



Implementación y configuración de un servidor

Universidad de Murcia
3º Grado en Ingeniería Informática
Asignatura: Servicios Telemáticos
Curso 2018/2019

Proyecto realizado por Nicolás Enrique Linares La Barba.

ÍNDICE

1. Introducción	3
2. Descripción del escenario desarrollado y versiones de software	4
3. Descripción de las configuraciones.....	7
3.1. Apache HTTP.....	7
3.2. DNS	9
3.3. SMTP/POP	14
3.4. OpenSSL X.509 (autenticación del servidor)	17
3.5. Apache SSL (autenticación del cliente).....	20
3.6. IPsec	23
4. Descripción de la implementación del servicio Web-SSTT HTTP	25
4.1. Descripción del protocolo HTTP.....	25
4.2. Características relevantes	25
4.3. Tipos de mensaje	26
4.3.1. HTTP Request	26
4.3.2. Métodos de petición	27
4.3.3. HTTP Response.....	27
4.3.4. Códigos de estado.....	27
4.4. Implementación	28
4.4.1. Parámetros de entrada y ejecución	28
4.4.2. Gestión de clientes.....	29
4.4.3. Procesamiento de la petición GET	30
4.4.4. Persistencia	38
5. Estado final del servidor – Puertos activos	40
6. Trazas representativas de los protocolos	41
7. Problemas encontrados en el proceso de desarrollo.....	48
8. Número de horas empleadas para cada apartado	49
9. Conclusiones y valoración personal.....	50

1. Introducción

La presente memoria se centra en el estudio de servicios telemáticos, que se define como el campo de la informática que analiza, diseña y se encarga de la gestión de aplicaciones de red y servicios de comunicaciones para la transferencia de información entre equipos.

A lo largo de la asignatura se han explicado ciertos servicios que se han puesto en práctica para su adecuado entendimiento y por su relevancia en las redes de comunicaciones actuales. Se pondrán en práctica los protocolos, estándares y tecnologías implicadas en la implementación de los servicios web como HTTP y Apache, de correo electrónico con SMTP/POP y de acceso a la red con DNS, así como los relacionados con la comunicación segura de los mismos mediante OpenSSL e IPsec.

La práctica se divide en un primer ejercicio de programación de un servidor web HTTP que permita procesar las peticiones GET que realiza un cliente a través de un navegador como Firefox, por ejemplo, mediante una conexión TCP. En este apartado se pondrán en práctica los conceptos de peticiones GET, cookies y persistencia, entre otros.

La segunda parte de la práctica conlleva la configuración de los servicios mencionados antes. Se comenzará configurando un servidor web Apache, que sustituirá al servidor HTTP programado en el primer ejercicio. A continuación, se utilizará un servicio DNS para establecer un dominio (sstt8654.org) y llevar a cabo la traducción de nombres tras cada petición del cliente. Se establecerá también un servidor de correo SMTP/POP para este dominio, donde se hará uso de la aplicación de correo Thunderbird para enviar mensajes entre dos usuarios.

Además, se tratarán aspectos de seguridad, como el acceso a una página web segura con HTTPS en el caso de Apache o realizando una protección en la comunicación a nivel de red (seguridad a nivel 3) mediante IPsec. Se pondrán en práctica conceptos como certificados, el protocolo IKEV2 (Protocolo Internet Key Exchange), las cabeceras AH o ESP o los modos de operación (túnel y transporte), entre otros.

En el siguiente punto se presenta el escenario donde se va a desplegar cada servicio, un apartado para cada una de las configuraciones realizadas y se incluirá otro apartado para sus respectivas trazas en Wireshark que muestran su correcto funcionamiento.

2. Descripción del escenario desarrollado y versiones de software

En el desarrollo de la práctica se hará empleo de VirtualBox para el escenario de red, el cual va a constar de dos equipos virtualizados proporcionados por la asignatura y con el sistema operativo de Ubuntu en su versión 16.04, Host1 para el servidor y Host2 para el cliente como se puede ver en la siguiente imagen:

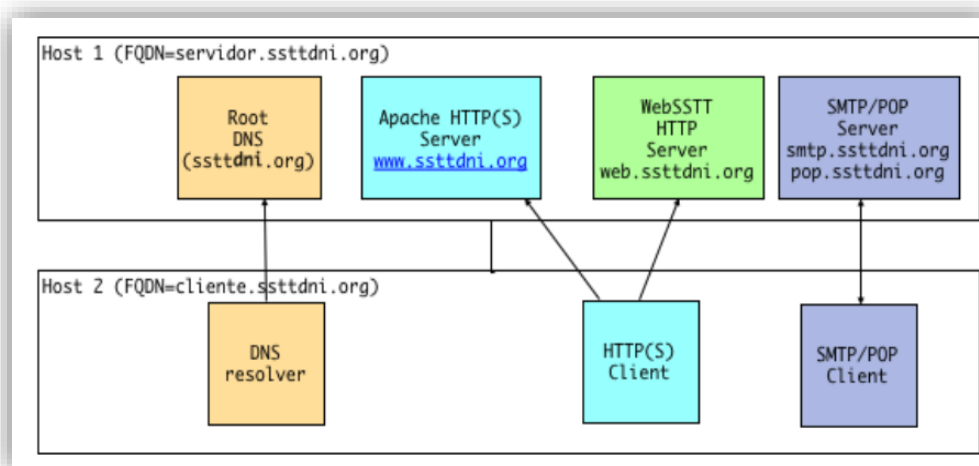


Figura 1. Escenario de red.

Como se ha mencionado en el párrafo anterior, el Host1 (servidor.sstt8654.org) será el servidor encargado de proveer los servicios al cliente, Host2 (cliente.sstt8654.org) en la imagen.

A continuación, se explican los protocolos de cada servicio que se debe desplegar en el servidor y el software que empleará cada uno.

Apache HTTP

Apache es un software open source de servidor HTTP desarrollado por Apache Software Foundation para sistemas Unix y Windows. Su característica más importante es su estructura modular, componiéndose de un núcleo básico y de módulos que se pueden agregar para completar o añadir funcionalidades, como el módulo SSL que veremos más adelante.

El protocolo HTTP, de sus siglas en inglés "Hypertext Transfer Protocol", es un protocolo de la capa de aplicación que trabaja sobre una conexión TCP en el puerto 80 y es la base de cualquier intercambio de datos en la web. Para más información, consultar el apartado [de programación web](#) donde se profundiza más en este protocolo.

Un servidor web HTTP como Apache no es más que el software que recibe la solicitud del cliente (HTTP Request) para acceder a una página web, la procesa y si dispone de esa página web se la manda al navegador del cliente para poder visualizarla (HTTP Response), en caso contrario dispone de mecanismos para resolver y notificar de los errores.

Se empleará Apache en la versión 2.4.18 (Ubuntu) como servidor HTTP en sistemas Linux.

Servicio DNS

El Sistema de Nombres de Dominio o **DNS** (*Domain Name System*) es un protocolo de la capa de aplicación que trabaja en el puerto 53 y permite identificar a cada Host de la red, ya que se encarga de la traducción de nombres de host a direcciones IP, ofreciéndonos una interfaz más fácil para navegar por internet.

DNS es una base de datos jerárquica y distribuida en un gran número de servidores. Ningún servidor DNS dispone de todas las traducciones a nombres de Host por lo que se lleva a cabo una comunicación entre ellos hasta que el nombre es resuelto.

En esta práctica se implementará un Servicio DNS para resolver las direcciones de los siguientes equipos:

- cliente.sstt8654.org
- servidor.sstt8654.org
- www.sstt8654.org
- web.sstt8654.org
- smtp.sstt8654.org
- pop.sstt8654.org
- dns.sstt8654.org

Se empleará BIND (*Berkeley Internet Name Domain*) en su versión 9.10.3, que es el servidor DNS para sistemas Unix más usado en internet.

Servicio de correo SMTP/POP

El **servicio de correo SMTP/POP** permitirá el intercambio de mensajes entre usuarios de forma asíncrona, es decir, sin necesidad de coordinarse, pues envían y reciben mensajes cuando les conviene.

El protocolo simple de transferencia de correo o SMTP (Simple Mail Transfer Protocol) trabaja en el puerto 25 y se encargará de enviar los mensajes entre el servidor de correo del emisor hasta el servidor de correo del destinatario, que para el caso de la práctica solo existirá un servidor de correo en el dominio definido; y con el protocolo de oficina de correos o POP (Post Office Protocol) trabaja en el puerto 110 y podremos transferir los mensajes desde el servidor de correo hasta el agente de usuario (destinatario).

En el servidor se hará uso del agente de transporte de correo Exim (*EX*perimental *I*nternet *M*ailer) en su versión 4.86 para la configuración de SMTP, y Dovecot en su versión 2.2.22 para el servicio de POP3. En el caso del cliente se deberá configurar el cliente de correo Thunderbird para la gestión de los mensajes.

Apache SSL

Servicio web Apache HTTP/ HTTPS como servidor web que permitirá entregar las páginas web que solicite el cliente. HTTP estará escuchando en el puerto 80 y HTTPS escuchará en el puerto 443 ofreciendo una conexión segura.

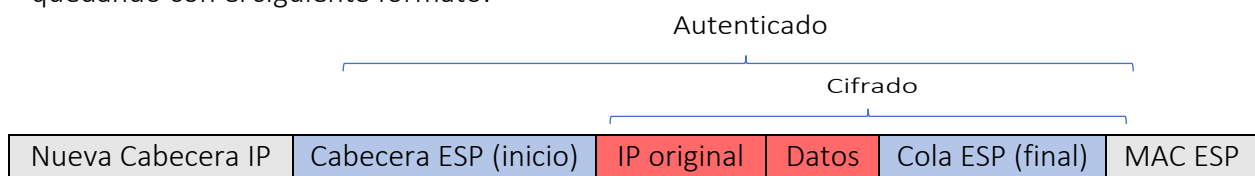
OpenSSL es un kit de herramientas con todas las funciones para los protocolos de Seguridad de la capa de transporte (TLS) y de la Capa de sockets seguros (SSL) que se empleará para crear los certificados de clave pública. Esto nos permitirá, más adelante, establecer una conexión segura entre el cliente y el servidor por medio de HTTPS al acceder a la página web <https://www.sstt8654.org>.

El Host1 como servidor será la Autoridad de certificación (CA) cuyo trabajo será validar las identidades y emitir los certificados de cliente y servidor utilizando la firma electrónica. Estos certificados asocian una entidad a su correspondiente clave pública. A su vez hará el papel de Autoridad de registro (RA) para certificar el servicio web www.sstt8654.org.

Se empleará Apache en la versión 2.4.18 (Ubuntu) como servidor HTTP en sistemas Linux y mediante la herramienta OpenSSL en su versión 1.0.2g se crearán los certificados digitales necesarios para la conexión segura.

IPsec

Se desplegará en el escenario propuesto el protocolo de seguridad de IP o **IPsec** que proporcionará seguridad en la capa de red. Se realizará una configuración **en modo túnel** para cifrar y autenticar la comunicación entre el cliente y el servidor. Y veremos cómo el paquete IP original se empaqueta entre dos cabeceras ESP y se añade la nueva cabecera IP destino, quedando con el siguiente formato:



Se empleará el software Strongswan, en ambos equipos, en su versión 5.3.5 que proporcionará los mecanismos de autenticación entre los hosts a través de los certificados creados con OpenSSL.

3. Descripción de las configuraciones

3.1. Apache HTTP

La configuración de este servicio se va a centrar en 2 estructuras de directorios relevantes que se muestran en la siguiente imagen, con los ficheros y directorios que se van a manipular:

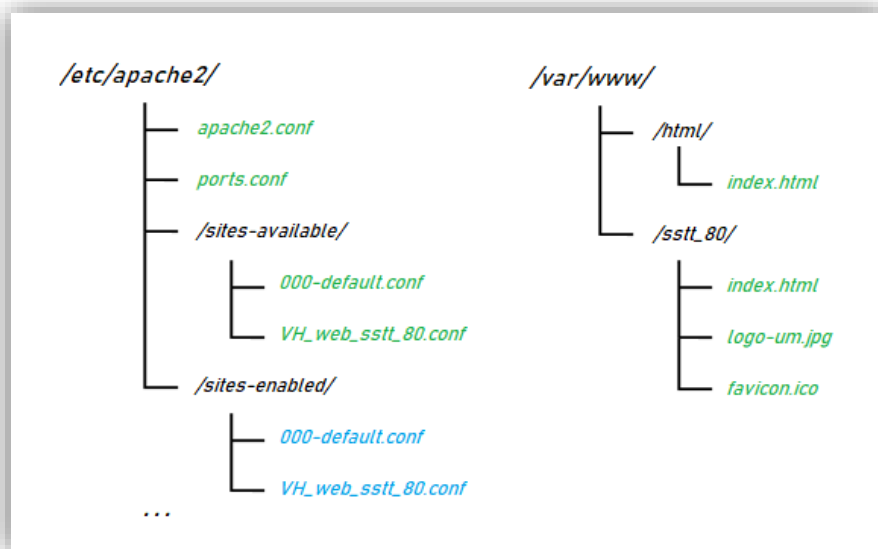


Figura 2. Estructura de directorios para Apache. En negro directorios, en verde ficheros y en azul enlaces simbólicos.

En la Figura 2, en el árbol de directorios de la izquierda, se encuentra el fichero **apache2.conf** el cual no va a modificarse, pero es importante saber que contiene la configuración global de Apache.

El fichero **ports.conf** define los puertos donde va a escuchar el servidor, es decir, donde va a recibir las peticiones HTTP. No se modifica, por convenio se encuentra en el puerto 80 (“Listen 80”).

Apache tiene la capacidad de servir peticiones en varios puertos a la vez, simulando varios servidores en una misma máquina (Virtual Hosts), los cuales se almacenan en el directorio “sites-available”.

En este directorio se encuentra por defecto el virtual host “000-default.conf” configurado en el puerto 80 para servir la página oficial de Apache que se encuentra en el directorio “/var/www/html/”.

Configuración del Virtual Host

Para esta configuración solo será necesario un Virtual Host que escuche en el puerto 80, el cual tiene la siguiente configuración:

```
<VirtualHost *:80>
    ServerAdmin alumno@sstt8654.org
    ServerName www.sstt8654.org
    DocumentRoot /var/www/sstt_80
    <Directory /var/www/sstt_80>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>
</VirtualHost>
```

Figura 3. Fichero “VH_web_sstt_80.conf”

- La cláusula “<VirtualHost *:80>” indica el comienzo de la definición del virtual host indicando que escuchará en todas sus interfaces de red en el puerto 80.
- ServerName indica el nombre del servidor al que responde la máquina virtual, siendo para esta práctica www.sstt8654.org.
- DocumentRoot indica donde se encuentran las páginas web. Para ello se ha creado el directorio “sstt_80” que contiene la página web proporcionada por la asignatura.
- Mediante la cláusula “<Directory /var/www/sstt_80>” se indica que solo al contenido del directorio “sstt_80” se va a aplicar los siguientes criterios de control de accesos:
 - *Options*: controla qué funciones están disponibles.
 - *Indexes*: para devolver la página index.html si se indica solo el directorio.
 - *FollowSymLinks*: para seguir enlaces simbólicos como en el caso de index.html que hace referencia al logo-um.jpg y al favicon.ico.
 - *MultiViews*: para elegir entre una opción si hay coincidencias.
 - *AllowOverride None*: para omitir el uso de archivos .htaccess.
 - *Order allow, deny*: las reglas *permitir* son evaluadas antes que las *denegar*.
 - *Allow from all*: todos los hosts tienen acceso.

Este fichero se debe crear en “sites-available” y al ejecutar “sudo a2ensite VH_web_sstt_80.conf” pasará a estar activado, apareciendo un enlace en el subdirectorio “sites-enabled”. En la Figura 4 se muestra como debe quedar finalmente estos directorios.

```
sites-available:
000-default.conf  VH_web_sstt_80.conf

sites-enabled:
000-default.conf  VH_web_sstt_80.conf
```

Figura 4. Virtual Host disponibles y activos.

3.2. DNS

Para la implantación de este servicio deberemos configurar tanto el servidor como el cliente. Además, haremos uso de los comandos *host* y *dig* para realizar consultas y verificar el comportamiento adecuado del servidor DNS.

Configuración en el Servidor

Una vez instalado Bind9 nos situamos en su directorio: */etc/bind/*. En esta carpeta se encuentran los ficheros *named* para la configuración global del demonio *bind*, pero solo modificaremos lo siguiente:

- En el fichero *named.conf.local* se define la zona local administrada por este servidor DNS. Se indica el nombre del dominio “*sstt8654.org*”, cuyo tipo de zona es *master*, es decir, que posee la mayor autoridad para esta zona, y se especifica con la sentencia *file* la ruta del fichero con la configuración de la zona. Se puede ver en la siguiente figura:

```
zone "sstt8654.org" IN {  
    type master;  
    file "/etc/bind/db.sstt8654.zone";  
};
```

Figura 5. Fichero *named.conf.local*.

- En el fichero *named.conf.options* completaremos la configuración global del servidor. Añadimos en la cláusula *forwarders* la IP del servidor DNS de la UMU (155.54.1.1) a la que redireccionará las peticiones cuando no pueda resolverlas, modificamos la entrada *dnssec-validation* a “no” para ... y por último modificamos la entrada *auth-nxdomain* a “yes” para permitir que el servidor responda con autoridad cuando devuelve respuestas de NXDOMAIN.

Quedando de la siguiente manera:

```
options {  
    directory "/var/cache/bind/";  
  
    forwarders {  
        155.54.1.1;  
    };  
  
    dnssec-validation no;  
  
    auth-nxdomain yes;    # conform to RFC1035  
    listen-on-v6 { any; };  
};
```

Figura 6. Fichero *named.conf.options*.

Ahora debemos crear el **fichero de zona de dominio** “db.sstt8654.org” que contendrá los **registros de recursos (RR)** que cada mensaje de respuesta DNS transporta. Este fichero se define con las siguientes entradas:

```
$TTL      604800
@         IN      SOA      dns.sstt8654.org.  root.sstt8654.org. (
                        1      ; serial
                        3600    ; actualizar (1 hora)
                        1800    ; reintentar (30 mins)
                        604800  ; expira (7 dias)
                        3600    ; minimo (1 hora)
) ;

@         IN      NS       dns.sstt8654.org.
@         IN      A        192.168.56.106

@         IN      MX       10      smtp.sstt8654.org.

cliente   IN      A        192.168.56.105
servidor  IN      A        192.168.56.106

www       IN      CNAME    servidor
web       IN      CNAME    servidor
pop       IN      CNAME    servidor
smtp      IN      A        192.168.56.106
dns       IN      A        192.168.56.106
```

Figura 7. Fichero db.sstt8654.org.

Directivas

- La directiva **\$TTL** indica el tiempo en segundos que la información sobre el registro permanecerá en caché.

Tipos de registros de recursos

- **SOA**: registro de recursos que define el nombre de la zona “sstt8654.org”, el nombre del servidor DNS que tiene autoridad para este dominio “dns.sstt8654.org” y también el correo del administrador del dominio “root.sstt8654.org”. Entre paréntesis se definen parámetros que afectan a todos los registros de la zona.
- **NS**: identifica que el servidor de nombres es “dns.sstt8654.org”.
- **MX**: identifica que el servidor de correo es “smtp.sstt8654.org”. Si no se especifica este registro, entonces se solicitará el registro A del dominio en su lugar. Por ello sigue funcionando, aunque no se ponga esta entrada en el fichero.
- **CNAME**: permite definir el alias “servidor” el cual hace referencia a la indicada con el registro tipo A.
- **A**: asocia un nombre con su dirección IP. Se definen los distintos nombres de equipos que están gestionados en este servidor DNS, son el caso de:

- cliente.sstt8654.org
- servidor.sstt8654.org
- www.sstt8654.org
- web.sstt8654.org
- pop.sstt8654.org

- smtp.sstt8654.org
- dns.sstt8654.org

Configuración en el Cliente

La configuración en el cliente es mínima, solo se deben modificar dos ficheros:

- /etc/resolv.conf: que define el servidor DNS al que consultará el host cuando necesite resolver cualquier dirección, por lo que pondremos la dirección IP del servidor DNS que se ha definido antes. Solo debe contener las siguientes entradas:
 - “search sstt8654.org”: dominio por defecto cuando no se indique.
 - “servername 192.168.56.106”: dirección IP del servidor DNS.
- /etc/resolvconf/resolv.conf.d/base: evita que el cliente DHCP modifique el fichero anterior (aunque han surgido problemas pues se modificaba igual). Se debe incluir la entrada:
 - “servername 192.168.56.106”: dirección IP del servidor DNS.

Comando dig

Esta herramienta permite realizar consultas a un servidor DNS para solicitar información sobre su dominio. Una vez configurado el servidor DNS y el cliente para esta práctica, realizamos la siguiente consulta desde el cliente: **dig ANY sstt8654.org**

Podemos utilizar ANY para especificar que se muestren todos los tipos de registros que se sirven en la zona DNS del dominio “sstt8654.org”, como vemos en la siguiente figura:

```

alumno@desktop: ~
alumno@desktop:~$ dig ANY sstt8654.org

; <<>> DiG 9.10.3-P4-Ubuntu <<>> ANY sstt8654.org
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: NOERROR, id: 28178
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;sstt8654.org.                IN      ANY

;; ANSWER SECTION:
sstt8654.org.                604800  IN      SOA     dns.sstt8654.org. root.sstt8654.org. 1 3600 1800 604800 3600
sstt8654.org.                604800  IN      NS      dns.sstt8654.org.
sstt8654.org.                604800  IN      A       192.168.56.106
sstt8654.org.                604800  IN      MX      10 smtp.sstt8654.org.

;; ADDITIONAL SECTION:
dns.sstt8654.org.            604800  IN      A       192.168.56.106
smtp.sstt8654.org.           604800  IN      A       192.168.56.106

;; Query time: 9 msec
;; SERVER: 192.168.56.106#53(192.168.56.106)
;; WHEN: Fri May 17 16:36:15 CEST 2019
;; MSG SIZE rcvd: 169

```

Figura 8. Fichero db.sstt8654.org.

Se puede observar en la sección “ANSWER SECTION” como los registros SOA, NS, MX y A configurados antes en el servidor coinciden con el resultado de la consulta realizada.

En la sección “ADDITIONAL SECTION” se mostrará el servidor de nombres autorizado para esa zona, al igual que aparece en la sección “AUTHORITY SECTION” si solo ejecutamos “dig sstt8654.org”.

Pruebas en el navegador del cliente

Escribimos en el buscado “http://www.sstt8654.org” y se accederá por HTTP a través del puerto 80 a la página web administrada por el Virtual Host definido en la configuración de Apache, pues es la página que se ha especificado en /var/www/sstt_80, como muestra la figura 9.



Figura 9. Ejemplo 1 de conexión no segura.

El ejemplo anterior vemos que ha sido el servidor de Apache quien ha respondido a la petición porque http va por defecto al puerto 80, pero si ahora escribimos el puerto donde escucha el servidor programado WEB-SSTT “http://www.sstt8654.org:8080” entonces nos responderá dicho servidor:



Figura 10. Ejemplo 2 de conexión no segura.

Por otro lado, si escribimos cualquiera de las siguientes URL:

- servidor.sstt8654.org
- web.sstt8654.org
- smtp.sstt8654.org
- pop.sstt8654.org
- dns.sstt8654.org

Entonces actuará el Virtual Host por defecto que trabaja también en el puerto 80 con la página especificada en el directorio /var/www/html/ por lo que nos muestra la página de Apache, según muestra la figura 11.

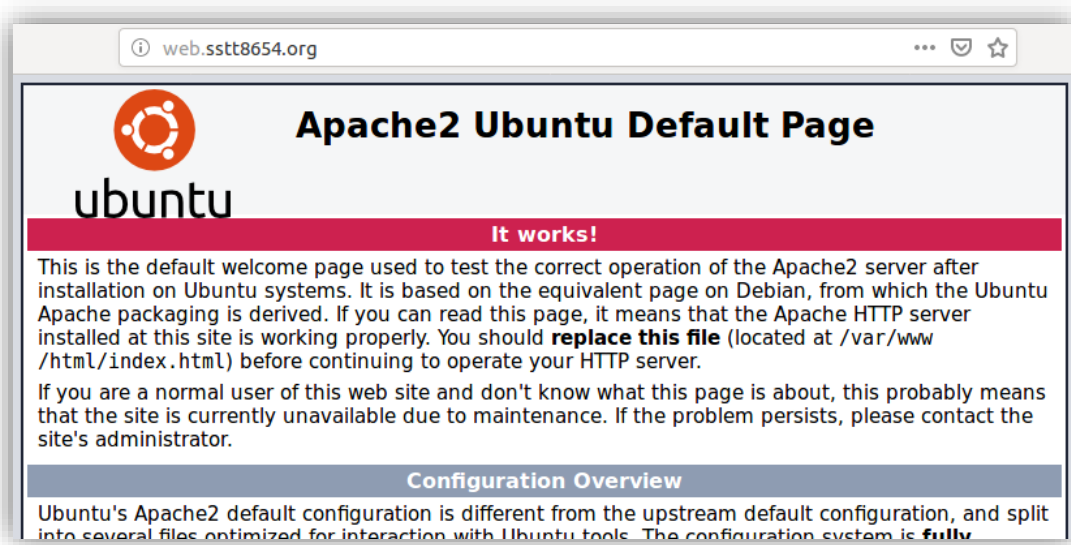


Figura 11. Ejemplo 3 de conexión no segura.

3.3. SMTP/POP

Configuración en el Servidor

Para comenzar la configuración, primero debemos declarar los nombres smtp.sstt8654.org y pop.sstt8654.org en el DNS como muestra la [figura 7](#). SMTP y DNS se relacionan a través del registro MX el cual especifica cómo debe ser encaminado un correo electrónico en internet, pues el registro MX apunta a los servidores a los cuales enviar un correo según una prioridad. Comprobamos su correcto funcionamiento mediante un ping a estos nombres.

– SMTP

Instalamos *exim4* y lo configuramos con el comando “dpkg-reconfigure exim4-config”, añadiendo lo siguiente:

- Tipo general del servidor:
Internet site; el correo se envía y recibe directamente usando SMTP.
- Nombre del sistema de correo:
sstt8654.org
- Direcciones IP en las que recibir conexiones SMTP:
(lo dejamos en vacío)
- Destinos de los que se acepta correo:
sstt8654.org
- Dominio para que se puede reenviar correo:
sstt8654.org
- Máquinas para las cuales reenviar correo:
192.168.56.0/24
- Limitar consultas DNS:
NO
- Formato de buzón de correo:
Formato <<Maildir>> en el directorio personal
- Dividir ficheros de configuración:
NO

– POP3

Instalamos *dovecot-pop3d* y en su directorio de configuración /etc/dovecot/conf.d modificamos los siguientes ficheros para la configuración básica de los usuarios creados en el servidor:

- **10-auth.conf:** buscamos la línea que ponga “disable_plaintext_auth = yes” y cambiamos su valor por “no”, lo que significa que el cliente puede iniciar sesión con autenticación de texto sin formato incluso cuando SSL / TLS no está habilitado en la conexión. Esto es inseguro, porque la contraseña de texto sin formato está expuesta a Internet.
- **10-mail.conf:** en este fichero se especifica el formato del buzón de correo que se va a emplear, debemos indicar la opción “mail_location = maildir:~/Maildir”.

– Creación de usuarios

Ahora debemos crear en el servidor los usuarios con los que se realizarán las pruebas, creamos el usuario *nombre1_8654* y *nombre2_8654* y ejecutamos con cada usuario en su directorio personal el comando “`maildirmake.dovecot ./Maildir`” para crear el directorio donde se gestionarán los mensajes de correo.

Cada usuario creado almacena sus correos en su directorio Maildir, si los correos son nuevos (no leídos) van a la carpeta `/home/usuario/Maildir/new` y si son leídos van a `/home/usuario/Maildir/cur`, salvo que se indique en el cliente que se borren los mensajes al leerlos.

Configuración en el Cliente

Como se ha comentado en el apartado 2, se utilizará el cliente de correo Thunderbird. No se debe realizar ninguna configuración a nivel de fichero, si no que abriremos la aplicación de correo y crearemos las cuentas de correo como se muestra en la siguiente secuencia de imágenes.

1. En configuración hacemos click en “Operaciones sobre la cuenta” y en “Añadir cuenta de correo...”.

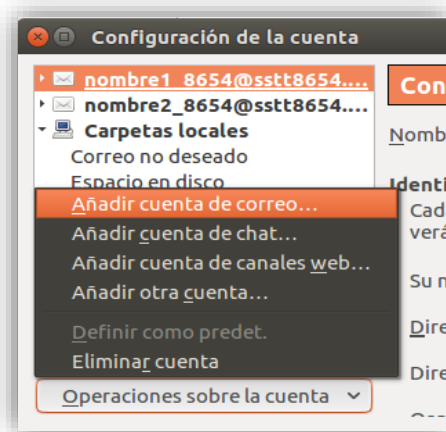


Figura 12. Ejemplo de creación de cuenta.

2. Introducimos los datos (mismo nombre y contraseña que el del usuario creado en el servidor) y le damos a continuar. Se cargará la configuración automáticamente y le damos al botón hecho. Nos aparecerá una advertencia indicando que no es seguro porque el servidor de correo no utiliza cifrado, pero le damos a “Entiendo los riesgos” y al botón hecho.

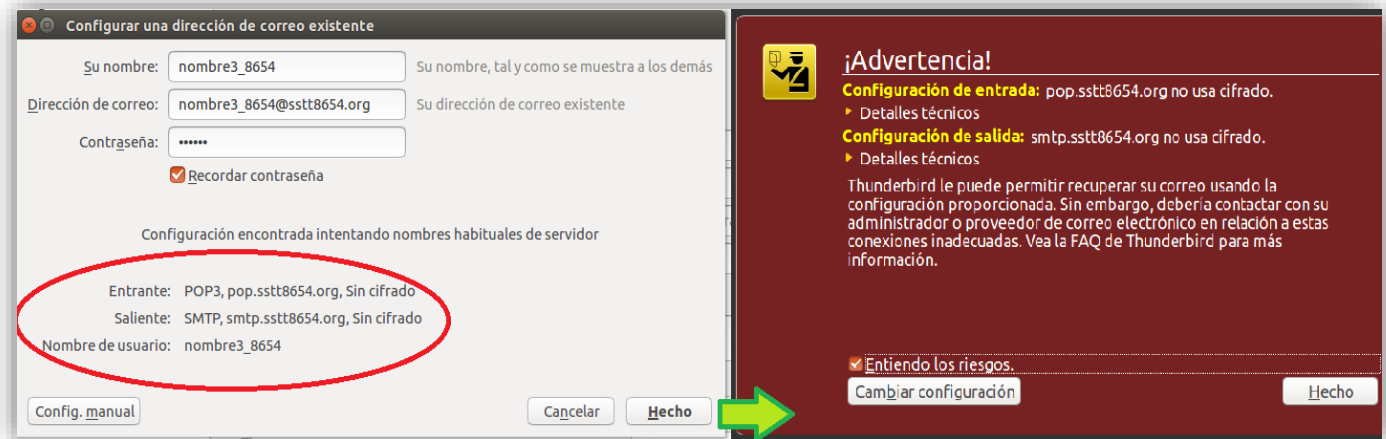


Figura 13. Ejemplo de creación de cuenta.

3. Por último, seleccionamos ambos usuarios y en su configuración, desmarcamos en la configuración del servidor la opción que indica “Dejar los mensajes en el servidor”.

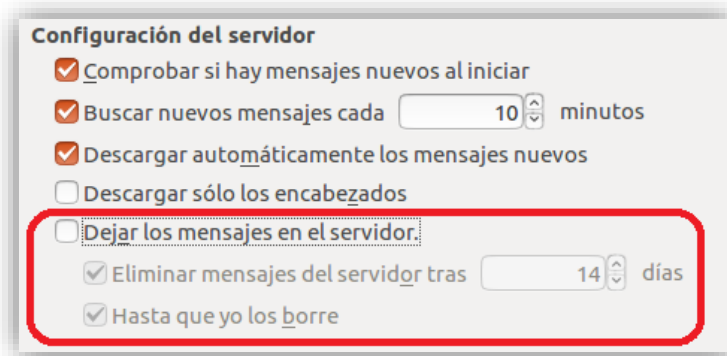


Figura 14. Ejemplo de creación de cuenta.

3.4. OpenSSL X.509 (autenticación del servidor)

Configuración inicial de la PKI

Se va a comenzar por la configuración básica para una PKI con OpenSSL que permitirá crear los certificados digitales para **autenticar al servidor**, para ello abrimos el fichero `/usr/lib/ssl/openssl.cnf` y realizamos las siguientes modificaciones:

- Buscamos la sección de “[CA_default]” y modificamos la primera línea para indicar el directorio raíz donde se almacenarán los certificados y otros ficheros:

dir = /home/alumno/demoCA

- Configuramos la estructura de nombres común que identificará a cada identidad. Buscamos la sección “[req_distinguished_name]” y modificamos las siguientes entradas:

- `countryName_default = ES`
- `stateOrProvinceName_default = Murcia`
- `0.organizationName_default = UMU`
- `organizationalUnitName_default = sstt8654`

Generar certificado de la CA

Las Autoridades de certificación (CA) son una autoridad fiable y de confianza, responsables de emitir los certificados digitales para aquellos que lo solicitan.

Para generar el certificado X.509 de la CA con cifrado RSA de 2048 bits y una duración válida de 10 años (3650 días) ejecutamos el siguiente comando:

openssl req -x509 -newkey rsa:2048 -keyout cakey.pem -out cacert.pem -days 3650

Nos pedirá introducir una palabra de paso (contraseña) y deberemos indicar los valores de la identidad de la CA, donde dejaremos los valores que especificamos en la configuración anterior, menos en el Common Name:

- Country Name (2 letter code) [ES]:
- State or Province Name (full name) [Murcia]:
- Locality Name (eg, city) []:
- Organization Name (eg, company) [UMU]:
- Organizational Unit Name (eg, section) [sstt8654]:
- Common Name (eg, YOUR name) []: **ca.sstt8654.org**
- Email Address []:

Este comando crea la clave privada de la CA (cakey.pem) protegida con una contraseña y el certificado X.509 de la CA que contiene la clave pública (cacert.pem). En este caso, se va a generar un certificado autofirmado ya que es una raíz de confianza.

Generar certificado para el servicio web www.sstt8654.org del servidor

La Autoridad de registro (RA), que se encuentra en el host1 de la [figura 1](#), debe emitir una solicitud de certificación para el servicio web www.sstt8654.org que se desea certificar. Para ello ejecuta el siguiente comando:

```
openssl req -new -nodes -newkey rsa:2048 -keyout serverkey.pem -out servercsr.pem
```

Nos pedirá los datos de la identidad que vamos a certificar, mantenemos los mismos valores menos el Common Name que debe ser la URL del servicio web:

- Country Name (2 letter code) [ES]:
- State or Province Name (full name) [Murcia]:
- Locality Name (eg, city) []:
- Organization Name (eg, company) [UMU]:
- Organizational Unit Name (eg, section) [sstt8654]:
- Common Name (eg, YOUR name) []: **www.sstt8654.org**
- Email Address []:

El resultado será clave privada (serverkey.pem) para el servicio web que se va a almacena en el servidor host1 y la solicitud de certificación (servercsr.pem). Esta solicitud debe ser firmada por la CA para generar el certificado X.509, con el siguiente comando:

```
openssl ca -keyfile private/cakey.pem -in servercsr.pem -out servercert.pem -days 400
```

Finalmente obtenemos el certificado X.509 (servercert.pem) con su clave pública para la URL www.sstt8654.org.

Configurar el navegador cliente

Se deberá copiar el certificado de la CA (cacert.pem) en el host2, el cliente, para poder importarlo en el navegador. Para ello nos situamos en Preferencias/Privacidad&Seguridad/ y le damos a "Ver certificados...". En la pestaña de Autoridades importamos dicho certificado, para el cual deberemos indicar el nivel de confianza (seleccionamos todos los campos para indicar que confiamos plenamente) y aceptamos. Se habrá sumado a la lista de certificados:

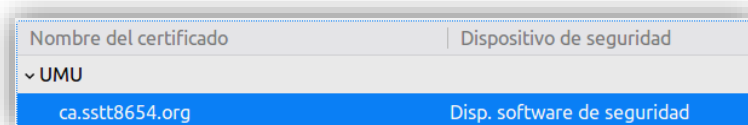


Figura 15. Certificado CA importado en el navegador cliente.

Configuración del Virtual Host en el servidor

Para poder establecer una conexión segura entre el cliente y Apache debemos comenzar creando otro Virtual Host, al igual que hicimos en el [apartado 3.1](#), pero esta vez para el puerto 443.

En el fichero `ports.conf`, que podemos ver en la figura 2, se encuentra la entrada *Listen 443* encerrada en una cláusula `<Ifmodule>` que se activa solo si el módulo de `ssl` se encuentra cargado. Por lo que no se va a modificar.

Para esta configuración solo será necesario crear un Virtual Host que escuche en el puerto 443, con la siguiente configuración:

```
<VirtualHost *:443>
    ServerAdmin alumno@sstt8654.org
    ServerName www.sstt8654.org
    DocumentRoot /var/www/sstt_443
    <Directory /var/www/sstt_443>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>
    SSLEngine on
    SSLCertificateFile      /home/alumno/demoCA/servercert.pem
    SSLCertificateKeyFile   /home/alumno/demoCA/serverkey.pem
    SSLCACertificateFile    /home/alumno/demoCA/cacert.pem
</VirtualHost>
```

Figura 16. Fichero "VH_web_sstt_443.conf"

Se mantienen las mismas entradas que en la figura 3, pero se cambiará:

- En la cláusula `<Directory>`, la ruta del directorio será `/var/www/sstt_443` para diferenciar las páginas web a las que se acceden con HTTP o con HTTPS.

Y se debe añadir las 4 últimas entradas que comienzan por SSL:

- `SSLEngine on`: habilita SSL para este host.
- `SSLCertificateFile`: certificado X.509 codificado en formato PEM del servidor.
- `SSLCertificateKeyFile`: clave privada codificado en formato PEM del servidor.
- `SSLCACertificateFile`: certificado CA codificado en formato PEM.

3.5. Apache SSL (autenticación del cliente)

En el apartado anterior se ha visto como crear los certificados del servicio web para autenticar el servidor, ahora completaremos ese caso para que **el servidor pueda autenticar al cliente**.

Configuración del Virtual Host en el servidor

Para la autenticación del cliente, se añaden dos entradas en el Virtual Host del puerto 443:

```
<VirtualHost *:443>
    ServerAdmin alumno@sstt8654.org
    ServerName www.sstt8654.org
    DocumentRoot /var/www/sstt_443
    <Directory /var/www/sstt_443>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>
    SSLEngine on
    SSLCertificateFile /home/alumno/demoCA/servercert.pem
    SSLCertificateKeyFile /home/alumno/demoCA/serverkey.pem
    SSLCACertificateFile /home/alumno/demoCA/cacert.pem
    SSLVerifyClient require
    SSLVerifyDepth 10
</VirtualHost>
```

Figura 17. Fichero final "VH_web_sstt_443.conf"

- SSLVerifyClient require: indica que el cliente debe presentar un certificado válido para su autenticación.
- SSLVerifyDepth 10: Profundidad máxima de los certificados de CA en la verificación del certificado del cliente.

Finalmente, la estructura de directorios tras haber configurado Apache y SSL para un virtual host en el puerto 80 y otro en el puerto 443 quedaría como vemos en la figura 18.

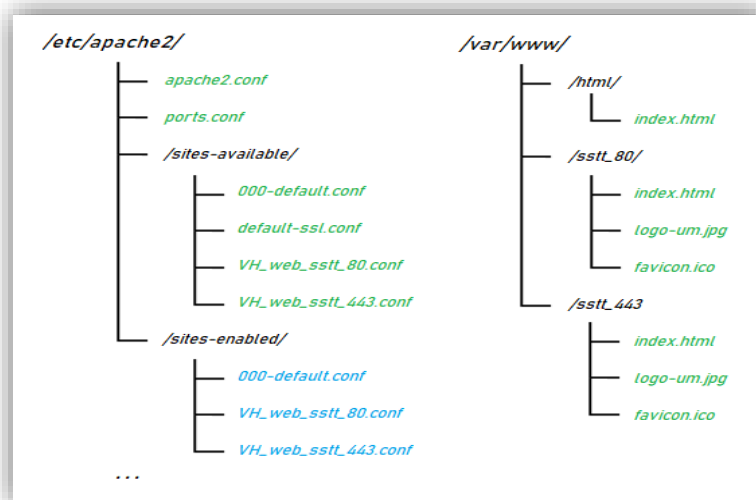


Figura 18. Estructura de directorios. En negro directorios, en verde ficheros y en azul enlaces simbólicos.

Generar certificado del cliente

Al igual que se hizo para “Generar certificado para el servicio web www.sstt8654.org del servidor” ejecutamos el mismo comando para emitir la solicitud de certificación para el cliente:

```
openssl req -new -nodes -newkey rsa:2048 -keyout clientkey.pem -out clientcsr.pem
```

Nos pedirá los datos de la identidad que vamos a certificar, mantenemos los mismos valores menos el Common Name que será “nicolas48658654S”:

- Country Name (2 letter code) [ES]:
- State or Province Name (full name) [Murcia]:
- Locality Name (eg, city) []:
- Organization Name (eg, company) [UMU]:
- Organizational Unit Name (eg, section) [sstt8654]:
- Common Name (eg, YOUR name) []: **nicolas48658654S**
- Email Address []:

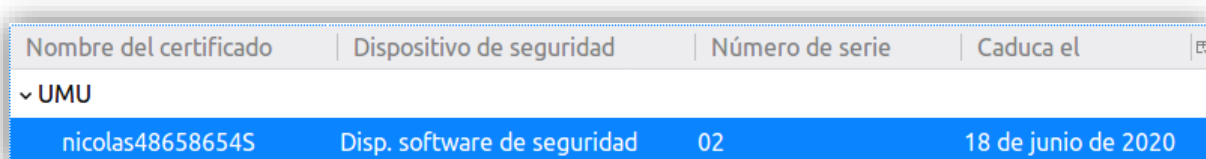
Dando como resultado la clave privada (clientkey.pem) para el servicio web que se va a almacena en el servidor host2 y la solicitud de certificación (clientcsr.pem). Esta solicitud debe ser firmada por la CA para generar el certificado X.509, con el siguiente comando:

```
openssl ca -keyfile private/cakey.pem -in clientcsr.pem -out clientcert.pem -days 400
```

Finalmente obtenemos el certificado X.509 (clientcert.pem) con su clave pública para autenticar al cliente.

Configurar el navegador cliente

Se deberá copiar el archivo clientcert.pfx en el host2, el cliente, para poder importarlo en el navegador. Para ello nos situamos en Preferencias/Privacidad&Seguridad/ y le damos a “Ver certificados...”. En la pestaña de “Sus certificados” importamos dicho archivo:



Nombre del certificado	Dispositivo de seguridad	Número de serie	Caduca el
▼ UMU			
nicolas48658654S	Disp. software de seguridad	02	18 de junio de 2020

Figura 19. Certificado de usuario importado en el navegador cliente.

De esta manera, y con todo configurado, realizamos la búsqueda en el servidor para la URL <https://www.sstt8654.org> en el navegador cliente. La primera vez nos aparecerá una ventana emergente, como la mostrada en la figura 20, que nos pedirá el certificado con el cual acceder a la página. Al aceptar la petición la página se habrá cargado correctamente indicando una conexión segura, lo podemos ver en la figura 21.

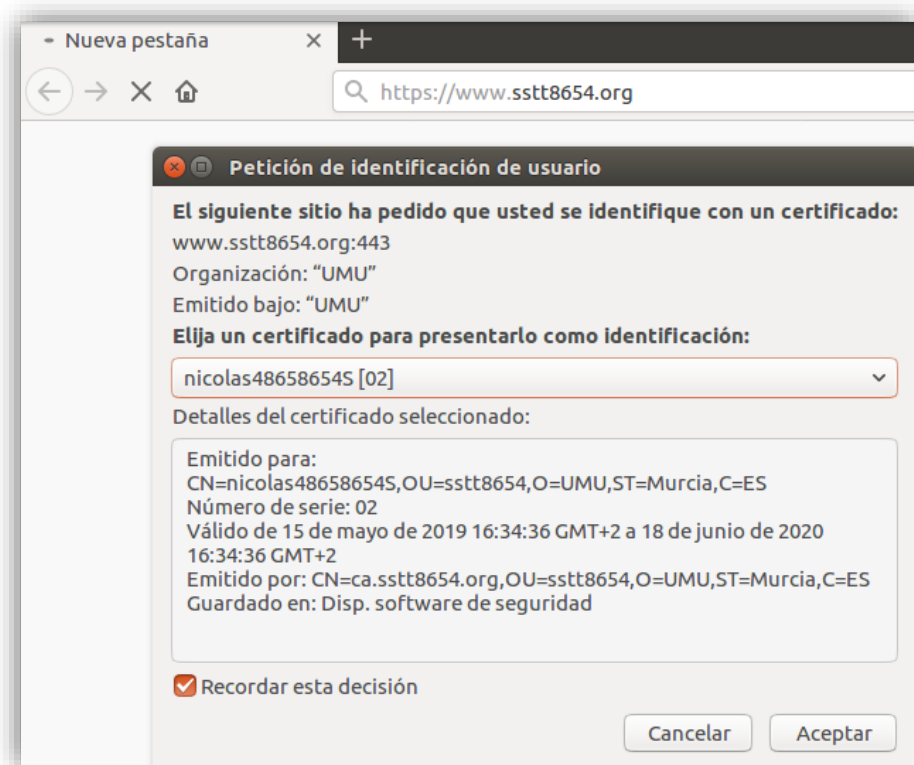


Figura 20. Ventana emergente.



Figura 21. Ejemplo de conexión segura.

3.6. IPsec

Tras la instalación del software Strongswan y su inicio con *ipsec start*, se va a configurar dicho servicio a través de los ficheros */etc/ipsec.conf* y */etc/ipsec.secrets* para ambos equipos, aunque con ciertas diferencias.

Configuración del servidor

En el primer fichero, *ipsec.conf*, se va a especificar la configuración de Strongswan asociada a la seguridad de IPsec. El fichero debe quedar como se ve en la figura 22.

- La primera sección, “conn %default”, establece los valores por defecto que heredan las demás secciones. Las líneas a destacar son:
 - *keyexchange=ikev2* que define el protocolo IKE que se encarga de la negociación y autenticación previa al intercambio de mensajes, así como de establecer el modo de operación (túnel).
 - *authby=pubkey* que indica autenticación mediante firmas de claves públicas.
- La segunda sección, “conn host-host”, establece la configuración específica entre el cliente y el servidor.
 - *Left* indica la IP del servidor.
 - *Leftcert* indica la ruta del certificado del servidor.
 - *Leftid* especifica la identidad del servidor.
 - *Right* indica la IP del cliente.
 - *Rightid* especifica la identidad del cliente.
 - *Type tunnel* para indicar el modo de operación.
 - *Auto=start* para ejecutar IKEv2 cuando se inicie ipsec.
 - *Esp=null-sha1* que indica que **no se cifrará, pero si autenticará por sha1** el tráfico entre los dos equipos.

```
config setup
conn %default
    ikelifetime=60m
    keylife=20m
    rekeymargin=3m
    keyingtries=1
    mobike=no
    keyexchange=ikev2
    authby=pubkey
conn host-host
    left=192.168.56.106
    leftcert=/etc/ipsec.d/certs/servercert.pem
    leftid="C=ES, ST=Murcia, O=UMU, OU=sstt8654, CN=www.sstt8654.org"
    right=192.168.56.105
    rightid="C=ES, ST=Murcia, O=UMU, OU=sstt8654, CN=nicolas48658654S"
    type=tunnel
    auto=start
    esp=null-sha1
```

Figura 22. Fichero de configuración *ipsec.conf* para el servidor.

En el segundo fichero, `ipsec.secrets`, va a contener el tipo de secreto RSA donde se especifica la ruta de la clave privada del servidor:

```
: RSA /etc/ipsec.d/private/serverkey.pem
```

Figura 23. Fichero de configuración `ipsec.secrets` para el servidor.

Configuración del cliente

La configuración del cliente es similar a la del servidor, solo deberemos invertir ciertos valores del fichero `ipsec.conf` y modificar la ruta de la clave privada del cliente en el fichero `ipsec.secrets`. La configuración de ambos ficheros se muestra en la figura 24 y 25, respectivamente.

```
config setup
conn %default
    ikelifetime=60m
    keylife=20m
    rekeymargin=3m
    keyingtries=1
    mobike=no
    keyexchange=ikev2
    authby=pubkey

conn host-host
    left=192.168.56.105
    leftcert=/etc/ipsec.d/certs/clientcert.pem
    leftid="C=ES, ST=Murcia, O=UMU, OU=sstt8654, CN=nicolas48658654S"
    right=192.168.56.106
    rightid="C=ES, ST=Murcia, O=UMU, OU=sstt8654, CN=www.sstt8654.org"
    type=tunnel
    auto=start
    esp=null-sha1
```

Figura 24. Fichero de configuración `ipsec.conf` para el cliente.

```
: RSA /etc/ipsec.d/certs/clientkey.pem
```

Figura 25. Fichero de configuración `ipsec.secrets` para el cliente.

4. Descripción de la implementación del servicio Web-SSTT HTTP

4.1. Descripción del protocolo HTTP

El protocolo HTTP, de sus siglas en inglés "Hypertext Transfer Protocol", es un protocolo de la **capa de aplicación** que trabaja sobre una **conexión TCP** en el **puerto 80** y es la base de cualquier intercambio de datos en la web.

HTTP es usado por los navegadores web (clientes) para la transferencia de información con los servidores web, que se basa en el **esquema petición/respuesta**. El protocolo define los métodos para la comunicación cliente – servidor que forman cada una de las peticiones, además de las respectivas respuestas y posibles errores. Algunos ejemplos de los métodos más conocidos de petición son: GET, POST, DELETE...; y también las respuestas más conocidas: 404 Not Found, 403 Forbidden, etc.

De esta manera, el servidor procesa las peticiones de los clientes y devuelve respuestas que pueden incluir información como páginas webs (HTML), imágenes, objetos, texto, etc. Por ejemplo, cuando un usuario común escribe la página que desea en su navegador, como Firefox o Google Chrome, eso se convierte en una petición GET (HTTP Request) a un servidor, el cual procesa y envía una respuesta (HTTP Response) correspondiente a la solicitud.

4.2. Características relevantes

Para el desarrollo de esta práctica se va a explicar dos características del protocolo que se van a implementar: el estado de una conexión y la persistencia.

HTTP es un **protocolo sin estado**, es decir, que no guarda información sobre el cliente por cada conexión con el servidor: si por ejemplo el cliente pide un *index.html*, y 10 segundos después vuelve a pedir la misma página, el servidor se la devuelve ya que desconoce si se la ha enviado antes o no. En el caso de que un servidor desee identificar a los clientes, HTTP hará uso de las cookies. Las cookies son un código que se inserta en una cabecera especial del mensaje HTTP para que el servidor pueda diferenciar a los distintos usuarios y responder dependiendo de estos.

Por otro lado, tenemos la persistencia, como se ha mencionado anteriormente TCP proporcionará una conexión fiable entre el cliente y el servidor, pero también **ofrece una conexión persistente**. Esto quiere decir que durante un periodo X de tiempo, se permitirá el intercambio de mensajes entre ambas partes sin que se cierre la conexión TCP, evitando así una sobrecarga de nuevas conexiones cuando, por ejemplo, se pidan distintos objetos de un mismo fichero base HTML.

Estas dos características mencionadas serán relevantes para el desarrollo de la práctica y serán explicadas con detalle posteriormente.

4.3. Tipos de mensaje

Como se ha mencionado anteriormente, HTTP sigue el esquema petición/respuesta, por lo que se definen mensajes de solicitud y mensajes de respuesta, cada uno con su respectivo formato.

4.3.1. HTTP Request

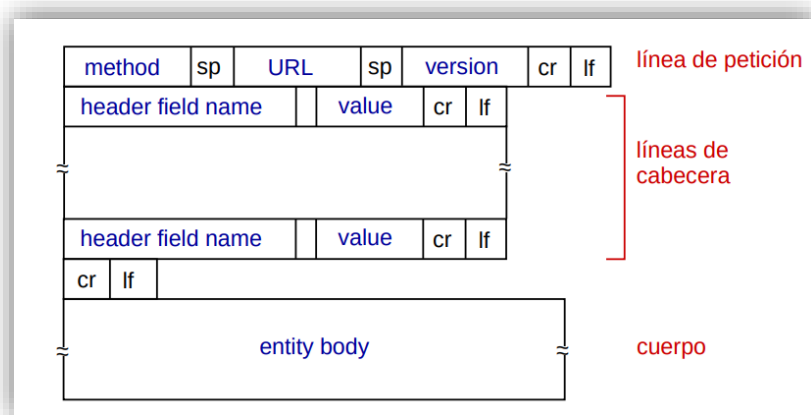


Figura 26. Formato del HTTP Request.

En la figura 26 se representa el formato de una solicitud (HTTP Request). Podemos observar que se compone tres componentes clave: la línea de petición, las cabeceras y el cuerpo donde se adjuntan los datos que queremos transmitir.

La **línea de petición** presenta tres campos:

- El método HTTP que define la operación que debe procesar el servidor.
- La URL o dirección del objeto que se desea.
- La versión del protocolo HTTP, siendo HTTP/1.1 la versión que trataremos en la práctica, aunque actualmente se encuentra la versión HTTP/2.

Las **líneas de cabecera** servirán para indicar al servidor características adicionales, ya sea de la conexión actual, del Host que realiza la petición, fecha y hora, tipos de datos, etc.

Y, por último, se separa con un retorno de carro y un salto de línea, **los datos** que se van a transmitir, en caso de que los haya.

El ejemplo mínimo de petición podría ser: "GET / HTTP/1.1"

4.3.2. Métodos de petición

Como hemos visto en el apartado anterior, los mensajes HTTP Request reservan un campo para el método de la petición. El protocolo define una gran cantidad de métodos que no se explicarán en este apartado pero que podemos encontrar a partir de la sección 9 del documento oficial del protocolo HTTP/1.1 ([RFC-2616](https://tools.ietf.org/html/rfc2616)).

En esta práctica **se va a trabajar con el método GET**, que permite realizar una solicitud para pedir un archivo al servidor (el que indiquemos en el campo URL).

4.3.3. HTTP Response

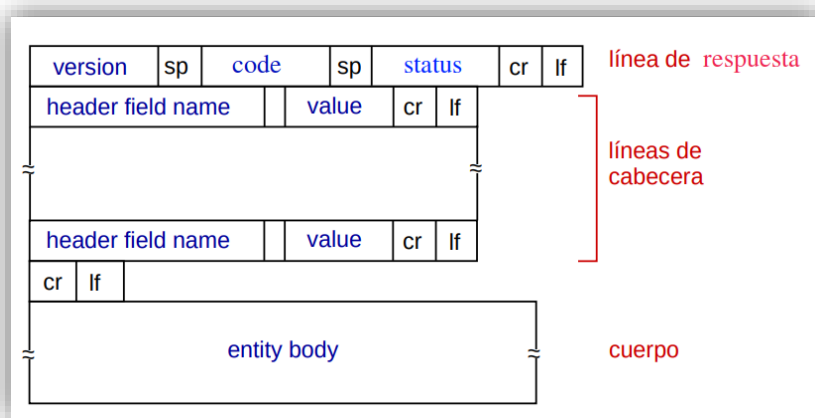


Figura 27. Formato del HTTP Response.

En la figura 27 vemos una estructura similar al formato de la petición, donde la línea de respuesta es la que cambia. En primer lugar, se indica la **versión HTTP** que están usando, en segundo lugar, un **código de estado** que indica si la petición se ha realizado correctamente o no y por último el **nombre del código** de estado que puede servir de breve descripción. Todo lo demás permanece igual.

El ejemplo mínimo de respuesta podría ser: "HTTP/1.1 200 OK"

4.3.4. Códigos de estado

Cuando el servidor responde a una petición, debe indicar en el mensaje el código de estado para **indicar** al cliente **qué ha pasado con la solicitud**. Existen multitud de códigos, según el mensaje que se quiere transmitir, los cuales se dividen en los siguientes formatos:

- **1xx:** Respuestas informativas.
- **2xx:** Peticiones correctas.

- **3xx:** Redirecciones.
- **4xx:** Errores del cliente.
- **5xx:** Errores del servidor.

Para esta práctica se han empleado los siguientes códigos de estado:

- **200 OK:** indica peticiones correctas.
- **400 Bad Request:** indica que el servidor no puede procesar la solicitud porque es errónea.
- **403 Forbidden:** el servidor rechaza procesar la solicitud.
- **404 Not Found:** recurso no encontrado.
- **405 Method Not Allowed:** el servidor no soporta el método de la solicitud.
- **415 Unsupported Media Type:** el servidor no soporta el tipo de recurso solicitado.
- **505 HTTP Version Not Supported:** el servidor no soporta la versión de la solicitud.

Para más información podemos consultar la sección 10 del documento oficial del protocolo HTTP/1.1 ([RFC-2616](https://tools.ietf.org/html/rfc2616)).

4.4. Implementación

En los siguientes apartados se va a explicar cada una de las funcionalidades que se han ido desarrollando en la plantilla en C proporcionada, indicando el código relevante e imágenes en caso de que sea necesario para facilitar la explicación.

4.4.1. Parámetros de entrada y ejecución

El formato de ejecución del servidor está constituido por el fichero ejecutable, el puerto donde estará activo y la carpeta de trabajo del servidor:

```
./web_sstt <PUERTO> <DIRECTORIO>
```

- **PUERTO:** Podremos ejecutar el programa en un puerto comprendido entre el 1024 y el 65536, ya que del 0 al 1023 están reservados para el sistema.
- **DIRECTORIO:** en este directorio se encontrarán los archivos con los que trabajará el servidor, como el fichero webserver.log para registrar comentarios sobre la ejecución, y las páginas web, imágenes, etc, como el fichero HTML base index.html y el logo-um.jpg.

Para ejecutar el servicio HTTP, nos situamos en el servidor donde deseamos que se ejecute y escribimos en el bash la siguiente línea, indicando como directorio de trabajo el directorio actual:

```
./web_sstt 8080 .
```

Para evitar problemas se realizan las siguientes comprobaciones de errores a la hora de ejecutar el programa:

Comprobación del número de argumentos para evitar equivocación:

```
alumno@server:~$ ./web_sstt
USO:  ./web_sstt  <PUERTO>  <DIRECTORIO>
```

Comprobación del valor del puerto, solo mayor que 1024:

```
alumno@server:~$ ./web_sstt 80 .
ERROR: "80" no es un puerto válido
```

Comprobación de la existencia del directorio:

```
alumno@server:~$ ./web_sstt 8080 no_existe
ERROR: el directorio "no_existe" no existe
```

Comprobación de los permisos del directorio:

```
alumno@server:~$ ./web_sstt 8080 dir_server/
ERROR: el directorio "dir_server/" no tiene permisos de escritura
```

```
alumno@server:~$ ./web_sstt 8080 dir_server/
ERROR: el directorio "dir_server/" no tiene permisos de lectura
```

4.4.2. Gestión de clientes

El programa es capaz de gestionar todas las peticiones web de distintos clientes. La comunicación comienza con el establecimiento de la conexión mediante el uso de sockets TCP, donde ambas partes reservan los recursos necesarios. Un socket es un canal de comunicación entre dos procesos de distinta máquina por el que se transmiten mensajes, y se define mediante un puerto y una dirección IP.

Con el socket abierto y escuchando en el puerto indicado, el servidor se queda bloqueado en la función *accept()* hasta que un cliente inicie la comunicación TCP.

Cuando se desbloquea esta función, el servidor crea un hilo con la función *fork()* para que sea el hijo el que se ocupe de la comunicación y el padre pueda continuar gestionando el socket para las peticiones entrantes de nuevos clientes.

Por su lado, el hijo cierra el socket anterior y se dedicará a procesar las peticiones del nuevo cliente en un nuevo socket.

4.4.3. Procesamiento de la petición GET

El proceso hijo procesará las peticiones del cliente mediante la función *process_web_request()*. Esta función se puede dividir en dos partes: la primera se encargará de analizar el mensaje HTTP para comprobar que no se hayan producido errores, la segunda se encargará de gestionar el recurso solicitado y la tercera es el envío de este recurso o de un mensaje de error en caso de haberlo.

En los siguientes apartados se profundizará en la implementación de las distintas partes de esta función.

4.4.3.1. Comprobaciones de la línea de petición

Se comienza leyendo la petición HTTP del socket y separando cada línea de la petición por un "@" para facilitar su análisis posteriormente.

Ahora se van a tratar los métodos de petición. Como solo se soporta GET vamos a comprobar que sea este método y rechazaremos cualquier otro. Para ello se llama a la función *isGetRequest()* pasando como parámetro el *buffer* que contiene la petición HTTP, un *path_file* para almacenar la ruta del recurso solicitado y el *descriptorFichero* para enviar los mensajes de error si los hubiere.

❖ isGetRequest():

Esta función se centra en detectar errores en la primera línea de la petición, es decir, en la línea que indica el tipo de método. Las comprobaciones que realiza son las siguientes:

1. Que la línea no contenga menos de 14 caracteres, ya que la petición mínima será "GET / HTTP/1.1" y también que no comience por un espacio.

Ahora se leerá la petición campo por campo con la función *strtok()*. Se guarda el campo del método para el final, pues primero debemos evitar otros errores.

2. Se comprueba que el segundo campo comience por `"/"`, lo que nos da indicios de una ruta, y se guarda en la variable `path_file` mencionada antes.
3. Lee el tercer campo (la versión HTTP) y comprueba que no hayan más de tres campos volviendo a llamar a `strtok()` y confirma que sea NULL.

En caso de no cumplir las condiciones se manda un HTTP Response del tipo 400 BAD REQUEST y se sale de la función. Si la línea comienza correctamente, tiene los caracteres mínimos y el número de campos es correcto. Solo queda comprobar que el primer campo es un `"GET"` y que el tercer campo es un `"HTTP/1.1"`.

4. Le damos prioridad al método de petición, en caso de no ser GET, se enviará un HTTP Response del tipo 405 METHOD NOT ALLOWED y se sale de la función.
5. Por último, si la versión HTTP no es `"HTTP/1.1"` contestamos con un HTTP Response del tipo 505 HTTP VERSION NOT SUPPORTED y se sale de la función.

Si la línea cumple con las condiciones la función termina retornando 1 (True).

4.4.3.2. *Comprobaciones de las cabeceras*

❖ `checkHeaders()`:

Esta función se llamará con el buffer que contiene la petición HTTP y un entero donde se almacenará el valor de la cookie, `cookie_counter`. Para empezar, salta la primera línea de petición y nos situamos en la primera cabecera. Mantendremos un array llamado `headers` donde se guardarán todas las cabeceras de la petición.

Debemos puntualizar que el formato de las cabeceras es:

`"Nombre: valor"`.

Por lo que vamos a comprobar que cada cabecera cumpla con este formato. Se hacen las siguientes operaciones:

1. Obtenemos el primer campo de la cabecera, que debe ser `"Nombre:"`.
2. Comprobamos que su último carácter sean dos puntos `":"` y que después haya un espacio.

Si no cumplen estas condiciones se saldrá de la función y se mandará un HTTP Response del tipo 400 BAD REQUEST, indicando que el formato de las cabeceras no son correctas. Por otro lado, si no se produce un error, se pasa a comprobar el tipo de cabecera, realizando una operación si es conveniente. El ejemplo más claro es con la cabecera "Cookie" del cual se obtiene su valor para tratarlo en funciones posteriores.

4.4.3.3. *Accesos ilegales*

Mediante la siguiente función vamos a descartar aquellas peticiones a recursos que se encuentren en directorios superiores de la jerarquía.

❖ `IsIllegalAccess()`:

Para comprobar si accede a un directorio superior, debemos fijarnos en si la ruta contiene dos puntos seguidos, lo que indica que sube un nivel en la jerarquía. Pero eso no es todo, ya que, si por ejemplo desciende dos niveles y solo asciende uno, eso sería correcto pues se mantiene por debajo del directorio raíz donde trabaja el servidor. Por ejemplo, si establecemos que el directorio de trabajo es `/home/alumno/`, y la del recurso pedido es `/directorio/./logo-um.jpg`, eso sería correcto:

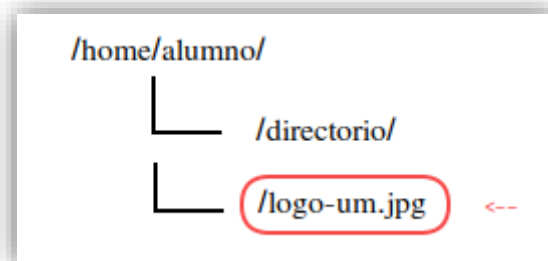


Figura 28. Ejemplo de directorio de trabajo.

De esta manera, la forma de resolver este problema es contar cuantas veces asciende (restamos 1) y cuantas desciende (sumamos 1). Si al terminar el valor es menor o igual a -1 será un acceso ilegal.

En caso de ser un acceso ilegal, mandaremos un HTTP Response del tipo 403 FORBIDDEN y se sale de la función. Si todo es correcto emplearemos la función `setPathFile()` para concatenar la el directorio de trabajo con la ruta del recurso pedido. Además se hace uso de la función `realpath()` para obtener la ruta absoluta del directorio de trabajo.

4.4.3.4. Evaluación del recurso solicitado

A continuación, se comprueba la extensión del archivo y se establece la cabecera “Content-Type” que indica el tipo de archivo que se va a devolver al cliente:

❖ `setContentType()`

A esta función le pasamos el array donde se va a almacenar la cabecera y la ruta del fichero pedido. Comienza buscando el nombre del fichero de la ruta y almacenándolo en *name_file* para realizar las siguientes comprobaciones:

1. Se comprueba si el fichero tiene extensión (solo aceptamos ficheros con extensión).
2. Si la tiene, comprobamos si se encuentra en la estructura *extensions*. Si está aprovechamos para formar la cabecera “Content-Type” del archivo.
3. En otro caso se envía un HTTP Response del tipo 415 UNSUPPORTED MEDIA TYPE y se sale de la función. Podemos ver en la siguiente traza un ejemplo y cómo se ve en el navegador:

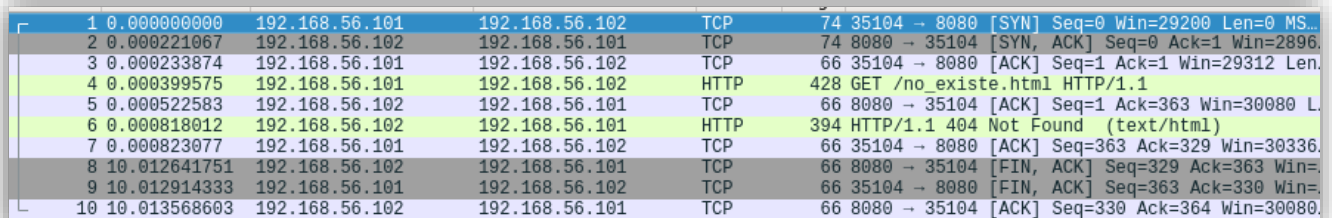
17	29.655766255	192.168.56.101	192.168.56.102	TCP	74	35110 → 8080	[SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK...
18	29.656003908	192.168.56.102	192.168.56.101	TCP	74	8080 → 35110	[SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS...
19	29.656018859	192.168.56.101	192.168.56.102	TCP	66	35110 → 8080	[ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=18...
20	29.656160229	192.168.56.101	192.168.56.102	HTTP	430	GET /fichero.hola HTTP/1.1	
21	29.656284498	192.168.56.102	192.168.56.101	TCP	66	8080 → 35110	[ACK] Seq=1 Ack=365 Win=30080 Len=0 TSval=...
22	29.656634636	192.168.56.102	192.168.56.101	HTTP	427	HTTP/1.1 415 Unsupported Media Type (text/html)	
23	29.656640553	192.168.56.101	192.168.56.102	TCP	66	35110 → 8080	[ACK] Seq=365 Ack=362 Win=30336 Len=0 TSva...
30	39.668050329	192.168.56.102	192.168.56.101	TCP	66	8080 → 35110	[FIN, ACK] Seq=362 Ack=365 Win=30080 Len=0...
31	39.668156749	192.168.56.101	192.168.56.102	TCP	66	35110 → 8080	[FIN, ACK] Seq=365 Ack=363 Win=30336 Len=0...
32	39.668402720	192.168.56.102	192.168.56.101	TCP	66	8080 → 35110	[ACK] Seq=363 Ack=366 Win=30080 Len=0 TSva...

Figura 29. Traza ejemplo de respuesta 415.



Figura 30. Vista en el navegador del error 415.

Después se lleva a cabo la comprobación de la existencia del archivo pedido, que en caso contrario se envía un HTTP Response del tipo 404 NOT FOUND y se sale de la función. De la misma forma, podemos ver el resultado con Wireshark y en el navegador:



1	0.000000000	192.168.56.101	192.168.56.102	TCP	74	35104 → 8080	[SYN] Seq=0 Win=29200 Len=0 MS...
2	0.000221067	192.168.56.102	192.168.56.101	TCP	74	8080 → 35104	[SYN, ACK] Seq=0 Ack=1 Win=2896...
3	0.000233874	192.168.56.101	192.168.56.102	TCP	66	35104 → 8080	[ACK] Seq=1 Ack=1 Win=29312 Len...
4	0.000399575	192.168.56.101	192.168.56.102	HTTP	428	GET /no_existe.html HTTP/1.1	
5	0.000522583	192.168.56.102	192.168.56.101	TCP	66	8080 → 35104	[ACK] Seq=1 Ack=363 Win=30080 L...
6	0.000818012	192.168.56.102	192.168.56.101	HTTP	394	HTTP/1.1 404 Not Found (text/html)	
7	0.000823077	192.168.56.101	192.168.56.102	TCP	66	35104 → 8080	[ACK] Seq=363 Ack=329 Win=30336...
8	10.012641751	192.168.56.102	192.168.56.101	TCP	66	8080 → 35104	[FIN, ACK] Seq=329 Ack=363 Win=...
9	10.012914333	192.168.56.101	192.168.56.102	TCP	66	35104 → 8080	[FIN, ACK] Seq=363 Ack=330 Win=...
10	10.013568603	192.168.56.102	192.168.56.101	TCP	66	8080 → 35104	[ACK] Seq=330 Ack=364 Win=30080...

Figura 31. Traza ejemplo de respuesta 404.



Figura 32. Vista en el navegador del error 404.

4.4.3.5. Gestión de Cookies

Para la gestión de las cookies debemos tener en cuenta que se mandarían siempre que recibamos una petición GET correcta, nunca cuando informemos de un error. Por ello, hasta que no confirmamos que la petición está bien formada no procedemos a su gestión.

En el apartado anterior 2.2.2. de comprobación de las cabeceras, se ha explicado que si la petición HTTP Request contiene una cabecera "Cookie" se recoge el valor de esta para tratarlo posteriormente, en caso de no contenerla significará que es la primera petición al servidor y su valor es 0.

Ahora se va a comprobar si la cookie supera las 10 peticiones para enviar un HTTP Response del tipo 403 FORBIDDEN, en cuyo caso podemos ver con Wireshark el envío de estos mensajes y la respuesta en el navegador:

64	11.818748648	192.168.56.101	192.168.56.102	HTTP	408 GET /logo-um.jpg HTTP/1.1
65	11.819128005	192.168.56.102	192.168.56.101	HTTP	417 HTTP/1.1 403 Forbidden (text/html)
66	11.819143336	192.168.56.101	192.168.56.102	TCP	66 35080 → 8080 [ACK] Seq=3842 Ack=43567 Wj

Figura 33. Traza ejemplo de respuesta 403.



Figura 34. Vista en el navegador del error 403.

Si, además, analizamos la traza anterior del Wireshark observamos como la cookie efectivamente tiene el valor 10:

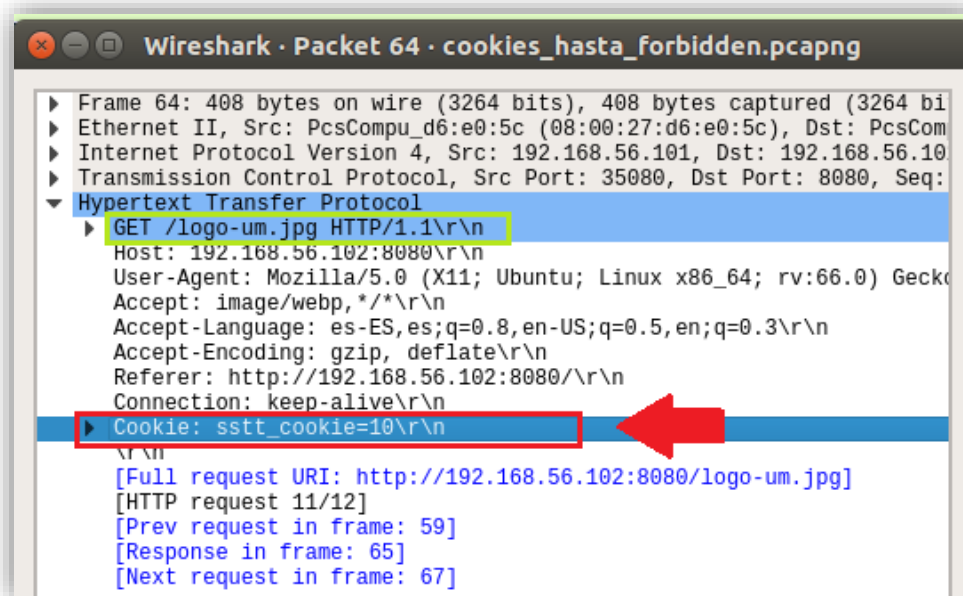


Figura 35. Ejemplo 1 del bloqueo con cookie igual a 10..

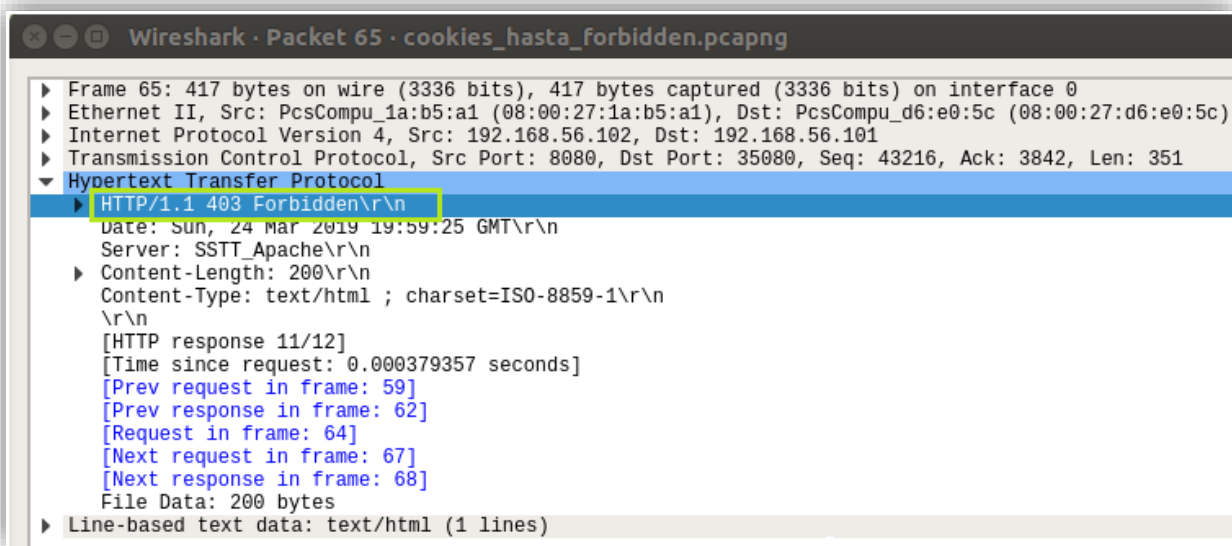


Figura 36. Ejemplo 2 del bloqueo con cookie igual a 10.

Por otro lado, si la cookie no supera las 10 peticiones, se incluirá la cabecera “Set-Cookie” con el valor incrementado en una unidad y con un tiempo de expiración de 120 segundos. Como podemos ver en las siguientes imágenes, la primera muestra una petición GET con valor de cookie 3, y en la segunda como el servidor le devuelve el valor de cookie 4:



Figura 37. Ejemplo 1 del aumento del valor de la cookie.

```
24 4.314633585 192.168.56.101 192.168.56.102 HTTP 437 GET / HTTP/1.1
25 4.315001976 192.168.56.102 192.168.56.101 TCP 307 8080 → 35080 [PSH, ACK] Seq=1364
26 4.315014527 192.168.56.101 192.168.56.102 TCP 66 35080 → 8080 [ACK] Seq=1364
27 4.315188326 192.168.56.102 192.168.56.101 HTTP 516 HTTP/1.1 200 OK (text/html)

> Frame 27: 516 bytes on wire (4128 bits), 516 bytes captured (4128 bits) on interface 0
> Ethernet II, Src: PcsCompu_1a:b5:a1 (08:00:27:1a:b5:a1), Dst: PcsCompu_d6:e0:5c (08:00:27:d6:e0:5c)
> Internet Protocol Version 4, Src: 192.168.56.102, Dst: 192.168.56.101
> Transmission Control Protocol, Src Port: 8080, Dst Port: 35080, Seq: 11292, Ack: 1364, Len: 450
> [2 Reassembled TCP Segments (691 bytes): #25(241), #27(450)]
v Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Date: Sun, 24 Mar 2019 19:59:17 GMT\r\n
    Server: SSTT_Apache\r\n
    Set-Cookie: sstt_cookie=4; Max-Age=120\r\n
  > Content-Length: 450\r\n
    Content-Type: text/html; charset=ISO-8859-1\r\n
    Keep-Alive: timeout=10, max=1000\r\n
    Connection: Keep-Alive\r\n
  \r\n
  [HTTP response 4/12]
  [Time since request: 0.000554741 seconds]
  [Prev request in frame: 19]
  [Prev response in frame: 22]
  [Request in frame: 24]
  [Next request in frame: 29]
  [Next response in frame: 34]
  File Data: 450 bytes
> Line-based text data: text/html (17 lines)
```

Figura 38. Ejemplo 2 del aumento del valor de la cookie.

4.4.3.6. Envío de la respuesta

En este punto solo queda construir la respuesta y mandarla. Para empezar, abrimos el recurso pedido y establecemos la cabecera “Content-Length” que indica el tamaño del fichero de datos sin contar la cabecera HTTP; también la cabecera “DATE” con la función setDate().

Finalmente, concatenamos todo en el array response y la cabecera HTTP quedará formada. La enviamos a través del socket indicando su tamaño y seguidamente leemos los bytes del recurso.

Se ha establecido un tamaño de buffer de 8096 bytes, por lo que los ficheros que superen este tamaño serán enviados por paquetes de 8096 hasta haber mandado todo. Al terminar cerramos con close() el fichero abierto.

En el caso de los mensajes de error, se ha aprovechado la función debug() de la plantilla para el control de errores. Mediante un switch se detecta el tipo de operación a realizar, donde los códigos 400, 403, 404, 405, 415 y 505 hacen referencia a los códigos de estado mencionados en el apartado 1.2.4. En estos casos se forma la respectiva línea del tipo de respuesta, se añaden las cabeceras “Date”, “Server”, “Content-Type” y “Content-Length” y se adjunta un mensaje HTML para que aparezca el error en pantalla.

4.4.4. Persistencia

Una conexión no persistente, establecería por cada solicitud del cliente al servidor una negociación TCP-SYN, y finalizaría con TCP-FIN. Por lo que si pedimos el fichero index.html al servidor, se estarían creando tres conexiones: una por el index.html, otra por el logo-um.jpg y otra por el favicon.ico. Podemos verlo en el siguiente ejemplo:

Time	Source	Destination	Protocol	Length	Info
1 0.000000000	192.168.56.102	192.168.56.101	TCP	66	8080 → 35124 [FIN, ACK] Seq=1 Ack=1 Win=227 Len=0 TSval=1677096 TSsec=0
2 0.000193536	192.168.56.101	192.168.56.102	TCP	66	35124 → 8080 [FIN, ACK] Seq=1 Ack=2 Win=386 Len=0 TSval=1943017 TSsec=0
3 0.000816166	192.168.56.102	192.168.56.101	TCP	66	8080 → 35124 [ACK] Seq=2 Ack=2 Win=227 Len=0 TSval=1677096 TSsec=0
4 34.230439173	192.168.56.101	192.168.56.102	TCP	74	35126 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1955550 TSsec=0
5 34.230827046	192.168.56.102	192.168.56.101	TCP	74	8080 → 35126 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1955550 TSsec=0
6 34.230853975	192.168.56.101	192.168.56.102	TCP	66	35126 → 8080 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1951575 TSsec=0
7 37.534729046	192.168.56.101	192.168.56.102	TCP	92	35126 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=26 TSval=1952401 TSsec=0
8 37.535109665	192.168.56.102	192.168.56.101	TCP	66	8080 → 35126 [ACK] Seq=1 Ack=27 Win=29056 Len=0 TSval=168880 TSsec=0
9 37.535891913	192.168.56.102	192.168.56.101	TCP	307	8080 → 35126 [PSH, ACK] Seq=1 Ack=27 Win=29056 Len=241 TSval=16880 TSsec=0
10 37.535909017	192.168.56.101	192.168.56.102	TCP	66	35126 → 8080 [ACK] Seq=27 Ack=242 Win=30336 Len=0 TSval=192401 TSsec=0
11 37.536228632	192.168.56.102	192.168.56.101	HTTP	516	HTTP/1.1 200 OK (text/html)
12 37.536240155	192.168.56.101	192.168.56.102	TCP	66	35126 → 8080 [ACK] Seq=27 Ack=692 Win=31360 Len=0 TSval=1952401 TSsec=0
13 47.547860753	192.168.56.102	192.168.56.101	TCP	66	8080 → 35126 [FIN, ACK] Seq=692 Ack=27 Win=29056 Len=0 TSval=168880 TSsec=0
14 47.548075714	192.168.56.101	192.168.56.102	TCP	66	35126 → 8080 [FIN, ACK] Seq=27 Ack=693 Win=31360 Len=0 TSval=195401 TSsec=0
15 47.548609158	192.168.56.102	192.168.56.101	TCP	66	8080 → 35126 [ACK] Seq=693 Ack=28 Win=29056 Len=0 TSval=1688983 TSsec=0
16 50.131663213	192.168.56.101	192.168.56.102	TCP	74	35128 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1955550 TSsec=0
17 50.131944676	192.168.56.102	192.168.56.101	TCP	74	8080 → 35128 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1955550 TSsec=0
18 50.131966618	192.168.56.101	192.168.56.102	TCP	66	35128 → 8080 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1955550 TSsec=0
19 52.811022774	192.168.56.101	192.168.56.102	TCP	93	35128 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=27 TSval=1956221 TSsec=0
20 52.811638774	192.168.56.102	192.168.56.101	TCP	66	8080 → 35128 [ACK] Seq=1 Ack=28 Win=29056 Len=0 TSval=1690299 TSsec=0
21 52.812661720	192.168.56.102	192.168.56.101	TCP	308	8080 → 35128 [PSH, ACK] Seq=1 Ack=28 Win=29056 Len=242 TSval=1690299 TSsec=0
22 52.812688257	192.168.56.101	192.168.56.102	TCP	66	35128 → 8080 [ACK] Seq=28 Ack=243 Win=30336 Len=0 TSval=1956221 TSsec=0
23 52.813714358	192.168.56.102	192.168.56.101	TCP	7306	8080 → 35128 [ACK] Seq=243 Ack=28 Win=29056 Len=720 TSval=1690299 TSsec=0
24 52.813754568	192.168.56.101	192.168.56.102	TCP	66	35128 → 8080 [ACK] Seq=28 Ack=7483 Win=44800 Len=0 TSval=1956221 TSsec=0
25 52.813795784	192.168.56.102	192.168.56.101	TCP	922	8080 → 35128 [PSH, ACK] Seq=7483 Ack=28 Win=29056 Len=856 TSval=1956221 TSsec=0
26 52.813804554	192.168.56.101	192.168.56.102	TCP	66	35128 → 8080 [ACK] Seq=28 Ack=8339 Win=47744 Len=0 TSval=1956221 TSsec=0
27 52.813881969	192.168.56.102	192.168.56.101	TCP	1514	8080 → 35128 [ACK] Seq=8339 Ack=28 Win=29056 Len=148 TSval=1690299 TSsec=0
28 52.813891044	192.168.56.101	192.168.56.102	TCP	66	35128 → 8080 [ACK] Seq=28 Ack=9787 Win=50560 Len=0 TSval=1956221 TSsec=0
29 52.814681520	192.168.56.102	192.168.56.101	HTTP	80	HTTP/1.1 200 OK (JPEG JFIF image)
30 52.814696210	192.168.56.101	192.168.56.102	TCP	66	35128 → 8080 [ACK] Seq=28 Ack=9801 Win=50560 Len=0 TSval=1956221 TSsec=0
31 62.816123641	192.168.56.102	192.168.56.101	TCP	66	8080 → 35128 [FIN, ACK] Seq=9801 Ack=28 Win=29056 Len=0 TSval=1690299 TSsec=0
32 62.816189452	192.168.56.101	192.168.56.102	TCP	66	35128 → 8080 [FIN, ACK] Seq=28 Ack=9802 Win=50560 Len=0 TSval=1956221 TSsec=0
33 62.816330898	192.168.56.102	192.168.56.101	TCP	66	8080 → 35128 [ACK] Seq=9802 Ack=29 Win=29056 Len=0 TSval=1692800 TSsec=0
41 69.674287201	192.168.56.101	192.168.56.102	TCP	74	35130 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1955550 TSsec=0
42 69.674688926	192.168.56.102	192.168.56.101	TCP	74	8080 → 35130 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1955550 TSsec=0
43 69.674717460	192.168.56.101	192.168.56.102	TCP	66	35130 → 8080 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=19560436 TSsec=0
45 71.733866272	192.168.56.101	192.168.56.102	TCP	93	35130 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=27 TSval=19560436 TSsec=0
46 71.734482857	192.168.56.102	192.168.56.101	TCP	66	8080 → 35130 [ACK] Seq=1 Ack=28 Win=29056 Len=0 TSval=1695021 TSsec=0
47 71.735703095	192.168.56.102	192.168.56.101	TCP	307	8080 → 35130 [PSH, ACK] Seq=1 Ack=28 Win=29056 Len=241 TSval=1695021 TSsec=0
48 71.735731235	192.168.56.101	192.168.56.102	TCP	66	35130 → 8080 [ACK] Seq=28 Ack=242 Win=30336 Len=0 TSval=1960051 TSsec=0
49 71.736552171	192.168.56.102	192.168.56.101	HTTP	384	HTTP/1.1 200 OK (image/ico)
50 71.736575341	192.168.56.101	192.168.56.102	TCP	66	35130 → 8080 [ACK] Seq=28 Ack=560 Win=31360 Len=0 TSval=1960951 TSsec=0
51 81.747802691	192.168.56.102	192.168.56.101	TCP	66	8080 → 35130 [FIN, ACK] Seq=560 Ack=28 Win=29056 Len=0 TSval=1697001 TSsec=0
52 81.747956456	192.168.56.101	192.168.56.102	TCP	66	35130 → 8080 [FIN, ACK] Seq=28 Ack=561 Win=31360 Len=0 TSval=1963001 TSsec=0
53 81.748356315	192.168.56.102	192.168.56.101	TCP	66	8080 → 35130 [ACK] Seq=561 Ack=29 Win=29056 Len=0 TSval=1697533 TSsec=0

Figura 39. Ejemplo de traza no persistente.

Gracias a la persistencia evitamos crear tantas conexiones, logrando mantener una sesión por cliente con un timeout de 10 segundos. Cuya ejecución pasaría a ser la siguiente, viéndose claramente cómo se reduce la cantidad de mensajes:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.56.101	192.168.56.102	TCP	74	35140 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=19930
2	0.000291804	192.168.56.102	192.168.56.101	TCP	74	8080 → 35140 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1
3	0.000326534	192.168.56.101	192.168.56.102	TCP	66	35140 → 8080 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=199308 TSsecr=172716
4	0.030960442	192.168.56.101	192.168.56.102	HTTP	437	GET / HTTP/1.1
5	0.031184893	192.168.56.102	192.168.56.101	TCP	66	8080 → 35140 [ACK] Seq=1 Ack=372 Win=30080 Len=0 TSval=1727271 TSsecr=1993
6	0.031608841	192.168.56.102	192.168.56.101	TCP	307	8080 → 35140 [PSH, ACK] Seq=1 Ack=372 Win=30080 Len=241 TSval=1727171 TSsecr=17
7	0.031614545	192.168.56.101	192.168.56.102	TCP	66	35140 → 8080 [ACK] Seq=372 Ack=242 Win=30336 Len=0 TSval=1993092 TSsecr=17
8	0.031828926	192.168.56.102	192.168.56.101	HTTP	516	HTTP/1.1 200 OK (text/html)
9	0.031841690	192.168.56.101	192.168.56.102	TCP	66	35140 → 8080 [ACK] Seq=372 Ack=692 Win=31360 Len=0 TSval=1993092 TSsecr=17
10	0.156991846	192.168.56.101	192.168.56.102	HTTP	407	GET /logo-um.jpg HTTP/1.1
11	0.157396710	192.168.56.102	192.168.56.101	TCP	308	8080 → 35140 [PSH, ACK] Seq=692 Ack=713 Win=31104 Len=242 TSval=1727202 TSsecr=17
12	0.157418006	192.168.56.101	192.168.56.102	TCP	66	35140 → 8080 [ACK] Seq=713 Ack=934 Win=32512 Len=0 TSval=1993124 TSsecr=17
13	0.157465001	192.168.56.102	192.168.56.101	TCP	4410	8080 → 35140 [ACK] Seq=934 Ack=713 Win=31104 Len=4344 TSval=1727202 TSsecr=17
14	0.157469109	192.168.56.101	192.168.56.102	TCP	66	35140 → 8080 [ACK] Seq=713 Ack=5278 Win=41216 Len=0 TSval=1993124 TSsecr=17
15	0.157636497	192.168.56.102	192.168.56.101	TCP	4410	8080 → 35140 [ACK] Seq=5278 Ack=713 Win=31104 Len=4344 TSval=1727202 TSsecr=17
16	0.157640755	192.168.56.101	192.168.56.102	TCP	66	35140 → 8080 [ACK] Seq=713 Ack=9622 Win=49792 Len=0 TSval=1993124 TSsecr=17
17	0.157649037	192.168.56.102	192.168.56.101	HTTP	936	HTTP/1.1 200 OK (JPEG JFIF image)
18	0.157650511	192.168.56.101	192.168.56.102	TCP	66	35140 → 8080 [ACK] Seq=713 Ack=10492 Win=52736 Len=0 TSval=1993124 TSsecr=17
19	10.157266350	192.168.56.101	192.168.56.102	TCP	66	[TCP Keep-Alive] 35140 → 8080 [ACK] Seq=712 Ack=10492 Win=52736 Len=0 TSval=1993124 TSsecr=17
20	10.157953847	192.168.56.102	192.168.56.101	TCP	66	[TCP Keep-Alive ACK] 8080 → 35140 [ACK] Seq=10492 Ack=713 Win=31104 Len=0 TSval=1729704 TSsecr=17
21	10.163018196	192.168.56.102	192.168.56.101	TCP	66	8080 → 35140 [FIN, ACK] Seq=10492 Ack=713 Win=31104 Len=0 TSval=1729704 TSsecr=17
22	10.163246772	192.168.56.101	192.168.56.102	TCP	66	35140 → 8080 [FIN, ACK] Seq=713 Ack=10493 Win=52736 Len=0 TSval=1995625 TSsecr=17
23	10.163828153	192.168.56.102	192.168.56.101	TCP	66	8080 → 35140 [ACK] Seq=10493 Ack=714 Win=31104 Len=0 TSval=1729704 TSsecr=17

Figura 40. Ejemplo de traza persistente.

Para implementar esta persistencia, se ha hecho uso de la función *select()* que permite monitorizar los sockets. Esta función se bloquea hasta que ocurre un cambio de estado en un socket, en este caso cuando hay nuevos caracteres para leer.

Al desbloquearse, permite al hilo leer nuevos datos del socket y procesar la petición. Pero también puede desbloquearse porque el tiempo de espera se ha terminado, en ese caso si se cierra la conexión TCP y el hilo terminará su ejecución.

5. Estado final del servidor – Puertos activos

Antes de mostrar las trazas representativas a cada servicio desplegado, se ha hecho uso del comando **nmap** en el servidor para mostrar que todos los procesos se encuentran en funcionamiento y escuchando en su respectivo puerto. En la figura 41, se puede observar el servicio de correo activo con SMTP escuchando en el puerto 25 y POP3 en el puerto 110; DNS en el 53; Apache se encuentra activo tanto en el puerto 80 para HTTP como en el puerto 443 para HTTPS; y además el servicio web programado WEB-SSTT HTTP en el puerto 8080.

```
alumno@server:~$ sudo nmap -sT -O localhost

Starting Nmap 7.01 ( https://nmap.org ) at 2019-05-19 10:48 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00058s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 993 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
110/tcp   open  pop3
443/tcp   open  https
8080/tcp  open  http-proxy
Device type: general purpose
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux_kernel:3
OS details: Linux 3.8 - 3.19
Network Distance: 0 hops
```

Figura 42. Puertos activos en el servidor con nmap.

O con más detalle podemos verlo con el comando **netstat -anp** en la figura 43.

Conexiones activas de Internet (servidores y establecidos)						
Proto	Recib	Enviad	Dirección local	Dirección remota	Estado	PID/Program name
tcp	0	0	0.0.0.0:110	0.0.0.0:*	ESCUCHAR	1204/dovecot
tcp	0	0	0.0.0.0:8080	0.0.0.0:*	ESCUCHAR	1769/web_sstt
tcp	0	0	192.168.56.106:53	0.0.0.0:*	ESCUCHAR	1168/named
tcp	0	0	10.0.2.15:53	0.0.0.0:*	ESCUCHAR	1168/named
tcp	0	0	127.0.0.1:53	0.0.0.0:*	ESCUCHAR	1168/named
tcp	0	0	0.0.0.0:22	0.0.0.0:*	ESCUCHAR	1192/sshd
tcp	0	0	0.0.0.0:25	0.0.0.0:*	ESCUCHAR	1636/exim4
tcp	0	0	127.0.0.1:953	0.0.0.0:*	ESCUCHAR	1168/named
tcp6	0	0	:::110	:::*	ESCUCHAR	1204/dovecot
tcp6	0	0	:::80	:::*	ESCUCHAR	1531/apache2
tcp6	0	0	:::53	:::*	ESCUCHAR	1168/named
tcp6	0	0	:::22	:::*	ESCUCHAR	1192/sshd
tcp6	0	0	:::25	:::*	ESCUCHAR	1636/exim4
tcp6	0	0	:::1:953	:::*	ESCUCHAR	1168/named
tcp6	0	0	:::443	:::*	ESCUCHAR	1531/apache2
udp	0	0	0.0.0.0:4500	0.0.0.0:*		2548/charon
udp	0	0	0.0.0.0:500	0.0.0.0:*		2548/charon
udp	0	0	192.168.56.106:53	0.0.0.0:*		1168/named
udp	0	0	10.0.2.15:53	0.0.0.0:*		1168/named
udp	0	0	127.0.0.1:53	0.0.0.0:*		1168/named
udp	0	0	0.0.0.0:68	0.0.0.0:*		763/dhclient
udp	0	0	0.0.0.0:68	0.0.0.0:*		792/dhclient
udp6	0	0	:::4500	:::*		2548/charon
udp6	0	0	:::500	:::*		2548/charon
udp6	0	0	:::53	:::*		1168/named
Activar zócalos de dominio UNIX (servidores y establecidos)						

Figura 43. Puertos activos en el servidor con netstat.

6. Trazas representativas de los protocolos

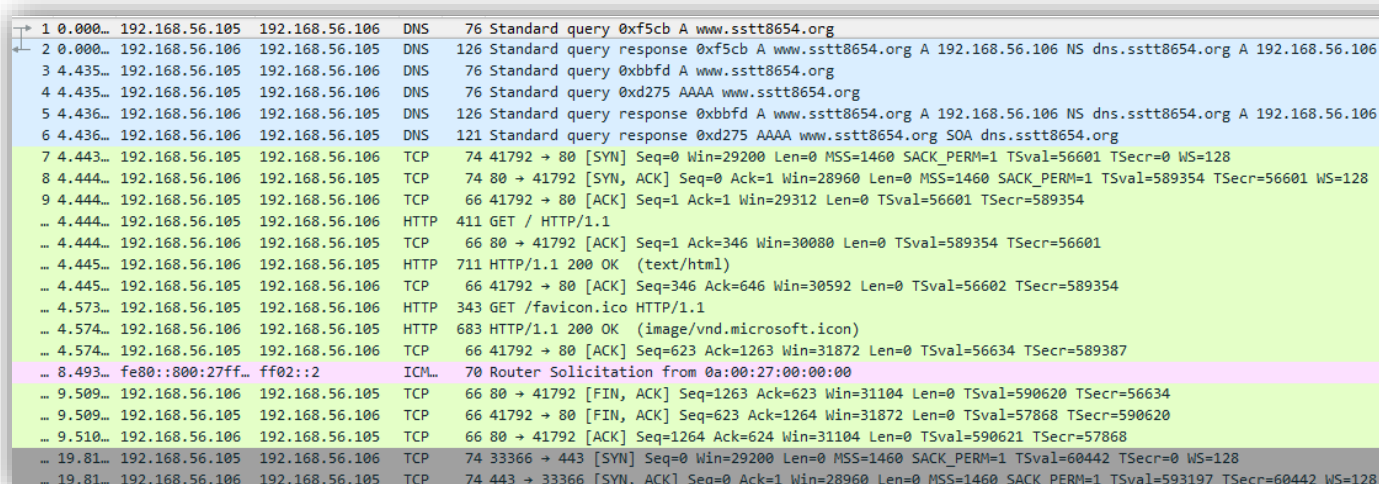
A continuación, se explican las trazas de los distintos protocolos empleados en el escenario.

- Intercambio DNS y el acceso a la web <http://www.sstt8654.org> (HTTP).

Como se ha mencionado en el apartado DNS de la [descripción del escenario](#), DNS se ha configurado para traducir las direcciones IP específicas del dominio, entre ellas la que veremos en este ejemplo.

Este protocolo actuará de la siguiente manera: al introducir la URL “<http://www.sstt8654.org>” en el navegador web se llevará a cabo la traducción de la página a su correspondiente IP. Para ello, el host2 (el cliente, siguiendo la descripción del escenario de la [figura 1](#)) desde donde estamos realizando la petición, se ha configurado para que todas las traducciones de nombres de dominio se envíen al DNS servidor (host1, con IP 192.168.56.106) del dominio “sstt8654.org”. Por lo que, una vez llegada la petición de traducción al servidor, comprueba si se encuentra en la lista de los nombres que gestiona (fichero de zona) y en ese caso devolverá su correspondiente dirección IP (que será la misma IP por estar en el mismo host1, 192.168.56.106).

En la figura 44, se puede observar toda la traza hasta que se consigue visualizar la página pedida. Se aprecian más consultas, ya que se realiza tanto para IPv4 (tipo A) y también para IPv6 (tipo AAAA).



1	0.000...	192.168.56.105	192.168.56.106	DNS	76 Standard query 0xf5cb A www.sstt8654.org
2	0.000...	192.168.56.106	192.168.56.105	DNS	126 Standard query response 0xf5cb A www.sstt8654.org A 192.168.56.106 NS dns.sstt8654.org A 192.168.56.106
3	4.435...	192.168.56.105	192.168.56.106	DNS	76 Standard query 0xbbfd A www.sstt8654.org
4	4.435...	192.168.56.105	192.168.56.106	DNS	76 Standard query 0xd275 AAAA www.sstt8654.org
5	4.436...	192.168.56.106	192.168.56.105	DNS	126 Standard query response 0xbbfd A www.sstt8654.org A 192.168.56.106 NS dns.sstt8654.org A 192.168.56.106
6	4.436...	192.168.56.106	192.168.56.105	DNS	121 Standard query response 0xd275 AAAA www.sstt8654.org SOA dns.sstt8654.org
7	4.443...	192.168.56.105	192.168.56.106	TCP	74 41792 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=56601 TSecr=0 WS=128
8	4.444...	192.168.56.106	192.168.56.105	TCP	74 80 → 41792 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=589354 TSecr=56601 WS=128
9	4.444...	192.168.56.105	192.168.56.106	TCP	66 41792 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=56601 TSecr=589354
...	4.444...	192.168.56.105	192.168.56.106	HTTP	411 GET / HTTP/1.1
...	4.444...	192.168.56.106	192.168.56.105	TCP	66 80 → 41792 [ACK] Seq=1 Ack=346 Win=30080 Len=0 TSval=589354 TSecr=56601
...	4.445...	192.168.56.106	192.168.56.105	HTTP	711 HTTP/1.1 200 OK (text/html)
...	4.445...	192.168.56.105	192.168.56.106	TCP	66 41792 → 80 [ACK] Seq=346 Ack=646 Win=30592 Len=0 TSval=56602 TSecr=589354
...	4.573...	192.168.56.105	192.168.56.106	HTTP	343 GET /favicon.ico HTTP/1.1
...	4.574...	192.168.56.106	192.168.56.105	HTTP	683 HTTP/1.1 200 OK (image/vnd.microsoft.icon)
...	4.574...	192.168.56.105	192.168.56.106	TCP	66 41792 → 80 [ACK] Seq=623 Ack=1263 Win=31872 Len=0 TSval=56634 TSecr=589387
...	8.493...	fe80::800:27ff...	ff02::2	ICMP	70 Router Solicitation from 0a:00:27:00:00:00
...	9.509...	192.168.56.106	192.168.56.105	TCP	66 80 → 41792 [FIN, ACK] Seq=1263 Ack=623 Win=31104 Len=0 TSval=590620 TSecr=56634
...	9.509...	192.168.56.105	192.168.56.106	TCP	66 41792 → 80 [FIN, ACK] Seq=623 Ack=1264 Win=31872 Len=0 TSval=57868 TSecr=590620
...	9.510...	192.168.56.106	192.168.56.105	TCP	66 80 → 41792 [ACK] Seq=1264 Ack=624 Win=31104 Len=0 TSval=590621 TSecr=57868
...	19.81...	192.168.56.105	192.168.56.106	TCP	74 33366 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=60442 TSecr=0 WS=128
...	19.81...	192.168.56.106	192.168.56.105	TCP	74 443 → 33366 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=593197 TSecr=60442 WS=128

Figura 44. Traza HTTP.

Si abrimos el mensaje número 2 de la figura 44 y analizamos su contenido, se pueden ver tres campos destacados en la figura 45, como:

- **Queries** que ha sido la solicitud para `www.sstt8654.org`.
- **Answers** que contiene la respuesta a la solicitud, indicando su dirección IP (`192.168.56.106`, como se ha mencionado antes) y el **Time to live** de `604800` segundos que se especificó en el fichero de zona para todos los nombres que se resuelvan.
- **Authoritative nameservers** establece el nombre del DNS (servidor de nombres autorizado) que es el responsable de traducir el nombre de la página.

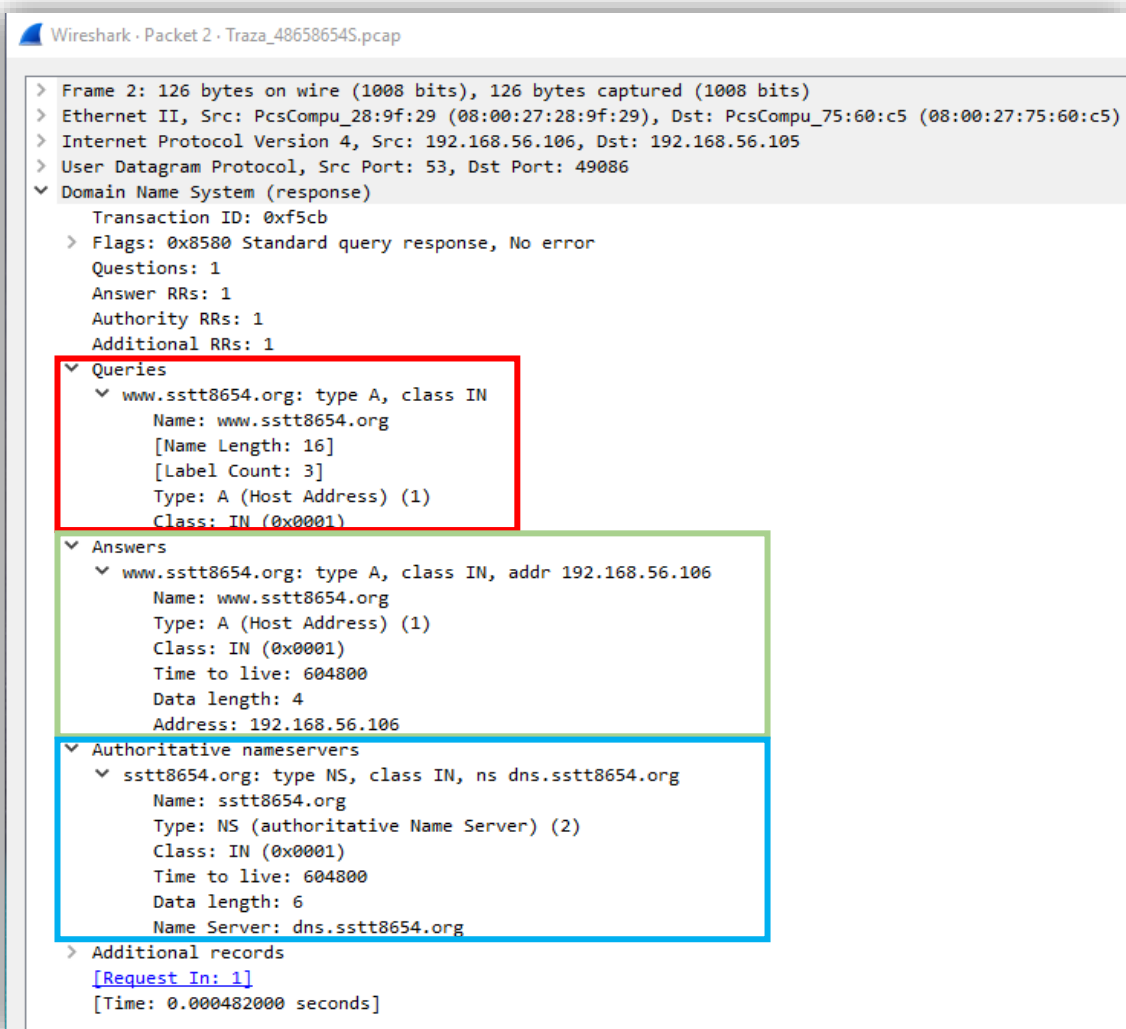


Figura 45. Paquete DNS response.

Como la parte de HTTP ya se explicó detalladamente en la primera entrega ([ver apartado del servidor web programado](#)), se omitirá esta explicación.

- Intercambio DNS y el acceso a la web <https://www.sstt8654.org> (HTTPS).

Para el acceso seguro a la página web, especificamos explícitamente en el navegador que queremos realizar una conexión segura a través de HTTPS. Como previamente se había accedido a la misma página y el navegador la ha guardado en una caché, se puede ver en la figura 46 como **no actúa el protocolo DNS**, pero en su caso realizaría el mismo proceso explicado en el punto anterior.

Al igual que para HTTP se produce la negociación TCP, pero ahora en lugar de intercambiar paquetes mediante el protocolo HTTP, se harán por medio del protocolo de transporte seguro TLS (*Transport Layer Security*).

TLS comienza realizando el *Handshake Protocol* que sirve para definir ciertos parámetros importantes para la seguridad de los mensajes y la autenticación del cliente y servidor, como los algoritmos de cifrado, y el intercambio de los certificados X.509 con las claves públicas.

Hecha esta negociación, los datos irán encapsulados en los paquetes cuya columna “info”, en la figura 46, pone “Application Data”.

Finalmente se cierra la conexión TCP con el servidor.

No.	Time	Source	Destination	Protocol	Length	Info
18	9.509103	192.168.56.106	192.168.56.105	TCP	66	80 → 41792 [FIN, ACK] Seq=1263 Ack=623 Win=31104 Len=0 TSval=590620 TSecr=56634
19	9.509649	192.168.56.105	192.168.56.106	TCP	66	41792 → 80 [FIN, ACK] Seq=623 Ack=1264 Win=31872 Len=0 TSval=57868 TSecr=590620
20	9.510500	192.168.56.106	192.168.56.105	TCP	66	80 → 41792 [ACK] Seq=1264 Ack=624 Win=31104 Len=0 TSval=590621 TSecr=57868
21	19.8131...	192.168.56.105	192.168.56.106	TCP	74	33366 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=60442 TSecr=0 WS=
22	19.8136...	192.168.56.106	192.168.56.105	TCP	74	443 → 33366 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=593197
23	19.8136...	192.168.56.105	192.168.56.106	TCP	66	33366 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=60444 TSecr=593197
24	19.8163...	192.168.56.105	192.168.56.106	TLSv1.2	1589	Client Hello
25	19.8167...	192.168.56.106	192.168.56.105	TCP	66	443 → 33366 [ACK] Seq=1 Ack=1524 Win=32128 Len=0 TSval=593198 TSecr=60444
26	19.8174...	192.168.56.106	192.168.56.105	TLSv1.2	218	Server Hello, Change Cipher Spec, Encrypted Handshake Message
27	19.8174...	192.168.56.105	192.168.56.106	TCP	66	33366 → 443 [ACK] Seq=1524 Ack=153 Win=30336 Len=0 TSval=60445 TSecr=593198
28	19.8178...	192.168.56.105	192.168.56.106	TLSv1.2	117	Change Cipher Spec, Encrypted Handshake Message
29	19.8186...	192.168.56.105	192.168.56.106	TLSv1.2	444	Application Data
30	19.8189...	192.168.56.106	192.168.56.105	TCP	66	443 → 33366 [ACK] Seq=153 Ack=1953 Win=34944 Len=0 TSval=593199 TSecr=60445
31	19.8195...	192.168.56.106	192.168.56.105	TLSv1.2	808	Application Data, Application Data, Application Data
32	19.8650...	192.168.56.105	192.168.56.106	TCP	66	33366 → 443 [ACK] Seq=1953 Ack=895 Win=31872 Len=0 TSval=60457 TSecr=593199
33	19.9361...	192.168.56.105	192.168.56.106	TLSv1.2	412	Application Data
34	19.9369...	192.168.56.106	192.168.56.105	TLSv1.2	9969	Application Data, Application Data
35	19.9369...	192.168.56.105	192.168.56.106	TCP	66	33366 → 443 [ACK] Seq=2299 Ack=10798 Win=51584 Len=0 TSval=60475 TSecr=593228
36	19.9625...	192.168.56.105	192.168.56.106	TLSv1.2	376	Application Data
37	19.9704...	192.168.56.106	192.168.56.105	TLSv1.2	741	Application Data, Application Data
38	20.0150...	192.168.56.105	192.168.56.106	TCP	66	33366 → 443 [ACK] Seq=2609 Ack=11473 Win=54528 Len=0 TSval=60494 TSecr=593237
39	24.9128...	192.168.56.106	192.168.56.105	TLSv1.2	97	Encrypted Alert
40	24.9129...	192.168.56.105	192.168.56.106	TCP	66	33366 → 443 [ACK] Seq=2609 Ack=11504 Win=54528 Len=0 TSval=61719 TSecr=594473
41	24.9136...	192.168.56.105	192.168.56.106	TLSv1.2	97	Encrypted Alert
42	24.9138...	192.168.56.106	192.168.56.105	TCP	66	443 → 33366 [FIN, ACK] Seq=11504 Ack=2609 Win=40704 Len=0 TSval=594473 TSecr=60494
43	24.9140...	192.168.56.105	192.168.56.106	TCP	66	33366 → 443 [FIN, ACK] Seq=2640 Ack=11505 Win=54528 Len=0 TSval=61719 TSecr=594473
44	24.9143...	192.168.56.106	192.168.56.105	TCP	66	443 → 33366 [ACK] Seq=11505 Ack=2641 Win=40704 Len=0 TSval=594473 TSecr=61719

Figura 46. Traza HTTPS.

- Intercambios DNS, SMTP y POP.

Para comprobar el funcionamiento de **SMTP/POP** se ha iniciado Thunderbird y se ha enviado un mensaje desde nombre1_sstt8654@sstt8654.org hasta nombre2_sstt8654@sstt8654.org.

Inicia el intercambio el protocolo DNS para poder traducir la dirección **smtp.sstt8654.org** que está definido como el servidor de correo y a donde será enviado el mensaje mediante el protocolo SMTP. Después se negocia la conexión con TCP y se lleva a cabo el intercambio de mensajes que establece SMTP.

No.	Time	Source	Destination	Protocol	Length	Info
114	92.8785...	192.168.56.105	192.168.56.106	DNS	77	Standard query 0x636d A smtp.sstt8654.org
115	92.8786...	192.168.56.105	192.168.56.106	DNS	77	Standard query 0xacbc AAAA smtp.sstt8654.org
116	92.8790...	192.168.56.106	192.168.56.105	DNS	127	Standard query response 0x636d A smtp.sstt8654.org A 192.168.56.106 NS dns.sstt8654.org A
117	92.8809...	192.168.56.106	192.168.56.105	DNS	122	Standard query response 0xacbc AAAA smtp.sstt8654.org SOA dns.sstt8654.org
118	92.8811...	192.168.56.105	192.168.56.106	TCP	74	34386 → 25 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=78711 TSecr=0 WS=128
119	92.8814...	192.168.56.106	192.168.56.105	TCP	74	25 → 34386 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=611474 TSecr=
120	92.8814...	192.168.56.105	192.168.56.106	TCP	66	34386 → 25 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=78711 TSecr=611474
121	93.7241...	192.168.56.106	192.168.56.105	SMTP	135	S: 220 server ESMTP Exim 4.86_2 Ubuntu Fri, 24 May 2019 17:05:31 +0200
122	93.7241...	192.168.56.105	192.168.56.106	TCP	66	34386 → 25 [ACK] Seq=1 Ack=70 Win=29312 Len=0 TSval=78921 TSecr=611685
123	93.7311...	192.168.56.105	192.168.56.106	SMTP	89	C: EHLO [192.168.56.105]
124	93.7315...	192.168.56.106	192.168.56.105	TCP	66	25 → 34386 [ACK] Seq=70 Ack=24 Win=29056 Len=0 TSval=611687 TSecr=78923
125	93.7319...	192.168.56.106	192.168.56.105	SMTP	187	S: 250-server Hello [192.168.56.105] [192.168.56.105] 250-SIZE 52428800 250-8BITMIME
126	93.7329...	192.168.56.105	192.168.56.106	SMTP	128	C: MAIL FROM:<nombre1_8654@sstt8654.org> BODY=8BITMIME SIZE=500
127	93.7333...	192.168.56.105	192.168.56.106	SMTP	74	S: 250 OK
128	93.7336...	192.168.56.106	192.168.56.106	SMTP	103	C: RCPT TO:<nombre2_8654@sstt8654.org>
129	93.7341...	192.168.56.106	192.168.56.105	SMTP	80	S: 250 Accepted
130	93.7491...	192.168.56.105	192.168.56.106	SMTP	72	C: DATA
131	93.7495...	192.168.56.106	192.168.56.105	SMTP	122	S: 354 Enter message, ending with "." on a line by itself
132	93.7567...	192.168.56.105	192.168.56.106	SMTP	566	C: DATA fragment, 500 bytes
133	93.7583...	192.168.56.105	192.168.56.106	SMTP IMF	69	from: nombre1_8654 <nombre1_8654@sstt8654.org>, subject: Prueba, (text/plain)
134	93.7591...	192.168.56.106	192.168.56.105	TCP	66	25 → 34386 [ACK] Seq=269 Ack=632 Win=30080 Len=0 TSval=611693 TSecr=78928
135	93.7664...	192.168.56.106	192.168.56.105	SMTP	94	S: 250 OK id=1hU8ID-0000X5-10
136	93.7827...	192.168.56.105	192.168.56.106	SMTP	72	C: QUIT
137	93.7832...	192.168.56.106	192.168.56.105	SMTP	97	S: 221 server closing connection
138	93.7835...	192.168.56.106	192.168.56.105	TCP	66	25 → 34386 [FIN, ACK] Seq=328 Ack=638 Win=30080 Len=0 TSval=611699 TSecr=78936
139	93.7895...	192.168.56.106	192.168.56.105	TCP	66	[TCP Retransmission] 25 → 34386 [FIN, ACK] Seq=328 Ack=638 Win=30080 Len=0 TSval=611704 T
140	93.7996...	192.168.56.105	192.168.56.106	TCP	78	34386 → 25 [ACK] Seq=638 Ack=329 Win=29312 Len=0 TSval=78940 TSecr=611699 SLE=328 SRE=329
141	93.8722...	192.168.56.105	192.168.56.106	TCP	66	34386 → 25 [FIN, ACK] Seq=638 Ack=329 Win=29312 Len=0 TSval=78958 TSecr=611699
142	93.8725...	192.168.56.106	192.168.56.105	TCP	66	25 → 34386 [ACK] Seq=329 Ack=639 Win=30080 Len=0 TSval=611722 TSecr=78958

Figura 47. Trazo SMTP.

Al no estar protegido, podemos ver perfectamente el mensaje en el paquete número 132, cuya columna “info” pone “C: Data fragment, 500 bytes”, como se ve en la figura 48.



Figura 48. Paquete SMTP (Data fragment).

Para el caso de recibir el mensaje, se procede de la misma manera: primero se traduce con DNS la dirección **pop.sstt8654.org** y después se establece la conexión TCP para comenzar el intercambio de mensajes POP como se ve en la figura 49.

En este caso, veremos el contenido del mensaje recuperado en el paquete número 162. Además, el paquete número 163 y 164 significan que el mensaje que se recupera del servidor va a ser borrado, tal cual se especificó en la [figura 14](#).

No.	Time	Source	Destination	Protocol	Length	Info
143	104.589...	192.168.56.105	192.168.56.106	TCP	74	49496 → 110 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=81638 TSecr=0 WS=
144	104.589...	192.168.56.106	192.168.56.105	TCP	74	110 → 49496 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=614403
145	104.589...	192.168.56.105	192.168.56.106	TCP	66	49496 → 110 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=81638 TSecr=614403
146	104.592...	192.168.56.106	192.168.56.105	POP	86	S: +OK Dovecot ready.
147	104.592...	192.168.56.105	192.168.56.106	TCP	66	49496 → 110 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=81638 TSecr=614403
148	104.614...	192.168.56.105	192.168.56.106	POP	72	C: CAPA
149	104.614...	192.168.56.106	192.168.56.105	TCP	66	110 → 49496 [ACK] Seq=21 Ack=7 Win=29056 Len=0 TSval=614409 TSecr=81643
150	104.614...	192.168.56.106	192.168.56.105	POP	149	S: +OK
151	104.632...	192.168.56.105	192.168.56.106	POP	78	C: AUTH PLAIN
152	104.632...	192.168.56.106	192.168.56.105	POP IMF	70	+
153	104.649...	192.168.56.105	192.168.56.106	POP	96	C: AG5vbWJyZTJfODY1NA8hbHVtbn8=
154	104.663...	192.168.56.106	192.168.56.105	POP	82	S: +OK Logged in.
155	104.672...	192.168.56.105	192.168.56.106	POP	72	C: STAT
156	104.673...	192.168.56.106	192.168.56.105	POP	77	S: +OK 1 833
157	104.675...	192.168.56.105	192.168.56.106	POP	72	C: LIST
158	104.675...	192.168.56.106	192.168.56.105	POP	93	S: +OK 1 messages:
159	104.697...	192.168.56.105	192.168.56.106	POP	72	C: UIDL
160	104.698...	192.168.56.106	192.168.56.105	POP	94	S: +OK
161	104.702...	192.168.56.105	192.168.56.106	POP	74	C: RETR 1
162	104.702...	192.168.56.106	192.168.56.105	POP	918	S: +OK 833 octets
163	104.726...	192.168.56.105	192.168.56.106	POP	74	C: DELE 1
164	104.727...	192.168.56.106	192.168.56.105	POP	93	S: +OK Marked to be deleted.
165	104.768...	192.168.56.105	192.168.56.106	TCP	66	49496 → 110 [ACK] Seq=83 Ack=1069 Win=30976 Len=0 TSval=81682 TSecr=614437
166	105.491...	192.168.56.105	192.168.56.106	POP	72	C: QUIT
167	105.495...	192.168.56.106	192.168.56.105	POP	102	S: +OK Logging out, messages deleted.
168	105.535...	192.168.56.105	192.168.56.106	TCP	66	49496 → 110 [ACK] Seq=89 Ack=1106 Win=30976 Len=0 TSval=81874 TSecr=614630
169	105.574...	192.168.56.105	192.168.56.106	TCP	66	49496 → 110 [FIN, ACK] Seq=89 Ack=1106 Win=30976 Len=0 TSval=81884 TSecr=614630
170	105.574...	192.168.56.106	192.168.56.105	TCP	66	110 → 49496 [ACK] Seq=1106 Ack=90 Win=29056 Len=0 TSval=614649 TSecr=81884

Figura 49. Traza POP.

- Uso de IKE e IPsec.

Se ha ejecutado el comando **ipsec restart** en el cliente para observar la **negociación IKEv2**, que comienza con los mensajes **IKE_SA_INIT** para establecer un canal seguro e **IKE_AUTH** para autenticar el cliente y el servidor. A partir de este momento todos los mensajes ya se encuentran cifrados y autenticados (cabecera ESP).

No.	Time	Source	Destination	Protocol	Length	Info
175	194.608...	192.168.56.105	192.168.56.106	ISAKMP	1166	IKE_SA_INIT MID=00 Initiator Request
176	194.622...	192.168.56.106	192.168.56.105	ISAKMP	523	IKE_SA_INIT MID=00 Responder Response
177	194.637...	192.168.56.105	192.168.56.106	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=9787) [Reassembled in #178]
178	194.637...	192.168.56.105	192.168.56.106	ISAKMP	286	IKE_AUTH MID=01 Initiator Request
179	194.645...	192.168.56.106	192.168.56.105	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=d6c3) [Reassembled in #180]
180	194.645...	192.168.56.106	192.168.56.105	ISAKMP	62	IKE_AUTH MID=01 Responder Response
181	194.663...	192.168.56.105	192.168.56.106	ESP	118	ESP (SPI=0xcc909bf5)
182	194.663...	192.168.56.105	192.168.56.106	ESP	118	ESP (SPI=0xcc909bf5)
183	194.663...	192.168.56.106	192.168.56.105	ESP	262	ESP (SPI=0xcae82c2e)
184	194.663...	192.168.56.106	192.168.56.105	DNS	220	Standard query response 0x5f81 A daisy.ubuntu.com A 162.213.33.132 A 162.213.33.132
185	194.663...	192.168.56.106	192.168.56.105	ESP	182	ESP (SPI=0xcae82c2e)
186	194.663...	192.168.56.106	192.168.56.105	DNS	137	Standard query response 0x8b0b AAAA daisy.ubuntu.com SOA ns1.canonical.com

Figura 50. Trazas de IKEv2 y ESP.

Si desmarcamos la opción de preferencias de protocolo “Attempt to detect/decode NULL encrypted ESP payloads” al hacer click derecho en un paquete ESP, podremos observar el contenido de todos los mensajes, ya que se especificó en la configuración que no se cifraran, sino que solo se autenticaran los mensajes.

- Acceso a la web <http://www.sstt8654.org> con IPsec.

Al acceder a dicha página se ha obtenido el resultado mostrado en la figura 51, el cual se ha descriptado con la opción que ofrece Wireshark antes mencionada.

No.	Time	Source	Destination	Protocol	Length	Info
317	290.836...	192.168.56.105	192.168.56.106	TCP	118	41832 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=128198
318	290.836...	192.168.56.106	192.168.56.105	TCP	118	80 → 41832 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TS
319	290.836...	192.168.56.106	192.168.56.105	TCP	74	[TCP Out-Of-Order] 80 → 41832 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=
320	290.836...	192.168.56.105	192.168.56.106	TCP	110	41832 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=128200 TSecr=660986
321	290.837...	192.168.56.105	192.168.56.106	HTTP	454	GET / HTTP/1.1
322	290.837...	192.168.56.106	192.168.56.105	TCP	110	80 → 41832 [ACK] Seq=1 Ack=346 Win=30080 Len=0 TSval=660987 TSecr=128200
323	290.837...	192.168.56.106	192.168.56.105	TCP	66	[TCP Dup ACK 322#1] 80 → 41832 [ACK] Seq=1 Ack=346 Win=30080 Len=0 TSval=
324	290.839...	192.168.56.106	192.168.56.105	HTTP	754	HTTP/1.1 200 OK (text/html)
325	290.839...	192.168.56.106	192.168.56.105	TCP	711	[TCP Retransmission] 80 → 41832 [PSH, ACK] Seq=1 Ack=346 Win=30080 Len=64
326	290.839...	192.168.56.105	192.168.56.106	TCP	110	41832 → 80 [ACK] Seq=346 Ack=646 Win=30592 Len=0 TSval=128200 TSecr=66098
327	290.977...	192.168.56.105	192.168.56.106	HTTP	386	GET /favicon.ico HTTP/1.1
328	290.978...	192.168.56.106	192.168.56.105	HTTP	726	HTTP/1.1 200 OK (image/vnd.microsoft.icon)
329	290.978...	192.168.56.106	192.168.56.105	TCP	683	[TCP Retransmission] 80 → 41832 [PSH, ACK] Seq=646 Ack=623 Win=31104 Len=
330	290.978...	192.168.56.105	192.168.56.106	TCP	110	41832 → 80 [ACK] Seq=623 Ack=1263 Win=31872 Len=0 TSval=128235 TSecr=6610
331	295.847...	PcsCompu_28:9f...	PcsCompu_75:60...	ARP	60	Who has 192.168.56.105? Tell 192.168.56.106
332	295.847...	PcsCompu_75:60...	PcsCompu_28:9f...	ARP	42	192.168.56.105 is at 08:00:27:75:60:c5
333	295.889...	192.168.56.106	192.168.56.105	TCP	110	80 → 41832 [FIN, ACK] Seq=1263 Ack=623 Win=31104 Len=0 TSval=662250 TSecr
334	295.889...	192.168.56.106	192.168.56.105	TCP	66	[TCP Out-Of-Order] 80 → 41832 [FIN, ACK] Seq=1263 Ack=623 Win=31104 Len=0
335	295.893...	192.168.56.105	192.168.56.106	TCP	110	41832 → 80 [FIN, ACK] Seq=623 Ack=1264 Win=31872 Len=0 TSval=129464 TSecr
336	295.894...	192.168.56.106	192.168.56.105	TCP	110	80 → 41832 [ACK] Seq=1264 Ack=624 Win=31104 Len=0 TSval=662251 TSecr=1294
337	295.894...	192.168.56.106	192.168.56.105	TCP	66	[TCP Dup ACK 336#1] 80 → 41832 [ACK] Seq=1264 Ack=624 Win=31104 Len=0 TSV

Figura 51. Trazas de HTTP con Ipsec.

Si abrimos el mensaje 321 referente a la petición GET como se ve en la figura 52, o cualquier otro, deberá aparecer 2 cabeceras IPv4 en su estructura lo que demuestra el correcto funcionamiento de IPsec en modo túnel.

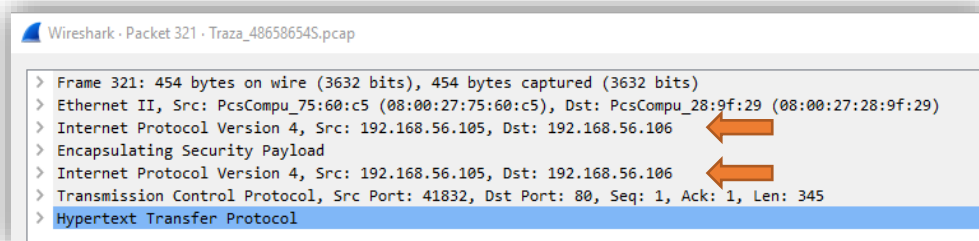


Figura 52. Paquete HTTP con IPsec.

- Acceso a la web <https://www.sstt8654.org> con IPsec.

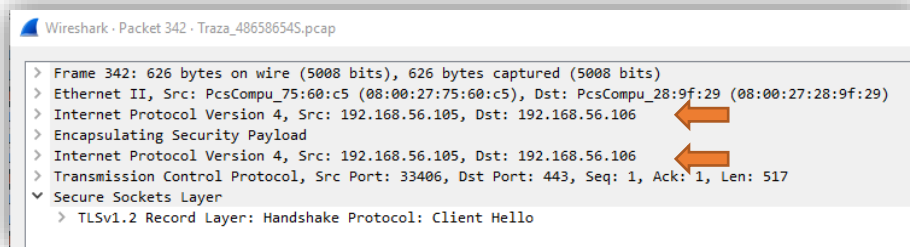


Figura 53. Paquete TLS con IPsec.

Lo mismo ocurrirá en este caso, donde observaremos que los paquetes TLS donde se encapsula la información sobre el intercambio HTTP, tienen también dos cabeceras IPv4.

- SMTP y POP con IPsec.

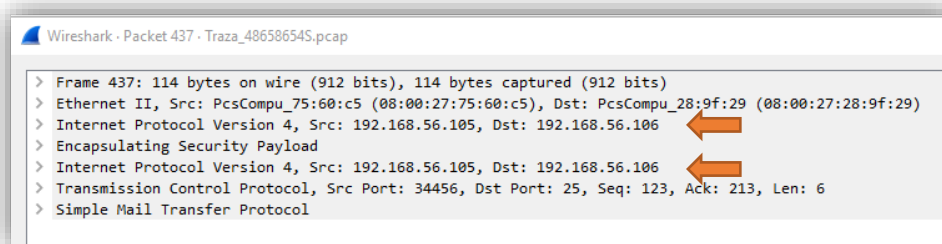


Figura 54. Paquete SMTP con IPsec.

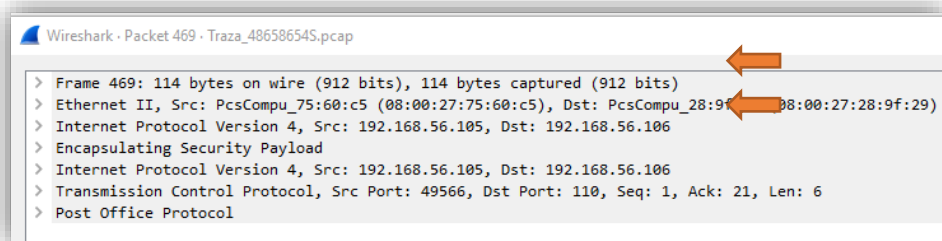


Figura 55. Paquete SMTP con IPsec.

También para SMTP y POP donde no varía su comportamiento, pero al igual que en los casos anteriores, IPsec se encarga de encapsular la IPv4 real en otra IP para ocultar su identidad, que debido al tipo de escenario en el que se está trabajando serán las mismas.

7. Problemas encontrados en el proceso de desarrollo

Durante la parte de programación, los problemas a destacar son con respecto a la manipulación de cadenas en el lenguaje C. A diferencia de otros lenguajes, como Java o C++, este no dispone de expresiones regulares para el análisis de cadenas, lo que dificulta mucho la comprobación de las peticiones del cliente. En su lugar se han utilizado las funciones de la librería `<string.h>`, con funciones como `strtok()`, `strcmp()`, `strcpy()`, `strcat()`, entre otros.

Otro problema ha sido comprender el funcionamiento de la función `select()` para el control de los sockets en la persistencia, así como adaptar el código para su implementación y correcto funcionamiento.

Por otro lado, aunque la parte de programación del servicio HTTP ha llevado más carga de trabajo en estas prácticas, ha sido la que mejor he comprendido desde un principio por lo que se ha hecho más ameno que la parte de configuración de los servicios, ya que debes comprender muy bien el servicio que estás configurando para no tener problemas.

Uno de los problemas surgidos en la parte de configuración de servicios ha sido con el cliente de correo Thunderbird que por una mala configuración en el servidor me daba ciertos problemas al crear las cuentas de correo y no encontraba la manera de reiniciar o restaurar por defecto el cliente de correo.

8. Número de horas empleadas para cada apartado

- Apache HTTP – 4 horas
- DNS – 5 horas
- SMTP/POP – 3 horas
- OpenSSL X.509 (autenticación del servidor) – 4 horas
- Apache SSL (autenticación del cliente) – 3 horas
- IPsec – 5 horas

- Programación Web-SSTT HTTP – 18 horas

TOTAL ESTIMADO = 42 horas

9. Conclusiones y valoración personal

A lo largo de la asignatura de Servicios Telemáticos se han presentado numerosos temas en teoría que se han podido llevar a la práctica y, como consecuencia de ello, ha servido para afianzar estos conocimientos y de gran utilidad para ponernos en práctica ante este tipo de situaciones en el ámbito de las redes informáticas.

Como valoración personal, creo que las prácticas han seguido fielmente el temario de teoría y por ello, remarco, ha sido de gran ayuda para terminar de comprender ciertos conceptos de la asignatura, como es el caso del apartado orientado a la seguridad con OpenSSL e IPsec cuya parte teórica no comprendí del todo.

Se ha de destacar que, en la parte de configuración de servicios, las diapositivas de la asignatura han resultado algo confusas y faltas de información, por lo que se ha tenido que emplear más tiempo en buscar información por internet o en acudir a una tutoría a preguntarle las dudas al profesor.