2020

SISTEMAS DISTRIBUIDOS

Sockets seguros

Murcia Gómez, Francisco Joaquín
48734281H

Universidad de Alicante

índice

Sockets seguros

Para la realización de esta práctica hemos segurizado los sockets de la practica 1 con la librería openSSL de C, SSl hace uso de certificados digitales para establecer comunicaciones seguras a través de Internet, dicha conexión es cifrada, en nuestro caso el servidor es el que otorga la clave de cifrado

SSL responde a dos principios:

- **El cifrado:** todos los datos que circulan entre cliente y servidor están cifrados por una clave
- **La autenticación:** se requiere de una autenticación (usuario y contraseña en nuestro caso) para poder empezar a intercambiar información con el servidor

El funcionamiento es muy simple:

1º) El cliente solicita el establecimiento de conexión

2º) El servidor envía el certificado y se establece una conexión segura

3º) El servidor pregunta por una autenticación

4º) El cliente se autentica

5º) Comienza la comunicación segura

- El servidor envía un mensaje cifrado
- El cliente lo descifra y le contesta con un mensaje cifrado
- El servidor lo descifra y le contesta con un mensaje cifrado

En nuestro caso usamos una clave de 2048 bits de openSSL que generamos con el siguiente comando(la clave es "1234"):

openssl req -x509 -days 365 -newkey rsa:2048 -keyout certServ.pem -out certServ.pem

Mi Código

Servidor

En primer lugar, comprobamos que el programa no se ejecuta como root, ya que sería una falta de seguridad grave

```
int main(int count, char *Argc[])
{
    SSL_CTX *ctx;
    int server;
    char *portnum;
    //Solo root/sudo puede ejecutar el servidor
    if(!isRoot())
    {
        printf("¡¡Este programa debe ser ejecutado por root!!");
        exit(0);
    }
    if ( count != 2 )
    {
        printf("Uso: %s <portnum>\n", Argc[0]);
        exit(0);
    }
```

A continuación, iniciamos el protocolo SSL abrimos el socket y realizamos la conexión con el cliente

```
// Inicialización de la librería SSL
SSL_library_init();
portnum = Argc[1];
ctx = InitServerCTX();        /* inicializa SSL */
LoadCertificates(ctx, "mycert.pem", "mycert.pem"); /* carga de certificados */
server = OpenListener(atoi(portnum));    /* creación del socket servidor */
while (1)
{
    struct sockaddr_in addr;
    socklen_t len = sizeof(addr);
    SSL *ssl;
    int client = accept(server, (struct sockaddr*)&addr, &len);  /* acepta conexiones entrantes */
    printf("Connection: %s:%d\n",inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));
    ssl = SSL_new(ctx);              /* obtención de contexto de conexión seguro */
    SSL_set_fd(ssl, client);      /* conversión del socket en socket seguro SSL */
    Servlet(ssl);         /* arranque del servicio de conexión */

}
close(server);          /* cierre del socket servidor */
SSL_CTX_free(ctx);         /* liberación de contexto seguro */
```

El módulo void Servlet(SSL* ssl){}, es donde administramos la conexión, en el enviamos y recibimos mensajes del cliente

Comprobamos que la credencial del cliente sean correctas

```
char buf[1024] = {0};
int sd, bytes;
const char* ServerResponse="ACCESO PERMITIDO. Pide tu deseo!!";
const char *cpValidMessage = "Usuario: SD   Contraseña: 12345678";
if ( SSL_accept(ssl) == FAIL )     /* do SSL-protocol accept */
    ERR_print_errors_fp(stderr);
else
{
    bytes = SSL_read(ssl, buf, sizeof(buf)); /* obtener respuesta */
    buf[bytes] = '\0';
    printf("Mensaje del cliente: \"%s\"\n", buf);
    if ( bytes > 0 )
    {
```

En caso afirmativo, enviaremos el menú y validaremos la opción elegida por el cliente

```
if(strcmp(cpValidMessage,buf) == 0)
{


    SSL_write(ssl, ServerResponse, strlen(ServerResponse)); /* envía respuesta */

    //enviamos el menu al cliente
    SSL_write(ssl,"¿Que desea hacer?\n[multiplicar] a*b\n[sumar] a+b\n[elevar] a^b\nEscriba la opcion: "
              , strlen("¿Que desea hacer?\n[multiplicar] a*b\n[sumar] a+b\n[elevar] a^b\nEscriba la opcion: "));


    bytes = SSL_read(ssl, buf, sizeof(buf));//obtenemos la operacion
    buf[bytes] = 0;


    int operacion=validar(buf);
```

Comprobamos que sea o multiplicar, sumar o elevar y solicitamos los operandos y mostramos el resultado

```
if(operacion!=-1){
    //introducimos los operadores
    SSL_write(ssl,"Introduce el operador 1: ", strlen("Introduce el operador 1: "));
    bytes = SSL_read(ssl, buf, sizeof(buf));
    buf[bytes] = 0;
    int op1=atoi(buf);
    SSL_write(ssl,"Introduce el operador 2: ", strlen("Introduce el operador 2: "));
    bytes = SSL_read(ssl, buf, sizeof(buf));
    buf[bytes] = 0;
    int op2=atoi(buf);
    int resul=realizarOperacion(operacion,op1,op2);
    char resultado[100]={0};
    sprintf(resultado,"El resultado es: %d \n",resul);

    SSL_write(ssl,resultado, strlen(resultado));//enviamos el resultado
```

Cliente

Cos conectamos al servidor con un método no seguro y solicitas una conexión segura

```
int main(int count, char *strings[])
{
    SSL_CTX *ctx;
    int server;
    SSL *ssl;
    char buf[1024];
    char acClientRequest[1024] = {0};
    int bytes;
    char *hostname, *portnum;
    if ( count != 3 )
    {
        printf("uso: %s <hostname> <portnum>\n", strings[0]);
        exit(0);
    }
    SSL_library_init();
    hostname=strings[1];
    portnum=strings[2];
    ctx = InitCTX();
    server = OpenConnection(hostname, atoi(portnum)); /* Conexión no-cifrada con servidor */
    ssl = SSL_new(ctx);       /* crea un nueva conexión SSL */
    SSL_set_fd(ssl, server);    /* asocia el descriptor no-cifrado a la conexión cifrada*/
    if ( SSL_connect(ssl) == FAIL )   /* conecta con el servidor de forma cifrada */
        ERR_print_errors_fp(stderr);
    else
    {
```

Nos autenticamos al servidor y ciframos los mensajes

```
    {
        char acUsername[16] = {0};
        char acPassword[16] = {0};
        const char *cpRequestMessage = "Usuario: %s   Contraseña: %s";
        printf("Introduce Usuario: ");
        scanf("%s",acUsername);
        printf("\nIntroduce Password: ");
        scanf("%s",acPassword);
        sprintf(acClientRequest, cpRequestMessage, acUsername,acPassword);   /* construye contestación */
        printf("\n\nAlgoritmos de cifrado: %s \n", SSL_get_cipher(ssl));
        SSL_write(ssl,acClientRequest, strlen(acClientRequest));   /* encripta y envía el mensaje */
        bytes = SSL_read(ssl, buf, sizeof(buf)); /* obtiene la respuesta y la desencripta */
        buf[bytes] = 0;
        printf("\nRecibido: \"%s\"\n", buf);
        if(!strcmp(buf,"NO SÉ DE QUÉ ME HABLAS. FUERA!!")){
        }else{
```

Recibimos el menú y le enviamos al servidor el topo de operación validado y los operandos

```
bytes = SSL_read(ssl, buf, sizeof(buf)); /* recibimos el menu */
buf[bytes] = 0;
printf(buf);

//eligimos el tipo de  operacion
char operacion[16]={0};
scanf("%s",operacion);
if(validar(operacion)==-1){
    printf("\nError, argumento incorrecto");
        exit(0);
}
SSL_write(ssl,operacion, strlen(operacion));


//operador1
bytes = SSL_read(ssl, buf, sizeof(buf));
buf[bytes] = 0;
printf(buf);
char op1[16]={0};
scanf("%s",op1);
SSL_write(ssl,op1, strlen(op1));
//operador2
bytes = SSL_read(ssl, buf, sizeof(buf));
buf[bytes] = 0;
printf(buf);
char op2[16]={0};
scanf("%s",op2);
SSL_write(ssl,op2, strlen(op2));


//resultado
bytes = SSL_read(ssl, buf, sizeof(buf));
buf[bytes] = 0;
printf(buf);
```

Por último, recibimos el resultado

```
    //resultado
    bytes = SSL_read(ssl, buf, sizeof(buf));
    buf[bytes] = 0;
    printf(buf);
```

Ejecución

Capturas de terminal

Para coompilar y ejecutar el programa en el cliente:

- gcc -Wall -o cliente cliente.c -L/usr/lib -lssl -lcrypto
- ./cliente 127.0.0.1 8090

Para coompilar y ejecutar el programa del servidor:

- gcc -Wall -o servidor servidor.c -L/usr/lib -lssl -lcrypto
- sudo ./servidor 8090

A continuación, mostrare el funcionamiento del Código

Ejecutamos el servidor con: ./servidor 8090

Ejecutamos el servidor con: ./ cliente 127.0.0.1 8090

1º) Introducimos el usuario "SD"

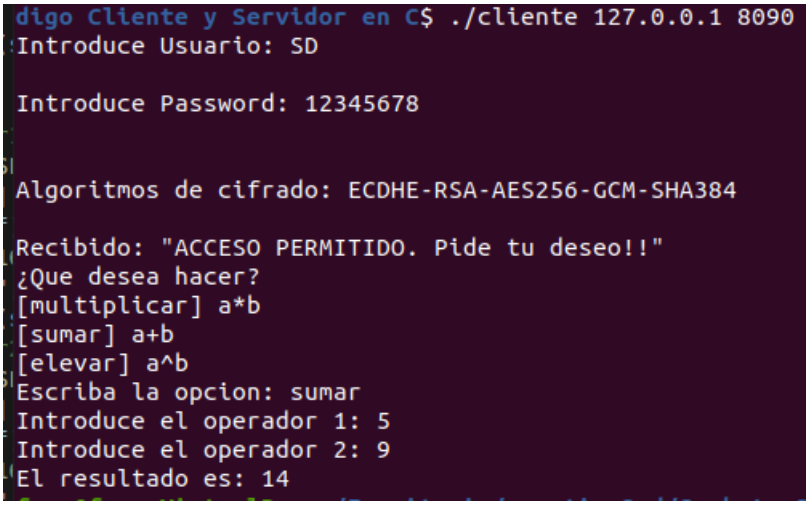
2º) Introducimos la contraseña "12345678"

3º) Escribimos el tipo de operación a realizar "sumar"

4º) Escribimos el operando 1 "5"

5º) Escribimos el operando 1 "9"

6º) Recibimos el resultado "14"

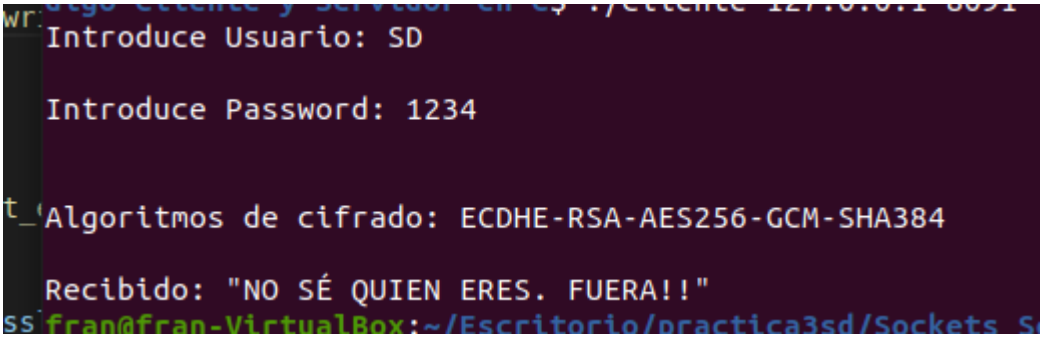


Si nos autenticamos más:

1º) Introducimos el usuario "SD"

2º) Introducimos la contraseña "1234"

3º) El servidor nos echa



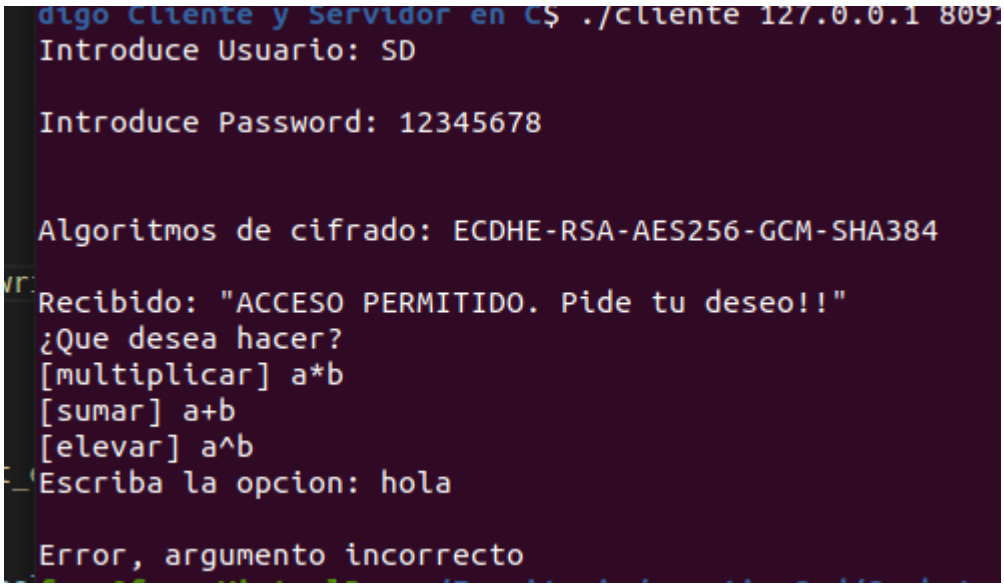
Si seleccionamos mal la operación:

1º) Introducimos el usuario "SD"

2º) Introducimos la contraseña "12345678"

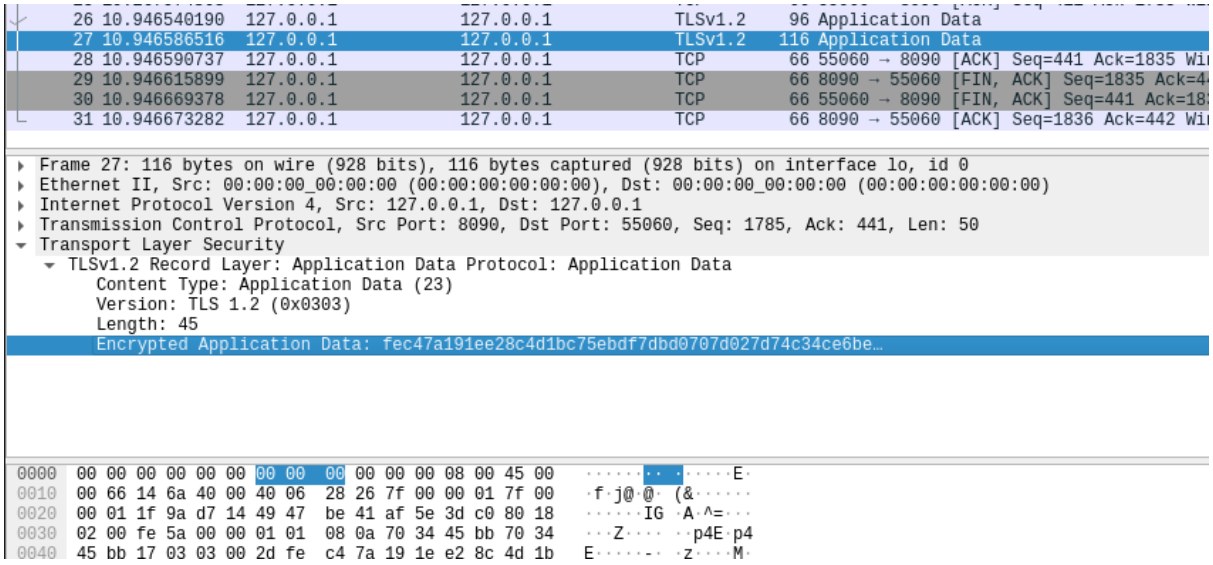
3º) Escribimos el tipo de operación a realizar "hola"

4º) Se detecta el error y se aborta

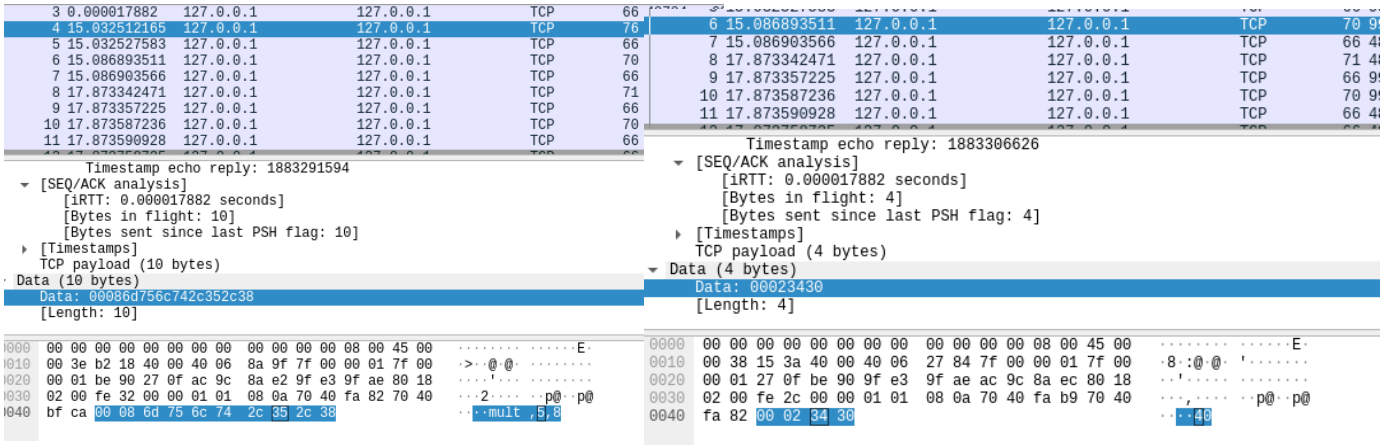


Capturas de paquetes

Si por ejemplo capturamos los paquetes con el programa de capturas de paquetes wireshark si inspecionamos los paquetes observamos que la informacion esta encriptada



Sin embargo, si capturamos los paquetes de la práctica anterior podremos ver los datos como los operadores (imagen de la izquierda) o el resultado (imagen de la derecha)



Como podemos deducir esto es una ganancia en seguridad importante ya que con un simple programa como wireshark pueden robar la información fácilmente