```c
//SSL-Server.c
#include <errno.h>
#include <unistd.h>
#include <malloc.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <resolv.h>
#include "openssl/ssl.h"
#include "openssl/err.h"

#define FAIL    -1

int OpenListener(int port)
{   int sd;
    struct sockaddr_in addr;

    sd = socket(PF_INET, SOCK_STREAM, 0);
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = INADDR_ANY;
    if ( bind(sd, (struct sockaddr*)&addr, sizeof(addr)) != 0 )
    {
        perror("can't bind port");
        abort();
    }
    if ( listen(sd, 10) != 0 )
    {
        perror("Can't configure listening port");
        abort();
    }
    return sd;
}

SSL_CTX* InitServerCTX(void)
{
    const SSL_METHOD *method;
    SSL_CTX *ctx;

    OpenSSL_add_all_algorithms();  /* load & register all cryptos, etc. */
    SSL_load_error_strings();   /* load all error messages */

    method = TLS_server_method();
    ctx = SSL_CTX_new(method);   /* create new context from method */
    if(ctx == NULL)
    {
        ERR_print_errors_fp(stderr);
        abort();
    }

    SSL_CTX_set_cipher_list(ctx, "ALL:eNULL");

    return ctx;
}

void LoadCertificates(SSL_CTX* ctx, char* CertFile, char* KeyFile)
{
    //New lines
```

```
62      if (SSL_CTX_load_verify_locations(ctx, CertFile, KeyFile) != 1)
63          ERR_print_errors_fp(stderr);
64
65      if (SSL_CTX_set_default_verify_paths(ctx) != 1)
66          ERR_print_errors_fp(stderr);
67      //End new lines
68
69      /* set the local certificate from CertFile */
70      if (SSL_CTX_use_certificate_file(ctx, CertFile, SSL_FILETYPE_PEM) <= 0)
71      {
72          ERR_print_errors_fp(stderr);
73          abort();
74      }
75      /* set the private key from KeyFile (may be the same as CertFile) */
76      SSL_CTX_set_default_passwd_cb_userdata(ctx, "12345678");
77      if (SSL_CTX_use_PrivateKey_file(ctx, KeyFile, SSL_FILETYPE_PEM) <= 0)
78      {
79          ERR_print_errors_fp(stderr);
80          abort();
81      }
82      /* verify private key */
83      if (!SSL_CTX_check_private_key(ctx))
84      {
85          fprintf(stderr, "Private key does not match the public certificate\n");
86          abort();
87      }
88
89      //New lines - Force the client-side have a certificate
90      //SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT,
NULL);
91      //SSL_CTX_set_verify_depth(ctx, 4);
92      //End new lines
93  }
94
95  void ShowCerts(SSL* ssl)
96  {
97      X509 *cert;
98      char *line;
99
100     cert = SSL_get_peer_certificate(ssl); /* Get certificates (if available) */
101     if ( cert != NULL )
102     {
103         printf("Server certificates:\n");
104         line = X509_NAME_oneline(X509_get_subject_name(cert), 0, 0);
105         printf("Subject: %s\n", line);
106         free(line);
107         line = X509_NAME_oneline(X509_get_issuer_name(cert), 0, 0);
108         printf("Issuer: %s\n", line);
109         free(line);
110         X509_free(cert);
111     }
112     else
113         printf("No certificates.\n");
114 }
115
116 void Servlet(SSL* ssl) /* Serve the connection -- threadable */
117 {
118     char buf[1024];
119     int sd, bytes;
120
121     char enter[3] = { 0x0d, 0x0a, 0x00 };
```

```
122
123     char output[1024];
124     strcpy(output, "HTTP/1.1 200 OK");
125     strcat(output, enter);
126     strcat(output, "Content-Type: text/html");
127     strcat(output, enter);
128     strcat(output, "Content-Length: 75");
129     strcat(output, enter);
130     strcat(output, enter);
131     strcat(output, "<html><body><h1>Hello World from https://localhost:5000!</h1>
</body></html>");
132
133     if ( SSL_accept(ssl) == FAIL )     /* do SSL-protocol accept */
134         ERR_print_errors_fp(stderr);
135     else
136     {
137         ShowCerts(ssl);        /* get any certificates */
138         bytes = SSL_read(ssl, buf, sizeof(buf)); /* get request */
139         if ( bytes > 0 )
140         {
141             buf[bytes] = 0;
142             printf("Client msg: \"%s", buf);
143             SSL_write(ssl, output, strlen(output)); /* send reply */
144         }
145         else
146             ERR_print_errors_fp(stderr);
147     }
148     sd = SSL_get_fd(ssl);       /* get socket connection */
149     SSL_free(ssl);         /* release SSL state */
150     close(sd);          /* close connection */
151 }
152
153 int main(int argc, char **argv)
154 {
155     SSL_CTX *ctx;
156     int server;
157     char portnum[]="5000";
158
159     char CertFile[] = "key/certificate.crt";
160     char KeyFile[] = "key/private_key.pem";
161
162     SSL_library_init();
163
164     ctx = InitServerCTX();        /* initialize SSL */
165     LoadCertificates(ctx, CertFile, KeyFile); /* load certs */
166     server = OpenListener(atoi(portnum));    /* create server socket */
167     while (1)
168     {
169         struct sockaddr_in addr;
170         socklen_t len = sizeof(addr);
171         SSL *ssl;
172
173         int client = accept(server, (struct sockaddr*)&addr, &len);  /* accept
connection as usual */
174         printf("Connection: %s:%d\n",inet_ntoa(addr.sin_addr),
ntohs(addr.sin_port));
175         ssl = SSL_new(ctx);              /* get new SSL state with context */
176         SSL_set_fd(ssl, client);      /* set connection socket to SSL state */
177         Servlet(ssl);         /* service connection */
178     }
179     close(server);          /* close server socket */
180     SSL_CTX_free(ctx);         /* release context */
```

```
}
```