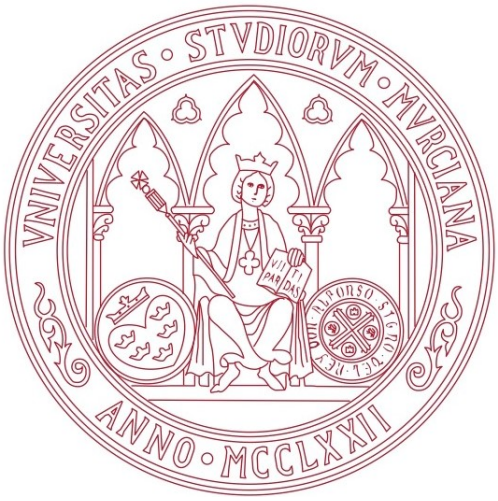
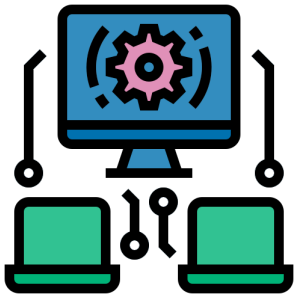


Servicios Telemáticos

MEMORIA DE PRÁCTICAS

Alejandro Segura García - G2.3 - 23837696J
Raúl Hernández Martínez - G2.2 - 24423648V
Tutorizado por: Ramón Sánchez Iborra
Convocatoria Mayo 2023



Índice

1. Introducción **4**

2. Escenario **4**

3. Servidor Web **5**

3.1. Descripción del escenario 5

3.2. Funcionalidad básica 5

3.3. Implementación 6

4. Servicios SMTP y POP **7**

4.1. Configuración SMTP 7

4.2. Configuración POP 8

4.3. Cliente 9

5. Servicio DNS **10**

5.1. Configuración del servicio 10

6. Servicios HTTP y HTTPS **13**

6.1. Configuración HTTP en el servidor 13

6.2. Configuración HTTPS en el servidor 14

6.3. Cliente 17

7. IPSec **17**

7.1. Servidor 17

7.2. Cliente 19

8. Análisis de trazas en wireshark. **19**

8.1. DNS y HTTP 19

8.2. DNS y HTTPS 22

8.3. DNS, SMTP e IPSec 23

8.4. DNS, POP e IPSec 25

8.5. Intercambio IKE e IPsec 26

9. Problemas encontrados **27**

10.Número de horas empleadas **27**

11.Conclusiones y valoraciones personales **28**

1. Introducción

Nuestro proyecto se basa en el enunciado de prácticas de la Servicios Telemáticos disponible en el apartado de recuersos de la asignatura. En este se comentan los servicios que debemos desplegar para poner en práctica los conocimientos adquiridos en las clases de teoría de este cuatrimestre. Se mencionan diferentes conceptos a lo largo del proyecto como son: un servidor web HTTP bastante simple programado en el lenguaje python, un servicio POP/SMTP, un servicio DNS y un servicio Apache HTTP/HTTPS. Todos estos conceptos los desarrollaremos más adelante.

2. Escenario

En nuestra práctica contamos con una máquina servidor con una dirección **IP 192.168.55.4** en la que se han instalado varios servivicos. Además, disponemos de una máquina cliente que utilizaremos para probar y verificar el correcto funcionamiento de los diferentes servicios desplagados.

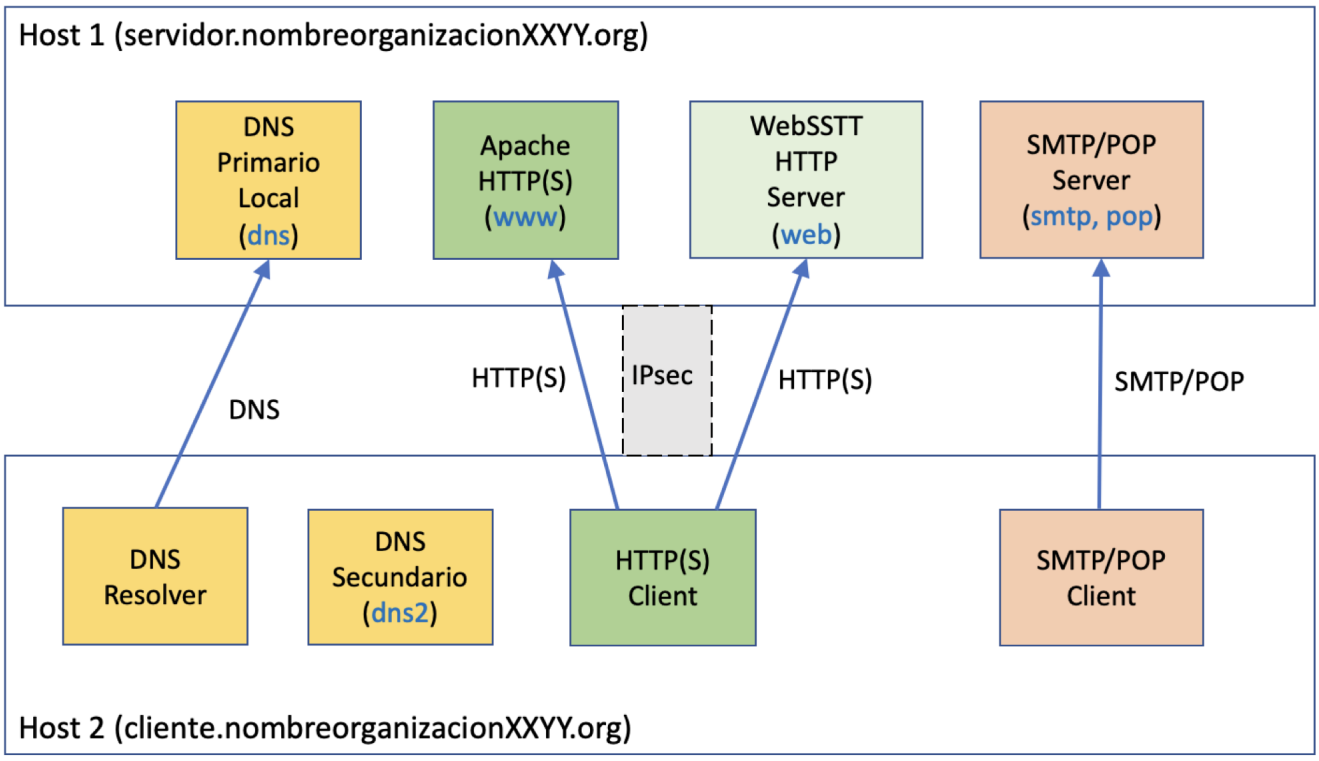


Figura 1: Escenario **objetivo** de la práctica.

Como podemos ver en la **Figura 1** en el servidor, que en nuestro caso sería **servidor.sstt4896.org**, tenemos varios servicios desplegados como son: servicio de **DNS**, servicio **Apache HTTP/HTTPS** y los servicios de **correo electrónico, POP/SMTP**.

3. Servidor Web

La finalidad de realizar este pequeño servidor es aprender a trabajar con las **cabeceras de HTTP** y afianzar nuestros conocimientos sobre la programación en el lenguaje **python**. Las cabeceras las veremos detalladamente en el apartado de análisis de trazas de **Wireshark**.

3.1. Descripción del escenario

El servicio está activo en el puerto 8096 de la máquina del servidor que hemos descrito anteriormente en el apartado 2. Inicialmente, partimos de un archivo **index.html** que contiene una imagen de la Universidad de Murcia con el escudo de la misma y un formulario html.

Figura 2: Página principal de nuestro sevidor web.

El programa esá diseñado para **escuchar solicitudes** de cualquier cliente, **crear un nuevo proceso** para cada solicitud recibida y manejarlas de manera concurrente. Una vez que se está atendiendo una solicitud en particular, se debe detectar el mensaje HTTP recibido y contestar con la información que se haya solicitado o generar una respuesta a esa solicitud. Aunque se pueden recibir cualquier tipo de mensaje HTTP, por defecto, este servicio solo puede responder a solicitudes de recursos almacenados en el servidor. Para ello, se deben utilizar diferentes llamadas al sistema y enviar toda la información siguendo el protoclo HTTP para que el navegador del cliente, en este caso Firefox, pueda interpretar la información y mostrarla correctamente.

3.2. Funcionalidad básica

- Gestión básica del método GET en peticiones HTTP request que provienen del cliente y su correspondiente respuesta.
- Gestionar una petición HTTP con el método POST básica de un formulario que se devolverá en la página index.html.

- Gestión básica de cookies en el servidor Web-SSTT HTTP. Se enviará una cookie con un contador de accesos al servidor de modo que al décimo acceso se denegará el acceso al contenido. El valor de la cookie variará para cada petición del usuario al recurso index.html del servidor, no para cada recurso. Esta cookie expirará a los 2 minutos de su creación.
- Implementar un mecanismo de persistencia HTTP. El servicio deberá mantener una conexión abierta durante un tiempo determinado y, en el caso de no recibir peticiones durante ese periodo de tiempo, se terminará la conexión.
- Implementar para la conexión persistente, además, un límite por número de peticiones, y no sólo por tiempo de la conexión TCP activa.
- El fichero HTML también hará referencia, al menos, a una imagen .gif o .jpg. La imagen deberá ser mayor de 8 Kbytes.
- Durante el lanzamiento del servidor, éste debe recibir como parámetro el puerto en el que ha de lanzarse el servicio.
- Verificación de que la petición HTTP es válida. Se debe verificar que la petición y sus cabeceras han sido definidas de acuerdo con la especificación de HTTP.
- Verificar que en la petición se ha incluido la cabecera Host.
- Las cabeceras que se tienen que incluir en la respuesta son:
  - Server
  - Content-Type
  - Content-Length
  - Date
  - Connection
  - Keep-Alive
  - Set-Cookie
- La cabecera Server deberá indicar el nombre completo del servidor del grupo. En nuestro caso Server: web.sstt4896.org.

3.3. Implementación

El método «**procces_web_request**» es donde se implementa la mayor parte de la funcionalidad del servicio, ya que este método es el encargado del procedimiento principal de los mensajes recibidos. Se encarga de la lectura de las peticiones HTTP y la comprobación de los errores que se pueden dar en este tipo de peticiones.

El siguiente código se bloquea hasta que llegan datos para leer al socket o hasta que pasa el tiempo indicado en TIMEOUT_CONNECTION.

```
rsublist, wsublist, xsublist = select.select([cs], [], [],
    TIMEOUT_CONNECTION)
    if not rsublist:
        print("\n\nHa saltado el Timeout.", file=sys.stderr)
        cerrar_conexion(cs)
        sys.exit(-1)
    else:
```

Una vez dentro se comprueban todos los posibles errores de la linea de solicitud, además de enviar su fichero de código html correspondiente para mostrar al usuario el error de forma legible.

El concepto de un socket es un flujo de bytes, por tanto, para enviar un recurso por él, enviamos las cabeceras correspondientes, junto con el recurso en bytes. Si el mensaje completo supera el tamaño máximo del socket (BUFSIZE), se enviara en varios mensajes como podemos ver a continuación.

```
if(os.stat(ruta).st_size + len(respuesta) <= BUFSIZE):
    with open(ruta, "rb") as f:
        buffer = f.read()
        to_send = respuesta.encode() + buffer
        enviar_mensaje(cs, to_send)
else:
    enviar_mensaje(cs, respuesta.encode())
    with open(ruta, "rb") as f:
        while (1):
            buffer = f.read(BUFSIZE)
                if(not buffer):
                    break
                enviar_mensaje(cs, buffer)
```

4. Servicios SMTP y POP

En nuestro servidor hemos desplegado los servicios de correo SMTP y POP. Para ello, ha sido necesario crear dos cuentas de usuario a nivel de sistema operativo cuyas carpetas de correo (Maildir) se han creado automaticamemnte.

Posteriormente, cuando hablemos de DNS, tendremos que tener en cuenta estos servicios para que nuestros clientes tengan acceso. Necesitaremos añadir dos entradas: smtp.sstt4896.org y pop.sstt4896.org. En el mismo DNS se ha añadido el registro tipo MX y otro tipo A.

4.1. Configuración SMTP

Inicialmente instalaremos y configuraremos SMTP siguiendo los pasos dados en las transparencias para que se lleve a cabo un buen funcionamiento.

Al configuar SMTP, en el menu interactivo lo hemos rellenado con la siguiente configuración:

```
Tipo general del servidor: // Primera opcion
```

```
Nombre del sistema de correo: // sstt4896.org
Direcciones IP en las que recibir conexiones SMTP // (en blanco);
Destinos de los que se acepta correo: // sstt4896.org
Dominio para que se puede reenviar correo: // sstt4896.org; org.es
Maquinas para las cuales reenviar correo: // 192.168.55.0/24
Limitar consultas DNS: // NO
Formato de buzon de correo: // Maildir
Dividir ficheros de configuracion: // NO
```

4.2. Configuración POP

Para desplegar el servicio POP en nuestro servidor, tendremos que instalarnos dovecot. Hay que configuar algunos archivos para el buen funcionamiento.

En el fichero **/etc/dovecot/conf.d/10-auth.conf** únicamente hay que descomentar dos líneas.

- **disable_plaintext_auth = no**. Para permitir la autenticación débil basada en texto plano, y así, poder analizar las trazas con mayor facilidad.

```
# Disable LOGIN command and all other plaintext authentications unless
# SSL/TLS is used (LOGINDISABLED capability). Note that if the remote IP
# matches the local IP (ie. you're connecting from the same computer), the
# connection is considered secure and plaintext authentication is allowed.
# See also ssl=required setting.
disable_plaintext_auth = no
```

Figura 3: Configuración disable_plaintext_auth.

- **auth_mechanims = plain**. Para activar la autenticación débil basada en texto plano.

```
# Space separated list of wanted authentication mechanisms:
#   plain login digest-md5 cram-md5 ntlm rpa apop anonymous gssapi otp skey
#   gss-spnego
# NOTE: See also disable_plaintext_auth setting.
auth_mechanisms = plain
```

Figura 4: Configuración auth_mechanims.

En el fichero **/etc/dovecot/conf.d/10-mail.conf** solamente hay que descomentar una línea.

- **mail_location = maildir: /Maildir**. Esto es para especificar el formato de los buzones de correo.

```
# See doc/wiki/Variables.txt for full list. Some examples:
#
#   mail_location = maildir:~/Maildir
#   mail_location = mbox:~/mail:INBOX=/var/mail/%u
#   mail_location = mbox:/var/mail/%d/%1n/%n:INDEX=/var/indexes/%d/%1n/%n
#
# <doc/wiki/MailLocation.txt>
#
mail_location = maildir:~/Maildir
```

Figura 5: Configuración mail_location.

4.3. Cliente

Una vez configurado los dos servicios pasamos a crear dos cuentas de usuarios. Hacemos click derecho en **Menú - Preferencias - Configuración de cuenta - Añadir cuenta de correo** y rellenamos los campos de nombre, dirección de correo y la contraseña. Las cuentas creadas son, uno de ellos con nombre «**raul**» y dirección de correo «**raul@sstt4896.org**» y otro con «**alejandro**» y dirección de correo«**alejandrol@sstt4896.org**», ambos tienen la contraseña «**sstt4896**».

Figura 6: Información de una cuenta en ThunderBird.

Al crear las cuentas, hay que indiciar que para el correo entrante se usará POP3 a través del puerto 110 y de SMTP para el correo saliente por el puerto 25. Como se dice en las transparencias no queremos seguridad de la conexión SSL y nuestra contraseña queremos que sea normal (tramitada de forma insegura).

Figura 7: Configuración servidor de ThunderBird.

En las imagenes mostradas son ejemplos de una cuenta de usuario, pero la configuración es igual para ambos.

5. Servicio DNS

DNS es uno de los protocolos de nivel de aplicación más utilizados en Internet. Su función es permitir el acceso a cualquier página web sin necesidad de conocer su dirección IP. La existencia de DNS es esencial para la navegación en Internet tal y como la conocemos hoy en día. Si bien es posible acceder a los sitios web mediante su dirección IP, resultaría difícil para las personas recordar más de unas pocas direcciones IP, lo que haría inaccesibles la gran mayoría de los sitios.

5.1. Configuración del servicio

Para la implementación y configuración del servidor DNS se ha utilizado el Software BIND, con distribución que contiene un servidor de nombres (named) y dos librerías resolvers. La información asociada con cada dominio es almacenada en resource records.

- Lo ficheros y directorios más importantes son el ejecutable (named) ubicado en **/usr/sbin/named**.
- El demonio de Ubuntu para la ejecución y parada de bind **/etc/init.d/bind9 start-stop-restart-status**.

- Directorios de configuración que se encuentran en **/etc/bind**, los ficheros **db.*** corresponden con la configuración de zonas en concreto.

```
alumno@server:/etc/bind$ ls
bind.keys  db.empty              db.sstt4896.org.zone.save  named.conf.options
db.0       db.local              named.conf                 rndc.key
db.127     db.root               named.conf.default-zones   zones.rfc1918
db.255     db.sstt4896.org.zone  named.conf.local
```

Figura 8: Ficheros configuración Bind.

Para la configuración de Bind se ha tenido que tocar dos ficheros y crear el fichero de zona de nuestra propia organización.

- El archivo **named.conf.options** encuentra en las opciones de configuración de servidor globales.

```
alumno@server:/etc/bind$ cat named.conf.options
options {
        directory "/var/cache/bind";

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        // forwarders {
        //      0.0.0.0;
        // };

        //========================================================================
        // If BIND logs error messages about the root key being expired,
        // you will need to update your keys.  See https://www.isc.org/bind-keys
    //========================================================================
        dnssec-validation no;

        auth-nxdomain no;    # conform to RFC1035
        listen-on-v6 { any; };
};
```

Figura 9: Fichero named.conf.options.

- **Directory** : directorio de trabajo, en nuestro caso /var/cache/bind
- **Forwarder** : direcciones IP de servidores DNS donde reenviar las peticiones para ser resueltas si no puede hacerlo nuestro servidor. Se encuentra comentado debido a que el servicio DNS deberá hacer un reenvío a los servidores DNS raíces en el caso de que no sepa realizar una resolución.
- **Dnssec-validation** : indicamos si queremos ofrecer DNSSEC en las respuestas, en nuestro caso no.

- El archivo **named.conf.local** define las características de la zona sstt4896.org que administra el servidor DNS.

```
alumno@server:/etc/bind$ cat named.conf.local
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "sstt4896.org" {
        type master;
        file "/etc/bind/db.sstt4896.org.zone";
        allow-transfer { 192.168.55.5; };
};
```

Figura 10: Fichero named.conf.local.

Destacamos la definición de la zona sstt4896.org indicando que el servidor tiene autoridad sobre dicha zona (type master). Con la directiva file indicamos la ruta del fichero que contiene los datos de configuración de la zona, en este caso «db.sstt4896.org.zone». Y con el allow-transfer establecemos los servidores esclavos que están autorizados para pedir una transferencia de información de la zona, en este caso sería la dirección IP del segundo servidor que hace de DNS secundario.

- Por último, el fichero **db.sstt4896.org.zone** es donde se encuentra toda la información de nuestra configuración.

```
alumno@server:/etc/bind$ cat db.sstt4896.org.zone
$TTL    3600
@ IN SOA dns1.sstt4896.org. root.dns1.sstt4896.org. (
        2       ; Serial
        604800  ; Refresh
        86400   ; Retry
        2419200 ; Expire
        604800) ; minimum
;
@       IN      NS      dns1.sstt4896.org.
@       IN      NS      dns2.sstt4896.org.
dns1.sstt4896.org.      IN      A       192.168.55.4
dns2.sstt4896.org.      IN      A       192.168.55.5
cliente.sstt4896.org.   IN      A       192.168.55.3
servidor.sstt4896.org.  IN      A       192.168.55.4
www.sstt4896.org.       IN      A       192.168.55.4
web.sstt4896.org.       IN      A       192.168.55.4
pop.sstt4896.org.       IN      A       192.168.55.4
sstt4896.org.           IN      MX      20      smtp.sstt4896.org.
smtp.sstt4896.org.      IN      A       192.168.55.4
```

Figura 11: Fichero db.sstt4896.org.zone.

$TTL 3600: Ajusta el valor del TTL predeterminado para toda la zona al valor indicado.

Para los registros DNS:

- **SOA** : Indica el inicio de los datos para una zona y define pámetros que afectan a todos los registros de la zona.
- **NS** : Identifica el servidor de nombres para el dominio.
- **MX** : Identifica dónde enviar el correo electrónico para el dominio.
- **A** : Convierte un nomre en una direccion IPv4.

6. Servicios HTTP y HTTPS

Para la implementación de estos servicios desplegamos un escenario compuesto por un servicio Apache HTTP/HTTPS en el servidor. Con la utilización de este servicio Apache nos centraremos en proporcionar una página web HTTP con servicio de login en el puerto estándar 80 y otra página web HTTPS con un servicio de certificados cliente/servidor basado en certificados X.509 en el puerto estándar 443. Una vez que hemos instalado apache, instalado en nuestro cliente el navegador web Firefox y añadida a nuestro servicio DNS la resolución del nombre www.sstt4896.org podemos pasar a desplegar ambos servicios.

6.1. Configuración HTTP en el servidor

Para la configuración en el servidor tendremos que crear un VirtualHost en el puerto 80. Para ello ejecutaremos el comando «**sudo a2ensite sstt4896.org**» para que se nos genere el fichero **/etc/apache2/sites enabled/sstt4896.org**, aquí es donde configuraremos nuestro VirtualHost.

Para la configuración de este VirtualHost primero tenemos que crear el fichero de contraseñas, para ello ejecutamos el comando «**htpasswd -c /etc/apache/passwords usuario**», le damos una contraseña a los usuarios, cuando se le ha añadido una contraseña a un usuario, se vuelve a repetir pero quitando la opción «**sstt**» porque sino reemplazaría el archivo. En nuestro caso la contraseña es «**sstt**».

Tras establecer una contraseña ya podemos establecer la configuración para el fichero de contraseñas en el VirtualHost. Tras establecer las contraseñas, establecemos la configuración para el fichero de contraseñas en el VirtualHost.

```
alumno@server:/etc/apache2$ cat passwords
raul48:$apr1$dCQvEgUI$eDhKD3NZWTCAKRwUoY1QV1
alejandro96:$apr1$ZTKHYSEK$SSMzjwbZGQ7MDOq07PNgR/
```

Figura 12: Fichero passwords.

```
alumno@server:/etc/apache2/sites-enabled$ cat sstt4896.conf
<VirtualHost *:80>
        ServerAdmin raul48@sstt4896.org
        ServerName www.sstt4896.org
        DocumentRoot /var/www/sstt4896
        <Directory /var/www/sstt4896>
                AllowOverride AuthConfig
                AuthType Basic
                AuthName "Acceso restringido a usuarios no registrados"
                AuthBasicProvider file
                AuthUserFile /etc/apache2/passwords
                Require user raul48 alejandro96
                Order allow,deny
                        allow from all
        </Directory>
</VirtualHost>
```

Figura 13: Fichero sites-enabled/sstt4896.conf - 80.

Como podemos ver en la primera línea de configuración del fichero estamos diciendo que el VirtualHost utilice el puerto 80 aclarando así que será HTTP.

Tras eso nos encontramos con **ServerAdmin** que es el correo del administrador, en nuestro caso es **raul48sstt4896.org**.

El **ServerName** establece el nombre del VirtualHost y el **DocumentRoot** indica el directorio donde se encuentra la página web.

Ahora, las opciones para el directorio, **AuthType** que decimos que sea Basic para que así las contraseñas vayan sin cifrar, **Requier user** que dice que los usuarios «raul» y «alejandro» podrán acceder si puede hacer login y por último **AuthUserFile /etc/apache2/passwords** que especifica el path al fichero de contraseñas.

6.2. Configuración HTTPS en el servidor

El objetivo es conseguir conexiones HTTPS con autenticación tanto de cliente como del servidor. Como paso previo para la configuración de HTTPS en Apache, hemos tenido que generar el material criptográfico necesario.

- Entidad de certificación (CA).
- Certificado de la CA (autofirmado).
- Clave privada de la CA.
- Certificado del servidor firmado por la CA.
- Clave privada del servidor.
- Certificado del cliente firmado por la CA.

- Clave privada del cliente.

Para generar estos certificados hemos creado nuestra propia PKI (Infraestructura de clave pública) situada en la CA. Su configuración es realizada en el fichero **/usr/lib/ssl/openssl.cnf**.

```
[ CA_default ]

dir             = /home/alumno/demoCA           # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
                                        # several ctificates with same subject.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber        # the current crl number
                                        # must be commented out to leave a V1 CRL
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE        = $dir/private/.rand    # private random number file

x509_extensions = usr_cert              # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default            # Certificate field options
```

Figura 14: Fichero openssl.cnd - CA_default.

El único cambio es en la opción **dir**, se ha cambiado de **./demoCA** al valor actual, es decir, **/home/alumno/demoCA**, con el propósito de actualizar al directorio que queremos que se almacene la información criptográfica que vayamos generando. Además, hemos configurado la estructura de nombres.

```
[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = ES
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Murcia

localityName                    = Locality Name (eg, city)

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = UMU

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, section)
organizationalUnitName_default  = sstt4896

commonName                      = Common Name (e.g. server FQDN or YOUR name)
commonName_max                  = 64

emailAddress                    = Email Address
emailAddress_max                = 64

# SET-ex3                       = SET extension number 3
```

Figura 15: Fichero openssl.cnd - req_distinguished_name.

Una vez hecho esto, necesitamos los certificados tanto para la CA, como para el servidor. El certificado de la ca (cacert.pem) irá autofirmado, y el del servidor (servercert.pem) irá firmado por la CA.

Para que todo esto funcionase ha sido necesario, al igual que hemos hecho en HTTP, crear una nueva VirtualHost, esta vez para HTTPS habilitando el puerto 443. Esta VirtualHost es idéntica a la de HTTP solo que ahora hemos añadido al final nuevas opciones.

```
<VirtualHost *:443>
        ServerAdmin raul48@sstt4896.org
        ServerName www.sstt4896.org
        DocumentRoot /var/www/sstt4896
        <Directory /var/www/sstt4896>
                AllowOverride none
                AuthType Basic
                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 16: Fichero sites-enabled/sstt4896.conf - 443.

Con **SSLEngine on** indicamos que queremos utilizar SSL, las tres siguientes opciones decimos donde se encuentra cada certificado, **SSLVerifyClient** se activa para autentificar al cliente y por último **SSLVerifyDepth** se indica la profundidad de verificación.

6.3. Cliente

Para garantizar una conexión segura entre el cliente y el servidor, es necesario configurar el navegador web del cliente. Esto implica que el navegador debe confiar en la Autoridad de Certificación (CA) que emite el certificado del servidor. Para lograr esto, es necesario copiar el certificado de la CA (cacert.pem) en el cliente y posteriormente importarlo en el navegador web.

Además, si se requiere la autenticación del cliente, es necesario copiar tanto el certificado como la clave privada del cliente en el navegador. Una forma de lograr esto es mediante el uso del formato de archivo **.pfx**, el cual contiene tanto el certificado del cliente como su clave privada, así como el certificado de la CA. Este archivo **.pfx** puede ser importado en el navegador para lograr la autenticación del cliente.

7. IPSec

Para la configuración de IPsec hemos utilizado el software **Strongswan** en ambos equipos, este software nos permite cifrar, autenticar y dar integridad a toda comunicación entre hosts. La configuración para el servidor y el cliente es muy similar.

7.1. Servidor

Los ficheros que hemos modificado para la configuración del servidor han sido **/etc/ipsec.conf** y **/etc/ipsec.secrets**. Como paso previo, hemos tenido que hacer los siguientes

cambios:

- Mover la clave privada del servidor (serverkey.pem) al directorio **/etc/ipsec.d/private/**.
- Mover el certificado del servidor (servercert.pem) al directorio **/etc/ipsec.d/certs/**.
- Situar el certificado de la CA (cacert.pem) en el directorio **/etc/ipsec.d/cacert/**.

Este es el fichero de configuración de **Strongswan**. Aquí se definen las características de la asociación de seguridad IPsec (AH, ESP, túnel o transporte).

```
alumno@server:/etc$ cat ipsec.conf
config setup
#Para cualquier conexión
conn %default
        ikelifetime=60m # Tiempo de vida de una IKE SA
        keylife=20m # Tiempo de vida de una Ipsec sa
        rekeymargin=3m
        keyingtries=1
        mobike=no
        keyexchange=ikev2 #Usamos IKEv2
        authby=pubkey

#Para la conexión específica entre estos dos hosts
conn host-host # “host-host” es simplemente una etiqueta
        left=192.168.55.4 #La IP de un equipo
        leftcert=/etc/ipsec.d/certs/servercert.pem
        leftid="C=ES, ST=Murcia, O=UMU, OU=sstt4896, CN=www.sstt4896.org"
        right=192.168.55.3 #La IP de otro equipo
        rightid="C=ES, ST=Murcia, O=UMU, OU=sstt4896, CN=raul48.org"
        type=transport # modo transporte
        auto=start # IKEv2 se ejecuta en el momento que haya un ipsec start
        ah=sha1-sha256-modp1024
        esp=null
```

Figura 17: Fichero /etc/ipsec.conf.

La opción **authby=pubkey** indica que la autenticación se realizará con clave pública. Posteriormente indicamos en **left** la dirección ip de la máquina que contiene el fichero (nuestro servidor), en **leftcert** decimos donde se encuentra el certificado del cliente, en **leftid** la identidad del servidor, en **rigth** indicamos la dirección ip de la máquina destino (cliente) y en **rigthid** la identidad del cliente.

En este archivo se realiza la configuración de claves, en este caso de indica el path a la clave privada del servidor .

```
alumno@server:/etc$ sudo cat ipsec.secrets
# /etc/ipsec.secrets - strongSwan IPsec secrets file
# 192.168.55.3 192.168.55.4
: RSA /etc/ipsec.d/private/serverkey.pem
```

Figura 18: Fichero /etc/ipsec.secrets.

7.2. Cliente

Como primer paso, tenemos que mover las claves del cliente, a los mismos directorios que hicimos con el servidor.

Respecto a los ficheros **ipsec.conf** y **ipsec.secrets** del cliente, las opciones son las mismas que en el servidor, pero ahora **left** pasa a ser el cliente y **right** pasa a ser el servidor.

```
alumno@desktop:~$ cat /etc/ipsec.conf
config setup
#Para cualquier conexión
conn %default
        ikelifetime=60m # Tiempo de vida de una IKE SA
        keylife=20m # Tiempo de vida de una Ipsec sa
        rekeymargin=3m
        keyingtries=1
        mobike=no
        keyexchange=ikev2 #Usamos IKEv2
        authby=pubkey

#Para la conexión especifica entre estos dos hosts
conn host-host # "host-host" es simplemente una etiqueta
        left=192.168.55.3 #La IP de un equipo
        leftcert=/etc/ipsec.d/certs/raulcert.pem
        leftid="C=ES, ST=Murcia, O=UMU, OU=sstt4896, CN=raul48.org"
        right=192.168.55.4 #La IP de otro equipo
        rightid="C=ES, ST=Murcia, O=UMU, OU=sstt4896, CN=www.sstt4896.org"
        type=transport # modo transporte
        auto=start # IKEv2 se ejecuta en el momento que haya un ipsec start
        ah=sha1-sha256-modp1024
        esp=null
```

Figura 19: Fichero /etc/ipsec.conf.

```
alumno@desktop:~$ sudo cat /etc/ipsec.secrets
# /etc/ipsec.secrets - strongSwan IPsec secrets file
# 192.168.55.4 192.168.55.3
: RSA /etc/ipsec.d/private/raulkey.pem
```

Figura 20: Fichero /etc/ipsec.secrets.

8. Análisis de trazas en wireshark.

8.1. DNS y HTTP

Para realizar este análisis del intercambio DNS y HTTP hemos puesto a capturar Wireshark en el cliente y desde el navegador Firefox hemos accedido a **http://www.sstt4896.org**.

61	16.997933	192.168.55.3	192.168.55.4	DNS	100	Standard query 0x919a A www.sstt4896.org
62	16.998070	192.168.55.3	192.168.55.4	DNS	100	Standard query 0x7824 AAAA www.sstt4896.org
63	16.998516	192.168.55.4	192.168.55.3	DNS	151	Standard query response 0x919a A www.sstt4896.org A 192.168.55.4 NS dns1.sstt4896.org A 192.168.55.4
64	16.998695	192.168.55.4	192.168.55.3	DNS	146	Standard query response 0x7824 AAAA www.sstt4896.org SOA dns1.sstt4896.org
65	16.999184	192.168.55.3	192.168.55.4	TCP	98	34550 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3640634 TSecr=0 WS=128
66	16.999430	192.168.55.4	192.168.55.3	TCP	98	80 → 34550 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=5094438 TSecr=3640634 WS=128
67	16.999460	192.168.55.3	192.168.55.4	TCP	90	34550 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3640634 TSecr=5094438
68	16.999564	192.168.55.3	192.168.55.4	HTTP	384	GET / HTTP/1.1
69	16.999717	192.168.55.4	192.168.55.3	TCP	90	80 → 34550 [ACK] Seq=1 Ack=295 Win=30080 Len=0 TSval=5094438 TSecr=3640634
70	17.000269	192.168.55.4	192.168.55.3	HTTP	851	HTTP/1.1 401 Unauthorized (text/html)
71	17.000318	192.168.55.3	192.168.55.4	TCP	90	34550 → 80 [ACK] Seq=295 Ack=762 Win=30848 Len=0 TSval=3640634 TSecr=5094438
72	21.586240	192.168.55.3	192.168.55.4	HTTP	423	GET / HTTP/1.1
73	21.587577	192.168.55.4	192.168.55.3	HTTP	719	HTTP/1.1 200 OK (text/html)
74	21.587641	192.168.55.3	192.168.55.4	TCP	90	34550 → 80 [ACK] Seq=628 Ack=1391 Win=32256 Len=0 TSval=3641781 TSecr=5095585

Figura 21: Acceso a http://www.sstt4896.org.

El cliente hace una consulta DNS de tipo **A** para conocer la IP de **www.sstt4896.org**. Luego hace la misma consulta, pero ahora es de tipo **AAAA**, que sirve para conocer la **IPv6**. Y lo que hay despues son las respuestas DNS a los dos mensajes previos.

Llega el **GET HTTP** del cliente hacia el servidor hacia el puerto 80.

El servidor contesta con un código **401 Unauthorized** indicando que no está autorizado para poder visitar esa página web. Esto ocurre por la directiva **Require** [**user**] que añadimos en la configuración de Apache. Por tanto ahora el cliente tendría que autenticarse mediante itroduciendo su usuario y contraseña para demostrar que es el usuario «raul48».

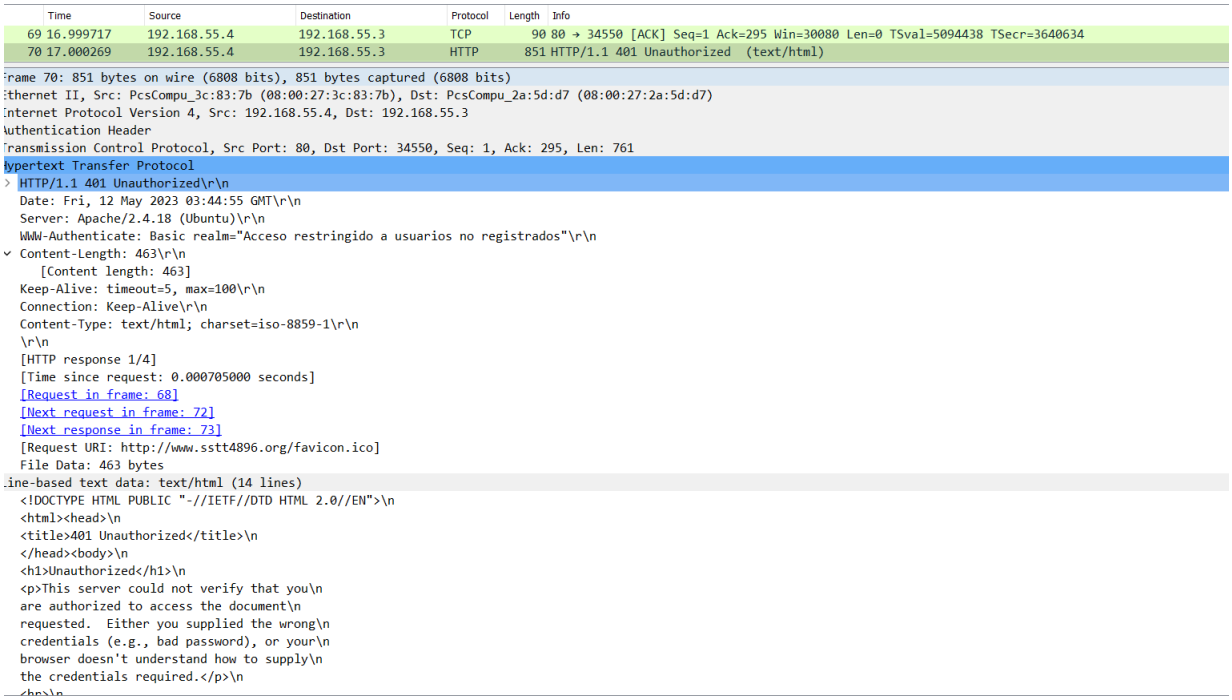


Figura 22: Código de error 401 enviado por el servidor.

Podemos destacar la cabecera **WWW-Authenticate** donde indica el texto que muestra en el navegador cuando insertamos el usuario y contrasña, la cabecera **Keep-Alive** donde indica un timeout de 5 segundos y como máximo se reciben en esta conexión 100 peticiones. Si nos fijamos en la secció de **Line-based text data** podemos ver una página de error que mostraríaa el navegador en caso de no poder autenticarnos.

Podemos ver que luego el cliente vuelve a hacer un GET, pero esta vez el servidor sí le devuelve el **index.html**. Podemos apreciar en la parte destacada, el cliente está autenticandose mediante sus credenciales dentro del propio **GET HTTP**.

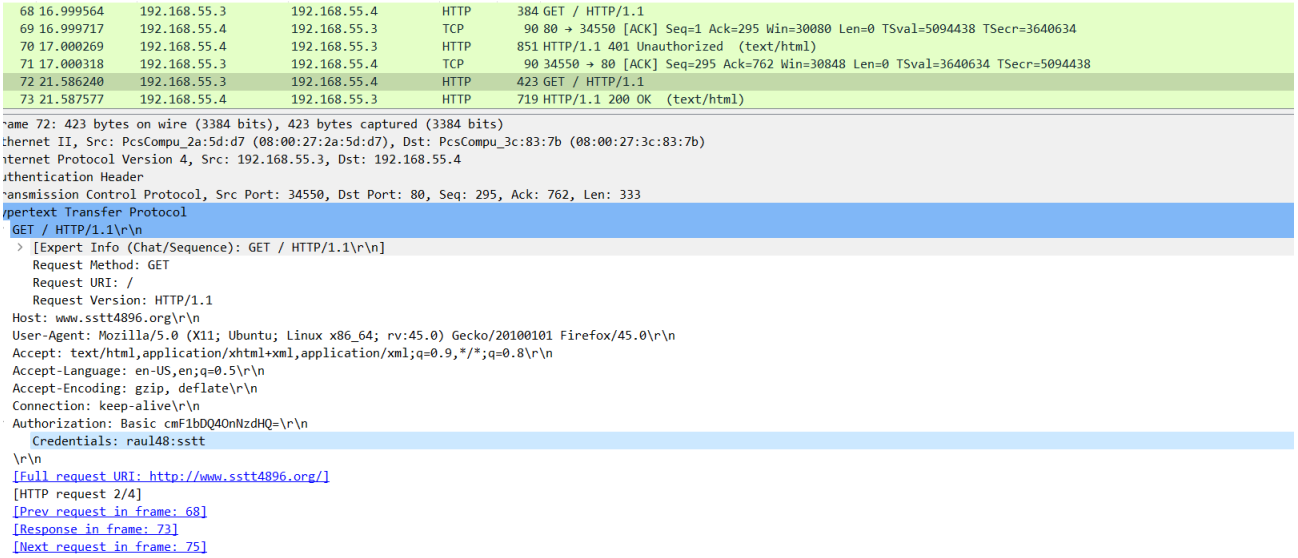


Figura 23: Consulta HTTP con autorización

8.2. DNS y HTTPS

Para poder ver un análisis de intercambio con HTTPS, tenemos que acceder a **https://www.sstt4896.org** en nuestro firefox, y ponernos a capturar con wireshark en el cliente.

59	18.049979	192.168.55.3	192.168.55.4	DNS	100	Standard query 0x44eb A www.sstt4896.org
60	18.050671	192.168.55.4	192.168.55.3	DNS	151	Standard query response 0x44eb A www.sstt4896.org A 192.168.55.4 NS dns1.sstt4896.org A 192.168.55.4
61	18.051685	192.168.55.3	192.168.55.4	DNS	100	Standard query 0x55a0 AAAA www.sstt4896.org
62	18.052111	192.168.55.4	192.168.55.3	DNS	146	Standard query response 0x55a0 AAAA www.sstt4896.org SOA dns1.sstt4896.org
63	18.057159	192.168.55.3	192.168.55.4	TCP	98	59594 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4204157 TSecr=0 WS=128
64	18.057557	192.168.55.4	192.168.55.3	TCP	98	443 → 59594 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=5657961 TSecr=4204157 WS=128
65	18.057594	192.168.55.3	192.168.55.4	TCP	90	59594 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4204157 TSecr=5657961
66	18.057807	192.168.55.3	192.168.55.4	TLSv1.2	280	Client Hello
67	18.058021	192.168.55.4	192.168.55.3	TCP	90	443 → 59594 [ACK] Seq=1 Ack=191 Win=30080 Len=0 TSval=5657961 TSecr=4204157
68	18.061624	192.168.55.4	192.168.55.3	TLSv1.2	1514	Server Hello
69	18.061673	192.168.55.3	192.168.55.4	TCP	90	59594 → 443 [ACK] Seq=191 Ack=1425 Win=32128 Len=0 TSval=4204158 TSecr=5657961
70	18.062007	192.168.55.4	192.168.55.3	TLSv1.2	1165	Certificate, Server Key Exchange, Certificate Request, Server Hello Done
71	18.062039	192.168.55.3	192.168.55.4	TCP	90	59594 → 443 [ACK] Seq=191 Ack=2500 Win=34944 Len=0 TSval=4204158 TSecr=5657962
72	27.864645	192.168.55.3	192.168.55.4	TLSv1.2	1442	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
73	27.864857	192.168.55.3	192.168.55.4	TLSv1.2	417	Application Data
74	27.865027	192.168.55.4	192.168.55.3	TCP	90	443 → 59594 [ACK] Seq=2500 Ack=1870 Win=35712 Len=0 TSval=5660413 TSecr=4206608
75	27.866305	192.168.55.4	192.168.55.3	TLSv1.2	1324	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
76	27.866371	192.168.55.3	192.168.55.4	TCP	90	59594 → 443 [ACK] Seq=1870 Ack=3734 Win=37888 Len=0 TSval=4206609 TSecr=5660413
77	27.866816	192.168.55.4	192.168.55.3	TLSv1.2	807	Application Data, Application Data, Application Data
78	27.866870	192.168.55.3	192.168.55.4	TCP	90	59594 → 443 [ACK] Seq=1870 Ack=4451 Win=40704 Len=0 TSval=4206609 TSecr=5660413
79	27.943013	192.168.55.3	192.168.55.4	TLSv1.2	434	Application Data
80	27.943860	192.168.55.4	192.168.55.3	TLSv1.2	1514	Application Data

Figura 24: Acceso a https://www.sstt4896.org.

Primero se lleva a cabo los mansjes DNS para la resolución del nombre **www.sstt4896.org**, y se lleva a cabo una conexión TCP.

Se realiza un **Client Hello** y el servidor le contesta con un **Server Hello**, una vez hecho esto el cliente y el servidor se intercambian los certificados para poder mantener una conexión segura entre ellos dos.

```
Frame 66: 280 bytes on wire (2240 bits), 280 bytes captured (2240 bits)
Ethernet II, Src: PcsCompu_2a:5d:d7 (08:00:27:2a:5d:d7), Dst: PcsCompu_3c:83:7b (08:00:27:3c:83:7b)
Internet Protocol Version 4, Src: 192.168.55.3, Dst: 192.168.55.4
Authentication Header
Transmission Control Protocol, Src Port: 59594, Dst Port: 443, Seq: 1, Ack: 1, Len: 190
Transport Layer Security
▾ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 185
  ▾ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 181
      Version: TLS 1.2 (0x0303)
    › Random: 757b8c65a038ed87396f52bc74be09e12906ac86ddde59764a8bd4079f40d3a9
      Session ID Length: 0
      Cipher Suites Length: 22
    ▾ Cipher Suites (11 suites)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
        Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
        Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
        Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
        Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
      Compression Methods Length: 1
    › Compression Methods (1 method)
      Extensions Length: 118
    › Extension: server_name (len=21)
    › Extension: renegotiation_info (len=1)
    › Extension: supported_groups (len=8)
    › Extension: ec_point_formats (len=2)
    › Extension: session_ticket (len=0)
    › Extension: next_protocol_negotiation (len=0)
```

Figura 25: Cliente Hello.

Se puede ver la versión de TLS que soporta el cliente Version: TLS 1.2.

Una vez recibida la propuesta de suite criptográfica por parte del cliente, es el servidor el que hace la elección. Es aquí en el mensaje **ServerHello** donde se indica el acuerdo del final. Sino, se cierra la conexión.

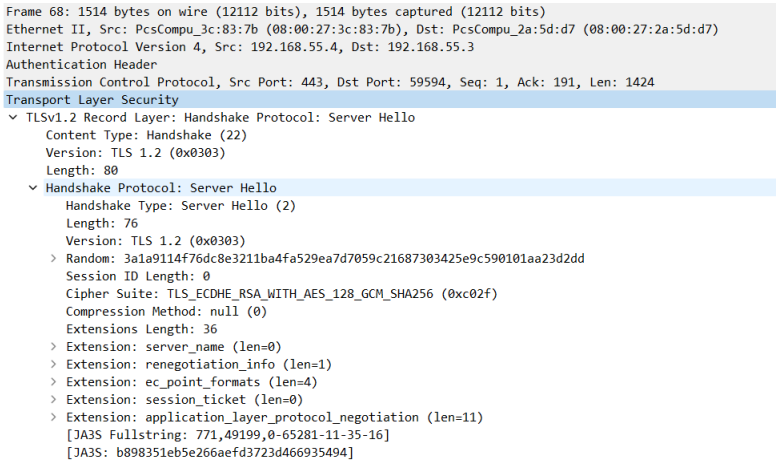


Figura 26: Server Hello.

El servidor envía su certificado junto con su clave, el certificado de la CA y solicita el certificado del cliente. El cliente manda su certificado junto con su clave.

8.3. DNS, SMTP e IPSec

Para realizar un análisis de las trazas respecto a DNS y SMTP, iniciamos Wireshark en el cliente, y mediante **Thunderbird** enviamos un correo desde la cuenta del usuario «raul48» al usuario «alejandro96».

1	0.000000	192.168.55.3	192.168.55.4	DNS	101	Standard query 0x334a A smtp.sstt4896.org
2	0.000033	192.168.55.3	192.168.55.4	DNS	101	Standard query 0x7a19 AAAA smtp.sstt4896.org
3	0.000770	192.168.55.4	192.168.55.3	DNS	187	Standard query response 0x334a A smtp.sstt4896.org A 192.168.55.4 NS dns1.sstt4896.org NS dns2.sstt4896.org A 192.168.55.4 A
4	0.009064	192.168.55.4	192.168.55.3	DNS	147	Standard query response 0x7a19 AAAA smtp.sstt4896.org SOA dns1.sstt4896.org
5	0.009828	192.168.55.3	192.168.55.4	TCP	98	60310 → 25 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4294965964 TSecr=0 WS=128
6	0.010272	192.168.55.4	192.168.55.3	TCP	98	25 → 60310 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=4294955793 TSecr=4294965964 WS=128
7	0.010314	192.168.55.3	192.168.55.4	TCP	90	60310 → 25 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4294965964 TSecr=4294955793
8	0.011904	192.168.55.4	192.168.55.3	SMTP	159	S: 220 server ESMTP Exim 4.86_2 Ubuntu Fri, 12 May 2023 17:12:03 +0200
9	0.011966	192.168.55.3	192.168.55.4	TCP	90	60310 → 25 [ACK] Seq=1 Ack=70 Win=29312 Len=0 TSval=4294965965 TSecr=4294955794
10	0.217698	192.168.55.3	192.168.55.4	SMTP	111	C: EHLO [192.168.55.3]
11	0.218313	192.168.55.4	192.168.55.3	TCP	90	25 → 60310 [ACK] Seq=70 Ack=22 Win=29056 Len=0 TSval=4294955845 TSecr=4294966016
12	0.218707	192.168.55.4	192.168.55.3	SMTP	207	S: 250-server Hello [192.168.55.3] [192.168.55.3] | SIZE 52428800 | 8BITMIME | PIPELINING | PRDR | HELP
13	0.218738	192.168.55.3	192.168.55.4	TCP	90	60310 → 25 [ACK] Seq=22 Ack=187 Win=29312 Len=0 TSval=4294966016 TSecr=4294955846
14	0.392079	192.168.55.3	192.168.55.4	SMTP	144	C: MAIL FROM:<raul@sstt4896.org> BODY=8BITMIME SIZE=463
15	0.392720	192.168.55.4	192.168.55.3	SMTP	98	S: 250 OK
16	0.392772	192.168.55.3	192.168.55.4	TCP	90	60310 → 25 [ACK] Seq=76 Ack=195 Win=29312 Len=0 TSval=4294966060 TSecr=4294955889
17	0.394556	192.168.55.3	192.168.55.4	SMTP	124	C: RCPT TO:<alejandro@sstt4896.org>
18	0.395209	192.168.55.4	192.168.55.3	SMTP	104	S: 250 Accepted
19	0.397033	192.168.55.3	192.168.55.4	SMTP	96	C: DATA
20	0.397638	192.168.55.4	192.168.55.3	SMTP	146	S: 354 Enter message, ending with "." on a line by itself
21	0.399936	192.168.55.3	192.168.55.4	SMTP	553	C: DATA fragment, 463 bytes
22	0.400224	192.168.55.3	192.168.55.4	SMTP/I…	93	from: raul <raul@sstt4896.org>, subject: Hola Alejandro, (text/plain)
23	0.400482	192.168.55.4	192.168.55.3	TCP	90	25 → 60310 [ACK] Seq=265 Ack=582 Win=30080 Len=0 TSval=4294955891 TSecr=4294966061
24	0.409174	192.168.55.4	192.168.55.3	SMTP	118	S: 250 OK id=1pxURA-0000Qz-78
25	0.415763	192.168.55.3	192.168.55.4	SMTP	96	C: QUIT
26	0.416337	192.168.55.4	192.168.55.3	SMTP	121	S: 221 server closing connection

Figura 27: Trazas de envio de correo SMTP.

Podemos ver primero un intercambio de mansajes DNS de tipo A para la resolución del nombre **smtp.sstt4896.org**. Al no conocer la IP del servidor de correo, necesita conocer la

IP del servidor SMTP. También se lleva a cabo una consulta DNS a un registro tipo AAAA de nuevo a **smtp.sstt4896.org**, por la IPv6. Y Se lleva a cabo una una conexión TCP.

Podemos ver los mensajes **HELO** de SMTP, done el cliente indica su IP al servidor. Y el servidor le responde con un **Server-Hello** con código 250 de confirmación y para que el cliente pueda continuar.

De seguido, el cliente indica su dirección de correo mediante el comando **MAIL FROM**, donde recibe un código 250 de confirmación.

El cliente indica cual quiere que sea el destinatario del correo mediante el comando **RCPT TO**, y de la misma forma, el servidor le devuelve un código 250 de confirmación.

Mediante el mansaje **DATA**, el cliente va a empezar a enviar datos, es decir, el contenido del correo, hasta que se indique **CRLF.CRLF** para indicar el final.

Por último, el cliente indica la finalización de la conexción TCP mediante el comando **QUIT**.

Figura 28: Contenido del correo.

Podemos ver que el asunto del correo es «Hola Alejandro», y el cuerpo del mismo es «Este mensaje es para la prueba de Wireshark».

Para ver que que **IPSec** esta funcionando correctamente, nos podemos dar cuenta de que además de tener las cabeceras de los distintios niveles de TCP, tenemos un nivel extra que se llama **Authentication Header**, que sirve para autenticar a las partes.

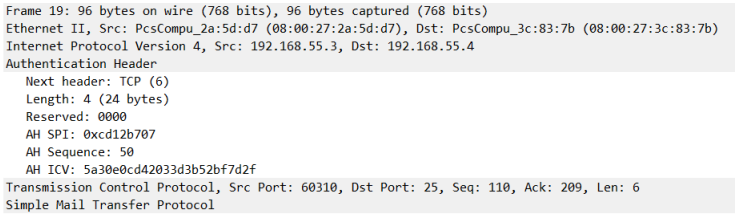
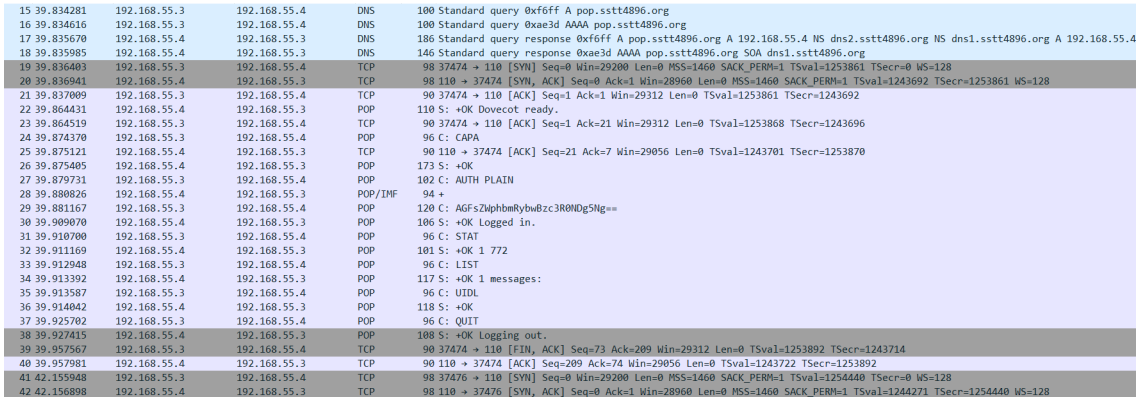


Figura 29: Cabecera Authentication Header.

8.4. DNS, POP e IPSec

Ahora vamos a abrir el correo desde «alejandro96@sstt4896.org» y mostrar su contenido. Para que un usuario pueda mostrar el contenido de sus correos, hará uso de POP.

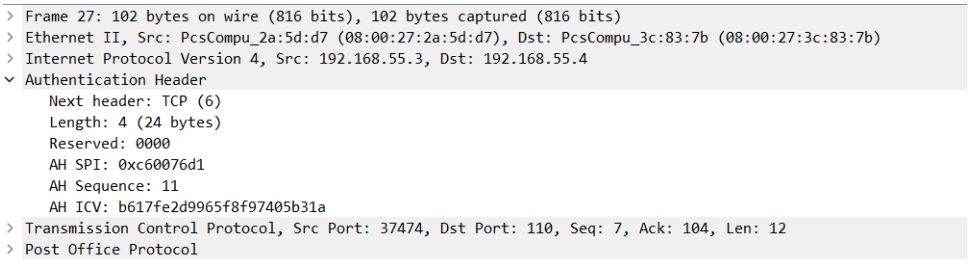


15	39.834281	192.168.55.3	192.168.55.4	DNS	100 Standard query 0xf6ff A pop.sstt4896.org
16	39.834616	192.168.55.3	192.168.55.4	DNS	100 Standard query 0xae3d AAAA pop.sstt4896.org
17	39.835670	192.168.55.4	192.168.55.3	DNS	186 Standard query response 0xf6ff A pop.sstt4896.org A 192.168.55.4 NS dns2.sstt4896.org NS dns1.sstt4896.org A 192.168.55.4
18	39.835985	192.168.55.4	192.168.55.3	DNS	146 Standard query response 0xae3d AAAA pop.sstt4896.org SOA dns1.sstt4896.org
19	39.836403	192.168.55.3	192.168.55.4	TCP	98 37474 → 110 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1253861 TSecr=0 WS=128
20	39.836941	192.168.55.4	192.168.55.3	TCP	98 110 → 37474 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1243692 TSecr=1253861 WS=128
21	39.837009	192.168.55.3	192.168.55.4	TCP	90 37474 → 110 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1253861 TSecr=1243692
22	39.864431	192.168.55.4	192.168.55.3	POP	110 S: +OK Dovecot ready.
23	39.864519	192.168.55.3	192.168.55.4	TCP	90 37474 → 110 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1253868 TSecr=1243696
24	39.874370	192.168.55.3	192.168.55.4	POP	96 C: CAPA
25	39.875121	192.168.55.4	192.168.55.3	TCP	90 110 → 37474 [ACK] Seq=21 Ack=7 Win=29056 Len=0 TSval=1243701 TSecr=1253870
26	39.875405	192.168.55.4	192.168.55.3	POP	173 S: +OK
27	39.879731	192.168.55.3	192.168.55.4	POP	102 C: AUTH PLAIN
28	39.880826	192.168.55.4	192.168.55.3	POP/IMF	94 +
29	39.881167	192.168.55.3	192.168.55.4	POP	120 C: AGFsZWphbmRybwBzc3R0NDg5Ng==
30	39.909070	192.168.55.4	192.168.55.3	POP	106 S: +OK Logged in.
31	39.910700	192.168.55.3	192.168.55.4	POP	96 C: STAT
32	39.911169	192.168.55.4	192.168.55.3	POP	101 S: +OK 1 772
33	39.912948	192.168.55.3	192.168.55.4	POP	96 C: LIST
34	39.913392	192.168.55.4	192.168.55.3	POP	117 S: +OK 1 messages:
35	39.913587	192.168.55.3	192.168.55.4	POP	96 C: UIDL
36	39.914042	192.168.55.4	192.168.55.3	POP	118 S: +OK
37	39.925702	192.168.55.3	192.168.55.4	POP	96 C: QUIT
38	39.927415	192.168.55.4	192.168.55.3	POP	108 S: +OK Logging out.
39	39.957567	192.168.55.3	192.168.55.4	TCP	90 37474 → 110 [FIN, ACK] Seq=73 Ack=209 Win=29312 Len=0 TSval=1253892 TSecr=1243714
40	39.957981	192.168.55.4	192.168.55.3	TCP	90 110 → 37474 [ACK] Seq=209 Ack=74 Win=29056 Len=0 TSval=1243722 TSecr=1253892
41	42.155948	192.168.55.3	192.168.55.4	TCP	98 37476 → 110 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1254440 TSecr=0 WS=128
42	42.156898	192.168.55.4	192.168.55.3	TCP	98 110 → 37476 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1244271 TSecr=1254440 WS=128

Figura 30: Trazas de recepción de correos POP.

Podemos ver primero un intercambio de mansajes DNS de tipo A para la resolución del nombre **pop.sstt4896.org**. Al no conocer la IP del servidor de correo, necesita conocer la IP del servidor SMTP. También se lleva a cabo una consulta DNS a un registro tipo AAAA de nuevo a **pop.sstt4896.org**, por la IPv6. Y Se lleva a cabo una una conexión TCP.

Una vez hecho esto el cliente se autentifica a través del comando **AUTH PLAIN** y el servidor lo confirma. Una vez el cliente autenticado con el comando **STAT** pide el número de mensajes sin leer y el tamaño de estos. Tras esto, el cliente con el comando **LIST** pide un listado de los mensajes y el servidor le devuelve esa lista.



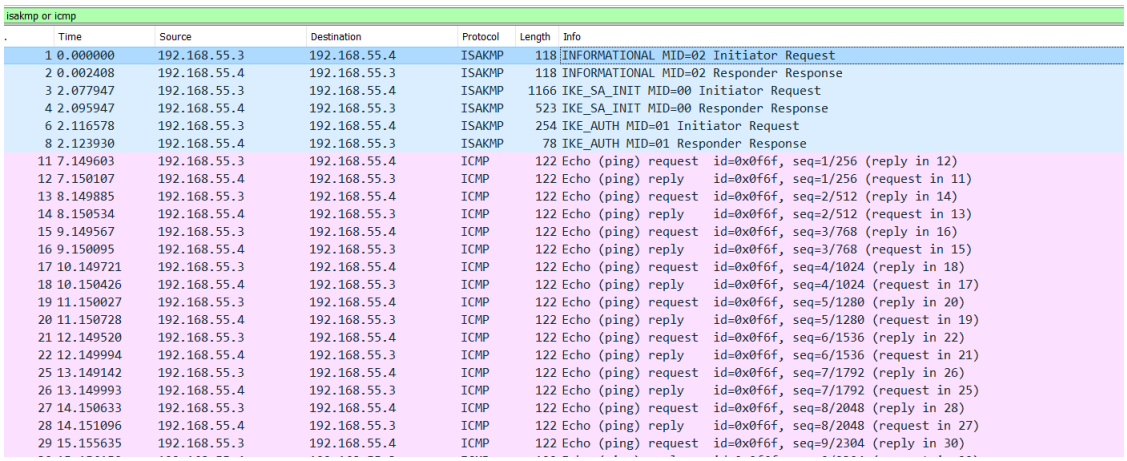
```
> Frame 27: 102 bytes on wire (816 bits), 102 bytes captured (816 bits)
> Ethernet II, Src: PcsCompu_2a:5d:d7 (08:00:27:2a:5d:d7), Dst: PcsCompu_3c:83:7b (08:00:27:3c:83:7b)
> Internet Protocol Version 4, Src: 192.168.55.3, Dst: 192.168.55.4
v Authentication Header
    Next header: TCP (6)
    Length: 4 (24 bytes)
    Reserved: 0000
    AH SPI: 0xc60076d1
    AH Sequence: 11
    AH ICV: b617fe2d9965f8f97405b31a
> Transmission Control Protocol, Src Port: 37474, Dst Port: 110, Seq: 7, Ack: 104, Len: 12
> Post Office Protocol
```

Figura 31: Cabecera Authentication Header.

De la misma forma, para ver que que **IPSec** esta funcionando correctamente, nos podemos dar cuenta de que además de tener las cabeceras de los distintios niveles de TCP, tenemos un nivel extra que se llama **Authentication Header**.

8.5. Intercambio IKE e IPsec

Vamos a generar las trazas reiniciando el servicio **Strongswan** para mostrar el intercambio IKE. De suguido se hace un ping al servidor.



isakmp or icmp

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.55.3	192.168.55.4	ISAKMP	118	INFORMATIONAL MID=02 Initiator Request
2	0.002408	192.168.55.4	192.168.55.3	ISAKMP	118	INFORMATIONAL MID=02 Responder Response
3	2.077947	192.168.55.3	192.168.55.4	ISAKMP	1166	IKE_SA_INIT MID=00 Initiator Request
4	2.095947	192.168.55.4	192.168.55.3	ISAKMP	523	IKE_SA_INIT MID=00 Responder Response
6	2.116578	192.168.55.3	192.168.55.4	ISAKMP	254	IKE_AUTH MID=01 Initiator Request
8	2.123930	192.168.55.4	192.168.55.3	ISAKMP	78	IKE_AUTH MID=01 Responder Response
11	7.149603	192.168.55.3	192.168.55.4	ICMP	122	Echo (ping) request id=0x0f6f, seq=1/256 (reply in 12)
12	7.150107	192.168.55.4	192.168.55.3	ICMP	122	Echo (ping) reply id=0x0f6f, seq=1/256 (request in 11)
13	8.149885	192.168.55.3	192.168.55.4	ICMP	122	Echo (ping) request id=0x0f6f, seq=2/512 (reply in 14)
14	8.150534	192.168.55.4	192.168.55.3	ICMP	122	Echo (ping) reply id=0x0f6f, seq=2/512 (request in 13)
15	9.149567	192.168.55.3	192.168.55.4	ICMP	122	Echo (ping) request id=0x0f6f, seq=3/768 (reply in 16)
16	9.150095	192.168.55.4	192.168.55.3	ICMP	122	Echo (ping) reply id=0x0f6f, seq=3/768 (request in 15)
17	10.149721	192.168.55.3	192.168.55.4	ICMP	122	Echo (ping) request id=0x0f6f, seq=4/1024 (reply in 18)
18	10.150426	192.168.55.4	192.168.55.3	ICMP	122	Echo (ping) reply id=0x0f6f, seq=4/1024 (request in 17)
19	11.150027	192.168.55.3	192.168.55.4	ICMP	122	Echo (ping) request id=0x0f6f, seq=5/1280 (reply in 20)
20	11.150728	192.168.55.4	192.168.55.3	ICMP	122	Echo (ping) reply id=0x0f6f, seq=5/1280 (request in 19)
21	12.149520	192.168.55.3	192.168.55.4	ICMP	122	Echo (ping) request id=0x0f6f, seq=6/1536 (reply in 22)
22	12.149994	192.168.55.4	192.168.55.3	ICMP	122	Echo (ping) reply id=0x0f6f, seq=6/1536 (request in 21)
25	13.149142	192.168.55.3	192.168.55.4	ICMP	122	Echo (ping) request id=0x0f6f, seq=7/1792 (reply in 26)
26	13.149993	192.168.55.4	192.168.55.3	ICMP	122	Echo (ping) reply id=0x0f6f, seq=7/1792 (request in 25)
27	14.150633	192.168.55.3	192.168.55.4	ICMP	122	Echo (ping) request id=0x0f6f, seq=8/2048 (reply in 28)
28	14.151096	192.168.55.4	192.168.55.3	ICMP	122	Echo (ping) reply id=0x0f6f, seq=8/2048 (request in 27)
29	15.155635	192.168.55.3	192.168.55.4	ICMP	122	Echo (ping) request id=0x0f6f, seq=9/2304 (reply in 30)

Figura 32: Trazas IKE e IPSec.

Podemos ver que omienza la negociación **IKEv2**, en este caso el papel de initiator lo toma el cliente. Este mensaje corresponde a la fase **IKE SA INIT** cuyo objetivo es el intercambio criptográfico mediante **Diffie-Hellman** para proteger los mensajes de la fase **IKE AUTH**. En este mensaje también se envéa el número aleatorio.

Comienza la fase de autenticación, donde el contenido va cifrado con la clave producida con **Diffie-Hellman**, en este mensaje el cliente se autentica y solicita la autenticación del servidor, además, se proponen unas suites criptográficas a negociar para ser utilizadas esta vez en IPsec y las políticas que han provocado esta negociación IKE.

En las anterias capturas de **Wireshark** podemos ver que **IPSec** esta funcionando correctamente en los demás servicios, ya que tenemos la cabecera **Authentication Header**.

9. Problemas encontrados

Los problemas con los que nos hemos podido encontar durante este cuatrimestre con respecto a las prácticas de esta asinatura no han sido verdaderamente problemas notables.

El problema más significativo es la aglomeración de todos los proyectos de todas las asignaturas como tal, pero es algo con lo que lidiamos desde el principio de la carrera.

Centrandonos en la práctica de esta asignatura, algo que nos preocupaba desde un principio era realizar el código de web_sstt.py en python ya que no es un lenguaje que hayamos practicado mucho, aun así nos resultó fácil ir cogiendo soltura con el. Python es una muy buena opción para realizar esta parte de la práctica porque es un lenguaje sencillo e intuitivo y nos facilita mucho el trabajo en algunos ámbitos. Al principio ibamos un poco perdidos con todas las cabeceras, pero con el material de teoría y las clases grabadas pudimos programar el servidor sin ningún problema.

Con los servicios de SMTP y POP tuvimos un pequeño problema al crear las cuentas en el espacio ThunderBird. Al encontrarnos este problema decimos hacerlo de forma manual, pero seguia sin funcionar. El problema estaba en que en uno de los ficheros de configuración habiamos comentado una linea sin darnos cuenta y por ese motivo surgian varios errores. Después de corregir ese pequeño "despiste" ya todo funcionaba correctamente.

Con respecto a apache HTTP no tuvimos ningún problema ya que es relativamente sencillo y no tuvimos ningún problema al realizarlo. Sin embargo, apache HTTPS si que nos ha resultado un poco más difícil a la hora de importar los certificados.

Finalmente, IPSec nos ha parecido sencillo gracias a las diapositivas puesto que están muy bien explicadas y son fáciles de seguir.

10. Número de horas empleadas

Vamos a dividir las horas dedicadas en cuatro apartados, siguiendo el formato de las verificaciones, con lo cual: V1, V2, V3 y documentación.

La primera parte de esta práctica, la V1, ha sido la más costosa a la hora de tiempo dedicado debido a lo que hemos mencionado anteriormente. Al tener que "recordar" a usar python y a entender toda la estructura del código hemos dedicado muchas horas a esta parte. Aproximadamente habremos empleado a la realización de esta práctica unas 17-18 horas. Son bastantes horas las que le hemos dedicado, pero si que es cierto que lo haciamos en pequeñas dosis y poco a poco con lo que con este método siempre se consume más tiempo por estar recordando que se hizo anteriormente.

A la parte de los servicios SMTP y POP si que le dedicamos más horas de lo normal en consecuencia del error del comentario del fichero que hemos mencionado antes. Le dimos muchas vueltas a lo mismo sin encontrar el error hasta que preguntamos en clase de prácticas y nos resolvió la duda. Las horas dedicadas oscilan alrededor de unas 9-10 horas.

La parte de la V3 ha sido sencillo ya que no nos han dado errores y como hemos dicho antes

es bastante fácil seguir las instrucciones de las diapositivas. Aproximamente hicimos esta parte la hicimos en poco más de una tarde, contabilizandolo en horas serían unas 7 horas.

Por último, la documentación la ibamos haciendo poco a poco mientras realizabamos la práctica aunque si que es cierto que hemos tenido que corregir y añadir muchas cosas en la recta final de esta. Para documentarlo todo, hemos tardado unas 8-9 horas.

En total, hemos tardado unas 44 horas en terminar la práctica. La práctica ha sido bastante densa en cuanto a contenidos asi que me parece un tiempo razonable.

11. Conclusiones y valoraciones personales

Como conclusión, nosotros dos estamos de acuerdo que es una de las prácticas más completas que hemos realizado hasta el momento. Está muy bien organizada y enlazada con la teoría. Esta asignatura está muy bien organizada con respecto a las clases de teoría y las clases de prácticas con respecto a contenidos. Otras asignaturas o las prácticas son totalmente diferentes a la teoría o las prácticas se dedican a reforzar la teoría. En este caso, la asignatura tiene un equilibrio ejemplar entre estas.

La práctica es bastante extensa, pero al ser en parejas la vemos bastante aceptable en cuanto a cantidad de trabajo. Pese a eso, hemos disfrutado la práctica porque es bastante visual y ves el resultado de lo que vas haciendo. Tambien, como hemos comentado antes, las diapositivas de la asignatura están muy bien explicadas y eso nos facilita mucho el trabajo.