ssl client in windows:

#include <iostream>

#include <stdio.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <memory.h>

#include <errno.h>

#include <sys/types.h>

#include <openssl/crypto.h>

#include <openssl/x509.h>

#include <openssl/pem.h>

#include <openssl/ssl.h>

#include <openssl/err.h>

#include <openssl/rand.h>

#define CHK\_NULL(x) if ((x)==NULL) exit (1)

#define CHK\_ERR(err,s) if ((err)==-1) { perror(s); exit(1); }

#define CHK\_SSL(err) if ((err)==-1) { ERR\_print\_errors\_fp(stderr); exit(2); }

using namespace std;

int main(int argc, char\* argv[])

{

int err = 0;

int sd;

struct sockaddr\_in sa;

SSL\_CTX\* ctx;

SSL\* ssl;

X509\* server\_cert;

char\* str;

char buf[4096];

const SSL\_METHOD \*meth;

SSLeay\_add\_ssl\_algorithms();

meth = TLSv1\_client\_method();

SSL\_load\_error\_strings();

ctx = SSL\_CTX\_new(meth); CHK\_NULL(ctx);

CHK\_SSL(err);

/\* Initialise Windows Sockets \*/

WORD wVersionRequested;

WSADATA wsaData;

wVersionRequested = MAKEWORD(2, 2);

err = WSAStartup(wVersionRequested, &wsaData);

if (err != 0) {

/\* Tell the user that we could not find a usable \*/

/\* WinSock DLL. \*/

return 0;

}

/\* Confirm that the WinSock DLL supports 2.2.\*/

/\* Note that if the DLL supports versions greater \*/

/\* than 2.2 in addition to 2.2, it will still return \*/

/\* 2.2 in wVersion since that is the version we \*/

/\* requested. \*/

if (LOBYTE(wsaData.wVersion) != 2 ||

HIBYTE(wsaData.wVersion) != 2) {

/\* Tell the user that we could not find a usable \*/

/\* WinSock DLL. \*/

WSACleanup();

return 0;

}

/\* The WinSock DLL is acceptable. Proceed. \*/

/\*---------------------------------------------\*/

/\* Initialize PRNG \*/

RAND\_screen();

/\* ----------------------------------------------- \*/

/\* Create a socket and connect to server using normal socket calls. \*/

sd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sd == -1)

{

err = WSAGetLastError();

cout << err << endl;

exit(err);

}

//CHK\_ERR(sd, "socket");

memset(&sa, '\0', sizeof(sa));

sa.sin\_family = AF\_INET;

inet\_pton(AF\_INET, "165.246.38.151", &(sa.sin\_addr.s\_addr)); // target IP

sa.sin\_port = htons(12194); /\* Server Port number \*/

err = connect(sd, (struct sockaddr\*) &sa,

sizeof(sa)); CHK\_ERR(err, "connect");

/\* ----------------------------------------------- \*/

/\* Now we have TCP conncetion. Start SSL negotiation. \*/

ssl = SSL\_new(ctx); CHK\_NULL(ssl);

SSL\_set\_fd(ssl, sd);

err = SSL\_connect(ssl); CHK\_SSL(err);

/\* Following two steps are optional and not required for

data exchange to be successful. \*/

/\* Get the cipher - opt \*/

printf("SSL connection using %s\n", SSL\_get\_cipher(ssl));

/\* Get server's certificate (note: beware of dynamic allocation) - opt \*/

server\_cert = SSL\_get\_peer\_certificate(ssl); CHK\_NULL(server\_cert);

printf("Server certificate:\n");

str = X509\_NAME\_oneline(X509\_get\_subject\_name(server\_cert), 0, 0);

CHK\_NULL(str);

printf("\t subject: %s\n", str);

free(str);

str = X509\_NAME\_oneline(X509\_get\_issuer\_name(server\_cert), 0, 0);

CHK\_NULL(str);

printf("\t issuer: %s\n", str);

free(str);

/\* We could do all sorts of certificate verification stuff here before

deallocating the certificate. \*/

X509\_free(server\_cert);

/\* --------------------------------------------------- \*/

/\* DATA EXCHANGE - Send a message and receive a reply. \*/

err = SSL\_write(ssl, "Hello World!", strlen("Hello World!")); CHK\_SSL(err);

err = SSL\_read(ssl, buf, sizeof(buf) - 1); CHK\_SSL(err);

buf[err] = '\0';

printf("Got %d chars:'%s'\n", err, buf);

SSL\_shutdown(ssl); /\* send SSL/TLS close\_notify \*/

/\* Clean up. \*/

closesocket(sd);

SSL\_free(ssl);

SSL\_CTX\_free(ctx);

WSACleanup();

printf("Hello World!\n");

return 0;

}