Win32 SChannel TLS message send (gross)

This works but it's super gross.

For TLS you have to Init like 3 times to finish the exchange and negotiation.

```
typedef struct {
    SOCKET socket;
    CredHandle hCredential;
    CtxtHandle hContext;
    char* ServerCertificate;
    unsigned int received;
    SecPkgContext_StreamSizes sizes;
} TLS_SOCKET;

// https://gist.github.com/mmozeiko/c0dfcc8fec527a90a02145d2cc0bfb6d
// Use SCHANNEL
int WINAPI WinMain(HINSTANCE  hInstance, HINSTANCE hPrevInstance, PSTR lpCmdLine, int nCmdShow){
    WORD wVersionRequired = MAKEWORD(2, 2);
    WSADATA wsaData;
    WSAStartup(wVersionRequired, &wsaData);
    // WS error codes https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-error-codes-2
    int result;
    // TLS_SOCKET;
    TLS_SOCKET tSocket = {0};
    tSocket.ServerCertificate = valloc(20000);

    // Create a socket
    tSocket.socket = WSASocketA(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0);
    if(tSocket.socket == INVALID_SOCKET){
        cprintf("Socket was invalid with error %ld\n", WSAGetLastError());
    }
```

```
    // Establish the peer
    struct sockaddr_in peer;
    peer.sin_family = AF_INET;
    peer.sin_addr.s_addr = inet_addr("127.0.0.1");
    peer.sin_port = htons(3333);

    // Connect to a peer.
    result = WSAConnect(tSocket.socket, (SOCKADDR*) &peer, sizeof(peer), NULL, NULL, NULL, NULL);
    if (result == SOCKET_ERROR) {
        result = WSAGetLastError();
        cprintf("WSAConnect returned error %ld\n", result);
    } else {
        cprintf("Connected.\n");
    }

    // Initialise SChannel
    // https://learn.microsoft.com/en-us/windows/win32/api/schannel/ns-schannel-schannel_cred
    // https://learn.microsoft.com/en-us/windows/win32/api/schannel/ns-schannel-sch_credentials
    SCHANNEL_CRED schannelCredential = {
        .dwVersion = SCHANNEL_CRED_VERSION,
        .dwFlags =  SCH_CRED_MANUAL_CRED_VALIDATION
        | SCH_CRED_NO_SERVERNAME_CHECK
        | SCH_CRED_NO_DEFAULT_CREDS,
        .grbitEnabledProtocols = SP_PROT_TLS1_2
    };



    // https://learn.microsoft.com/en-us/windows/win32/api/sspi/nf-sspi-acquirecredentialshandlea
     if (AcquireCredentialsHandleA(NULL, UNISP_NAME_A, SECPKG_CRED_OUTBOUND, NULL, &schannelCredential,
NULL, NULL, &tSocket.hCredential, NULL) != SEC_E_OK){
        WSACleanup();
        return -1;
    }



    CtxtHandle* context = NULL;
    int count = 0;
```

```
   // Perform the TLS Handshake.
    SecBuffer inbuffers[2] = { 0 };
    inbuffers[0].BufferType = SECBUFFER_TOKEN;
    inbuffers[0].pvBuffer = tSocket.ServerCertificate;
    inbuffers[0].cbBuffer = 0;
    inbuffers[1].BufferType = SECBUFFER_EMPTY;

    SecBuffer outbuffers[1] = { 0 };
    outbuffers[0].pvBuffer = halloc(40000);
    outbuffers[0].cbBuffer = 40000;
    outbuffers[0].BufferType = SECBUFFER_TOKEN;

    SecBufferDesc indesc = { SECBUFFER_VERSION, ARRAYSIZE(inbuffers), inbuffers };
    SecBufferDesc outdesc = { SECBUFFER_VERSION, ARRAYSIZE(outbuffers), outbuffers };

    DWORD flags = ISC_REQ_USE_SUPPLIED_CREDS | ISC_REQ_ALLOCATE_MEMORY | ISC_REQ_CONFIDENTIALITY |
ISC_REQ_REPLAY_DETECT | ISC_REQ_SEQUENCE_DETECT | ISC_REQ_STREAM;
    // https://learn.microsoft.com/en-us/windows/win32/api/sspi/nf-sspi-initializesecuritycontexta
        SECURITY_STATUS sec = InitializeSecurityContextA(
            &tSocket.hCredential,
            NULL,
            NULL,
            flags,
            0,
            0,
            NULL,
            0,
            &tSocket.hContext,
            &outdesc,
            &flags,
            NULL);

    //After the first round this will have a cert in it and the buffer type will become SECBUFFER_EXTRA
        if (sec == SEC_E_OK)
        {
            cprintf("OK!");
            char* buffer = "Hello world!";
            int size = 0;
            size= bstrlen(buffer);
            send(tSocket.socket, buffer, size, 0);
```

```
    // tls handshake completed
}
else if (sec == SEC_I_INCOMPLETE_CREDENTIALS)
{
    cprintf("INCOMPLETE");
    // server asked for client certificate, not supported here
    result = -1;
}
else if (sec == SEC_I_CONTINUE_NEEDED)
{
    // need to send data to server
    char* buffer = outbuffers[0].pvBuffer;
    int size = outbuffers[0].cbBuffer;
    cprintf("CONTINUE with data %s, %u\n", buffer, size);

    // DWORD dwBytes = 0;
    // char* buffer2 = valloc(50000);
    // dwBytes = recv(tSocket.socket, buffer2, 50000, 0);
    // cprintf("Recv %u bytes '%s'.\n", dwBytes, buffer2);

    while (size != 0)
    {
        int d = send(tSocket.socket, buffer, size, 0);
        if (d <= 0)
        {
            cprintf("Sent %u bytes.\n", d);
        }
        size -= d;
        buffer += d;
        cprintf("Sent %u bytes.\n", d);
    }
    // REceive the response
    int r = recv(tSocket.socket, tSocket.ServerCertificate, 4000, 0);
    cprintf("RECV  %u bytes.\n", r);
    inbuffers[0].cbBuffer = r;
    inbuffers[0].BufferType = SECBUFFER_TOKEN;


    //Request the TLS Certificate to negotiate
sec = InitializeSecurityContextA(
```

```
&tSocket.hCredential,
&tSocket.hContext,
NULL,
flags,
0,
0,
&indesc,
0,
&tSocket.hContext,
&outdesc,
&flags,
NULL);

cprintf("SEC: %x\n", sec);
if (sec == SEC_E_BUFFER_TOO_SMALL){
    cprintf("Buffers too small\n");
    // outbuffers[0].pvBuffer = var

}

if (sec != SEC_E_OK && sec != SEC_I_CONTINUE_NEEDED) {
    cprintf("InitializeSecurityContext failed: 0x%x\n", sec);
    return 1;
}

if (outbuffers[0].cbBuffer > 0) {
   send(tSocket.socket, (char *)outbuffers[0].pvBuffer, outbuffers[0].cbBuffer, 0);
}

if (sec == SEC_I_CONTINUE_NEEDED){
    //We need to negotiate further.
    //Receive into the inbuffer.
    cprintf("Further negoatiation.");
    // REceive the response
    int r = recv(tSocket.socket, tSocket.ServerCertificate, 4000, 0);
    cprintf("RECV  %u bytes.\n", r);
    inbuffers[0].cbBuffer = r;
    inbuffers[0].BufferType = SECBUFFER_TOKEN;
    sec = InitializeSecurityContextA(
        &tSocket.hCredential,
```

```
            &tSocket.hContext,
            NULL,
            flags,
            0,
            0,
            &indesc,
            0,
            &tSocket.hContext,
            &outdesc,
            &flags,
            NULL);

        cprintf("Sec= %x.\n", sec);
        if(sec == SEC_E_OK){
            cprintf("SEC_E_OK.\n");

        }

    }

    //FreeContextBuffer(outbuffers[0].pvBuffer);
    if (size != 0)
    {
        // failed to fully send data to server
        result = -1;
        cprintf("Oh");
    }
}
else if (sec != SEC_E_INCOMPLETE_MESSAGE)
{
    cprintf("INCMP_MSG");

    // SEC_E_CERT_EXPIRED - certificate expired or revoked
    // SEC_E_WRONG_PRINCIPAL - bad hostname
    // SEC_E_UNTRUSTED_ROOT - cannot vertify CA chain
    // SEC_E_ILLEGAL_MESSAGE / SEC_E_ALGORITHM_MISMATCH - cannot negotiate crypto algorithms
    result = -1;
}

if(sec == SEC_E_CERT_EXPIRED){
```

```
            cprintf("EXPIRED");
        }


        if(sec == SEC_E_WRONG_PRINCIPAL){
            cprintf("PRINCIPAl");
        }

        if(sec == SEC_E_ILLEGAL_MESSAGE || sec == SEC_E_ALGORITHM_MISMATCH){
            cprintf("ALGORITHM");
        }

        // if (sec == SEC_E_OK)
        // {
        //     cprintf("OK!");
        //     char* buffer = "Hello world!";
        //     int size = 0;
        //     size= bstrlen(buffer);
        //     send(tSocket.socket, buffer, size, 0);
        //     // tls handshake completed
        // }

        // int r = recv(tSocket.socket, tSocket.ServerCertificate + tSocket.received , 20000, 0);
        // if (r == 0)
        // {
        //     // server disconnected socket
        //     return 0;
        // }
        // else if (r < 0)
        // {
        //     // socket error
        //     result = -1;
        //     cprintf("SOCKET ERROR");
        //     break;
        // }
        // tSocket.received = r;
        // cprintf("RECEIVED %u", r);

// Get the sizes for the context
    // SECURITY_STATUS secStatus;
```

```
    SECURITY_STATUS secStatus = QueryContextAttributes(&tSocket.hContext, SECPKG_ATTR_STREAM_SIZES,
&tSocket.sizes);
    if (secStatus != SEC_E_OK) {
        cprintf("QueryContextAttributes failed: 0x%x\n", secStatus);
        char* data = valloc(10000);
        int r = recv(tSocket.socket, data, 10000, 0);
        char* a = "asd123";
        cprintf("Received %d bytes: '%s'", r, data);
        int k = send(tSocket.socket, a, 6, 0);
        cprintf("sent %d bytes: '%s'", r, data);
        return -1;
    } else {
        cprintf("SIZES MAX-MESSAGE: %d\n", tSocket.sizes.cbMaximumMessage);
        cprintf("SIZES HEAD-MESSAGE: %d\n", tSocket.sizes.cbHeader);
    }




//ENCRYPT AND SEND 'ASD"

    #define MAXIMUM_MESSAGE 20000
    SecBufferDesc Message;
    SecBuffer Buffers[4];

    // Prepare the message to be encrypted

    char* msg = valloc(MAXIMUM_MESSAGE);
    // Allocate the header
    Buffers[0].cbBuffer = tSocket.sizes.cbHeader;
    Buffers[0].pvBuffer = msg;
    Buffers[0].BufferType = SECBUFFER_STREAM_HEADER;

    // Allocate the data
    Buffers[1].pvBuffer = msg + tSocket.sizes.cbHeader;
    bmemcpy(msg + tSocket.sizes.cbHeader, "asd123", 7);
    Buffers[1].cbBuffer = 7;
    Buffers[1].BufferType = SECBUFFER_DATA;

    // Allocate a trailer?
```

```
    Buffers[2].BufferType = SECBUFFER_STREAM_TRAILER;
    Buffers[2].pvBuffer = msg + tSocket.sizes.cbHeader + 7;
    Buffers[2].cbBuffer = tSocket.sizes.cbTrailer;

    // NULL
    Buffers[3].BufferType = SECBUFFER_EMPTY;

    Message.ulVersion = SECBUFFER_VERSION;
    Message.cBuffers = 3;
    Message.pBuffers = Buffers;

    // Encrypt the message
    secStatus = EncryptMessage(&tSocket.hContext, 0, &Message, 0);
    if (secStatus == SEC_E_OK) {
        cprintf("EncryptMessage OK");
    } else if (secStatus == SEC_E_ENCRYPT_FAILURE){
        cprintf("EncryptMessage failed: 0x%x\n", secStatus);
        cprintf("EncryptMessage failed: SEC_E_ENCRYPT_FAILURE\n");
        if (secStatus == SEC_E_BUFFER_TOO_SMALL){
            cprintf("Small\n");
        }
        if (secStatus == SEC_E_INVALID_TOKEN){
            cprintf("TOKEN\n");
        }
        if (secStatus == SEC_E_CONTEXT_EXPIRED){
            cprintf("EXP\n");
        }
    }

    // Send the encrypted message
    int total = Buffers[0].cbBuffer + Buffers[1].cbBuffer + Buffers[2].cbBuffer;
    int sB = 0;
    while (sB < total) {
        sB += send(tSocket.socket, msg + sB, total - sB, 0);
    }
    cprintf("\nSB: %d, T: %d\n", sB, Buffers[0].cbBuffer + Buffers[1].cbBuffer + Buffers[2].cbBuffer );

    return 0;
}
```

Revision #1
Created 8 August 2024 23:16:47 by lepus
Updated 8 August 2024 23:17:34 by lepus