a representarse en tablas donde la primera fila va a representar el nombre del campo, y la segunda fila el número de octetos que ocupa (=el tamaño). Si hablamos de un valor en concreto de algún octeto, lo vamos a representar con la sintaxis X'hh'.

1. CLIENT GREETING

   Este mensaje se genera cuando un cliente desea establecer conexión con un proxy SOCKS. El cliente abre una conexión TCP al puerto SOCKS (1080) y espera la confirmación del servidor SOCKS. En el mensaje, el cliente envía una lista de métodos de autenticación que soporta, para que el servidor SOCKS seleccione uno que autentique al cliente.

VER	NMETHODS	METHODS
1	1	1 to 255

   Donde los campos representan:

   - VER = X'05' (versión de SOCKS5 en este caso).
   - NMETHOD = número de métodos de autenticación soportados
   - METHOD = métodos de autenticación, 1 byte por método. Se pueden ver numerados en https://en.wikipedia.org/wiki/SOCKS#SOCKS5 todos los métodos de autenticación soportados en el protocolo.

2. SERVER CHOICE

   Si el servidor acepta la conexión TCP, selecciona uno de los métodos de autenticación del campo métodos del cliente, y envía el mensaje de selección al cliente.

VER	METHOD
1	1

   Si el valor de METHOD es X'FF', ninguno de los métodos listados por el cliente es aceptado, y el cliente DEBE cerrar la conexión.

- Valores definidos para METHOD:
    - X’00’ NO AUTHENTICATION REQUIRED
    - X’01’ GSSAPI
    - X’02’ USERNAME/PASSWORD (RFC 1929)
    - X’03’ to X’7F’ IANA ASSIGNED
    - X’80’ to X’FE’ RESERVED FOR PRIVATE METHODS
    - X’FF’ NO ACCEPTABLE METHODS

3. AUTHENTICATION PROTOCOL

   Si el servidor SOCKS ha seleccionado un método en el anterior mensaje, cliente y servidor seguidamente entran en una subnegociación de autenticación con el método escogido.

4. CLIENT REQUEST

Cuando el método de autenticación se completa, el cliente envía los detalles de la solicitud.

VER	CMD	RSV	ATYP	DTS.ADDR	DST.PORT
1	1	X’00’	1	Variable	2

Donde los campos representan:

- VER = X’05’ (versión de SOCKS5 en este caso).
- CMD
    - CONNECT X'01' (El cliente quiere establecer una conexión TCP con servidor).
    - BIND X'02' (El cliente espera que el servidor establezca una conexión TCP con él).
    - UDP ASSOCIATE X'03' (El cliente quiere establecer una conexión UDP con servidor).

- RSV RESERVED
- ATYP = tipo de dirección de la siguiente dirección
    - IPv4 address: X'01' (4 octetos en DST.ADDR)
    - DOMAINNAME: X'03' (nombre de dominio, donde el 1er octeto contiene el número de octetos del nombre que le sigue. No hay octeto de terminación NULL)
    - IPv6 address: X'04' (16 octetos en DST.ADDR)
- DST.ADDR = dirección destino deseada
- DST.PORT = puerto deseado

5. SERVER REPLY

El servidor SOCKS evalúa la solicitud basándose en origen y destino, y devuelve una o más respuestas según sea apropiado con el tipo de solicitud.

VER	REP	RSV	ATYP	BND.ADDR	BND.PORT
1	1	X'00'	1	Variable	2

Donde los campos antes no mencionados son:

- REP Reply field:
    - X'00' succeeded
    - X'01' general SOCKS server failure
    - X'02' connection not allowed by ruleset
    - X'03' Network unreachable
    - X'04' Host unreachable
    - X'05' Connection refused
    - X'06' TTL expired
    - X'07' Command not supported
    - X'08' Address type not supported

- X'09' to X'FF' unassigned

- BND.ADDR = dirección con la que el proxy se conecta al destino
- BND.PORT = puerto asociado

6. SERVER REQUEST

   Petición que el server SOCKS hace actuando como si fuera el cliente final, con el servidor final.

7. SERVER RESPONSE

   Respuesta del servidor que será reenviada desde el proxy SOCKS al cliente final.

3. Evaluación de la implementación estudiada

En este apartado vamos a referirnos concretamente al servicio del que trata el trabajo. En primer lugar, destacaremos la funcionalidad de Squid, sus características, el equipamiento necesario para su implementación (tanto en hardware como en software), y como segundo punto realizaremos un análisis comparativo con implementaciones existentes en el mercado que puedan ser reconocidas como competencia, al ofrecer un tipo de servicio similar.

3.1. Referencias y características de la implementación estudiada

3.1.1. Breve Descripción de la solución adoptada

Basándonos en la información que nos da la página web oficial del servicio [4], Squid es un servidor proxy web con caché, y es conocida como una de las aplicaciones más populares y de referencia para esta función. Está desarrollado en software libre y publicado bajo la licencia de GNU GPL. Soporta los protocolos HTTP, HTTPS, FTP, GOPHER Y WHAIS.

Según relata en su introducción [5], el servidor proxy Squid tiene dos modos de funcionamiento principales, de los que se sacan diferentes funcionalidades. Estos son el modo directo, que trabaja en el lado cliente, y el modo "reverse", del lado del servidor. Más adelante explicamos más a fondo cada función.

En la siguiente tabla abarcamos sus características más destacables:

FUNCIONALIDADES	Proxy directo - Reducir uso de banda ancha de la red, y reducir tiempo de navegación del cliente al poder ser el propio proxy quien le proporciona los recursos necesarios de su caché local. - Establecer políticas de acceso. - Monitorizar tráfico de los usuarios que se conectan al proxy. Filtrar peticiones y respuestas mediante detectores de virus/malware. - Proporcionar cierto grado de privacidad al usuario, no exponiendo sus datos directamente a internet. *Reverse* proxy - Reducir uso de ancho de banda al cachear contenido estático de páginas web que no varían frecuentemente, como documentos CSS, JavaScript, imágenes, etc., y no tener que solicitarlos y volverlos a reenviar por cada petición. - Para reducir el tiempo medio de carga de las páginas y mejorar la experiencia de navegación de los clientes. - Para aliviar la carga de un servidor web muy ocupado actuando como servidor medio. - Para balanceo de carga en un sitio web de varios servidores.
REFERENCIAS BIBLIOGRÁFICAS	[4], [5], [6],[9]
DESARROLLADOR	Según [6], originalmente financiado por una subvención de NSF, y desarrollado por voluntarios, entre los que más se han dedicado: Duane Wessels (coordinador), Amos Jeffries, Henrik Nordström o Alex Rousskov.

VERSIÓN ESTUDIADA (FECHA)	Squid-v5.2 (3 de octubre de 2021)
ÚLTIMA VERSIÓN (FECHA)	Squid-v5.3 (7 diciembre de 2021) squid-5.3-20211214-r832aa256c (15 de diciembre 2021) (*bug-fixed update*)

3.1.2. Equipamiento necesario

HARDWARE NECESARIO	**Equipo cliente, equipo donde implementar el proxy Squid y con reserva de memoria caché, equipo servidor.**
INFRAESTRUCTURA DE RED REQUERIDA	Proxy directo: Para conexión HTTP: Equipo cliente con acceso a servidor proxy, equipo proxy con acceso a equipo servidor, que tenga implementado el servidor proxy Squid y equipo servidor HTTTP con servidor web implementado (en los escenarios usaremos el servidor Apache). Proxy inverso: Para conexión HTTP: Equipo servidor web con Apache (aislado por el proxy, todas las peticiones le llegan desde el equipo proxy), equipo proxy con Squid implementado que reciba peticiones de equipos clientes, equipos clientes con conexión a proxy HTTP.
ELEMENTOS SOFTWARE REQUERIDOS PARA INSTALACIÓN	Librerias: libc, libcap, libcrypt, openssl, libdb, glibc, gcc, make, c++, libecap, gss api, libltdl, libldap, libpam, cyrus-sasl-lib, libxml, perl, rtld (GNU-HASH), squid-migration-script.
S.O CONCRETO USADO	CentOS 7

3.1.3. Características y funcionalidades

Como se comentó previamente en el punto 3.1.1, el servidor proxy Squid trabaja de una manera u otra dependiendo de su modo de funcionamiento. En ambos modos, el papel de la caché es esencial, ya que la función principal es que, a partir de cierta petición de un cliente, el proxy almacene en su memoria local archivos y recursos de respuesta de los servidores que se acceden, para que al recibir otra petición similar del mismo cliente o de otros, pueda ser el proxy el que proporcione la información y no sea necesario reenviarla al servidor para recibir la misma respuesta nuevamente.

Vamos a centrarnos en los dos modos de funcionamiento principales, tomando como referencia [5]:

- El modo "directo", la forma más simple, que permite la comunicación indirecta de la parte cliente con los servidores que tengan como objetivo. Un caso de uso para este modo sería el siguiente: cuando iniciamos una solicitud de un recurso del servidor de destino, el servidor proxy se hace con nuestra conexión y se representa a sí mismo como un cliente para el servidor de destino, solicitando el recurso en nuestro nombre. Si se recibe una respuesta, el servidor proxy nos la devuelve, dando la sensación de que nos hemos comunicado con el servidor de destino. A partir de este caso básico se puede utilizar con funcionalidades más avanzadas como filtración de solicitudes basándose en varias reglas (basadas en dirección IP, protocolo, contenido, etc.), modificación de peticiones/respuestas, o almacenamiento local de respuestas (en caché) para satisfacer la misma solicitud de otros clientes, ahorrando ancho de banda, y potenciando la experiencia de navegación de los usuarios finales.

- El modo *"reverse"*, que actúa en la parte del servidor, y es una técnica que consiste en almacenar localmente las respuestas o recursos de un servidor web (o red de servidores de un mismo sitio web) para que las siguientes peticiones al mismo recurso puedan ser satisfechas desde la copia local en el servidor proxy, actuando como un servidor medio. El servidor proxy o la caché web comprueba si la copia almacenada localmente sigue siendo válida antes de servirla. En el caso de varios servidores de un mismo recurso, también puede servir como balanceador de carga, controlando la cantidad de peticiones que envía a cada servidor.

3.2. Comparativa de soluciones existentes en el mercado

	SQUID	DANTE	DELEGATE	SOCAT
Última versión/Fecha	V5.3 7 dic 2021	V1.4.3 abril 2021	V9.9.12 28 sept 2014	V1.7.3.3 13 oct 2019
SO soportado	Linux, Windows, FreeBSD, NetBSD, Open Solaris	Linux, OpenBSD, FreeBSD, AIX, SunOS	Linux, Unix, Windows, OS/2, Windows Mobile/CE	Linux, OpenBSD, FreeBSD, MacOS, Open Solaris, AIX
Equipo de desarrollo	Subvencionado originalmente por NSF y desarrollado por voluntarios (Henrik Nordström es el presidente).	Inferno Nettverk A/S (empresa de ingeniería software y consultora de software UNIX y sistemas embebidos)	Yutaka Sato (Japón)	Gerhard Rieger (desarrollador de software considerado experto en el campo de Linux security.
Licencia	GNU General Public License (En desarrollo y gratuito) [4]	BSD/CMU-Type License (En desarrollo y gratuito) [8]	Licensed by AIST (En desarrollo y gratuito) [7]	GNU General Public License v2 (En desarrollo y gratuito)
Lenguaje	C++	C++	C++	C
Funcionalidad (diferencias)	Servidor proxy web con caché. Soporta los protocolos HTTP, HTTPS, FTP, GOPHER Y WHAIS. Funcionalidad de Proxy SOCKS está en Testing.	Se divide en parte cliente SOCKS y parte servidor SOCKS. Ofrece la posibilidad de que las aplicaciones (que trabajen SOBRE TCP /UDP) que no soportan SOCKS, se apoyen en Dante para comunicarse	Servidor proxy con caché, actúa como proxy HTTP y como proxy SOCKS. Soporta HTTP, FTP, POP3, IMAP, LDAP, Telnet, SMTP, DNS.	Es una herramienta de línea de comandos. Establece dos flujos de bytes bidireccionales y transfiere datos entre ellos. La comunicación se puede producir en

		con servidores que soporten SOCKS.		protocolos que trabajan sobre TCP, UDP o SCTP

3.3. Clientes para el servicio

Del lado del cliente, enviaremos las peticiones web desde un navegador web, concretamente Firefox.

3.3.1. Referencias y características del cliente adoptado

En la siguiente tabla analizamos las características más importantes del servicio cliente:

Versión	Autor	Requisitos (Linux)
v. 78.7.0esr	Corporación Mozilla	- Procesador a 233 Mhz - 64 MB de RAM - 50 MB de espacio libre en disco - Kernel Linux 2.2.14 (mínimo)

3.3.2. Comparativa de clientes existentes en el mercado

Aquí podemos ver una tabla comparativa con aplicaciones que se consideran de la competencia al ofrecer un mismo servicio: permite ver la información que contiene una página web. El navegador interpreta el código en el que está escrita la página web y lo presenta en pantalla permitiendo al usuario interactuar con su contenido y navegar.

	Mozilla Firefox	Google Chrome	Opera	Tor Browser
Última versión	v. 95.0.1 21 dic 2021	v. 96.0.4664.110 7 enero 2022	v. 82.0.4227.43 28 dic 2021	v. 11.0.2 21 dic 2021
S.O. soportado	Linux, Windows, MacOS, Android, Ubuntu Touch, FreeBSD, Unix, iOS	Linux, Windows, MacOS, Android, iOS	Linux, Windows, MacOS, Android, iOS	Linux, Windows, MacOS, Android, BSD
Lenguaje	C, C++, JavaScript, Rust, XML User	C++	C++	Multilanguage
Licencia (*)	GNU General Public License	Freeware (licencia	Freeware (licencia	GNU General Public License

	Mozilla Public License	copyright)	copyright)	
URL de distribución	https://www.mozilla.org/es-ES/firefox/new/	https://www.google.com/chrome/	https://www.opera.com/	https://www.torproject.org/download/

En el caso de Tor Browser, se diferencia de las demás en que puede navegar en redes anónimas, que es el mayor uso que se le da. Todos los servicios son gratuitos.

4. Proceso de instalación y Uso del cliente

Aunque el software de Mozilla Firefox ya está instalado en el sistema, procederemos a su borrado del sistema operativo y reinstalación para demostrar el correcto funcionamiento, apoyándonos en el instalador de paquetes de CentOS, *yum*.

4.1. Obtención del software del cliente

Ejecutando el comando

yum info firefox

4.2. Instalación del cliente

4.2.1. Primera instalación del cliente

En la instalación por repositorio, desde CentOS 7 vamos a apoyarnos en el instalador de paquetes *yum*, que simplemente ejecutando el comando

```
yum install firefox
```

Nos instalaría automáticamente el navegador Mozilla Firefox.

4.2.2. Desinstalación del cliente

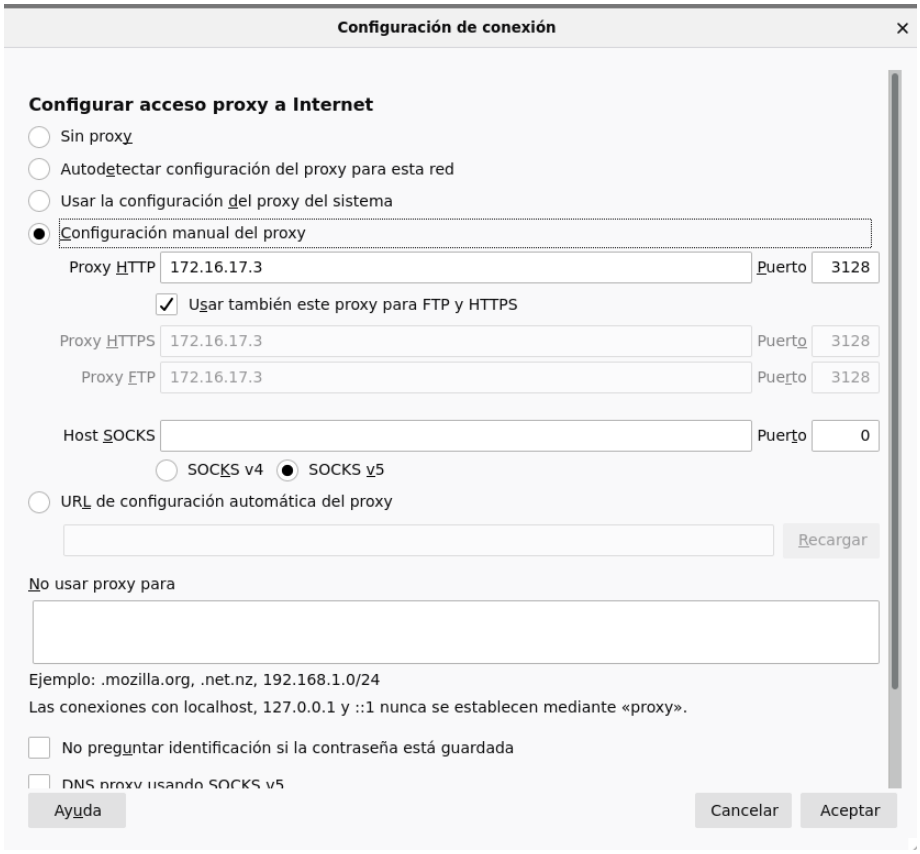
En la desinstalación por repositorio, desde CentOS 7 vamos a apoyarnos en el instalador de paquetes *yum*, que simplemente ejecutando el comando

```
yum remove firefox
```

Nos desinstalaría automáticamente el navegador.

4.3. Configuración del cliente

Ya que el trabajo consiste en la instalación y administración de un proxy, nos conviene saber de qué manera podemos configurar nuestro navegador para que se pueda conectarbcon el proxy en modo directo y le permita la comunicación indirecta con el resto (ocultando la IP del equipo cliente). En el submenú de la esquina superior del navegador, vamos al apartado Preferencias < General < Configuración de red, y en el botón de Configuración se pulsa y nos sale esta ventana:



Desde la que podemos configurar el proxy manualmente, introduciendo su IP y puerto en el que está escuchando. Hay opción para proxy HTTP, HTTPS, FTP Y SOCKS.

5. Proceso de instalación/administración del servidor

Este apartado lo vamos a dedicar a detallar paso a paso la parte de instalación y administración del servicio estudiado. Todos estos pasos van a estar registrados en un script para que a la hora de la defensa y de probar el servicio, todo el proceso se automatice. El script de instalación es *instalar_servicio.sh*.

5.1. Obtención del software del servidor

El proyecto Squid tiene como página oficial del servicio http://www.squid-cache.org donde:

- Registra todas las versiones estables en el apartado /Versions: http://www.squid-cache.org/Versions/ (en nuestro caso nos interesa la versión 3.5), y por cada versión da la opción de descarga del paquete correspondiente para la instalación por código fuente del servicio. Las extensiones de descarga de los paquetes son .tar.gz/.tar.bz2/.tar.xz.

- También ofrece una guía para la instalación por paquetes binarios a partir de instaladores de paquetes como *yum*, *apt*, etc. Dependiendo del SO que se esté utilizando, en https://wiki.squid-cache.org/SquidFaq/BinaryPackages

Nosotros nos centraremos en la versión 3.5 de Squid, y con la ayuda de la herramienta wget obtendremos el paquete de http://www.squid-cache.org/Versions/v3/3.5/squid-3.5.28.tar.gz .

5.2. Instalación del servidor

Se procede a explicar los pasos de instalación del servicio Squid tanto desde código fuente como desde el instalador de paquetes *yum*, desde el SO CentOS 7.

5.2.1. Primera instalación del servicio

5.2.1.1. Instalación desde código fuente (Compilación)

Antes de comenzar la instalación del servicio Squid, conviene comprobar que todas las dependencias están correctamente implementadas en el SO. En el caso de CentOS 7 de las máquinas del laboratorio, todas las dependencias ya las lleva integradas, pero igualmente podemos desinstalarlas y volverlas a instalar para comprobar el correcto funcionamiento. Las dependencias las instalaremos desde la herramienta *yum*, mediante los siguientes comandos:

```
yum install gcc
```

```
yum install make

yum install glibc

yum install libcap

yum install libecap

yum install libcom_err

yum install openssl-libs

yum install libdb

yum install libtool-ltdl

yum install libstdc++

yum install libxml2

yum install perl

yum install shadow-utils

yum install systemd
```

Con las dependencias instaladas, procedemos a la descarga del paquete de código fuente y su descompresión:

```
wget http://www.squid-cache.org/Versions/v3/3.5/squid-3.5.28.tar.gz

tar -xvzf squid-3.5.28.tar.gz

cd squid-3.5.28/
```

Lo siguiente va a ser la compilación del código fuente. Para ello, vamos a apoyarnos en el script de *configure* para generar los *makefiles* que adapten el software a los requisitos que queremos imponer al servicio. Todas las opciones de las que dispone este script se pueden ver desde el comando `./configure --help` con la respectiva descripción y uso de cada una. Para compilar usamos:

```
./configure --prefix=/usr/local/squid --sysconfdir=/etc/squid --with-logdir=/var/log/squid --with-pidfile=/var/run/squid.pid --enable-storeio=ufs,aufs --enable-auth --enable-removalpolicies=lru,heap --enable-useragent-log --enable-refererlog --bindir=/usr/sbin

make
```

```
su -

make install
```

Con la instalación del Squid ya ejecutada, se genera automáticamente un nuevo usuario efectivo del servicio, de nombre por defecto *squid* (se puede modificar con la opción `--with-default-user` del *configure* o desde el fichero de configuración de squid). A este usuario hay que hacerle propietario de los siguientes directorios, que son el directorio del usuario efectivo por defecto, el directorio donde se almacenan los ficheros log y el directorio de la memoria caché (se autogeneran con la instalación):

```
chown squid:squid /var/spool/squid/

chown squid:squid /var/log/squid/

chown squid:squid /var/cache/squid/
```

5.2.1.2. Instalación desde paquetes/repositorios

En la instalación por repositorio de paquetes binarios, para CentOS 7 vamos a apoyarnos en el instalador de paquetes *yum,* que simplemente ejecutando el comando

```
yum install squid
```

Nos instalaría automáticamente el servicio proxy Squid en el sistema con la versión 3.5.

5.2.2. Actualización del servicio

5.2.2.1. Actualización desde código fuente

Para la actualización por código fuente, será necesario desinstalar el servicio como se muestra en el punto siguiente 5.2.3.1, y luego descargar el paquete de código fuente de la página oficial de squid con la versión que nos interesa, siguiendo los pasos de 5.2.1.1 y teniendo en cuenta posibles pequeños cambios en los comandos al ser una nueva versión.

5.2.2.2. Actualización desde paquetes/repositorios

En la actualización por repositorio de paquetes binarios, desde CentOS 7 vamos a apoyarnos en el instalador de paquetes *yum,* que simplemente ejecutando el comando

```
yum update squid
```

Nos actualiza automáticamente el servicio proxy Squid.

5.2.3. Desinstalación del servicio

5.2.3.1. Desinstalación desde código fuente

Para la desinstalación del servicio por código fuente, hay que introducir una serie de comandos para el borrado total de los documentos del sistema. Todos estos comandos están ordenadamente recogidos en el script del trabajo *instalar_servicio.sh*.

```
cd /opt/squid/squid-3.5.28

make uninstall

cd

rm -rf /opt/squid/

rm -rf /usr/local/squid

rm -rf /etc/squid

rm -rf /var/spool/squid/

rm -rf /var/log/squid/

rm -rf /var/cache/squid/

rm /lib/systemd/system/squid.service

rm /usr/sbin/squid

userdel -r squid
```

5.2.3.2. Desinstalación desde paquetes/repositorios

En la desinstalación por repositorio de paquetes binarios, desde CentOS 7 vamos a apoyarnos en el instalador de paquetes *yum*, que simplemente ejecutando el comando

```
yum remove squid
```

Nos desinstalaría automáticamente el servicio proxy Squid.

5.3. Configuración del servidor

El fichero desde el que vamos a realizar todas las operaciones de configuración va a ser */etc/squid/squid.conf*, y todas estas operaciones van a ser llevadas a cabo por directivas. En [9] podemos ver la larga lista de directivas que existen para el servicio y se puede consultar como se usa cada una.

Vamos a analizar un poco la configuración inicial que se obtiene con la instalación del servicio, que son los requisitos mínimos para que el servicio squid funcione correctamente.

```
#
# Recommended minimum configuration:
#

# Example rule allowing access from your local networks.
# Adapt to list your (internal) IP networks from where browsing
# should be allowed
acl localnet src 10.0.0.0/8     # RFC1918 possible internal network
acl localnet src 172.16.0.0/12  # RFC1918 possible internal network
acl localnet src 192.168.0.0/16 # RFC1918 possible internal network
acl localnet src fc00::/7       # RFC 4193 local private network range
acl localnet src fe80::/10      # RFC 4291 link-local (directly plugged) machines

acl SSL_ports port 443
acl Safe_ports port 80          # http
acl Safe_ports port 21          # ftp
acl Safe_ports port 443         # https
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535  # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
acl Safe_ports port 591         # filemaker
acl Safe_ports port 777         # multiling http
acl CONNECT method CONNECT
```

La directiva *acl* genera agrupaciones de valores (IP o puertos en este caso, también puede ser de nombres de dominio por ejemplo), a los que se pueden hacer referencia, como mas adelante veremos, con un nombre. En el caso del grupo *localnet* vemos como se refiere a todas las direcciones IP de ámbito privado.

```
# Example rule allowing access from your local networks.
# Adapt localnet in the ACL section to list your (internal) IP networks
# from where browsing should be allowed
http_access allow localnet
http_access allow localhost

# And finally deny all other access to this proxy
http_access deny all
```

La directiva http_access tiene como utilidad general permitir o denegar el acceso en función de las listas de acceso definidas

Para permitir o denegar un mensaje recibido en un puerto HTTP, HTTPS o FTP utiliza la siguiente sintaxis:

```
http_access allow|deny [!]aclname ...
```

Define la parte de seguridad del servicio. Los http_access se interpretan por orden de lectura, de manera que todos los accesos que se permitan (allow) que estén después de la sentencia `http_access deny all`, no tendrán validez.

```
# Squid normally listens to port 3128
http_port 3128
```

Para los puertos que se abran de escucha. También permite definir el modo de actuación del proxy, como veremos en los escenarios del punto 7. A lo largo del documento iremos viendo más directivas y los usos que se les da más específicamente, como las de la habilitación del caché de Squid.

6. Puesta en funcionamiento del servicio

En este apartado vamos a enumerar los pasos que seguimos para hacer posible el arranque del servicio, tras haber finalizado con la instalación del punto anterior. Va a ser necesario la creación de los ficheros log, darle permisos al usuario efectivo de Squid para poder editar en esos ficheros, y configurar la caché antes del arranque. El sistema de arranque en el que nos vamos a apoyar va a ser Systemd.

6.1. Arranque del servicio con el sistema

Para arrancar el servicio Squid con Systemd, el repositorio nos proporciona un fichero *squid.service* con la siguiente estructura:

```
[Unit]
Description=Squid Web Proxy Server
After=network.target

[Service]
Type=simple
ExecStart=/usr/sbin/squid -sYC -N
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process

[Install]
WantedBy=multi-user.target
```

Que nos indica en la variable *ExecStart* la ubicación del ejecutable que inicia cuando se ejecuta por línea de comandos `systemctl start squid.service`. Este fichero hay que ubicarlo en */lib/systemd/system* para que el servicio de arranque Systemd pueda interpretarlo, y el ejecutable de Squid deberá situarse en */usr/sbin*.

```
cp /opt/squid/squid-3.5.28/tools/systemd/squid.service
/lib/systemd/system
```

```
cp /usr/local/squid/sbin/squid /usr/sbin
```

Una vez realizado, ejecutando el comando anterior se iniciaría el servicio, y con el comando:

```
systemctl status squid.service
```

Comprobaríamos que el servicio está en funcionamiento.

Otra manera de comprobar que el servicio corre correctamente es mediante los logs, tema que abarcamos en el siguiente punto.

6.2. Administración y monitorización del funcionamiento

En este punto es esencial la parte de configurar los ficheros log y los permisos para el usuario *squid*, ya que su acceso es requisito esencial para que el servidor Squid arranque correctamente.

6.2.1. Configuración y Uso de los ficheros de registro

Tras la instalación del servicio e implementar su arranque desde Systemd, es requisito esencial la creación de los ficheros log y concesión de permisos de escritura al usuario efectivo *squid*, ya que, si Squid recibe un error de escritura para un archivo de registro, sale y se reinicia. La razón principal es llamar la atención del usuario, ya que Squid quiere asegurarse de que no se pierda ninguna información de registro importante, especialmente si el sistema está siendo abusado o atacado.

Squid maneja tres ficheros log,

- *cache.log*, que en mi caso he preferido renombrar como *squid.log*, debido a que puede tender a error por el nombre que se le da por defecto. No informa sobre el estado de la cache, sino sobre el servicio en sí. En este fichero se imprimen los mensajes de información del servicio (arranque, puertos que abre, creación de procesos, finalización del servicio…) y debugging. Ante cualquier fallo del servicio, conviene acudir a este fichero. Su ruta y nombre del fichero se puede configurar desde *squid.conf* con la directiva cache_log.

- *access.log*, en el que se registra una línea por cada cliente que hace un Request al proxy. Su ruta y nombre del fichero se puede configurar desde *squid.conf* con la directiva access_log.

- *store.log*, donde el usuario *squid* introduce una línea por cada objeto que entra o sale de la memoria caché. Su ruta y nombre del fichero se puede configurar desde *squid.conf* con la directiva cache_store_log.

Por tanto, los pasos a seguir para que el usuario squid pueda tener acceso serían, en primer lugar, introducir en el fichero de configuración */etc/squid/squid.conf* las ubicaciones de los tres ficheros log, con estas líneas:

/etc/squid/squid.conf

```
#Logs

cache_log /var/log/squid/squid.log

access_log /var/log/squid/access.log

cache_store_log /var/log/squid/store.log
```

Tras esto, tenemos que crear los ficheros:

```
touch /var/log/squid/squid.log

touch /var/log/squid/access.log

touch /var/log/squid/store.log
```

Y hacer propietario al usuario *squid* de los ficheros:

```
chown squid:squid /var/log/squid/access.log

chown squid:squid /var/log/squid/store.log

chown squid:squid /var/log/squid/squid.log
```

6.2.2. Generación de los directorios caché

Para habilitar la memoria caché en nuestro sistema, vamos a tener que tocar el fichero de configuración de squid y ejecutar el servicio con una opción que permite la creación de los directorios de almacenamiento caché.

En *squid.conf* vamos a utilizar la directiva `cache_dir`, con la sintaxis siguiente [5]:

```
cache_dir STORAGE_TYPE DIRECTORY SIZE_IN_Mbytes L1 L2 [OPTIONS]
```

- `STORAGE_TYPE:` modo de almacenamiento para almacenar archivos y directorios en las unidades de disco. Los modelos más comunes son ufs, aufs y diskd. El caso de ufs, que es el que usaremos, es un esquema de almacenamiento muy simple y todas las transacciones de E/S se realizan utilizando el proceso principal de Squid. Esto es bueno para servidores con menos carga y discos de alta velocidad, pero puede suponer un problema para cachés con alta demanda, ya que hay peligro de retardos por bloqueo del proceso principal de Squid en la búsqueda de archivos en caché. En el caso de aufs, se usan hilos (librería pthreads) para estas búsquedas de archivos, con lo que el proceso principal puede seguir atendiendo otras peticiones y no se bloquea. Para diskd sería un proceso similar que aufs, pero en vez de usar hilos, llama a procesos externos.

- `DIRECTORY:` directorio donde se generan los directorios de almacenamiento caché.

- `SIZE_IN_Mbytes:` cantidad de disco dedicado a la caché.

- `L1, L2:` los dos últimos valores especifican el número de directorios de primer y segundo nivel que se quiere crear.

Las líneas a tener en cuenta serían las siguientes:

/etc/squid/squid.conf

```
cache_dir ufs /var/cache/squid 100 16 256

# Leave coredumps in the first cache dir

coredump_dir /var/cache/squid

#Cache user

cache_effective_user squid
```

Una vez añadidas las opciones de la caché, solo queda introducir la orden para la creación de los directorios y subdirectorios (*swap directories*). Esto se consigue añadiendo la opción -z al ejecutable de squid:

```
squid -z
```

ó

```
/usr/local/squid/sbin/squid -z
```

6.2.3. Arranque del servicio en modo detallado (verbose)

No procede. Para mirar la información y debugging del servicio, se accede a los logs, en nuestro caso a */var/log/squid/squid.log*, que nos detalla el arranque del servicio y los fallos que puede haber mientras corre.

6.2.4. Seguridad del servicio

El servicio tiene la capacidad de controlar los accesos al proxy con el dúo de directivas *acl* y *http_access (*del que hemos hablado antes en el apartado 5.3*).* Pueden definir estas reglas para direcciones IP, nombres de dominio o puertos. También da cierto grado de anonimato al ocultar la dirección IP de sus clientes para realizarlas solicitudes que se le envían.

Donde no tiene seguridad es en el cifrado de paquetes, que es un factor que depende del protocolo que se esté usando.

6.3. Tests y Pruebas del correcto arranque del servicio

Arranque del servicio con Systemd:

```
systemctl status squid
```

```
[root@192 squid-3.5.28]# systemctl status squid.service
● squid.service - Squid Web Proxy Server
   Loaded: loaded (/usr/lib/systemd/system/squid.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2022-01-05 09:33:29 CET; 1h 1min ago
 Main PID: 55353 (squid)
   CGroup: /system.slice/squid.service
           ├─55353 /usr/sbin/squid -sYC -N
           └─55354 (unlinkd)

Jan 05 09:33:30 192.168.93.130 squid[55353]:         0 Objects expired.
Jan 05 09:33:30 192.168.93.130 squid[55353]:         0 Objects cancelled.
Jan 05 09:33:30 192.168.93.130 squid[55353]:         0 Duplicate URLs purged.
Jan 05 09:33:30 192.168.93.130 squid[55353]:         0 Swapfile clashes avoided.
Jan 05 09:33:30 192.168.93.130 squid[55353]:   Took 0.01 seconds (  0.00 objects/sec).
Jan 05 09:33:30 192.168.93.130 squid[55353]: Beginning Validation Procedure
Jan 05 09:33:30 192.168.93.130 squid[55353]:   Completed Validation Procedure
Jan 05 09:33:30 192.168.93.130 squid[55353]:   Validated 0 Entries
Jan 05 09:33:30 192.168.93.130 squid[55353]:   store_swap_size = 0.00 KB
Jan 05 09:33:31 192.168.93.130 squid[55353]: storeLateRelease: released 0 objects
```

Desde los logs en */var/log/squid/squid.log* podemos comprobar si todo ha ido bien en el arranque:

```
2022/01/07 16:12:12| Configuring Parent server2.example.com/80/0
2022/01/07 16:12:12| Squid plugin modules loaded: 0
2022/01/07 16:12:12| Adaptation support is off.
2022/01/07 16:12:12| Accepting NAT intercepted HTTP Socket connections at local=[::]:3128 remo
te=[::] FD 19 flags=41
2022/01/07 16:12:12| Done reading /var/cache/squid swaplog (16 entries)
2022/01/07 16:12:12| Finished rebuilding storage from disk.
2022/01/07 16:12:12|        16 Entries scanned
2022/01/07 16:12:12|         0 Invalid entries.
2022/01/07 16:12:12|         0 With invalid flags.
2022/01/07 16:12:12|        16 Objects loaded.
2022/01/07 16:12:12|         0 Objects expired.
2022/01/07 16:12:12|         0 Objects cancelled.
2022/01/07 16:12:12|         0 Duplicate URLs purged.
2022/01/07 16:12:12|         0 Swapfile clashes avoided.
2022/01/07 16:12:12|   Took 0.06 seconds (248.45 objects/sec).
2022/01/07 16:12:12| Beginning Validation Procedure
2022/01/07 16:12:12|   Completed Validation Procedure
2022/01/07 16:12:12|   Validated 16 Entries
2022/01/07 16:12:12|   store_swap_size = 476.00 KB
2022/01/07 16:12:12| storeLateRelease: released 0 objects
```

También podríamos comprobarlo mediante el comando `ps ax` para comprobar que se ha creado el proceso padre Squid, como *root*, y el proceso hijo llevado a cabo por el usuario *squid*.

```
45269 ?        Ss     0:00 squid
45271 ?        S      0:00 (squid-1) --kid squid-1
```

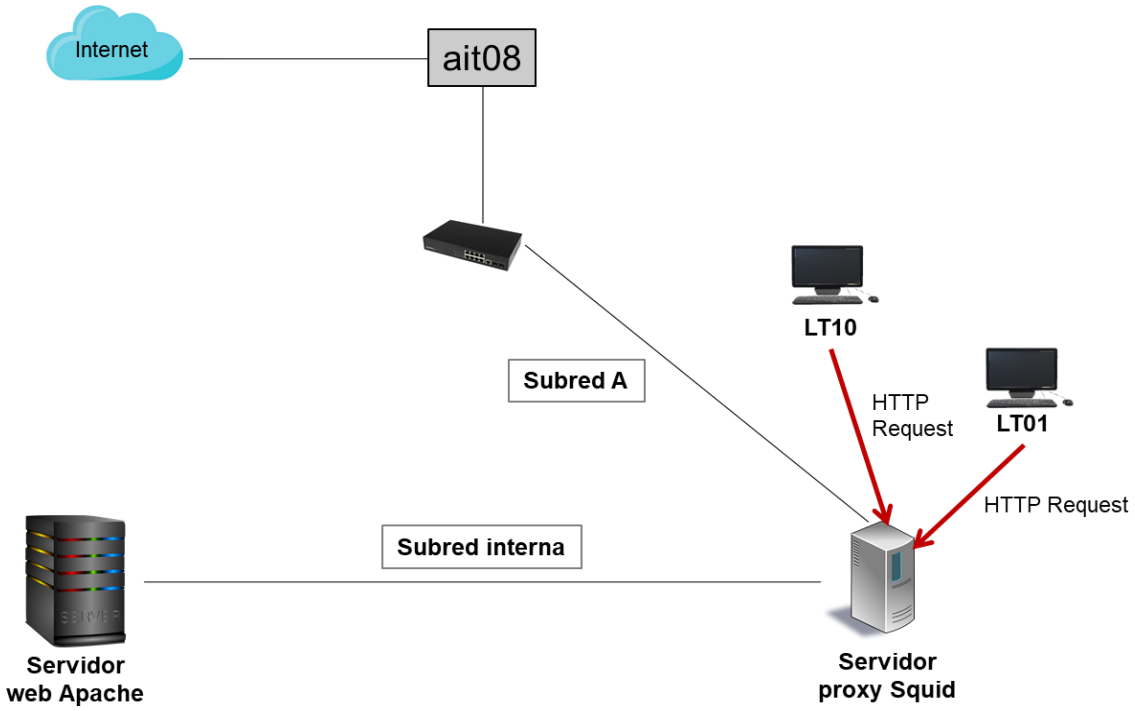
7. Diseño de los Escenarios de Defensa

7.1. Escenario de Defensa 1: <Proxy en modo inverso: acelerador>

En el primer escenario vamos a demostrar el uso que se hace de la caché cuando el servidor Squid trabaja en modo inverso. Buscamos ver cómo trabaja el proxy en el lado del servidor. Para ello, vamos a aislar un equipo que implementará un servidor web Apache de la red local del laboratorio, para que trabaje por detrás del servidor Squid y sólo se pueda acceder a él mediante este proxy. Vamos a comprobar como el servidor proxy almacena el contenido web de las respuestas que reenvía provenientes del servidor, y como ante nuevas peticiones de la misma página web, ya no las reenvía al servidor final, sino que es capaz de atenderlas él mismo consultando su caché local, actuando como servidor final de manera transparente, sin que ningún usuario de los equipos clientes tenga que ser consciente de ello. Es lo que se conoce como un “Acelerador”, ahorrando ancho de banda y liberando carga de peticiones al servidor web.

7.1.1. Escenario 1: Esquema de la red

El esquema de red tiene similitudes con el escenario que se monta en la P02 de la asignatura para el encaminamiento IP.



En el escenario se muestra como el servidor Apache se encuentra aislado en una subred interna que solo puede ser accedida a través del equipo proxy. El equipo proxy tiene también un servidor DHCP que se encarga de dar una dirección de la subred interna al equipo servidor web al arrancarse. Mediante las peticiones web de los equipos de la subred A vamos a ser capaces de comprobar el funcionamiento del proxy Squid y su caché HTTP. Configuramos también el servidor Apache para que ofrezca dos sitios web distintos (páginas de contenido sencillo) y como se proporciona uno y otro.

Para la configuración de Apache, modificamos el fichero de configuración *httpd.conf* añadiendo:

/etc/httpd/conf/httpd.conf

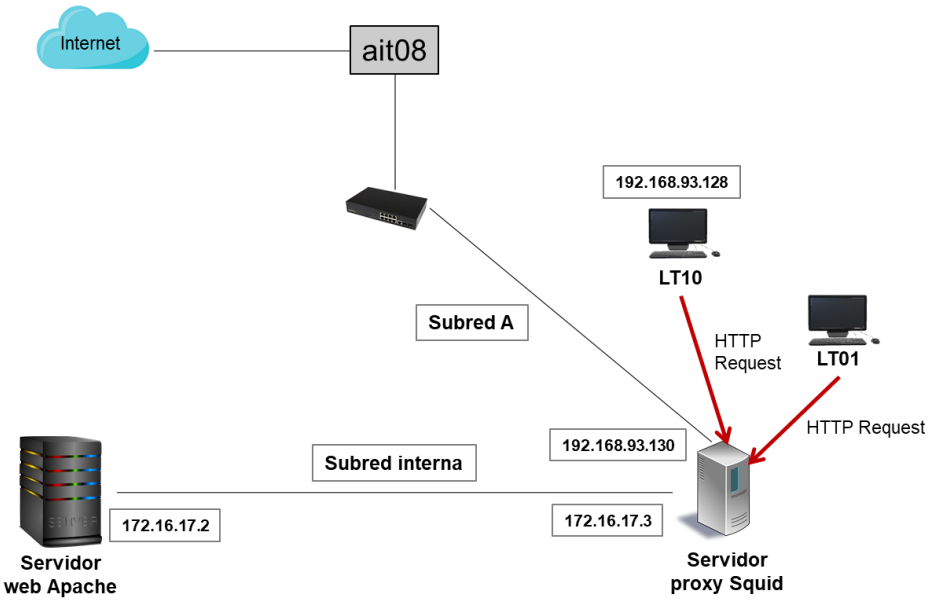
```
Alias /web /var/www/html/squid1

<Directory "var/www/html/squid1">
 Options Indexes FollowSymLinks
 Allowoverride None
 Order allow,deny
 Allow from all
</Directory>
```

Y reiniciamos Apache para que se lea el fichero de configuración:

```
service httpd restart
```

7.1.2. Escenario 1: Configuración del servidor

Para este ejemplo concreto vamos a asignar las siguientes direcciones de red a LT10, al equipo que implementa el proxy Squid y al servidor Apache:



Para que el servidor Squid actúe como acelerador, es necesario modificar su fichero *squid.conf* con nueva configuración, aparte de la que ya contiene. Vamos a hacer uso de las directivas `http_port`, `cache_peer` y `cache_peer_access`, cada una con el siguiente uso y sintaxis, sacada de la documentación de squid [9]:

- Sintaxis: `http_port port [mode] [options]`
    - `port:` Puerto que escucha
    - `mode:` modo de funcionamiento (en este caso va a actuar como un acelerador).
    - `options:` usaremos la opción `defaultsite=domainname,` y determina el hostname del sitio que el proxy debe considerar para su actuar como acelerador.

  Sentencia usada: `http_port 80 accel defaultsite=server1.example.com`

- Sintaxis: `cache_peer hostname type http_port icp_port [options]`
    - `hostname:` del equipo que cachea los recursos.
    - `type`: jerarquía del equipo que cachea. Como obtiene los recursos del servidor, su jerarquía es `parent`. Si cacheara de otro proxy vecino que controla el mismo sitio web, su parentesco sería `sibling`.
    - `http_port`: puerto donde el proxy acepta las HTTP Requests.
    - `icp_port`: Se utiliza para consultar las cachés de los vecinos sobre los objetos. Se establece en 0 si el peer no soporta ICP o HTCP.
    - `[options]:`
        - `no-query`: deshabilita peticiones ICP de otros vecinos proxy.
        - `originserver`: hace que se contacte con este padre como si fuera el servidor origen.

  Sentencia usada:

```
cache_peer server1.example.com parent 80 0 no-query originserver
```

- `cache_peer_access` para establecer reglas de acceso al servidor web interno (peer).

Por tanto, lo que se añade al archivo de configuración es:

/etc/squid/squid.conf

```
http_port 80 accel defaultsite=server1.example.com

cache_peer server1.example.com parent 80 0 no-query originserver
name=AceleradorLab

cache_peer_access AceleradorLab allow localnet
cache_peer_access AceleradorLab deny all
```

En las reglas de acceso al servidor interno, se entiende como localnet, como se explica en 5.3, por las direcciones IP que abarcan la Subred del laboratorio.

Para identificar al servidor con ese nombre de dominio, vamos a añadir en el fichero */etc/hosts* una línea que relacione la IP del servidor web Apache con ese nombre de dominio (a falta de DNS en la subred interna).

/etc/hosts

```
172.16.17.2 server1.example.com server1
```

7.1.3. Escenario 1: Tests y Pruebas del Escenario

Tras realizarla parte de configuración del fichero de squid, reiniciamos el servicio desde Systemd,

```
systemctl restart squid
```

El servicio suele tardar 30 segundos en finalizar antes de volver a iniciarse, para dar tiempo a atender a las peticiones que queden.

Ahora, desde un navegador web de la Subred A del laboratorio (esquematizado arriba), hacemos una petición al servidor proxy, que actúa como servidor web, aunque realmente lo que hace es cubrir al servidor final.

Desde LT10 se ejecuta el navegador: `firefox http://192.168.93.130/web &`

Resultado obtenido:

Si miramos los logs de store.log: `cat /var/log/squid/store.log`

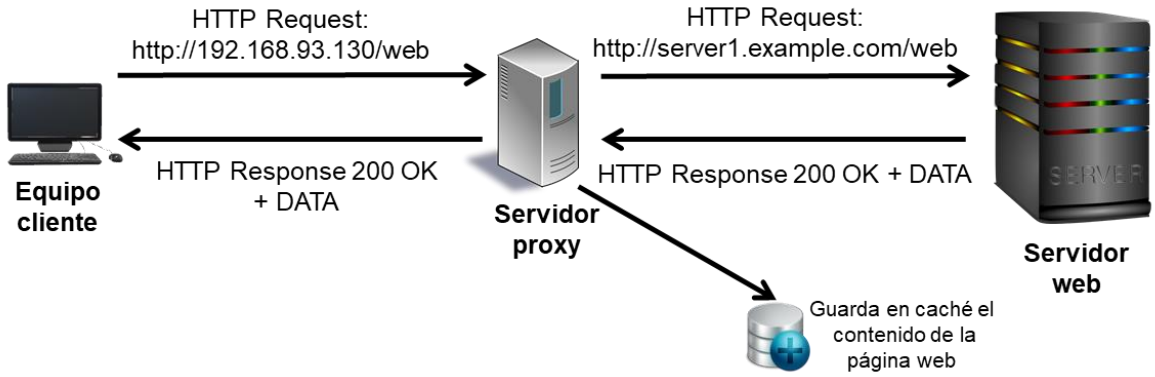
```
1641405156.471 SWAPOUT 00 00000000 A4ECF3642718B8FE23ED787976F66DBE  301 1641405156        -1
       -1 text/html 234/234 GET http://192.168.93.130/web
1641405156.521 SWAPOUT 00 00000001 56C200EA53A6B49A4A9035891DD9EA35  200 1641405156 1610631478
        -1 text/html 634/634 GET http://192.168.93.130/web/
1641405156.770 SWAPOUT 00 00000002 4BA6C65899CA7825526E7D520E634F20  200 1641405156 1610631498
        -1 image/png 10453/10453 GET http://192.168.93.130/web/imagenes/miSitio.png
1641405156.778 SWAPOUT 00 00000003 0394C2561A7C33DF97A85408FCF2C76F  200 1641405156 1610631496
        -1 image/png 3666/3666 GET http://192.168.93.130/web/imagenes/logo.png
1641405156.780 SWAPOUT 00 00000004 1A70FDCB07BE784CE297F91F6A698F8C  200 1641405156 1610631494
        -1 image/gif 2151/2151 GET http://192.168.93.130/web/imagenes/logoUS.gif
1641405196.472 RELEASE -1 FFFFFFFF 8949B4D7A1E961556F3A1C0DFD3B9C31   ?         ?         ?
```

Vemos como almacena en caché los archivos de la página html que hemos accedido. De esta manera, como demostramos en el siguiente punto, en las siguientes peticiones va a actuar como si fuese el servidor final, respondiendo él mismo.

7.1.4. Escenario 1: Análisis del intercambio real de mensajes de red

La parte importante del escenario se muestra aquí, donde se comprende la acción que realiza el proxy como acelerador. La primera vez que un equipo cliente solicita acceso al sitio web, se produce un paso de mensajes en el que la petición llega al proxy, y este la reenvía al servidor interno para que responda. La respuesta pasa por el proxy de nuevo para ser

reenviada al cliente, y a su vez el mismo proxy guarda en la caché y registra (visto anteriormente) en los logs el contenido (imágenes, html, etc.) de la respuesta HTTP.



La demostración de los mensajes en Wireshark (desde la máquina del proxy) para la petición y respuesta entre proxy (172.16.17.3) y servidor Apache (172.16.17.2) es la siguiente:

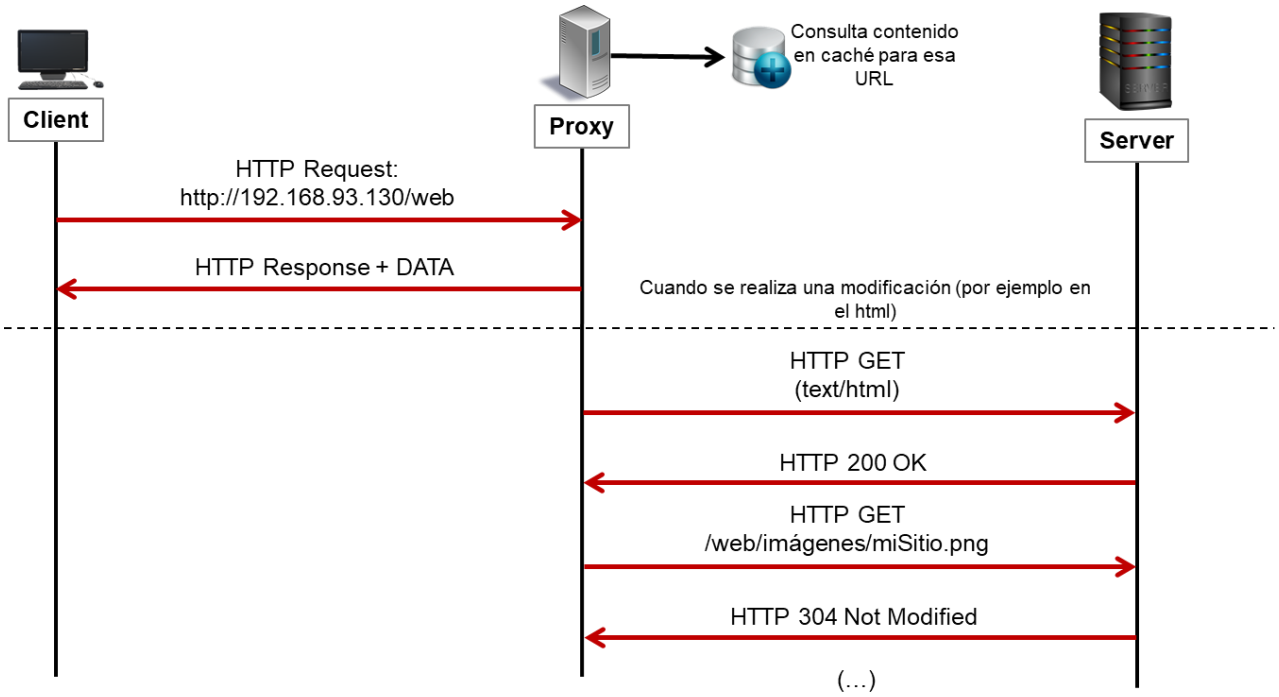
1	0.000000000	172.16.17.1	239.255.255.250	SSDP	214	M-SEARCH * HTTP/1.1
2	1.001540423	172.16.17.1	239.255.255.250	SSDP	214	M-SEARCH * HTTP/1.1
3	2.004175205	172.16.17.1	239.255.255.250	SSDP	214	M-SEARCH * HTTP/1.1
4	3.004501056	172.16.17.1	239.255.255.250	SSDP	214	M-SEARCH * HTTP/1.1
8	6.494432077	172.16.17.3	172.16.17.2	HTTP	528	GET /web/ HTTP/1.1
10	6.495276703	172.16.17.2	172.16.17.3	HTTP	958	HTTP/1.1 200 OK (text/html)
12	6.545333321	172.16.17.3	172.16.17.2	HTTP	495	GET /web/imagenes/miSitio.png HTTP/1.1
15	6.546040850	172.16.17.2	172.16.17.3	HTTP	689	HTTP/1.1 200 OK (JPEG JFIF image)
17	6.547139246	172.16.17.3	172.16.17.2	HTTP	492	GET /web/imagenes/logo.png HTTP/1.1
21	6.547726824	172.16.17.3	172.16.17.2	HTTP	494	GET /web/imagenes/logoUS.gif HTTP/1.1
32	6.549615116	172.16.17.2	172.16.17.3	HTTP	1074	HTTP/1.1 200 OK (GIF89a)
51	6.555963746	172.16.17.2	172.16.17.3	HTTP	23053	HTTP/1.1 200 OK (PNG)
53	6.568158873	172.16.17.3	172.16.17.2	HTTP	445	GET /favicon.ico HTTP/1.1

Podemos comprobar el envío desde el servidor al proxy de todos los recursos de la página vía HTTP.

Y el reenvío del proxy (193.168.93.130) a la máquina del cliente (193.168.93.128):

10817	1207.965328537	192.168.93.128	192.168.93.130	HTTP	470	GET http://192.168.93.130/web HTTP/1.1
10820	1207.978617617	192.168.93.130	192.168.93.128	HTTP	300	HTTP/1.1 301 Moved Permanently (text/html)
10826	1208.010796476	192.168.93.128	192.168.93.130	HTTP	406	GET http://detectportal.firefox.com/success.txt HTTP/1.1
10828	1208.022055261	192.168.93.128	192.168.93.130	HTTP	445	GET http://192.168.93.130/web/ HTTP/1.1
10830	1208.027687509	192.168.93.130	192.168.93.128	HTTP	700	HTTP/1.1 200 OK (text/html)
10837	1208.201115345	192.168.93.128	239.255.255.250	SSDP	214	M-SEARCH * HTTP/1.1
10838	1208.263904821	192.168.93.128	192.168.93.130	HTTP	412	GET http://192.168.93.130/web/imagenes/miSitio.png HTTP/1.1
10842	1208.274299916	192.168.93.128	192.168.93.130	HTTP	409	GET http://192.168.93.130/web/imagenes/logo.png HTTP/1.1
10847	1208.278655815	192.168.93.128	192.168.93.130	HTTP	411	GET http://192.168.93.130/web/imagenes/logoUS.gif HTTP/1.1
10854	1208.283163819	192.168.93.130	192.168.93.128	HTTP	2327	HTTP/1.1 200 OK (JPEG JFIF image)
10858	1208.284809522	192.168.93.130	192.168.93.128	HTTP	3732	HTTP/1.1 200 OK (PNG)
10862	1208.287522408	192.168.93.130	192.168.93.128	HTTP	2217	HTTP/1.1 200 OK (GIF89a)

Para las siguientes peticiones similares del mismo equipo cliente o de otros, ya no es necesario solicitar la respuesta al servidor, ya que el proxy la puede gestionar por su cuenta El proxy irá comprobando que la página del servidor no haya sido actualizada si se vuelve a solicitar la petición en un tiempo, o si se hace alguna modificación en algún recurso.



La demostración desde Wireshark para la comunicación entre el proxy (172.16.17.3) y el servidor Apache (172.16.17.2) tras hacer una modificación en el archivo html del sitio web:

557	1498.113612	172.16.17.3	172.16.17.2	HTTP	445 GET /favicon.ico HTTP/1.1
559	1498.114921	172.16.17.2	172.16.17.3	HTTP	511 HTTP/1.1 404 Not Found (text/html)
582	1560.036160	172.16.17.1	239.255.255.250	SSDP	214 M-SEARCH * HTTP/1.1
583	1561.037689	172.16.17.1	239.255.255.250	SSDP	214 M-SEARCH * HTTP/1.1
584	1562.039618	172.16.17.1	239.255.255.250	SSDP	214 M-SEARCH * HTTP/1.1
588	1562.548499	172.16.17.3	172.16.17.2	HTTP	609 GET /web/ HTTP/1.1
590	1562.549421	172.16.17.2	172.16.17.3	HTTP	959 HTTP/1.1 200 OK (text/html)
592	1562.649863	172.16.17.3	172.16.17.2	HTTP	577 GET /web/imagenes/miSitio.png HTTP/1.1
593	1562.650469	172.16.17.2	172.16.17.3	HTTP	266 HTTP/1.1 304 Not Modified
595	1562.657923	172.16.17.3	172.16.17.2	HTTP	575 GET /web/imagenes/logo.png HTTP/1.1
596	1562.658655	172.16.17.2	172.16.17.3	HTTP	267 HTTP/1.1 304 Not Modified
597	1562.663422	172.16.17.3	172.16.17.2	HTTP	575 GET /web/imagenes/logoUS.gif HTTP/1.1
598	1562.664092	172.16.17.2	172.16.17.3	HTTP	265 HTTP/1.1 304 Not Modified

Podemos comprobar como envía código 304 (Not Modified) para los archivos que no han sido modificados y 200 (OK) para los que sí.

Los mensajes que se envían entre proxy (172.16.17.3) y servidor (172.16.17.2) 3cuando hay una nueva petición de un cliente y el sitio web no se ha modificado es:

557	1498.113612(	172.16.17.3	172.16.17.2	HTTP	445 GET /favicon.ico HTTP/1.1
559	1498.114921!	172.16.17.2	172.16.17.3	HTTP	511 HTTP/1.1 404 Not Found (text/html)

Y entre el proxy (193.168.93.130) que envía los recursos desde su caché y la máquina del cliente (193.168.93.128):

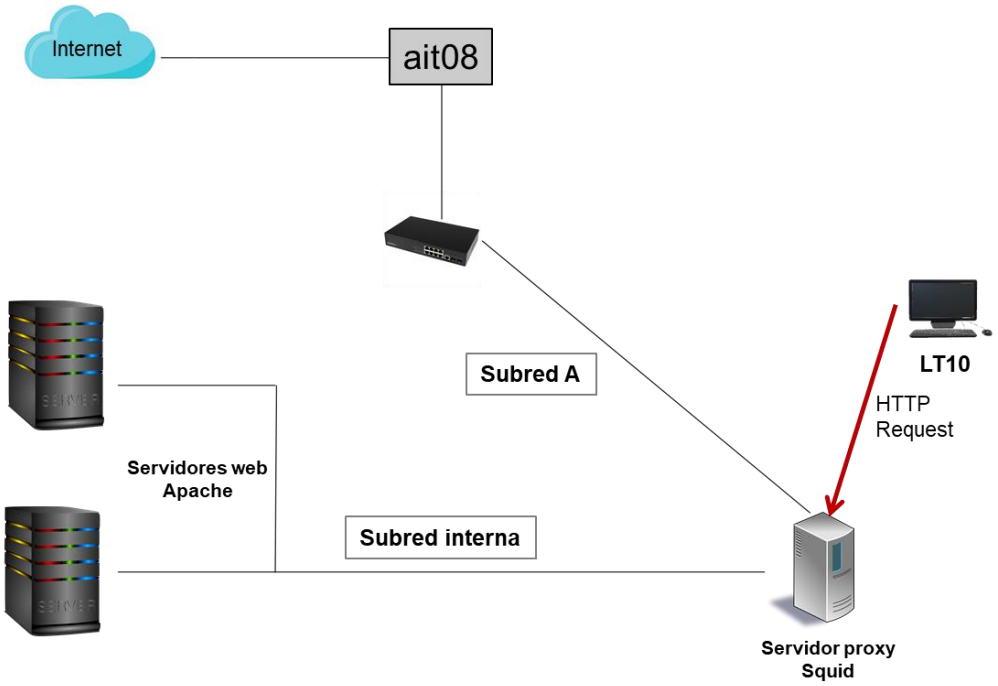
416	137.797562643	192.168.93.128	192.168.93.130	HTTP	493 GET /web/ HTTP/1.1
418	137.797798939	192.168.93.130	192.168.93.128	HTTP	1010 HTTP/1.1 200 OK (text/html)
430	137.981454704	192.168.93.128	192.168.93.130	HTTP	449 GET /web/imagenes/miSitio.png HTTP/1.1
433	137.981791030	192.168.93.130	192.168.93.128	HTTP	2633 HTTP/1.1 200 OK (JPEG JFIF image)
445	138.009128460	192.168.93.128	192.168.93.130	HTTP	446 GET /web/imagenes/logo.png HTTP/1.1
476	138.010503772	192.168.93.128	192.168.93.130	HTTP	448 GET /web/imagenes/logoUS.gif HTTP/1.1
481	138.010752002	192.168.93.130	192.168.93.128	HTTP	2574 HTTP/1.1 200 OK (GIF89a)
511	138.026369918	192.168.93.130	192.168.93.128	HTTP	4789 HTTP/1.1 200 OK (PNG)
513	138.071490190	192.168.93.128	192.168.93.130	HTTP	436 GET /favicon.ico HTTP/1.1
515	138.073126765	192.168.93.130	192.168.93.128	HTTP	275 HTTP/1.1 404 Not Found (text/html)

7.2. Escenario de Defensa 2: < Proxy en modo inverso: balanceador de carga>

En este segundo escenario vamos a mostrar otra funcionalidad que nos puede aportar el proxy Squid en modo de funcionamiento inverso, el balanceador de carga. Esta funcionalidad se genera para escenarios en los que el proxy actúa como acelerador de múltiples servidores web de una misma red. En nuestro caso vamos a demostrar cómo se configuraría el balanceo de carga entre dos servidores a los que los clientes solo tienen acceso mediante el proxy acelerador.

7.2.1. Escenario 2: Esquema de la red

El escenario en el que queremos trabajar se apoya en la topología del anterior, cambiando el número de servidores web dentro de la red interna:



Este sería el escenario ideal que buscamos, pero en el laboratorio al ser la subred interna cableada, no podemos tener más de dos equipos físicos dentro de la subred interna (problema del que me he podido dar cuenta a medida que redactaba este escenario). Se me pasaron por la cabeza varias alternativas para solventar el problema:

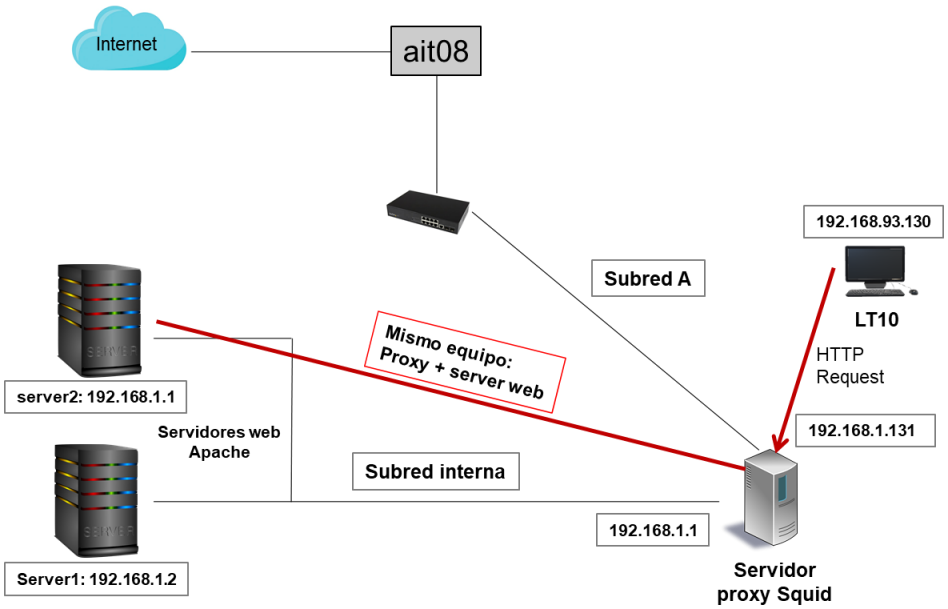
- Cablear la eth0 de un servidor interno a la eth1 del otro servidor y activar el reenvío (*ip_forward*). Ambas interfaces de un equipo servidor pertenecerían a la misma red

interna para unir el otro servidor. Por falta de tiempo para probarlo, y con ciertas dudas, no me fie.

- Activar el servidor apache únicamente en el servidor web interno (como en el escenario 1), y hacer `set down eth0` para aislar el equipo, pero sin desactivar la IP. Ofreceríamos dos páginas distintas en un mismo equipo, ayudándonos de la directiva VirtualHost basada en IP en la configuración de Apache. Mi prioridad aún así era tener dos equipos físicos, así que opté por la tercera opción.

- Que el servidor interno actúe de server1 y el servidor proxy también actúe de server2, ambos con Apache activado. En squid cambiamos la configuración para que escuche en el puerto 8080 (como acelerador) y de esta manera el servidor Apache puede escuchar desde el puerto 80. Las solicitudes de los clientes externos tendrán que ir al puerto 8080, y el proxy dependiendo del nombre de dominio que le llegue, reenviará la petición al servidor local o al otro servidor interno. Demostramos a continuación esta alternativa.

La solución que damos con las direcciones de las interfaces de cada equipo que usamos son:



7.2.2. Escenario 2: Configuración del servidor

Para el servidor Squid nos apoyamos en directivas similares al primer escenario. Esta sería la modificación que hacemos sobre el fichero de configuración inicial (sin las modificaciones del escenario 1, sino el del apartado 5.3):

/etc/squid/squid.conf

```
http_port 8080 accel
cache_peer server1.example.com parent 80 0 no-query originserver
name=server_1
acl sites_server_1 dstdomain server1.example.com
cache_peer_access server_1 allow sites_server_1

cache_peer server2.example.com parent 80 0 no-query originserver
name=server_2
acl sites_server_2 dstdomain server2.example.com
cache_peer_access server_2 allow sites_server_2
```

Importante que Squid escucha en el puerto 8080 actuando como reverse proxy (acelerador) para que el proceso apache pueda escuchar desde el puerto 80.

Mediante la directiva `acl` imponemos los nombres de dominio con los que se accede a cada servidor. El servidor proxy se va a identificar con ambos nombres de dominio de los servidores internos para los clientes externos, de manera que le lleguen las peticiones:

http://server1.example.com y http://server2.example.com

Por tanto, en el cliente que hace la solicitud tendremos registrado en su */etc/hosts* (algo que tendríamos que configurar en el servidor DNS del laboratorio, si quisiésemos).

/etc/hosts (cliente LT10)

```
192.168.93.131 server1.example.com server2.example.com
```

Y el servidor proxy en su */etc/hosts* tiene identificado cada nombre de dominio con cada IP de servidor que pertenece.

/etc/hosts (servidor Squid)

```
192.168.1.2 server1.example.com
192.168.1.1 server2.example.com
```

Reiniciamos el servicio Squid con `systemctl restart squid` para que se apliquen los cambios del fichero de configuración, y accedemos desde LT10 a:

http://server1.example.com o http://server2.example.com

7.2.3. Escenario 2: Tests y Pruebas del Escenario

Si observamos los logs en *squid.log:*

```
2022/01/07 14:43:56| Accepting reverse-proxy HTTP Socket connections at local=[::]:8080 remote=[::] FD 19 flags=9
2022/01/07 14:43:56| Accepting HTTP Socket connections at local=[::]:3128 remote=[::] FD 20 flags=9
```

Vemos como actúa en cada puerto que escucha.

Cuando hacemos la petición desde el equipo cliente a http://server1.example.com nos sale:

Si la petición es a http://server2.example.com nos sale:

Sitio web de servidor 2

Podemos también mirar los accesos en el fichero *access.log* para el primer acceso a server1:

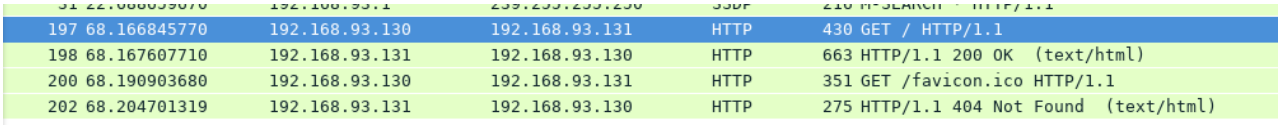
```
27 192.168.93.130 TCP_REFRESH_UNMODIFIED/304 249 GET http://server1.example.com:8080/ - FIRSTUP_PARENT/192.168.1.2
 0 192.168.93.130 TCP_MEM_HIT/200 597 GET http://server1.example.com:8080/ - HIER_NONE/- text/html
13 192.168.93.130 TCP_MISS/404 488 GET http://server1.example.com:8080/favicon.ico - FIRSTUP_PARENT/192.168.1.2 tex
```

Y para el acceso a server2:

```
1641563668.101     10 192.168.93.130 TCP_REFRESH_FAIL_OLD/200 695 GET http://server2.example.c
om:8080/ - ANY_OLD_PARENT/192.168.1.1 text/html
1641563668.137      1 192.168.93.130 TCP_MISS/503 4184 GET http://server2.example.com:8080/fav
icon.ico - ANY_OLD_PARENT/192.168.1.1 text/html
```

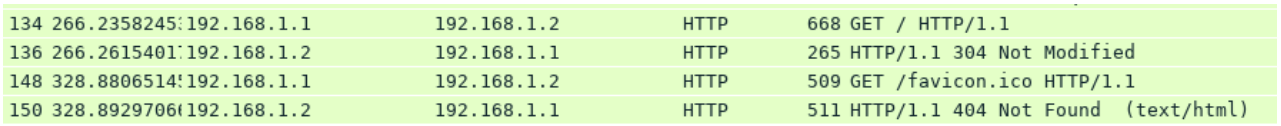
7.2.4. Escenario 2: Análisis del intercambio real de mensajes de red

Para la solicitud http://server1.example.com, desde wireshark se puede demostrar como se hace la petición y recibe la respuesta (entre cliente con IP 192.168.93.130 y proxy con IP 192.168.93.131):



197	68.166845770	192.168.93.130	192.168.93.131	HTTP	430 GET / HTTP/1.1
198	68.167607710	192.168.93.131	192.168.93.130	HTTP	663 HTTP/1.1 200 OK (text/html)
200	68.190903680	192.168.93.130	192.168.93.131	HTTP	351 GET /favicon.ico HTTP/1.1
202	68.204701319	192.168.93.131	192.168.93.130	HTTP	275 HTTP/1.1 404 Not Found (text/html)

Y si vemos el wireshark entre proxy (192.168.1.1) y server1 (192.168.1.2) obtenemos:

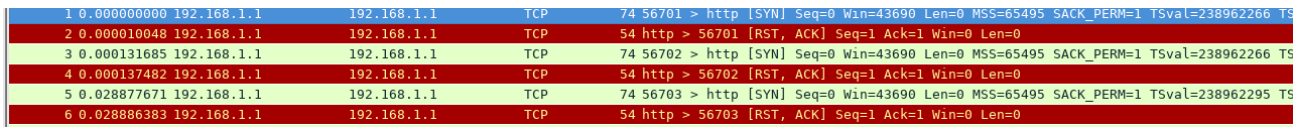


134	266.2358245	192.168.1.1	192.168.1.2	HTTP	668 GET / HTTP/1.1
136	266.2615401	192.168.1.2	192.168.1.1	HTTP	265 HTTP/1.1 304 Not Modified
148	328.8806514	192.168.1.1	192.168.1.2	HTTP	509 GET /favicon.ico HTTP/1.1
150	328.8929706	192.168.1.2	192.168.1.1	HTTP	511 HTTP/1.1 404 Not Found (text/html)

Para la solicitud http://server1.example.com, en wireshark entre cliente y servidor sale similar a los mensajes anteriores, pero con un campo Host distinto:

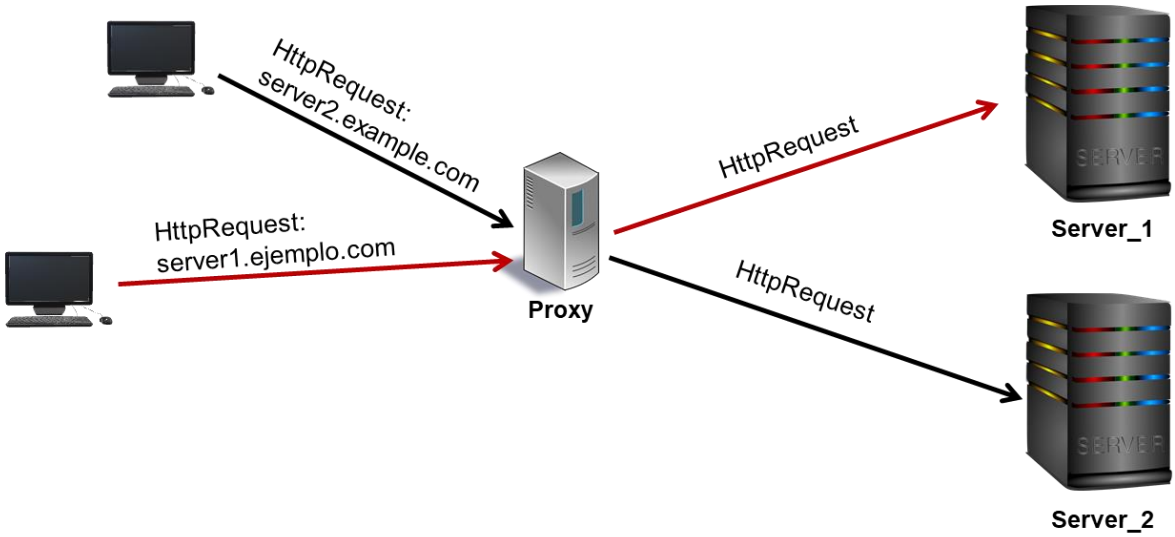
```
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
  Host: server2.example.com:8080\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
```

Y la petición pasaría por la interfaz lo del equipo del proxy:



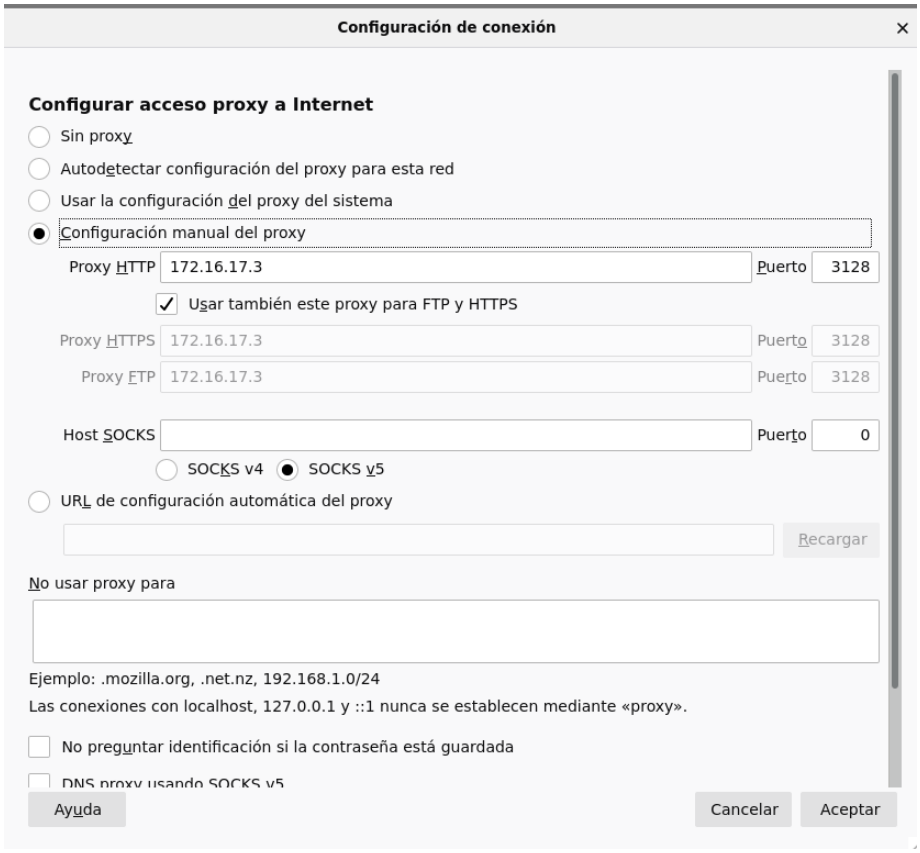
1	0.000000000	192.168.1.1	192.168.1.1	TCP	74 56701 > http [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=238962266 TS
2	0.000010048	192.168.1.1	192.168.1.1	TCP	54 http > 56701 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.000131685	192.168.1.1	192.168.1.1	TCP	74 56702 > http [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=238962266 TS
4	0.000137482	192.168.1.1	192.168.1.1	TCP	54 http > 56702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	0.028877671	192.168.1.1	192.168.1.1	TCP	74 56703 > http [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=238962295 TS
6	0.028886383	192.168.1.1	192.168.1.1	TCP	54 http > 56703 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Y el esquema gráfico sería algo así:



7.3. Escenario de Defensa 3: < Proxy en modo transparente>

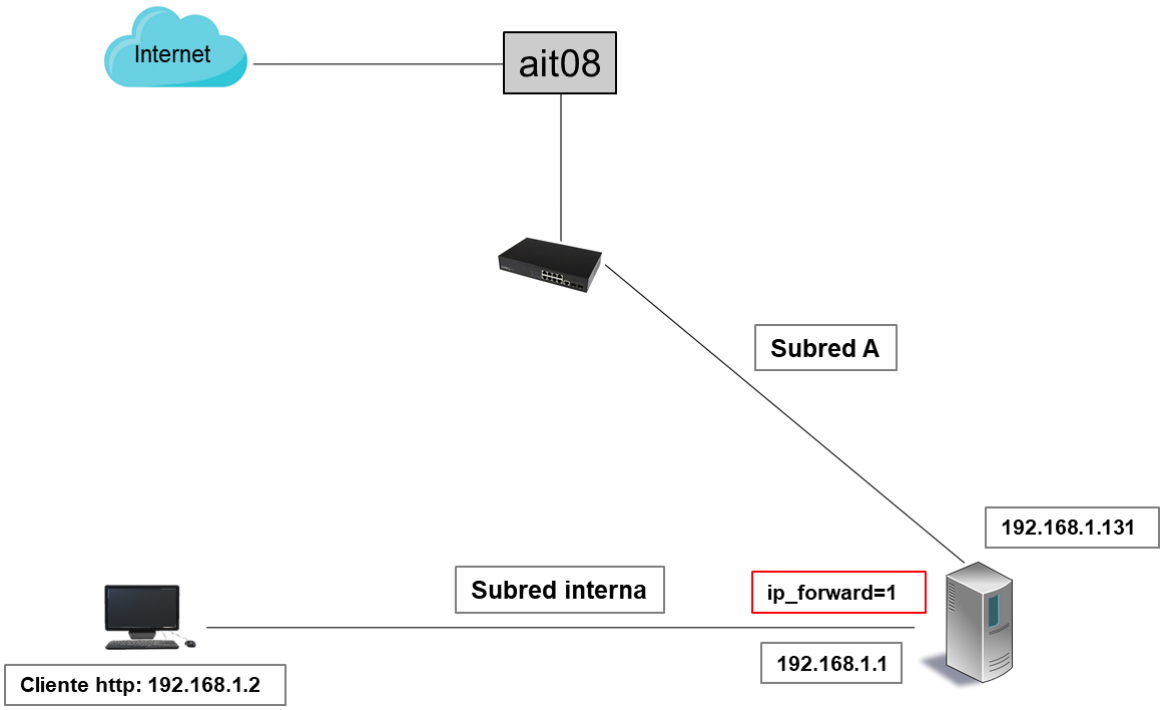
Normalmente, cuando queremos realizar una conexión indirecta a partir de un proxy, es necesario que sepamos la dirección y puerto a la que nos conectamos, y si accedemos desde el navegador, configurarlo en la siguiente pantalla:



Esto sería el uso más básico para un proxy directo. Nosotros lo que buscamos en este escenario es que un cliente (aislado en una subred, como en los escenarios previos pasaba con los servidores web) que quiere conectarse a un servidor web de fuera, pueda hacerlo realizando la solicitud http sin necesidad de configurar nada. De ahí el nombre de transparente, porque el cliente no necesita saber que el proxy es el que reenvía su petición y permite su conexión http. Para ello, nos vamos a apoyar en el servicio de **iptables**.

7.3.1. Escenario 3: Esquema de la red

El esquema del escenario junto con las direcciones de cada interfaz de los equipos que vamos a usar es el siguiente:



7.3.2. Escenario 3: Configuración del servidor

Para configurar el servidor proxy, únicamente tenemos que coger el fichero de configuración de Squid de inicio, y añadirle la sentencia `intercept` al puerto de escucha:

/etc/squid/squid.conf

```
http_port 3128 intercept
```

Que es la manera de decirle a squid que con toda petición que le llegue al puerto 3128, actúe como proxy directo transparente. ¿Cómo haremos que las peticiones http/s de un cliente lleguen a ese puerto? Con *iptables*, aplicamos las siguientes reglas:

```
service iptables start

iptables -t nat -A PREROUTING -p tcp -s 192.168.1.0/29 --dport 80 -j
REDIRECT --to-port 3128

  iptables -t nat -A PREROUTING -p tcp -s 192.168.1.0/29 --dport 443 -j
REDIRECT --to-port 3128

iptables -t nat -A POSTROUTING -s 192.168.1.0/29 -d 0.0.0.0/0 -o eth0 -j
MASQUERADE
```

Con estas reglas lo que generamos es que con todas las peticiones TCP que lleguen por el puerto 80 o 443 (http y https) con dirección IP origen de la subred interna, se modifique IP destino poniéndose el propio equipo proxy como destinatario y con puerto destino el de Squid. Entonces, actuará el proxy realizando la petición como si fuera el cliente final. Es indispensable, como marca el esquema, activar el reenvío.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

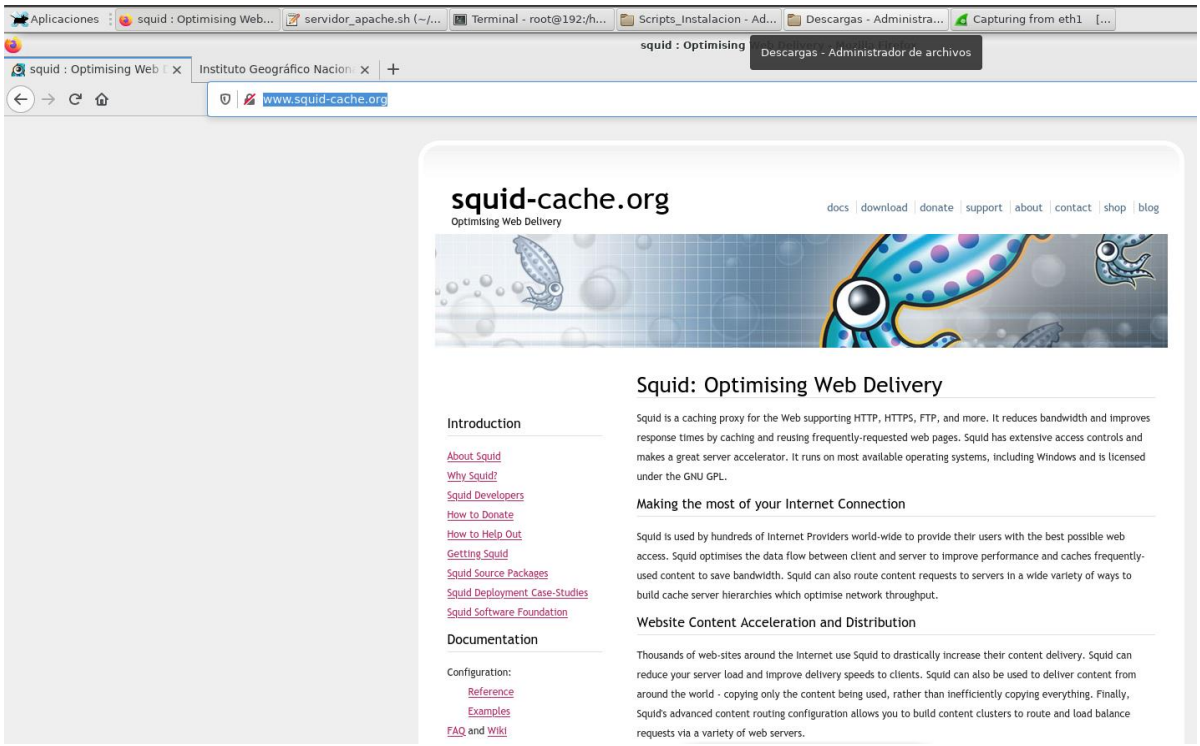
Es importante que en la tabla de encaminamiento IP del cliente, la entrada *default* (su *gateway*) sea el servidor proxy.

```
[root@192 Scripts_Instalacion]# ip r
default via 192.168.1.1 dev eth1
172.16.182.0/24 dev vmnet1 proto kernel scope link src 172.16.182.1
192.168.1.0/29 dev eth1 proto kernel scope link src 192.168.1.2
192.168.117.0/24 dev vmnet8 proto kernel scope link src 192.168.117.1
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1
[root@192 Scripts_Instalacion]#
```

7.3.3. Escenario 3: Tests y Pruebas del Escenario

Accediendo desde el equipo cliente a la página oficial de squid: http://www.squid-cache.org/

```
firefox http://www.squid-cache.org/ &
```



Desde los logs de access.log podemos comprobar el acceso del equipo cliente (192.168.1.2)

```
1641568938.341    607 192.168.1.2 TCP_MISS/200 8631 GET http://www.squid-cache.org/ - ORIGINAL
_DST/212.199.163.170 text/html
1641568938.852    322 192.168.1.2 TCP_MISS/200 3906 GET http://www.squid-cache.org/default.css
 - ORIGINAL_DST/212.199.163.170 text/css
1641568939.095    205 192.168.1.2 TCP_MISS/200 438 GET http://www.squid-cache.org/Images/img2.
gif - ORIGINAL_DST/212.199.163.170 image/gif
1641568939.122    220 192.168.1.2 TCP_MISS/200 776 GET http://www.squid-cache.org/Images/img1.
gif - ORIGINAL_DST/212.199.163.170 image/gif
1641568939.122    216 192.168.1.2 TCP_MISS/200 768 GET http://www.squid-cache.org/Images/img3.
gif - ORIGINAL_DST/212.199.163.170 image/gif
1641568939.128    216 192.168.1.2 TCP_MISS/200 421 GET http://www.squid-cache.org/Images/img5.
gif - ORIGINAL_DST/212.199.163.170 image/gif
1641568939.145    226 192.168.1.2 TCP_MISS/200 754 GET http://www.squid-cache.org/Images/img8.
gif - ORIGINAL_DST/212.199.163.170 image/gif
1641568939.170    317 192.168.1.2 TCP_MISS/200 29127 GET http://www.squid-cache.org/Images/img
4.jpg - ORIGINAL_DST/212.199.163.170 image/jpeg
1641568939.433    255 192.168.1.2 TCP_MISS/200 1742 GET http://www.squid-cache.org/favicon.ico
 - ORIGINAL_DST/212.199.163.170 image/vnd.microsoft.icon
```

Si accedemos a Google:

Esto se debe a que no se pueden hacer pasar por un proxy transparentes conexiones a los puertos http seguros (443). No se trata de un error, es una característica de las conexiones SSL, en este caso se tendría que hacer nat o enmascarar todo lo que vaya al puerto 443 en vez de hacerlo pasar por el proxy squid en modo transparente.

7.3.4. Escenario 3: Análisis del intercambio real de mensajes de red

Para el acceso a la página oficial de squid (http), el intercambio de mensajes en wireshark entre cliente y proxy sería:

145	188.898601938	192.168.1.2	212.199.163.170	HTTP	425 GET / HTTP/1.1
152	189.504927581	212.199.163.170	192.168.1.2	HTTP	168 HTTP/1.1 200 OK (text/html)
155	189.693710412	192.168.1.2	212.199.163.170	HTTP	388 GET /default.css HTTP/1.1
161	190.016379849	212.199.163.170	192.168.1.2	HTTP	3652 HTTP/1.1 200 OK (text/css)
164	190.017816273	192.168.1.2	212.199.163.170	HTTP	388 GET /Images/img4.jpg HTTP/1.1
166	190.055133151	192.168.1.2	212.199.163.170	HTTP	399 GET /Images/img2.gif HTTP/1.1
171	190.067039147	192.168.1.2	212.199.163.170	HTTP	399 GET /Images/img1.gif HTTP/1.1
176	190.071057765	192.168.1.2	212.199.163.170	HTTP	399 GET /Images/img3.gif HTTP/1.1
181	190.075973174	192.168.1.2	212.199.163.170	HTTP	399 GET /Images/img5.gif HTTP/1.1
186	190.084309225	192.168.1.2	212.199.163.170	HTTP	399 GET /Images/img8.gif HTTP/1.1
199	190.259709308	212.199.163.170	192.168.1.2	HTTP	185 HTTP/1.1 200 OK (GIF89a)
203	190.286458471	212.199.163.170	192.168.1.2	HTTP	522 HTTP/1.1 200 OK (GIF89a)
207	190.286903909	212.199.163.170	192.168.1.2	HTTP	514 HTTP/1.1 200 OK (GIF89a)
210	190.292191251	212.199.163.170	192.168.1.2	HTTP	168 HTTP/1.1 200 OK (GIF89a)
214	190.309619326	212.199.163.170	192.168.1.2	HTTP	500 HTTP/1.1 200 OK (GIF89a)
222	190.334264275	212.199.163.170	192.168.1.2	HTTP	2305 HTTP/1.1 200 OK (JPEG JFIF image)
225	190.341314785	192.168.1.2	212.199.163.170	HTTP	346 GET /favicon.ico HTTP/1.1
231	190.597034848	212.199.163.170	192.168.1.2	HTTP	1472 HTTP/1.1 200 OK (image/vnd.microsoft.icon)

Aquí puede demostrarse como actúa el proxy en modo transparente, porque el cliente ignora que el mensaje le va a llegar, su destino es el servidor web final.

Y para el acceso desde el proxy al servidor web de squid:

290 169.637453349	192.168.93.131	212.199.163.170	HTTP	514	GET / HTTP/1.1
292 169.835781343	212.199.163.170	192.168.93.131	HTTP	8605	HTTP/1.1 200 OK (text/html)
299 170.145717242	192.168.93.131	212.199.163.170	HTTP	477	GET /default.css HTTP/1.1
301 170.348501324	212.199.163.170	192.168.93.131	HTTP	3880	HTTP/1.1 200 OK (text/css)
313 170.458497484	192.168.93.131	212.199.163.170	HTTP	477	GET /Images/img4.jpg HTTP/1.1
317 170.492393036	192.168.93.131	212.199.163.170	HTTP	488	GET /Images/img2.gif HTTP/1.1
321 170.511421035	192.168.93.131	212.199.163.170	HTTP	488	GET /Images/img1.gif HTTP/1.1
325 170.513315401	192.168.93.131	212.199.163.170	HTTP	488	GET /Images/img3.gif HTTP/1.1
329 170.518488874	192.168.93.131	212.199.163.170	HTTP	488	GET /Images/img5.gif HTTP/1.1
333 170.531464026	192.168.93.131	212.199.163.170	HTTP	488	GET /Images/img8.gif HTTP/1.1
337 170.592257720	212.199.163.170	192.168.93.131	HTTP	412	HTTP/1.1 200 OK (GIF89a)
340 170.618615693	212.199.163.170	192.168.93.131	HTTP	750	HTTP/1.1 200 OK (GIF89a)
342 170.618770941	212.199.163.170	192.168.93.131	HTTP	742	HTTP/1.1 200 OK (GIF89a)
348 170.625254324	212.199.163.170	192.168.93.131	HTTP	395	HTTP/1.1 200 OK (GIF89a)
351 170.642419666	212.199.163.170	192.168.93.131	HTTP	728	HTTP/1.1 200 OK (GIF89a)
355 170.666512771	212.199.163.170	192.168.93.131	HTTP	14581	HTTP/1.1 200 OK (JPEG JFIF image)
365 170.782516755	192.168.93.131	212.199.163.170	HTTP	435	GET /favicon.ico HTTP/1.1
367 170.929866251	212.199.163.170	192.168.93.131	HTTP	1716	HTTP/1.1 200 OK (image/vnd.microsoft.icon)

Con Host:

```
▸ GET / HTTP/1.1\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Host: www.squid-cache.org\r\n
```

8. Interfaz gráfica de administración del servidor

No procede.

9. Deficiencias del servicio

Como se puede ver en el Escenario 3, el modo transparente es bastante poco útil ya que no puede atender peticiones HTTPS (por razones que explicamos en ese apartado), y las HTTP dependiendo del sitio web lo atiende bien o a medias, como es el caso del acceso a http://www.ign.es/web/ign/portal, que ofrece un servicio bastante pobre y lento.

10. Ampliaciones/mejoras del servicio

Es verdad que Squid es un proxy especializado en HTTP, por lo tanto no les ha llegado a interesar mucho el hecho de ampliar la funcionalidad con SOCKS, supongo que porque para eso los usuarios acuden a Dante o Delegate (siempre se recomiendan esos dos servicios en los distintos foros si se habla de proxy SOCKS en squid). Lo que sí está claro es que sería una ampliación bastante importante para el crecimiento del proyecto, y ya se ha intentado de

hecho. Es una funcionalidad que está en Testing, aunque lleva sin actualizarse desde septiembre de 2015.

11. Incidencias y principales problemas detectados

Problemas en el uso de Squid como proxy SOCKS, a la hora de ejecutar el ./configure con los parámetros que se indican en https://wiki.squid-cache.org/Features/Socks?highlight=%28%5CbCategoryFeature%5Cb%29%7C%28Status.%2Acomplete%29%7C%28Status.%2AObsolete%29%7C%28faqlisted.yes%29

12. Resumen y Conclusiones

A pesar de haber probado el servicio a pequeña escala y en escenarios muy liberados de tráfico, Squid demuestra que tiene alternativas inteligentes y eficaces para la protección de datos de sus clientes y la protección y transparencia ante los servidores web. Y que además sea un servicio OpenSource lo hace todavía más interesante, un proyecto donde cualquiera puede aportar su granito de arena y que no para de crecer. También tiene documentación bastante buena y bastantes foros de ayuda que lo hacen más popular.

Me pica la curiosidad por la parte de SOCKS al no haber podido dedicarle todo el tiempo que me hubiera gustado.