



# SSII

## PSI-SECTLS

El presente informe documenta los procedimientos seguidos y por consiguiente los resultados obtenidos para resolver las cuestiones del proyecto PSI-SECTLS, el cual está destinado a probar la seguridad aportada por el protocolo SSL/TLS.

Fco. Javier Delgado Vallano

03 de Marzo del 2015



# ÍNDICE

Historial de versiones.....	Pág. 2
Planificación.....	Pág. 2
Introducción.....	Pág. 3
Tarea 1.....	Pág. 4
Tarea 2.....	Pág. 11
Tarea 3.....	Pág. 17
Conclusión.....	Pág. 17

## HISTORIAL DE VERSIONES

Versión	Fecha	Descripción
1.0	03/03/2015	Creación y estructuración del documento, desarrollo de planificación e introducción.
1.1	04/03/2015	Desarrollo de tarea 1.
1.2	05/03/2015	Desarrollo de tarea 2, tarea3 y de la conclusión.
1.3	06/03/2015	Finalización del documento.

## PLANIFICACIÓN

Tareas	Tiempo estimado	Tiempo total	Tiempo desviación
Informe	8 h.	7 h.	1 h.
Tarea 1	3 h.	2.5 h.	0.5 h.
Tarea 2	4 h.	3 h.	1 h.
Tarea 3	0.5 h.	0.5 h.	0 h.
Reto	4 h.	X h.	X h.

## INTRODUCCIÓN

PSI-SECTLS es un proyecto que consiste en probar la seguridad aportada por el protocolo SSL/TLS en la comunicación cliente-servidor, desarrollando dos opciones, una sin seguridad y otra bajo el protocolo SSL/TLS. Para ello se han barajado las opciones de *Python* y *Java* para desarrollar una aplicación, debido a su numerosa lista de librerías y gran comunidad que tienen a sus espaldas, pero finalmente fue el primero el elegido debido a ventajas como son:

- Código dinámico.
- Mayor eficiencia con respecto a *Java*.
- No necesita clases predefinidas, por lo que el número de líneas de código se ve reducido.

Mediante estas opciones a desarrollar se intenta demostrar las diferencias existentes entre las comunicaciones seguras y las no seguras, comprobando de este modo la importancia de términos como la confidencialidad, autenticidad e integridad.

Además de esto se ha utilizado OpenSSL, para la realización de un *KeyStore*, mediante el cual la comunicación bajo TLS puede ser posible.

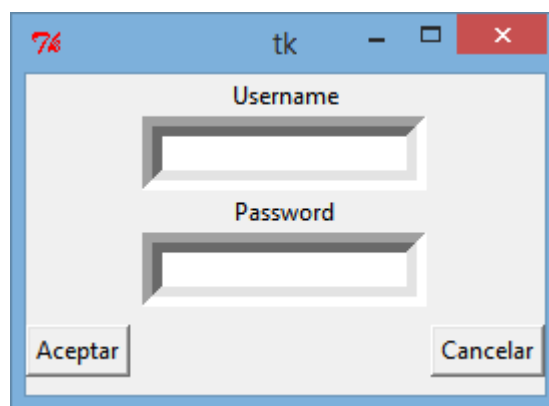
El proceso consiste en la emisión de mensajes, desde el cliente, que contengan un usuario y su contraseña; mientras que el servidor se encargará de verificar que estos son los correctos.

## TAREA 1

La primera tarea a realizar consiste en desarrollar una comunicación entre cliente-servidor mediante un canal no seguro.

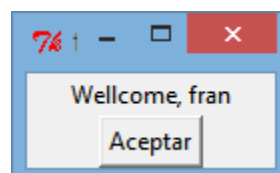
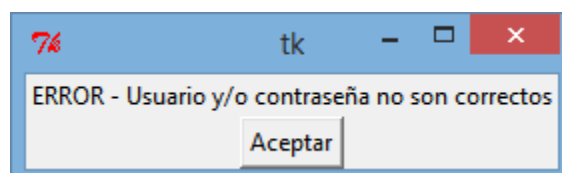
Para comenzar esta tarea se han realizado dos *scripts*, “cliente.py” y “servidor.py”, los cuales serán tanto emisor como receptor respectivamente. El cliente será el encargado de, tras establecer conexión con el *socket* del servidor, emitir mensajes en los que van contenidos un usuario y una contraseña que deben introducirse en una ventana que aparecerá al ejecutar el cliente.

A continuación podemos el formulario en el que debemos introducir estos datos:

A Tkinter window titled 'tk' with a blue title bar. It contains two text input fields, one labeled 'Username' and one labeled 'Password', both with a 3D effect. At the bottom, there are two buttons: 'Aceptar' on the left and 'Cancelar' on the right.

Una vez procesada la información emitida en el servidor, el cliente recibirá y mostrará una ventana en la que notificará de si el usuario y la contraseña son correctos, además de un mensaje de bienvenida en el caso de que la autenticación sea satisfactoria.

A continuación se muestran las imágenes de los mensajes que el sistema mostraría para cada uno de los casos, para el caso satisfactorio y para el fallido respectivamente.

A small Tkinter window titled 'tk' with a blue title bar. It displays the text 'Wellcome, fran' (note the typo) and has a single 'Aceptar' button at the bottom.A Tkinter window titled 'tk' with a blue title bar. It displays the text 'ERROR - Usuario y/o contraseña no son correctos' and has a single 'Aceptar' button at the bottom.

Finalmente, se muestra el código del *script* que toma la función de cliente, el cual tras haberse conectado a un *socket* (el del servidor) previamente especificado, envía al servidor el mensaje, y previo al cierre de la conexión recibe un mensaje del servidor notificando de si el usuario y contraseña introducidos son correctos.

```
# -*- coding: utf-8 -*-
import socket
from tkinter import *
import pickle

def cliente():

    def insertar():

        u = e1.get()
        p = e2.get()

        msg = [u, p]

        print ('enviando...\n ' + str(msg))

        sock.sendall(pickle.dumps(msg))

        root.destroy()

        return msg

    def cancelarPanel():

        root.destroy()

    # Creando un socket TCP/IP
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Conecta el socket en el puerto cuando el servidor esté escuchando
    server_address = ( 'Localhost', 4444)

    print ( 'conectando a ' + server_address[0] + ' y puerto ' +
str(server_address[1]))

    sock.connect(server_address)

    try:

        root = Tk()

        username = Label(root , text = "Username")
        e1 = Entry(root , bd = 10)

        password = Label(root , text = "Password")
        e2 = Entry(root , bd = 10)

        aceptar = Button (root , text = "Aceptar", command = insertar)
        cancelar = Button (root , text = "Cancelar", command = cancelarPanel
    )
```

```

username.pack()
e1.pack()
password.pack()
e2.pack()
aceptar.pack(side = LEFT)
cancelar.pack(side = RIGHT)

root.mainloop()

while True:

    data = sock.recv(1024).decode("utf_8")

    if (data):

        #amount_received += len(data)
        print ( 'recibiendo..\n' + str(data))

        root = Tk()

        msg = Label(root , text = str(data))
        accept = Button (text = "Aceptar" , command = cancelarPanel)

        msg.pack()
        accept.pack()

        root.mainloop()

        break

    break

finally:
    print ( 'cerrando socket\n\n' )
    sock.close()

if __name__ == '__main__':

    cliente()

```

En cuanto al servidor, el cual es el encargado de recibir dichos mensajes y de comprobar si la información recibida es la correcta, es decir, de si el usuario y contraseña recibidos coinciden con los que este mantiene almacenados. Por tanto, el servidor actúa de la siguiente manera:

- En primer lugar, tras haber recibido el mensaje, se procede a comprobar si los datos recibidos son correctos.
- A continuación, se comprueba que se haya realizado el paso anterior correctamente. Llegado a este punto, caben dos posibilidades, una en la que se haya realizado esto satisfactoriamente y otra en la que se ha producido un fallo y por tanto los datos recibidos no coinciden con los almacenados en el servidor.

En caso de que no se haya podido descifrar el mensaje, el servidor notificará al cliente con un mensaje como el siguiente:

*ERROR – Usuario y/o contraseña no son correctos*

Además de esto, se creará un archivo de texto en el que se almacene tanto el momento en el que el resultado fue negativo junto con el mensaje y la notificación pertinente.

En cambio, si dicho resultado es satisfactorio se notificará con:

*Wellcome, {Username}*

- Finalmente, el servidor envía al cliente la notificación de si el proceso ha sido o no satisfactorio.

A continuación se muestra el código del *script* que permitirá las funciones del servidor:

```
# -*- coding: utf-8 -*-
import socket
import datetime
from tkinter import *
import pickle

USERNAME = "fran"
PASSWORD = "fran"

def servidor():

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Enlace de socket y puerto
    server_address = ('localhost', 4444)
    print ( 'Empezando a Levantar ' + server_address[0] + ' puerto ' +
str(server_address[1]))
    sock.bind(server_address)

    # Escuchando conexiones entrantes
    sock.listen(1)

    recibidos = []

    while True:
        # Esperando conexion
        print ( 'Esperando para conectarse\n')
        connection, client_address = sock.accept()

        try:
            print ( 'conexion desde ' + client_address[0])

            # Recibe los datos en trozos y retransmite
            while True:
                data = connection.recv(1024)
```



```

d = pickle.loads(data)

recibidos.append(d)

#QUITAR
print ( 'recibido..\n' + str(d))

username = str(d[0])
password = str(d[1])

notifi = ""

if (USERNAME == username and PASSWORD == password):

    notifi = "Wellcome, " + username

else:

    outfile = open('error.txt', 'a')
    notifi = "ERROR - Usuario y/o contraseña no son  

correctos"

    out = str(datetime.datetime.now()) + " - " + str(d) + " - " + notifi + " \n"
    outfile.write(out)
    outfile.close()

if data:

    print ( 'Enviando mensaje de vuelta al cliente')
    connection.sendall(notifi.encode('utf_8'))
    print(notifi)

else:

    print ( 'No hay mas datos de ' + client_address[0])

    break

break

except EOFError:
    ret = []

finally:
    # Cerrando conexion

    connection.close()

if __name__ == '__main__':

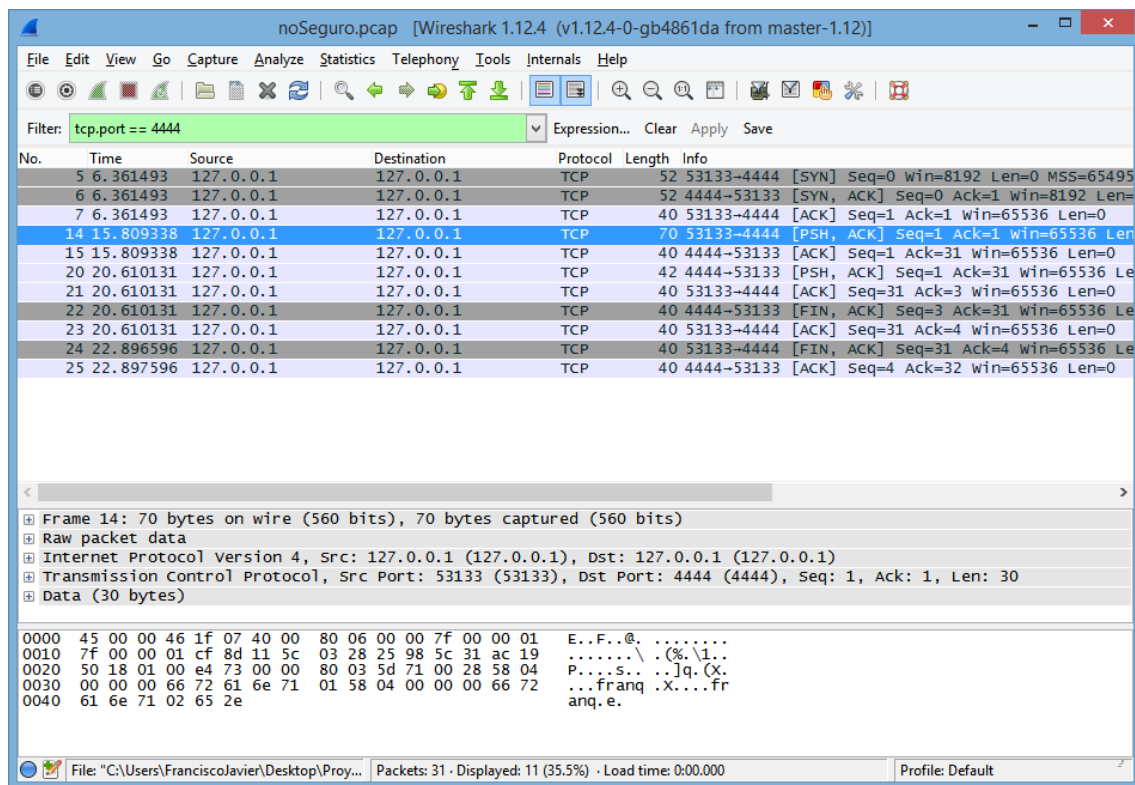
    servidor()

```

Tras realizar esto, se procede a realizar un análisis del tráfico de datos que fluye durante la comunicación entre cliente-servidor mediante la herramienta *RawCap* que nos devuelve un archivo que posteriormente se puede analizar con programas del estilo de *Wireshark*, el cual hemos utilizado en este caso.

Por tanto, al obtener dicho archivo comprobamos con *Wireshark* el resultado de la comunicación en cuestión.

A continuación se muestra la imagen del tráfico TCP recogido en la comunicación que se ha llevado a cabo entre cliente y servidor en *localhost* y puerto 4444.

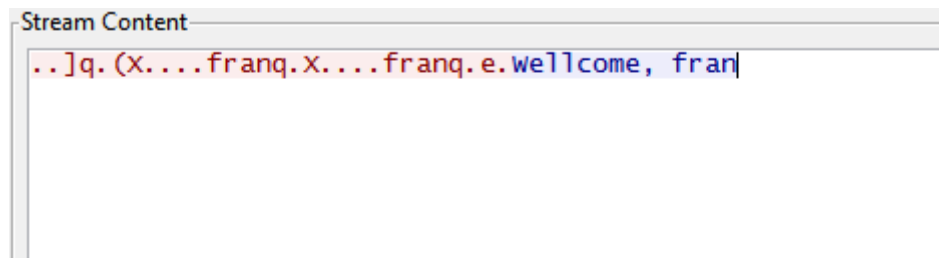


Al final de la imagen, podemos observar la sencillez con la que se puede obtener el usuario y contraseña en este caso en comunicaciones no seguras, ya que al final de la captura se muestran estos datos, los cuales son:

*Usuario: fran*

*Contraseña: fran*

Más claramente, podemos ver en la siguiente visualización dichos datos y el mensaje que envía el servidor para notificar del resultado que en este caso es satisfactorio.



```
..]q.(x....franq.x....franq.e.welcome, fran
```

The image shows a window titled "Stream Content" with a text area containing the string `..]q.(x....franq.x....franq.e.welcome, fran`. The text is displayed in a monospaced font with syntax highlighting: the opening bracket `[` is red, the closing bracket `]` is blue, and the rest of the string is black. The window has a standard title bar and a scroll bar on the right.

## TAREA 2

La segunda tarea propuesta para el presente proyecto, consiste en desarrollar otra comunicación cliente-servidor, pero en esta ocasión bajo un canal seguro, es decir, usando el protocolo SSL/TLS, el cual provee a dicha comunicación de autenticidad, confidencialidad e integridad.

Para ello se han realizado dos *scripts* similares a los expuestos en el caso anterior solo que estos difieren en que se realiza la implementación de *sockets* SSL bajo la librería *SSL* de *Python*.

Por tanto, el funcionamiento es el mismo y la implementación de esta comunicación también a excepción de las líneas de código que se han tenido de insertar.

A continuación podemos ver el código de ambos *scripts*:

- Para el cliente:

```
# -*- coding: utf-8 -*-
import socket
from tkinter import *
import pickle
import ssl
import pprint

def cliente():

    def insertar():

        u = e1.get()
        p = e2.get()

        msg = [u, p]

        print ( 'enviando...\n ' + str(msg))

        ssl_sock.sendall(pickle.dumps(msg))

        root.destroy()

        return msg

    def cancelarPanel():

        root.destroy()

    # Creando un socket TCP/IP
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    ssl_sock = ssl.wrap_socket(sock, ca_certs="server.crt",
    cert_reqs=ssl.CERT_REQUIRED, ssl_version = ssl.PROTOCOL_TLSv1)

    # Conecta el socket en el puerto cuando el servidor esté escuchando
```

```

server_address = ('Localhost', 4444)

print ('conectando a ' + server_address[0] + ' y puerto ' +
str(server_address[1]))

ssl_sock.connect(server_address)

#Muestra informacion de certificado
#print (repr(ssl_sock.getpeername()))
#print (ssl_sock.cipher())
#print (pprint.pformat(ssl_sock.getpeercert()))

try:

    root = Tk()

    username = Label(root , text = "Username")
    e1 = Entry(root , bd = 10)

    password = Label(root , text = "Password")
    e2 = Entry(root , bd = 10)

    aceptar = Button (root , text = "Aceptar", command = insertar)
    cancelar = Button (root , text = "Cancelar", command = cancelarPanel
)

    username.pack()
    e1.pack()
    password.pack()
    e2.pack()
    aceptar.pack(side = LEFT)
    cancelar.pack(side = RIGHT)

    root.mainloop()

while True:

    data = ssl_sock.recv(1024).decode("utf_8")

    if (data):

        print ( 'recibiendo..\n' + str(data))

        root = Tk()

        msg = Label(root , text = str(data))
        acept = Button (text = "Aceptar" , command = cancelarPanel)

        msg.pack()
        acept.pack()

        root.mainloop()

        break

    break

finally:
    print ('cerrando socket\n\n')

```

```

        ssl_sock.close()

if __name__ == '__main__':
    cliente()

```

○ Para el servidor:

```

# -*- coding: utf-8 -*-
import socket
import datetime
from tkinter import *
import pickle
import ssl

USERNAME = "fran"
PASSWORD = "fran"

def servidor():

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Enlace de socket y puerto
    server_address = ('localhost', 4444)
    print ( 'Empezando a Levantar ' + server_address[0] + ' puerto ' +
str(server_address[1]))
    sock.bind(server_address)

    # Escuchando conexiones entrantes
    sock.listen(1)

    recibidos = []

    while True:
        # Esperando conexion
        print ( 'Esperando para conectarse\n')
        connection, client_address = sock.accept()

        ssl_sock = ssl.wrap_socket(connection, server_side=True,
certfile="server.crt", keyfile="server.key", ssl_version =
ssl.PROTOCOL_TLSv1)

        try:
            print ( 'conexion desde ' + client_address[0])

            # Recibe los datos en trozos y retransmite
            while True:

                data = ssl_sock.recv(1024)

                d = pickle.loads(data)

                recibidos.append(d)

                print ( 'recibido..\n' + str(d))

```

```

username = str(d[0])
password = str(d[1])

if (USERNAME == username and PASSWORD == password):

    notifi = "Wellcome, " + username

else:

    outfile = open('error.txt', 'a')
    notifi = "ERROR - Usuario y/o contraseña no son  

correctos"

    out = str(datetime.datetime.now()) + " - " + str(d) + " -  

" + notifi + " \n"

    outfile.write(out)
    outfile.close()

if data:

    print ('Enviando mensaje de vuelta al cliente')
    ssl_sock.sendall(notifi.encode('utf_8'))
    print(notifi)

else:

    print ('No hay mas datos de ' + client_address[0])

    break

break

except EOFError:
    ret = []

finally:
    # Cerrando conexion

    ssl_sock.shutdown(socket.SHUT_RDWR)
    ssl_sock.close()

if __name__ == '__main__':

    servidor()

```

Cabe destacar que previamente a la finalización de estos *scripts* se ha realizado mediante *OpenSSL*, la generación de claves y certificado autofirmados para el servidor, consiguiendo de este modo el correcto funcionamiento de esta comunicación bajo el protocolo TLS.

Esto se ha hecho posible mediante la ejecución de las siguientes líneas de comandos en el *prompt*:

```
openssl genrsa -des3 -out server.orig.key 2048
```

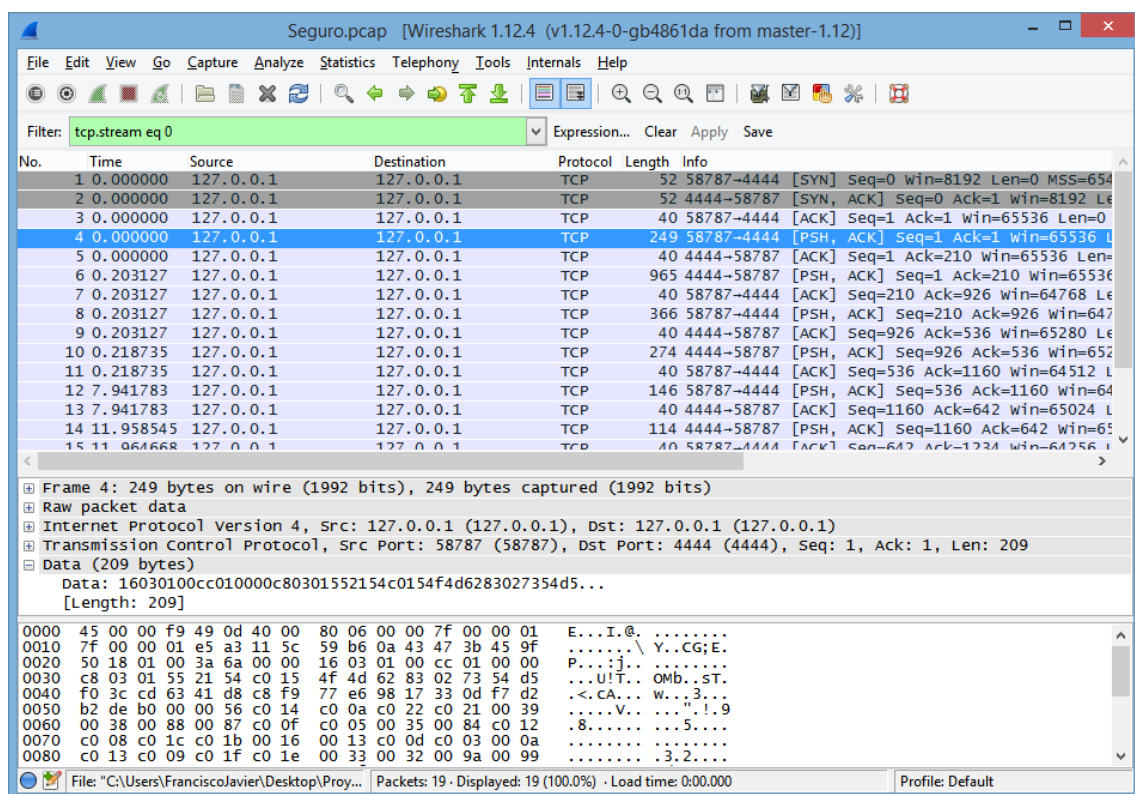
```
openssl rsa -in server.orig.key -out server.key
```

```
openssl req -new -key server.key -out server.csr
```

```
openssl x509 -req -days 365 -in -server.csr -signkey server.key -out server.crt
```

A la hora de generar el *server.key* ha hecho falta rellenar varios campos, los cuales corresponden al servidor, para que este en las comunicaciones bajo el protocolo SSL/TLS puede autenticarse y de este modo se pueda llevar a cabo una comunicación.

Llegados a este punto ya se puede realizar dicha comunicación de forma segura entre cliente-servidor, por lo que tras realizar un análisis del tráfico de red en *localhost* y en el puerto 4444, obtenemos los siguientes resultados.



Al contrario del caso anterior, ahora nos dificulta en mayor proporción la obtención del nombre de usuario y la contraseña, ya que como podemos observar al final de la imagen no podemos verificar que existan resultados semejantes a estos. Para corroborar esto, a continuación se muestra el resultado del flujo TCP.



## Stream Content

```

.....U!T..QMD..ST..<.CA...W...3
.....V...
"!.9.8.....5.....
...
.....3.2.....E.D.../...A.....I.....
.4.2...
.....
.....#.....:...6..U!T.e...:n.../p...i(H...0.)...5.....#.....P...L..I..FO..B0..*....O.q..Z-0
..*H..
.....0c1.0...U...ES1.0...U...Sevilla1.0...U...Sevilla1.0...U.
..US1
0...U...SSII1.0...U...FranVallano0..
150405145354Z.
160404145354Z0c1.0...U...ES1.0...U...Sevilla1.0...U...Sevilla1.0...U.
..US1
0...U...SSII1.0...U...FranVallano0.."0
..*H..
.....0..
.....{...f....|G.....\J.Z..c.g.Ve..p."k...B.O>..Z...q.r{%0CW7=..B.OU.X...G..7dA.GF&..}...}...}'.....|E....p.h.....&..'].S...Z...g.#d..
$......+...2..0....t8F.. 'V...Y.....b}/L.:N.B^5:....I..D.....e.
r.M...<.x...2...8T..>Z{..H.S!.....0
..*H..
.....'.C.=/vt....}.r.....K.cjvJ['...[/ 'y.h$C]...5.{...2AA.<34H.[...Y...H(1.dsc...<..C...SE...Od.\..
).!. ..h...C.y...?.Y./\J\.[...Ht...y...ueD.)...Q..Gt...%....0
.Z.>is./.....$;...6.=...<E..2.X.#q.v)...?.j.k...9P1.I..B...).e.v...3...X...."
K.....g1.H..Z...5...E"...G...:RX.5.^..."8 ..#.....>.+f.f...6...y...U.CBBI.:u.n0D.<...C{...`..o~#4.....30.k2...p.s
[T.....JY
.C.O.....a..Ra..].DX.B....j
.&.....cc%.....f.3u$*|7j..V....|p..10>.%G.....#...(. ..).....?MtS.....0.n.....<...jb.....?..)-FWi.j..1.|..1.v.Q@.....
N.U.h.5o1.0.5j=...x4X.....X3..m..*.....w.O... ..0...S`.....?..2.....1..UZ[
.:L(.Uw.....e.....t...HO..a..{.....%@
b...M)Gz.Q.....~)...(0)...!}!w_K..@*p.....CKQ...j.!E.9..A..Q(.J.._F$. ....19).....&v.@2..W..D...=>..&..}.H....@]/g7T.....S!.....k.wdo..a
$V4..)w.mXh;V...y.%...3V...4R.....S{.....M.d.b...8...9...w.....l.q.....!..SO...U...5|).pU.S..(..|

```

Finalmente podemos comprobar que los únicos datos visibles son los correspondientes al certificado que intercambia el servidor con el cliente. Hay que tener en cuenta que en el presente proyecto el servidor utiliza un *keyStore* compartido con el cliente cosa que realmente no tiene sentido si la comunicación se realizara bajo otra red que no fuese *localhost* sobre todo por temas de seguridad. Por tanto, en esos casos, en los que se utilizan dos máquinas distintas, tanto el servidor como el cliente deben contar con su propio *KeyStore*, de modo que sólo sea accesible por él.

## TAREA 3

La tercera tarea propuesta consiste en aportar un *feedback* en el que se aporten posibles mejoras que se puedan realizar al producto entregado.

Algunas de las mejoras aplicables pueden ser:

- Almacenar cifrados, los usuarios y contraseñas guardados en el servidor, en una base de datos.
- Modificar el servidor para que se inicie automáticamente con el inicio del sistema.
- Eliminar cada cierto tiempo los mensajes almacenados tanto en cliente como en servidor.
- Implementar técnicas de cifrado y verificadores de la integridad para los mensajes intercambiados.

## CONCLUSIÓN

Las comunicaciones como son en este caso, cliente-servidor, debemos implementar además de las técnicas aprendidas anteriormente como son aquellas que permiten garantizar tanto la confidencialidad y la integridad, debemos pensar en la autenticidad. Son 3 cosas que el protocolo TLS nos proporciona, pero no por ello debemos tenerlo como única opción.

Además de esto, cabe mencionar el aprendizaje de las técnicas existentes para implementar el protocolo SSL/TLS, así como su funcionamiento; e incluso también, el uso y funcionamiento de *OpenSSL*.