Seguridad en Redes de Comunicación

Práctica 4. SSL / TLS

Jorge Lanza Calderón

Luis Sánchez González

Departamento de Ingeniería de Comunicaciones

PRÁCTICA 4
SSL / TLS

1. Objetivos de la Práctica

El desarrollo de la práctica pretende complementar el aprendizaje de los conceptos teóricos que se han visto durante las sesiones de teoría.

Los principales objetivos de esta práctica son:

- Analizar y comprender los mecanismos y protocolos involucrados en SSL.
- Monitorizar y analizar las distintas fases de una sesión SSL.
- Configurar un servidor HTTP (Apache) para acceso a servicios web seguros.

2. Introducción

Durante las sesiones teóricas se han descrito los aspectos fundamentales y las fases que componen una sesión TLS.

A lo largo de la práctica utilizaremos el marco que nos facilita OpenSSL para experimentar con el establecimiento y uso de sesiones TLS. Utilizaremos los comandos de OpenSSL que nos permiten emular un servidor y un cliente SSL/TLS genéricos. Variando los distintos parámetros de dichos comandos estudiaremos en detalle los distintos protocolos de TLS.

A la conclusión de la práctica experimentaremos con el uso de SSL en un servidor real HTTP. Apache es la implementación de servidor HTTP más usada en entornos Linux. Mediante la configuración de los parámetros básicos de este servidor aseguraremos el acceso a los servicios Web provistos por éste.

3. Desarrollo de la Práctica

La práctica se desarrollará en un entorno Linux, para lo cual los datos de acceso a la cuenta definidos son:

```
Nombre de usuario: Alumnos General
Contraseña: telematica
```

3.1 Servidor y cliente SSL/TLS genéricos

A lo largo de las prácticas anteriores hemos hecho uso de distintas herramientas que se aglutinan dentro de OpenSSL. En esta práctica usaremos las herramientas `s_server` y `s_client`. Se trata de un servidor TLS y un cliente TLS respectivamente. Los comandos:

```
$ man s_server
$ man s_client
```

nos permiten acceder a todas las opciones de ambos comandos respectivamente. También se puede acceder a esta información en https://www.openssl.org/docs/apps/s_server.html y https://www.openssl.org/docs/apps/s_client.html.

Utilizando ambos comandos con los parámetros adecuados generaremos diferentes sesiones TLS que monitorizaremos y estudiaremos. El comando básico (utilizando el mayor número de parámetros por defecto) para la ejecución del servidor TLS es:

```
$ sudo openssl s_server -accept 443 -no_ticket -cert certs/server.crt -key certs/server.key
```

- Utilizamos el puerto 443 (**`-accept 443`**) para facilitar el análisis de los protocolos dado que Wireshark sólo empleará los disectores de SSL si entiende que el tráfico está protegido por SSL cosa que asume cuando detecta segmentos TCP desde o hacia el puerto 443 (puerto bien conocido para HTTPS).
- Emplearemos tanto el certificado de servidor (**`-cert`**) como la clave privada (**`-key`**) que tenéis dentro del directorio `/home/alumnos/Asignaturas/SRC/ssl/certs`.

Ejecutando este comando el servidor quedará en espera:

```
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
```

Podéis comprobarlo mediante el comando `netstat` (utilizad las opciones adecuadas).

En otra terminal ejecutaremos el comando del cliente de forma que éste se conecte al servidor que acabamos de lanzar.

Antes de establecer la sesión TLS iniciaremos una captura de Wireshark para su posterior análisis.

El comando básico (utilizando el mayor número de parámetros por defecto) para la ejecución del cliente TLS es:

```
$ openssl s_client -no_ticket -connect localhost:443
```

- Lanza el cliente indicándole que se conecte al servidor que está escuchando en el puerto 443 del localhost (**`-connect localhost:443`**).

Tanto en la terminal del cliente como en la del servidor se muestra un log con información sobre el establecimiento de la sesión TLS y su resultado. **Capturad el intercambio de paquetes usando Wireshark y analizad el intercambio.**

¿Qué versión de TLS se emplea? ¿Cuál es la Cipher Suite escogida durante el handshake TLS? ¿Se usa compresión en TLS?

Si el establecimiento ha sido correcto, cliente y servidor estarán conectados y esto os permitirá enviar mensajes entre uno y otro. Cualquier cosa escrita en una terminal aparecerá en la otra.

En una conexión establecida hay una serie de caracteres especiales que resultan en acciones especiales (Sección CONNECTED COMMANDS en el manual). El envío de una `Q` en el servidor cierra tanto cliente como servidor (si se envía desde el cliente sólo para al cliente).

Parad cliente y servidor mediante este comando.

3.2 Verificación de certificados

3.2.1 Verificación de certificado de servidor

En el log que se muestra en la terminal del cliente podéis ver como el cliente ha recibido el certificado del servidor y al tratar de verificarlo el proceso resulta en una serie de errores:

```
depth=0 O = Simple, CN = TLS Server
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 O = Simple, CN = TLS Server
verify error:num=27:certificate not trusted
verify return:1
depth=0 O = Simple, CN = TLS Server
verify error:num=21:unable to verify the first certificate
verify return:1
```

El modo de funcionamiento básico de TLS permite la autenticación del servidor. En nuestro caso la cadena de certificación (PKI) del certificado de nuestro servidor está descrita en el log:

```
Certificate chain
 0 s:/O=Simple/CN=TLS Server
   i:/DC=org/DC=simple/O=Simple Inc/OU=Simple Signing CA/CN=Simple Signing CA
```

En realidad la PKI completa tiene un nivel más:

```
Root-CA
|
|--Signing CA
   |
   |--TLS Server
```

La primera opción que utilizaremos será la de verificar el certificado del servidor y actuar en función del resultado de esta verificación:

```
$ openssl s_client -no_ticket -connect localhost:443 -verify 2 -verify_return_error
```

- **`-verify`** especifica la longitud máxima de la cadena de certificación del servidor. Puede variar entre 0 y N. En nuestro caso 2 son los niveles de nuestra PKI por lo que resulta adecuado.
- **`-verify_return_error`** hace que ante cualquier fallo en la verificación no se continúe con la sesión TLS.

Tanto en la terminal del cliente como en la del servidor se muestra un log con información sobre el establecimiento de la sesión TLS y su resultado. **Capturad el intercambio de paquetes usando Wireshark y analizad el intercambio.**

¿Qué nuevo protocolo TLS aparece en la captura? ¿Cuál es el error por el que la sesión no se completa? ¿Quién envía el Record con el protocolo Alert?

¿Ocurriría algo distinto si variásemos el valor de la longitud de la cadena a otro valor menor que 2?¿Por qué?

- La opción **`-CAfile`** recibe el fichero que contiene el/los certificados de CA(s) confiables que se utilizan durante la verificación.

En el directorio `/home/alumnos/Asignaturas/SRC/ssl/ca` se encuentran los certificados tanto de la Root CA (`root-ca.crt`) como de la Signing CA (`signing-ca.crt`) por separado y en un único archivo en el cual se concatenan ambos (`signing-ca-chain.pem`).

¿Qué certificado utilizarías para que la verificación fuera correcta?

¿Podrías utilizar diferentes ficheros dependiendo del valor pasado en la opción **`-verify`**?

Completad el siguiente comando con el fichero adecuado:

```
$ openssl s_client -no_ticket -connect localhost:443 -verify 2
-verify_return_error -CAfile <Trusted CA(s) certificate>
```

Tanto en la terminal del cliente como en la del servidor se muestra un log con información sobre el establecimiento de la sesión TLS y su resultado.

Capturad el intercambio de paquetes usando Wireshark y analizad el intercambio.

¿Hay algún cambio en la captura respecto al establecimiento de sesión TLS en el que no se hacía la verificación?

3.2.2 Verificación de certificado de cliente. Autenticación mutua.

La verificación del servidor permite al usuario del servicio confiar en el proveedor pero TLS también permite la autenticación en el otro sentido con lo que sería posible establecer controles de acceso al servicio utilizando el propio TLS.

Detened el servidor y volved a lanzarlo con el comando:

```
$ sudo openssl s_server -accept 443 –no_ticket -cert certs/server.crt -key
certs/server.key -Verify 2 -verify_return_error
```

- La opción **`-Verify`** hace que el servidor solicite un certificado por parte del cliente fallando el establecimiento de la sesión TLS si éste no lo envía. La profundidad de verificación también es variable en este caso entre 0 y N. Al igual que en el caso de la verificación del servidor fijamos el valor a 2.

En la terminal del cliente, completad el siguiente comando con el certificado adecuado:

```
$ openssl s_client -no_ticket -connect localhost:443 -verify 2
-verify_return_error -CAfile <Trusted CA(s) certificate>
```

Tanto en la terminal del cliente como en la del servidor se muestra un log con información sobre el establecimiento de la sesión TLS y su resultado. **Capturad el intercambio de paquetes usando Wireshark y analizad el intercambio.**

¿Cuál es el error por el que la sesión no se completa?

¿Quién envía el Record con el protocolo Alert?

- Emplearemos tanto el certificado `fred.crt` (**`-cert`**) como la clave privada `fred.key` (**`-key`**) dentro del directorio `/home/alumnos/Asignaturas/SRC/ssl/certs`.
- El password que protege a la clave privada es **telematica**.

```
$ openssl s_client -no_ticket -connect localhost:443 -verify 2 -verify_return_error -CAfile <Trusted CA(s) certificate> -cert certs/fred.crt -key certs/fred.key
```

Tanto en la terminal del cliente como en la del servidor se muestra un log con información sobre el establecimiento de la sesión TLS y su resultado. **Capturad el intercambio de paquetes usando Wireshark y analizad el intercambio.**

¿Cuál es el error por el que la sesión no se completa?

¿Quién envía el Record con el protocolo Alert?

¿Qué hace falta para que se pueda completar el handshake completamente?

Resolved el problema, capturad el intercambio de paquetes usando Wireshark y analizad el intercambio.

¿Qué opción u opciones faltaban? ¿En qué lado, cliente o servidor?

¿Qué **`Trusted CA(s) certificate`** has usado?

¿Existe más de una posibilidad?

Si existieran diferentes soluciones, ¿en qué se diferencia el handshake cuando se elige cada una de ellas?

3.3 Negociación de Cipher Suites

¿Cuantás Cipher Suites es capaz de usar el cliente? ¿Cuantás Cipher Suites es capaz de usar el servidor?

- La opción **`-cipher`**, valida tanto en el servidor como en el cliente, permite definir las Cipher Suites que se pueden usar para la sesión TLS. En https://www.openssl.org/docs/apps/ciphers.html podéis encontrar todas las Cipher Suites disponibles en OpenSSL.

Si queremos que tanto cliente como servidor sólo puedan utilizar la Cipher Suite `SSL_RSA_WITH_NULL_MD5` entonces los comandos a emplear serán:

```
$ sudo openssl s_server -accept 443 -no_ticket -cert certs/server.crt -key certs/server.key -cipher NULL-MD5

$ openssl s_client -no_ticket -connect localhost:443 -cipher NULL-MD5
```

Es posible indicar más de una Cipher Suite concatenando los identificadores de éstas separados por ":". Así por ejemplo, **`NULL-MD5:DES-CBC3-SHA`** permitiría usar tanto `SSL_RSA_WITH_NULL_MD5` como `TLS_RSA_WITH_3DES_EDE_CBC_SHA`. El orden en el que se introducen los identificadores de cada Cipher Suite establece la prioridad de uso en caso de múltiples coincidencias entre la lista del cliente y la del servidor.

Ejecutad los comandos necesarios que os permitan establecer sesiones TLS que utilicen las siguientes Cipher Suites.

- TLS_RSA_WITH_NULL_MD5
- Intercambio de llaves basado en RSA, Cifrado con AES128 CBC e integridad basado en SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- Intercambio de llaves basado en Diffie-Hellman efímero autenticado con RSA, Cifrado con AES256 CBC e integridad basado en SHA. En este caso añadid al comando del cliente la opción **`-tls1`**.
- TLS_RSA_WITH_AES_256_CBC_SHA256

Como dijimos al principio, una vez establecida la sesión TLS correctamente, cualquier cosa escrita en una de las terminales aparece en la otra. Para cada sesión **enviad mensajes 3, 10 y 20 bytes de longitud** (hay que tener en cuenta que el carácter Nueva Linea se añade a todos los mensajes por lo que por ejemplo escribir `hola` equivaldría a enviar 5 bytes).

Capturad todos los intercambios de paquetes usando Wireshark y analizadlos.

¿Qué Records van cifrados? ¿Qué podéis decir de la longitud de los Records cifrados?

3.4 Acceso a un servidor HTTPS

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP.

La arquitectura del servidor Apache es muy modular. El servidor consta de una sección core y diversos módulos que aportan mucha de la funcionalidad que podría considerarse básica para un servidor web. Para el desarrollo de esta práctica, el principal módulo que vamos a emplear es `mod_ssl` que habilita comunicaciones seguras vía TLS.

La mayor parte de la configuración del servidor se realiza en los ficheros `/etc/apache2/apache2.conf` y `/etc/apache2/conf-available/httpd.conf`. En esta práctica trabajaremos sobre este último. Cualquier cambio en este archivo requiere reiniciar el servidor:

```
$ sudo service apache2 restart
```

El contenido del fichero `httpd.conf` en los PCs del laboratorio es:

```
<VirtualHost _default_:443>
 ServerAdmin webmaster@localhost

 DocumentRoot /var/www

 LogLevel warn
 ErrorLog /var/log/apache2/error.log
 CustomLog /var/log/apache2/ssl_access.log combined

 SSLEngine             on
 SSLCertificateFile    /home/alumnos/Asignaturas/SRC/ssl/certs/webserver.crt
 SSLCertificateKeyFile /home/alumnos/Asignaturas/SRC/ssl/certs/webserver.key

#SSLVerifyClient       require
#SSLVerifyDepth        10
#SSLCACertificateFile  /home/alumnos/Asignaturas/ssl/ca/signing-ca-chain.pem
</VirtualHost>
```

De las distintas opciones de configuración, las que aplican a esta práctica son:

```
SSLCertificateFile    /home/alumnos/Asignaturas/SRC/ssl/certs/webserver.crt
SSLCertificateKeyFile /home/alumnos/Asignaturas/SRC/ssl/certs/webserver.key
SSLVerifyClient       require
SSLCACertificateFile  /home/alumnos/Asignaturas/ssl/ca/signing-ca-chain.pem
```

en las que se especifica el certificado y la clave privada del servidor HTTPS; la necesidad de verificar al cliente; y la CA confiable para el servidor.

Inicialmente estas dos últimas están comentadas en el archivo de configuración por lo que las sesiones TLS que se establezcan para el acceso al servidor HTTPS sólo autenticarán al servidor.

3.4.1 Configuración del acceso básico al servidor HTTPS.

Abrid el navegador (se recomienda usar Firefox) y acceded a vuestro servidor tanto HTTP como HTTPS. Capturad el intercambio de paquetes usando Wireshark y analizad el intercambio.

¿Habéis tenido algún problema al acceder al servidor HTTPS?

¿A qué se han debido esos problemas?

Los navegadores suelen tener cargadas por defecto las principales autoridades de certificación. En nuestro caso, la PKI que hemos usado durante la práctica no será confiable de manera nativa para Firefox. Es posible configurar nuevas autoridades de certificación confiables para el navegador:

`Edit > Preferences > Advanced > Certificates > View Certificates > Authorities`

y ahí `Importar` el certificado de la CA confinable que se quiere añadir.

¿Qué certificado habéis importado para solucionar los problemas?

¿Se han solucionado los problemas añadiendo la CA?

Capturad el intercambio de paquetes usando Wireshark y analizad el intercambio.

3.4.2 Verificación de certificado de cliente. Autenticación mutua.

Una vez solucionados todos los problemas en el acceso básico HTTPS, editaremos el archivo `httpd.conf` y descomentaremos las líneas comentadas. Después reiniciaremos el servidor.

Abrid el navegador y acceded a vuestro servidor mediante HTTPS. Capturad el intercambio de paquetes usando Wireshark y analizad el intercambio.

¿Habéis tenido algún problema al acceder al servidor HTTPS?

¿A qué se han debido esos problemas?

También es posible cargar certificados de usuario en Firefox de forma que el navegador tenga con qué responder a las peticiones del servidor si las hubiera.

`Edit > Preferences > Advanced > Certificates > View Certificates > Your Certificates`

y ahí `Importar` el certificado de usuario (debe ser el certificado en formato PKCS12 que incluya tanto la clave privada como el certificado).

Capturad el intercambio de paquetes usando Wireshark y analizad el intercambio.