

SSPI Kerberos for delegation

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2008-12/msg00168.htm>

- *From:* Kasparov <ganesh.tambat@xxxxxxxxxx>
 - *Date:* Thu, 18 Dec 2008 08:21:39 -0800 (PST)
-

Hi,

We want the authentication to happen without providing credentials anywhere. For this we are planning to use Windows SSPI. Now the problem I am facing is once the authentication from client to server is successful, I want to use the

security context created in server to connect back and authenticate to the client. I am trying to find out if this thing is possible with SSPI.

For this I have written an application (prototype) with client side and server side. I am connecting to the server and after successfully authentication at the server side I try to connect back (to the server component on the client

machine, running in the same process and thread as client) using the same procedure I used in the client side. My understanding is since I have impersonated I will be able to authenticate the same user again. But I always get the error

"No credentials are available in the security package" and the credentials might have expired. I have verified that both the user accounts are trusted for delegation and "cannot be delegated" tab is unchecked. Also both the servers are

trusted for delegation for any service for Kerberos.

I have tried doing this with NTLM and Kerberos. With NTLM it looks like the reverse connection is successful but after impersonation at the client side if I try to print the user name it says "Anonymous Logon". With Kerberos I am

getting various errors (not enough memory, credentials are expired etc) and I am still working on that part. The reason I am trying with Kerberos is that I learned from some articles on Net that such a thing (delegation) is possible only

SSPI Kerberos for delegation

with Kerberos.

Surprisingly I am not able to find out a single working solution of this kind. I have pasted my entire code below. Can anyone help me on this and advise me on how this can be achieved ? Please see the code samples below:

Client (Logged in user: win\gtambat)

```
*****
```

```
#include <stdio.h>
#include <windows.h>
#include <string.h>
#include <malloc.h>
```

```
#define SECURITY_WIN32 1
#include <security.h>
```

```
#pragma hdrstop
```

```
//
```

```
=====
// IMPORTANT NOTICE --- Check out
// http://www.mvps.org/security/sspi.html
// for the gory details of how this works
//
=====
```

```
void auth( SOCKET s, CredHandle& cred, CtxtHandle& cliCtx,
const char *tokenSource, const char *name = NULL,
const char *pwd = NULL, const char *domain = NULL );
```

```
PSecurityFunctionTable pf = NULL;
```

```
void initSecLib( HINSTANCE& hSec );
```

```
bool Ganesh_auth( SOCKET s, CredHandle& cred, CtxtHandle& srvCtx )
```

```
{
```

```
int rc;
```

```
bool haveToken = true;
```

```
SecPkgInfo *secPackInfo;
```

```
int bytesReceived = 0, bytesSent = 0;
```

```
char buf[256];
```

```
DWORD bufsiz = sizeof buf;
```

```
HANDLE threadRet;
```

```
puts( "auth() entered" );
```

```
rc = (pf->QuerySecurityPackageInfo)( "Kerberos", &secPackInfo );
```

```
printf( "QSPI(): %08xh\n", rc );
```

SSPI Kerberos for delegation

SSPI Kerberos for delegation

```
if ( rc != SEC_E_OK )
haveToken = false;

TimeStamp useBefore;

rc = (pf->AcquireCredentialsHandle)( NULL, "Kerberos",
SECPKG_CRED_BOTH,
NULL, NULL, NULL, NULL, &cred, &useBefore );

rc = GetLastError();

printf( "ACH(): %08xh\n", rc );
if ( rc != SEC_E_OK )
haveToken = false;

// input and output buffers
SecBufferDesc obd, ibd;
SecBuffer ob, ib;

DWORD ctxAttr;

bool haveContext = false;

while ( 1 )
{
// prepare to get the server's response
ibd.ulVersion = SECBUFFER_VERSION;
ibd.cBuffers = 1;
ibd.pBuffers = &ib; // just one buffer
ib.BufferType = SECBUFFER_TOKEN; // preping a token here

// receive the client's POD
// MACHINE-DEPENDENT CODE! (Besides, we assume that we
// get the length with a single read, which is not guaranteed)
recv( s, (char *) &ib.cbBuffer, sizeof ib.cbBuffer, 0 );
bytesReceived += sizeof ib.cbBuffer;
ib.pvBuffer = LocalAlloc( 0, ib.cbBuffer );

char *p = (char *) ib.pvBuffer;
int n = ib.cbBuffer;
while ( n )
{
rc = recv( s, p, n, 0 );
// if ( rc == SOCKET_ERROR )
// wserr( rc, "recv" );
// if ( rc == 0 )
// wserr( 999, "recv" );
bytesReceived += rc;
n -= rc;
p += rc;
}
}
```

SSPI Kerberos for delegation

```
// by now we have an input buffer

obd.ulVersion = SECBUFFER_VERSION;
obd.cBuffers = 1;
obd.pBuffers = &ob; // just one buffer
ob.BufferType = SECBUFFER_TOKEN; // preping a token here
ob.cbBuffer = secPackInfo->cbMaxToken;
ob.pvBuffer = LocalAlloc( 0, ob.cbBuffer );

// rc = (pf->AcceptSecurityContext)( &cred, haveContext? &srvCtx:
NULL,
// &ibd, 0, SECURITY_NATIVE_DREP, &srvCtx, &obd, &ctxAttr,
// &useBefore );

rc = (pf->AcceptSecurityContext)( &cred, haveContext? &srvCtx: NULL,
&ibd, 0, SECURITY_NATIVE_DREP /*SECURITY_NETWORK_DREP*/, &srvCtx,
&obd, &ctxAttr,
&useBefore );

printf( "ASC(): %08xh\n", rc );

if ( ib.pvBuffer != NULL )
{
LocalFree( ib.pvBuffer );
ib.pvBuffer = NULL;
}

if ( rc == SEC_I_COMPLETE_AND_CONTINUE || rc ==
SEC_I_COMPLETE_NEEDED )
{
if ( pf->CompleteAuthToken != NULL ) // only if implemented
(pf->CompleteAuthToken)( &srvCtx, &obd );
if ( rc == SEC_I_COMPLETE_NEEDED )
rc = SEC_E_OK;
else if ( rc == SEC_I_COMPLETE_AND_CONTINUE )
rc = SEC_I_CONTINUE_NEEDED;
}

// send the output buffer off to the server
// warning --- this is machine-dependent! FIX IT!
if ( rc == SEC_E_OK || rc == SEC_I_CONTINUE_NEEDED )
{
if ( ob.cbBuffer != 0 )
{
send( s, (const char *) &ob.cbBuffer, sizeof ob.cbBuffer, 0 );
bytesSent += sizeof ob.cbBuffer;
send( s, (const char *) ob.pvBuffer, ob.cbBuffer, 0 );
bytesSent += ob.cbBuffer;
}
LocalFree( ob.pvBuffer );
```

SSPI Kerberos for delegation

```
ob.pvBuffer = NULL;
}

if ( rc != SEC_I_CONTINUE_NEEDED )
break;

haveContext = true;

// loop back for another round
puts( "looping" );
}

// we arrive here as soon as InitializeSecurityContext()
// returns != SEC_I_CONTINUE_NEEDED.

if ( rc != SEC_E_OK )
{
printf( "Oops! ASC() returned %08xh!\n", rc );
haveToken = false;
}

rc = GetLastError();

GetUserName( buf, &bufsiz );

// now we try to use the context
rc = (pf->ImpersonateSecurityContext)( &srvCtx );
printf( "ImpSC(): %08xh\n", rc );
if ( rc != SEC_E_OK )
{
printf( "Oops! ImpSC() returns %08xh!\n", rc );
haveToken = false;
}
else
{
char buf[256];
DWORD bufsiz = sizeof buf;
GetUserName( buf, &bufsiz );
printf( "user name: \"%s\"\n", buf );
(pf->RevertSecurityContext)( &srvCtx );
printf( "RSC(): %08xh\n", rc );
}

(pf->FreeContextBuffer)( secPackInfo );

printf( "auth() exiting (%d received, %d sent)\n", bytesReceived,
bytesSent );
return haveToken;
}

int Ganesh_Server_Impl()
```

SSPI Kerberos for delegation

```
{
int rc, port = 12000, addrlen;
bool haveToken;
SOCKET sock, s;
WSADATA wsadata;
PSEVENT pse;
SOCKADDR_IN addr;
HINSTANCE hSecLib;

initSecLib( hSecLib );

rc = WSASStartup( 2, &wsadata );
// wserr( rc, "WSASStartup" );

sock = socket( AF_INET, SOCK_STREAM, 0 );
if ( sock == INVALID_SOCKET )
{
rc = -1;
}
else
{
rc = 0;
}
// wserr( 999, "socket" );

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;

// try numeric protocol first
if ( port > 0 && port < 32768 )
addr.sin_port = htons( (short) port );
else
{
}

rc = bind( sock, (SOCKADDR *) &addr, sizeof addr );
// wserr( rc, "bind" );

rc = listen( sock, 2 );
// wserr( rc, "listen" );

CredHandle cred;
CtxHandle srvCtx;

while ( 1 )
{
addrLen = sizeof addr;
s = accept( sock, (SOCKADDR *) &addr, &addrLen );
if ( s == INVALID_SOCKET )
{
rc = -1;
}
```

SSPI Kerberos for delegation

```
}
else
{
rc = 0;
}
// wserr( s, "accept" );

haveToken = Ganesh_auth( s, cred, srvCtx );

// now we talk to the client
printf( "haveToken = %s\n\n", haveToken? "true": "false" );
send( s, (const char *) &haveToken, sizeof haveToken, 0 );

// clean up
(pf->DeleteSecurityContext)( &srvCtx );
(pf->FreeCredentialHandle)( &cred );
closesocket( s );
}
}

// wserr() displays winsock errors and aborts. No grace there.
void wserr( int rc, const char * const funcname )
{
if ( rc == 0 )
return;

fprintf( stderr, "\nWinsock error %d [gle %d] returned by %s().\n"
"Sorry, no bonus!\n", rc, WSAGetLastError(), funcname );
WSACleanup();
exit( rc );
}

void auth( SOCKET s, CredHandle& cred, CtxtHandle& cliCtx,
const char *tokenSource, const char *name /* = NULL */,
const char *pwd /* = NULL */, const char *domain /* = NULL */)
{
int rc, rcISC;
SecPkgInfo *secPackInfo;
SEC_WINNT_AUTH_IDENTITY *nameAndPwd = NULL;
int bytesReceived = 0, bytesSent = 0;

puts( "auth() entered" );

// the arguments to ISC() is not const ... for once, I decided
// on creating writable copies instead of using a brutalizing cast.
char *myTokenSource;
myTokenSource = _strdup( tokenSource );

if ( name != NULL )
```

SSPI Kerberos for delegation

```
{
nameAndPwd = (SEC_WINNT_AUTH_IDENTITY *) malloc( sizeof
SEC_WINNT_AUTH_IDENTITY );
memset( nameAndPwd, '\0', sizeof *nameAndPwd );
nameAndPwd->Domain = (byte *) _strdup( domain? domain: "" );
nameAndPwd->DomainLength = domain? strlen( domain ): 0;
nameAndPwd->User = (byte *) _strdup( name? name: "" );
nameAndPwd->UserLength = name? strlen( name ): 0;
nameAndPwd->Password = (byte *) _strdup( pwd? pwd: "" );
nameAndPwd->PasswordLength = pwd? strlen( pwd ): 0;
nameAndPwd->Flags = SEC_WINNT_AUTH_IDENTITY_ANSI;
}
```

```
rc = (pf->QuerySecurityPackageInfo)( "Kerberos", &secPackInfo );
printf( "QSPI(): %08xh\n", rc );
```

```
TimeStamp useBefore;
```

```
rc = (pf->AcquireCredentialsHandle)( NULL, "Kerberos",
SECPKG_CRED_BOTH,
NULL, nameAndPwd, NULL, NULL, &cred, &useBefore );
printf( "ACH(): %08xh\n", rc );
```

```
// input and output buffers
SecBufferDesc obd, ibd;
SecBuffer ob, ib;
```

```
DWORD ctxReq, ctxAttr;
```

```
// from sspi.h:
// #define ISC_REQ_DELEGATE 0x00000001
// #define ISC_REQ_MUTUAL_AUTH 0x00000002
// #define ISC_REQ_REPLAY_DETECT 0x00000004
// #define ISC_REQ_SEQUENCE_DETECT 0x00000008
// #define ISC_REQ_CONFIDENTIALITY 0x00000010
// #define ISC_REQ_USE_SESSION_KEY 0x00000020
// #define ISC_REQ_PROMPT_FOR_CREDS 0x00000040
// #define ISC_REQ_USE_SUPPLIED_CREDS 0x00000080
// #define ISC_REQ_ALLOCATE_MEMORY 0x00000100
// #define ISC_REQ_USE_DCE_STYLE 0x00000200
// #define ISC_REQ_DATAGRAM 0x00000400
// #define ISC_REQ_CONNECTION 0x00000800
// #define ISC_REQ_CALL_LEVEL 0x00001000
// #define ISC_REQ_EXTENDED_ERROR 0x00004000
// #define ISC_REQ_STREAM 0x00008000
// #define ISC_REQ_INTEGRITY 0x00010000
ctxReq = ISC_REQ_REPLAY_DETECT | ISC_REQ_SEQUENCE_DETECT |
ISC_REQ_CONFIDENTIALITY | ISC_REQ_DELEGATE;
```

```
bool haveInbuffer = false;
bool haveContext = false;
```

SSPI Kerberos for delegation

```
ib.pvBuffer = NULL;

while ( 1 )
{
obd.ulVersion = SECBUFFER_VERSION;
obd.cBuffers = 1;
obd.pBuffers = &ob; // just one buffer
ob.BufferType = SECBUFFER_TOKEN; // preping a token here
ob.cbBuffer = secPackInfo->cbMaxToken;
ob.pvBuffer = LocalAlloc( 0, ob.cbBuffer );

rcISC = (pf->InitializeSecurityContext)( &cred, haveContext?
&cliCtx: NULL,
myTokenSource, ctxReq, 0, SECURITY_NATIVE_DREP /
*SECURITY_NETWORK_DREP*/, haveInbuffer? &ibd: NULL,
0, &cliCtx, &obd, &ctxAttr, &useBefore );
printf( "ISC(): %08xh\n", rcISC );

if ( ib.pvBuffer != NULL )
{
LocalFree( ib.pvBuffer );
ib.pvBuffer = NULL;
}

if ( rcISC == SEC_I_COMPLETE_AND_CONTINUE || rcISC ==
SEC_I_COMPLETE_NEEDED )
{
if ( pf->CompleteAuthToken != NULL ) // only if implemented
(pf->CompleteAuthToken)( &cliCtx, &obd );
if ( rcISC == SEC_I_COMPLETE_NEEDED )
rcISC = SEC_E_OK;
else if ( rcISC == SEC_I_COMPLETE_AND_CONTINUE )
rcISC = SEC_I_CONTINUE_NEEDED;
}

// send the output buffer off to the server
// warning --- this is machine-dependent! FIX IT!
if ( ob.cbBuffer != 0 )
{
send( s, (const char *) &ob.cbBuffer, sizeof ob.cbBuffer, 0 );
bytesSent += sizeof ob.cbBuffer;
send( s, (const char *) ob.pvBuffer, ob.cbBuffer, 0 );
bytesSent += ob.cbBuffer;
}
LocalFree( ob.pvBuffer );
ob.pvBuffer = NULL;

if ( rcISC != SEC_I_CONTINUE_NEEDED )
break;

// prepare to get the server's response
```

SSPI Kerberos for delegation

```
ibd.ulVersion = SECBUFFER_VERSION;
ibd.cBuffers = 1;
ibd.pBuffers = &ib; // just one buffer
ib.BufferType = SECBUFFER_TOKEN; // preping a token here

// receive the server's response
// MACHINE-DEPENDENT CODE! (Besides, we assume that we
// get the length with a single read, which is not guaranteed)
recv( s, (char *) &ib.cbBuffer, sizeof ib.cbBuffer, 0 );
bytesReceived += sizeof ib.cbBuffer;
ib.pvBuffer = LocalAlloc( 0, ib.cbBuffer );

char *p = (char *) ib.pvBuffer;
int n = ib.cbBuffer;
while ( n )
{
    rc = recv( s, p, n, 0 );
    if ( rc == SOCKET_ERROR )
        wserr( rc, "recv" );
    if ( rc == 0 )
        wserr( 999, "recv" );
    bytesReceived += rc;
    n -= rc;
    p += rc;
}

// by now we have an input buffer and a client context

haveInbuffer = true;
haveContext = true;

// loop back for another round
puts( "looping" );
}

// we arrive here as soon as InitializeSecurityContext()
// returns != SEC_I_CONTINUE_NEEDED.

if ( rcISC != SEC_E_OK )
    printf( "Oops! ISC() returned %08xh!\n", rcISC );

(pf->FreeContextBuffer)( secPackInfo );
printf( "auth() exiting (%d sent, %d received)\n", bytesSent,
bytesReceived );
free( myTokenSource );
if ( nameAndPwd != 0 )
{
    if ( nameAndPwd->Domain != 0 )
        free( nameAndPwd->Domain );
    if ( nameAndPwd->User != 0 )
        free( nameAndPwd->User );
}
```

SSPI Kerberos for delegation

```
if ( nameAndPwd->Password != 0 )
free( nameAndPwd->Password );
free( nameAndPwd );
}
}

void initSecLib( HINSTANCE& hSec )
{
PSecurityFunctionTable (*pSFT)( void );

hSec = LoadLibrary( "security.dll" );
pSFT = (PSecurityFunctionTable (*)( void )) GetProcAddress( hSec,
"InitSecurityInterfaceA" );
if ( pSFT == NULL )
{
puts( "security.dll load messed up ..." );
exit( 1 );
}

pf = pSFT();
if ( pf == NULL )
{
puts( "no function table?!?" );
exit( 1 );
}
}

int main( int argc, char *argv[] )
{
int rc, port, i, errors;
HINSTANCE hSecLib;

unsigned long naddr;
SOCKET sock;
WSADATA wsadata;
PHOSTENT phe;
PSEVENT pse;
SOCKADDR_IN addr;
//const char *tokenSource = "Authsamp", *server = "server-machine";
const char *tokenSource = "win\\gtambat1", *server = "server-
machine";
const char *portstr = "11000", *user = 0, *pwd = 0, *domain = 0;

errors = 0;
for ( i = 1; i < argc; ++ i )
{
if ( argv[i][0] != '-' && argv[i][0] != '/' )
```

SSPI Kerberos for delegation

```
{
printf( "\"%s\" is not a valid switch.\n", argv[i] );
++ errors;
continue;
}

switch ( argv[i][1] )
{
case 's':
if ( i >= argc - 1 )
{
printf( "\"%s\" requires an argument.\n", argv[i] );
++ errors;
}
else if ( server != 0 )
{
printf( "\"%s\" has already been used.\n", argv[i ++] );
++ errors;
}
else
server = argv[++ i];
break;
case 'p':
if ( i >= argc - 1 )
{
printf( "\"%s\" requires an argument.\n", argv[i] );
++ errors;
}
else if ( portstr != 0 )
{
printf( "\"%s\" has already been used.\n", argv[i ++] );
++ errors;
}
else
portstr = argv[++ i];
break;
case 't':
if ( i >= argc - 1 )
{
printf( "\"%s\" requires an argument.\n", argv[i] );
++ errors;
}
else if ( tokenSource != 0 )
{
printf( "\"%s\" has already been used.\n", argv[i ++] );
++ errors;
}
else
tokenSource = argv[++ i];
break;
case 'd':
```

SSPI Kerberos for delegation

```
if ( i >= argc - 1 )
{
printf( "\"%s\" requires an argument.\n", argv[i] );
++ errors;
}
else if ( domain != 0 )
{
printf( "\"%s\" has already been used.\n", argv[i++] );
++ errors;
}
else
domain = argv[++ i];
break;
case 'u':
if ( i >= argc - 2 )
{
printf( "\"%s\" requires two arguments.\n", argv[i++] );
++ errors;
}
else if ( user != 0 )
{
printf( "\"%s\" has already been used.\n", argv[i] );
i += 2;
++ errors;
}
else
{
user = argv[++ i];
pwd = argv[++ i];
}
break;
default:
printf( "\"%s\" is not a valid switch.\n", argv[i] );
++ errors;
break;
}
}

if ( server == 0 )
{
puts( "A server name or IP address must be specified." );
++ errors;
}

if ( portstr == 0 )
{
puts( "A port name or port number must be specified." );
++ errors;
}

if ( user == 0 && domain != 0 )
```

SSPI Kerberos for delegation

```
puts( "No user name was specified, ignoring the domain." );

if ( errors )
{
puts( "\nusage: client -s your.server.com -p serverport" );
puts( " [-t token-source] [-u user pwd [-d domain]]" );
puts( "token-source is _required_ for Kerberos and should be
your" );
puts( "current logon name (e.g., \"MYDOMAIN\\felixk\")." );
puts( "If -u is absent, your current credentials will be used." );
return 1;
}

initSecLib( hSecLib );

rc = WSASStartup( 2, &wsadata );
wserr( rc, "WSASStartup" );

sock = socket( AF_INET, SOCK_STREAM, 0 );
if ( sock == INVALID_SOCKET )
wserr( 999, "socket" );

addr.sin_family = AF_INET;
// try numeric IP address first (inet_addr)
naddr = inet_addr( server );
if ( naddr != INADDR_NONE )
{
addr.sin_addr.s_addr = naddr;
}
else
{
phe = gethostbyname( server );
if ( phe == NULL )
wserr( 1, "gethostbyname" );
addr.sin_addr.s_addr = *( unsigned long * ) ( phe->h_addr );
memcpy( ( char * ) &addr.sin_addr, phe->h_addr, phe->h_length );
}

// try numeric protocol first
port = atoi( portstr );
if ( port > 0 && port < 32768 )
addr.sin_port = htons( (short) port );
else
{
pse = getservbyname( portstr, "tcp" );
if ( pse == NULL )
wserr( 1, "getservbyname" );
addr.sin_port = pse->s_port;
}

CredHandle cred;
```

SSPI Kerberos for delegation

```
CtxtHandle cliCtx;

rc = connect( sock, (SOCKADDR *) &addr, sizeof addr );
wserr( rc, "connect" );

struct sockaddr name;
int namelen = sizeof name;;
rc = getsockname( sock, &name, &namelen );
wserr( rc, "getsockname()" );
printf( "I am %u.%u.%u.%u\n", (unsigned int) (unsigned char)
name.sa_data[2],
(unsigned int) (unsigned char) name.sa_data[3], (unsigned int)
(unsigned char) name.sa_data[4],
(unsigned int) (unsigned char) name.sa_data[5] );

auth( sock, cred, cliCtx, tokenSource, user, pwd, server ); // this
does the real work
// Added by Ganesh
//auth( sock, cred, cliCtx, tokenSource, "ganesh", "ganesh", "client-
machine" ); // this does the real work

// use the authenticated connection here
bool haveToken = false;
rc = recv( sock, (char *) &haveToken, sizeof haveToken, 0 );
if ( rc != sizeof haveToken )
wserr( 999, "result-recv" );

if ( haveToken )
puts( "That seems to have worked." );
else
puts( "Oops! Wrong user name or password?" );

// the server is probably impersonating us by now
// this is where the client and server talk business

// clean up
(pf->DeleteSecurityContext)( &cliCtx );
(pf->FreeCredentialHandle)( &cred );

rc = closesocket( sock );
wserr( rc, "closesocket" );

rc = WSACleanup();
wserr( rc, "WSACleanup" );

int j;
puts("/n/n Should I start server implemenation within client ? : ");
scanf("%d",&j);
```

SSPI Kerberos for delegation

```
rc = Ganesh_Server_Impl();
```

```
__try
{
FreeLibrary( hSecLib );
}
__except ( 1 )
{
puts( "Freelibrary( security.dll ) caused an access violation.
Yuck." );
}

return 0;
}
```

```
*****
```

```
Server (Logged in user win\gtambat1)
```

```
*****
```

```
#include <stdio.h>
#include <windows.h>
#include <string.h>
#include <malloc.h>

#define SECURITY_WIN32 1
#include <sspi.h>
//#include <issperr.h> // uncomment if you have an old Platform SDK

#pragma hdrstop

//
=====
// IMPORTANT NOTICE --- Check out
// http://www.mvps.org/security/sspi.html
// for the gory details of how this works
//
=====
```

```
PSecurityFunctionTable pf = NULL;
```

```
int Ganesh_Client_Impl(CtxtHandle &srvCtx);
```

```
void initSecLib( HINSTANCE& hSec );
```

```
void Ganesh_auth( SOCKET s, CredHandle& cred, CtxtHandle& cliCtx,
const char *tokenSource, const char *name /* = NULL */,
```

SSPI Kerberos for delegation

```
const char *pwd /* = NULL */, const char *domain /* = NULL */);

void Ganesh_auth( SOCKET s, CredHandle& cred, CtxtHandle& cliCtx,
const char *tokenSource, const char *name /* = NULL */,
const char *pwd /* = NULL */, const char *domain /* = NULL */)
{
int rc, rcISC;
SecPkgInfo *secPackInfo;
SEC_WINNT_AUTH_IDENTITY *nameAndPwd = NULL;
int bytesReceived = 0, bytesSent = 0;

//puts( "auth() entered" );

// the arguments to ISC() is not const ... for once, I decided
// on creating writable copies instead of using a brutalizing cast.
char *myTokenSource;
myTokenSource = _strdup( tokenSource );

if ( name != NULL )
{
nameAndPwd = (SEC_WINNT_AUTH_IDENTITY *) malloc( sizeof
SEC_WINNT_AUTH_IDENTITY );
memset( nameAndPwd, '\0', sizeof *nameAndPwd );
nameAndPwd->Domain = (byte *) _strdup( domain? domain: "" );
nameAndPwd->DomainLength = domain? strlen( domain ): 0;
nameAndPwd->User = (byte *) _strdup( name? name: "" );
nameAndPwd->UserLength = name? strlen( name ): 0;
nameAndPwd->Password = (byte *) _strdup( pwd? pwd: "" );
nameAndPwd->PasswordLength = pwd? strlen( pwd ): 0;
nameAndPwd->Flags = SEC_WINNT_AUTH_IDENTITY_ANSI;
}

rc = (pf->QuerySecurityPackageInfo)( "kerberos", &secPackInfo );
//printf( "QSPI(): %08xh\n", rc );

TimeStamp useBefore;

rc = GetLastError();

rc = (pf->AcquireCredentialsHandle)( NULL, "kerberos",
SECPKG_CRED_BOTH,
NULL, nameAndPwd, NULL, NULL, &cred, &useBefore );
//printf( "ACH(): %08xh\n", rc );

rc = GetLastError();
// input and output buffers
SecBufferDesc obd, ibd;
SecBuffer ob, ib;

DWORD ctxReq, ctxAttr;
```

SSPI Kerberos for delegation

```
// from sspi.h:
// #define ISC_REQ_DELEGATE 0x00000001
// #define ISC_REQ_MUTUAL_AUTH 0x00000002
// #define ISC_REQ_REPLAY_DETECT 0x00000004
// #define ISC_REQ_SEQUENCE_DETECT 0x00000008
// #define ISC_REQ_CONFIDENTIALITY 0x00000010
// #define ISC_REQ_USE_SESSION_KEY 0x00000020
// #define ISC_REQ_PROMPT_FOR_CREDS 0x00000040
// #define ISC_REQ_USE_SUPPLIED_CREDS 0x00000080
// #define ISC_REQ_ALLOCATE_MEMORY 0x00000100
// #define ISC_REQ_USE_DCE_STYLE 0x00000200
// #define ISC_REQ_DATAGRAM 0x00000400
// #define ISC_REQ_CONNECTION 0x00000800
// #define ISC_REQ_CALL_LEVEL 0x00001000
// #define ISC_REQ_EXTENDED_ERROR 0x00004000
// #define ISC_REQ_STREAM 0x00008000
// #define ISC_REQ_INTEGRITY 0x00010000
ctxReq = ISC_REQ_REPLAY_DETECT | ISC_REQ_SEQUENCE_DETECT |
ISC_REQ_CONFIDENTIALITY | ISC_REQ_DELEGATE;

bool haveInbuffer = false;
bool haveContext = false;
ib.pvBuffer = NULL;

while ( 1 )
{
    obd.ulVersion = SECBUFFER_VERSION;
    obd.cBuffers = 1;
    obd.pBuffers = &ob; // just one buffer
    ob.BufferType = SECBUFFER_TOKEN; // preping a token here
    ob.cbBuffer = secPackInfo->cbMaxToken;
    ob.pvBuffer = LocalAlloc( 0, ob.cbBuffer );

    rcISC = (pf->InitializeSecurityContext)( &cred, haveContext?
&cliCtx: NULL,
myTokenSource, ctxReq, 0, SECURITY_NATIVE_DREP /
*SECURITY_NETWORK_DREP*/, haveInbuffer? &ibd: NULL,
0, &cliCtx, &obd, &ctxAttr, &useBefore );
    // printf( "ISC(): %08xh\n", rcISC );

    if ( ib.pvBuffer != NULL )
    {
        LocalFree( ib.pvBuffer );
        ib.pvBuffer = NULL;
    }

    if ( rcISC == SEC_I_COMPLETE_AND_CONTINUE || rcISC ==
SEC_I_COMPLETE_NEEDED )
    {
        if ( pf->CompleteAuthToken != NULL ) // only if implemented
```

SSPI Kerberos for delegation

```
(pf->CompleteAuthToken)( &cliCtx, &obd );
if ( rcISC == SEC_I_COMPLETE_NEEDED )
rcISC = SEC_E_OK;
else if ( rcISC == SEC_I_COMPLETE_AND_CONTINUE )
rcISC = SEC_I_CONTINUE_NEEDED;
}

// send the output buffer off to the server
// warning -- this is machine-dependent! FIX IT!
if ( ob.cbBuffer != 0 )
{
send( s, (const char *) &ob.cbBuffer, sizeof ob.cbBuffer, 0 );
bytesSent += sizeof ob.cbBuffer;
send( s, (const char *) ob.pvBuffer, ob.cbBuffer, 0 );
bytesSent += ob.cbBuffer;
}
LocalFree( ob.pvBuffer );
ob.pvBuffer = NULL;

if ( rcISC != SEC_I_CONTINUE_NEEDED )
break;

// prepare to get the server's response
ibd.ulVersion = SECBUFFER_VERSION;
ibd.cBuffers = 1;
ibd.pBuffers = &ib; // just one buffer
ib.BufferType = SECBUFFER_TOKEN; // preping a token here

// receive the server's response
// MACHINE-DEPENDENT CODE! (Besides, we assume that we
// get the length with a single read, which is not guaranteed)
rc = recv( s, (char *) &ib.cbBuffer, sizeof ib.cbBuffer, 0 );
bytesReceived += sizeof ib.cbBuffer;
ib.pvBuffer = LocalAlloc( 0, ib.cbBuffer );

char *p = (char *) ib.pvBuffer;
int n = ib.cbBuffer;
while ( n )
{
rc = recv( s, p, n, 0 );
// if ( rc == SOCKET_ERROR )
// wserr( rc, "recv" );
// if ( rc == 0 )
// wserr( 999, "recv" );
bytesReceived += rc;
n -= rc;
p += rc;
}

// by now we have an input buffer and a client context
```

SSPI Kerberos for delegation

```
haveInbuffer = true;
haveContext = true;

// loop back for another round
// puts( "looping" );
}

// we arrive here as soon as InitializeSecurityContext()
// returns != SEC_I_CONTINUE_NEEDED.

if ( rcISC != SEC_E_OK )
{
printf( "Oops! ISC() returned %08xh!\n", rcISC );
}

(pf->FreeContextBuffer)( secPackInfo );
//printf( "auth() exiting (%d sent, %d received)\n", bytesSent,
bytesReceived );
free( myTokenSource );
if ( nameAndPwd != 0 )
{
if ( nameAndPwd->Domain != 0 )
free( nameAndPwd->Domain );
if ( nameAndPwd->User != 0 )
free( nameAndPwd->User );
if ( nameAndPwd->Password != 0 )
free( nameAndPwd->Password );
free( nameAndPwd );
}
}

// wserr() displays winsock errors and aborts. No grace there.
void wserr( int rc, const char * const funcname )
{
if ( rc == 0 )
return;

fprintf( stderr, "\nWinsock error %d [gle %d] returned by %s().\n"
"Sorry, no bonus!\n", rc, WSAGetLastError(), funcname );
WSACleanup();
exit( rc );
}

bool auth( SOCKET s, CredHandle& cred, CtxtHandle& srvCtx )
{
int rc;
bool haveToken = true;
```

SSPI Kerberos for delegation

```
SecPkgInfo *secPackInfo;
int bytesReceived = 0, bytesSent = 0;
char buf[256];
DWORD bufsiz = sizeof buf;

puts( "auth() entered" );

rc = (pf->QuerySecurityPackageInfo)( "kerberos", &secPackInfo );
printf( "QSPI(): %08xh\n", rc );
if ( rc != SEC_E_OK )
    haveToken = false;

TimeStamp useBefore;

rc = (pf->AcquireCredentialsHandle)( NULL, "kerberos",
    SECPKG_CRED_BOTH,
    NULL, NULL, NULL, NULL, &cred, &useBefore );

// rc = (pf->AcquireCredentialsHandle)( NULL, "kerberos",
// SECPKG_CRED_BOTH,
// NULL, NULL, NULL, NULL, &cred, &useBefore );

printf( "ACH(): %08xh\n", rc );
if ( rc != SEC_E_OK )
    haveToken = false;

// input and output buffers
SecBufferDesc obd, ibd;
SecBuffer ob, ib;

DWORD ctxAttr;

bool haveContext = false;

while ( 1 )
{
    // prepare to get the server's response
    ibd.ulVersion = SECBUFFER_VERSION;
    ibd.cBuffers = 1;
    ibd.pBuffers = &ib; // just one buffer
    ib.BufferType = SECBUFFER_TOKEN; // preping a token here

    // receive the client's POD
    // MACHINE-DEPENDENT CODE! (Besides, we assume that we
    // get the length with a single read, which is not guaranteed)
    recv( s, (char *) &ib.cbBuffer, sizeof ib.cbBuffer, 0 );
    bytesReceived += sizeof ib.cbBuffer;
    ib.pvBuffer = LocalAlloc( 0, ib.cbBuffer );

    char *p = (char *) ib.pvBuffer;
    int n = ib.cbBuffer;
}
```

SSPI Kerberos for delegation

```
while ( n )
{
rc = recv( s, p, n, 0 );
if ( rc == SOCKET_ERROR )
wserr( rc, "recv" );
if ( rc == 0 )
wserr( 999, "recv" );
bytesReceived += rc;
n -= rc;
p += rc;
}

// by now we have an input buffer

obd.ulVersion = SECBUFFER_VERSION;
obd.cBuffers = 1;
obd.pBuffers = &ob; // just one buffer
ob.BufferType = SECBUFFER_TOKEN; // preping a token here
ob.cbBuffer = secPackInfo->cbMaxToken;
ob.pvBuffer = LocalAlloc( 0, ob.cbBuffer );

rc = (pf->AcceptSecurityContext)( &cred, haveContext? &srvCtx: NULL,
&ibd, 0,/* SECURITY_NATIVE_DREP */ SECURITY_NATIVE_DREP /
*SECURITY_NETWORK_DREP*/, &srvCtx, &obd, &ctxAttr,
&useBefore );
printf( "ASC(): %08x\n", rc );

if ( ib.pvBuffer != NULL )
{
LocalFree( ib.pvBuffer );
ib.pvBuffer = NULL;
}

if ( rc == SEC_I_COMPLETE_AND_CONTINUE || rc ==
SEC_I_COMPLETE_NEEDED )
{
if ( pf->CompleteAuthToken != NULL ) // only if implemented
(pf->CompleteAuthToken)( &srvCtx, &obd );
if ( rc == SEC_I_COMPLETE_NEEDED )
rc = SEC_E_OK;
else if ( rc == SEC_I_COMPLETE_AND_CONTINUE )
rc = SEC_I_CONTINUE_NEEDED;
}

// send the output buffer off to the server
// warning --- this is machine-dependent! FIX IT!
if ( rc == SEC_E_OK || rc == SEC_I_CONTINUE_NEEDED )
{
if ( ob.cbBuffer != 0 )
{
send( s, (const char *) &ob.cbBuffer, sizeof ob.cbBuffer, 0 );
}
```

SSPI Kerberos for delegation

```
bytesSent += sizeof ob.cbBuffer;
send( s, (const char *) ob.pvBuffer, ob.cbBuffer, 0 );
bytesSent += ob.cbBuffer;
}
LocalFree( ob.pvBuffer );
ob.pvBuffer = NULL;
}

if ( rc != SEC_I_CONTINUE_NEEDED )
break;

haveContext = true;

// loop back for another round
puts( "looping" );
}

// we arrive here as soon as InitializeSecurityContext()
// returns != SEC_I_CONTINUE_NEEDED.

if ( rc != SEC_E_OK )
{
printf( "Oops! ASC() returned %08xh!\n", rc );
haveToken = false;
}

GetUserName( buf, &bufsiz );
// now we try to use the context
rc = (pf->ImpersonateSecurityContext)( &srvCtx );
printf( "ImpSC(): %08xh\n", rc );
if ( rc != SEC_E_OK )
{
printf( "Oops! ImpSC() returns %08xh!\n", rc );
haveToken = false;
}
else
{
char buf[256];
DWORD bufsiz = sizeof buf;

printf( "haveToken = %s\n\n", haveToken? "true": "false" );
send( s, (const char *) &haveToken, sizeof haveToken, 0 );

GetUserName( buf, &bufsiz );
printf( "user name: \"%s\"\n", buf );
//(pf->RevertSecurityContext)( &srvCtx );
printf( "RSC(): %08xh\n", rc );
}

//(pf->FreeContextBuffer)( secPackInfo );
```

SSPI Kerberos for delegation

```
printf( "auth() exiting (%d received, %d sent)\n", bytesReceived,
bytesSent );
return haveToken;
}

int Ganesh_Client_Impl(CtxtHandle &srvCtx)
{
char buf[256];
DWORD bufsiz = sizeof buf;
int rc = 0;

//CtxtHandle tmpCtx = srvCtx;

SecPkgContext_Lifespan lspan;
rc = (pf->QueryContextAttributes>(&srvCtx, SECPKG_ATTR_LIFESPAN,
&lspan);

rc = GetLastError();

GetUserName( buf, &bufsiz );
//rc = (pf->ImpersonateSecurityContext)( &srvCtx );

rc = GetLastError();
GetUserName( buf, &bufsiz );

int port, i, errors;
unsigned long naddr;
SOCKET sock;
WSADATA wsadata;
PHOSTENT phe;
PSERVENT pse;
SOCKADDR_IN addr;
HINSTANCE hSecLib;
const char *tokenSource = "win\\gtambat", *server = "client-machine";
//const char *tokenSource = "PUN\\gtambat/*"Authsamp"*/, *server =
"client-machine";
const char *portstr = "12000", *user = 0, *pwd = 0, *domain = 0;

errors = 0;
if ( server == 0 )
{
//puts( "A server name or IP address must be specified." );
++ errors;
}

if ( portstr == 0 )
{
//puts( "A port name or port number must be specified." );
++ errors;
}
```

SSPI Kerberos for delegation

```
if ( user == 0 && domain != 0 )
{
}
// puts( "No user name was specified, ignoring the domain." );

if ( errors )
{
/* puts( "\nusage: client -s your.server.com -p serverport" );
puts( " [-t token-source] [-u user pwd [-d domain]]" );
puts( "token-source is _required_ for Kerberos and should be
your" );
puts( "current logon name (e.g., \"MYDOMAIN\\felixk\")." );
puts( "If -u is absent, your current credentials will be used." );*/
return 1;
}

initSecLib( hSecLib );

rc = WSASStartup( 2, &wsadata );
//wserr( rc, "WSASStartup" );

sock = socket( AF_INET, SOCK_STREAM, 0 );
if ( sock == INVALID_SOCKET )
{
rc = -1;
}
else
{
rc = 0;
}

addr.sin_family = AF_INET;
// try numeric IP address first (inet_addr)
naddr = inet_addr( server );
if ( naddr != INADDR_NONE )
{
addr.sin_addr.s_addr = naddr;
}
else
{
phe = gethostbyname( server );
// if ( phe == NULL )
// wserr( 1, "gethostbyname" );
addr.sin_addr.s_addr = *( unsigned long *) (phe->h_addr);
memcpy( (char *) &addr.sin_addr, phe->h_addr, phe->h_length );
}

// try numeric protocol first
port = atoi( portstr );
if ( port > 0 && port < 32768 )
```

SSPI Kerberos for delegation

```
addr.sin_port = htons( (short) port );
else
{
pse = getservbyname( portstr, "tcp" );
// if ( pse == NULL )
// wserr( 1, "getservbyname" );
addr.sin_port = pse->s_port;
}

CredHandle cred;
CtxtHandle cliCtx;

rc = connect( sock, (SOCKADDR *) &addr, sizeof addr );
//wserr( rc, "connect" );

struct sockaddr name;
int namelen = sizeof name;;
rc = getsockname( sock, &name, &namelen );
/*wserr( rc, "getsockname()" );
printf( "I am %u.%u.%u.%u\n", (unsigned int) (unsigned char)
name.sa_data[2],
(unsigned int) (unsigned char) name.sa_data[3], (unsigned int)
(unsigned char) name.sa_data[4],
(unsigned int) (unsigned char) name.sa_data[5] );*/

Ganesh_auth( sock, cred, cliCtx, tokenSource, user, pwd, domain ); //
this does the real work

// use the authenticated connection here
bool haveToken = false;
rc = recv( sock, (char *) &haveToken, sizeof haveToken, 0 );
if ( rc != sizeof haveToken )
{
rc = 999;
}

// wserr( 999, "result-recv" );

if ( haveToken )
puts( "That seems to have worked." );
else
puts( "Oops! Wrong user name or password?" );

// the server is probably impersonating us by now
// this is where the client and server talk business

// clean up
(pf->DeleteSecurityContext)( &cliCtx );
(pf->FreeCredentialHandle)( &cred );
```

SSPI Kerberos for delegation

```
rc = closesocket( sock );
wserr( rc, "closesocket" );

rc = WSACleanup();
wserr( rc, "WSACleanup" );

__try
{
FreeLibrary( hSecLib );
}
__except ( 1 )
{
// puts( "Freelibrary( security.dll ) caused an access violation.
Yuck." );
}

return 0;
}

struct CapName
{
DWORD bits;
const char *name;
const char *comment;
} capNames[] = {
{ SECPKG_FLAG_INTEGRITY, "SECPKG_FLAG_INTEGRITY", "Supports integrity
on messages" },
{ SECPKG_FLAG_PRIVACY, "SECPKG_FLAG_PRIVACY", "Supports privacy
(confidentiality)" },
{ SECPKG_FLAG_TOKEN_ONLY, "SECPKG_FLAG_TOKEN_ONLY", "Only security
token needed" },
{ SECPKG_FLAG_DATAGRAM, "SECPKG_FLAG_DATAGRAM", "Datagram RPC
support" },
{ SECPKG_FLAG_CONNECTION, "SECPKG_FLAG_CONNECTION", "Connection
oriented RPC support" },
{ SECPKG_FLAG_MULTI_REQUIRED, "SECPKG_FLAG_MULTI_REQUIRED", "Full 3-
leg required for re-auth." },
{ SECPKG_FLAG_CLIENT_ONLY, "SECPKG_FLAG_CLIENT_ONLY", "Server side
functionality not available" },
{ SECPKG_FLAG_EXTENDED_ERROR, "SECPKG_FLAG_EXTENDED_ERROR", "Supports
extended error msgs" },
{ SECPKG_FLAG_IMPERSONATION, "SECPKG_FLAG_IMPERSONATION", "Supports
impersonation" },
{ SECPKG_FLAG_ACCEPT_WIN32_NAME, "SECPKG_FLAG_ACCEPT_WIN32_NAME",
"Accepts Win32 names" },
{ SECPKG_FLAG_STREAM, "SECPKG_FLAG_STREAM", "Supports stream
semantics" },
{ SECPKG_FLAG_NEGOTIABLE, "SECPKG_FLAG_NEGOTIABLE", "Can be used by
the negotiate package" },
{ SECPKG_FLAG_GSS_COMPATIBLE, "SECPKG_FLAG_GSS_COMPATIBLE", "GSS
Compatibility Available" },
```

SSPI Kerberos for delegation

```
{ SECPKG_FLAG_LOGON, "SECPKG_FLAG_LOGON", "Supports common  
LsaLogonUser" },  
{ SECPKG_FLAG_ASCII_BUFFERS, "SECPKG_FLAG_ASCII_BUFFERS", "Token  
Buffers are in ASCII" },  
{ 0xffffffffL, "(fence)", "(fence)" }  
};
```

```
void initSecLib( HINSTANCE& hSec )  
{  
PSecurityFunctionTable (*pSFT)( void );
```

```
hSec = LoadLibrary( "security.dll" );  
pSFT = (PSecurityFunctionTable (*)( void )) GetProcAddress( hSec,  
"InitSecurityInterfaceA" );  
if ( pSFT == NULL )  
{  
exit( 1 );  
}
```

```
pf = pSFT();  
if ( pf == NULL )  
{  
exit( 1 );  
}
```

```
SECURITY_STATUS rc;  
DWORD numPacks = 0, i, j;  
SecPkgInfo *pPacks = NULL;
```

```
rc = (pf->EnumerateSecurityPackages)( &numPacks, &pPacks );  
if ( rc != 0 )  
{  
exit( 1 );  
}
```

```
for ( i = 0; i < numPacks; ++ i )  
{  
for ( j = 0; capNames[j].bits != 0xffffffffL; ++ j )  
{  
//if ( ( capNames[j].bits & pPacks[i].fCapabilities ) == capNames  
[j].bits )  
//printf( " %s (%s)\n", capNames[j].name, capNames  
[j].comment );  
}  
}
```

```
if ( pPacks != NULL )  
(pf->FreeContextBuffer)( pPacks );  
}
```

SSPI Kerberos for delegation

```
int main( int argc, char *argv[] )
{

int rc, port, addrlen;
HINSTANCE hSecLib;
CtxtHandle srvCtx;

bool haveToken;
SOCKET sock, s;
WSADATA wsadata;
PSEVENT pse;
SOCKADDR_IN addr;
HANDLE threadRet;
char buf[256];
DWORD bufsiz = sizeof buf;

if ( argc != 2 )
{
puts( "usage: server portnumber" );
return 1;
}

initSecLib( hSecLib );

rc = WSASStartup( 2, &wsadata );
wserr( rc, "WSASStartup" );

sock = socket( AF_INET, SOCK_STREAM, 0 );
if ( sock == INVALID_SOCKET )
wserr( 999, "socket" );

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;

// try numeric protocol first
port = atoi( argv[1] );
if ( port > 0 && port < 32768 )
addr.sin_port = htons( (short) port );
else
{
pse = getservbyname( argv[1], "tcp" );
if ( pse == NULL )
wserr( 1, "getservbyname" );
addr.sin_port = pse->s_port;
}

rc = bind( sock, (SOCKADDR *) &addr, sizeof addr );
wserr( rc, "bind" );
```

SSPI Kerberos for delegation

```
rc = listen( sock, 2 );
wserr( rc, "listen" );

CredHandle cred;

while ( 1 )
{
    addrlen = sizeof addr;
    s = accept( sock, (SOCKADDR *) &addr, &addrlen );
    if ( s == INVALID_SOCKET )
        wserr( s, "accept" );

    haveToken = auth( s, cred, srvCtx );

    // now we talk to the client
    printf( "haveToken = %s\n\n", haveToken? "true": "false" );
    send( s, (const char *) &haveToken, sizeof haveToken, 0 );

    // clean up

    // Modified by Ganesh
    /*(pf->DeleteSecurityContext)( &srvCtx );
    (pf->FreeCredentialHandle)( &cred );
    closesocket( s );*/
    break;
}

int j;
puts("/n/n Should I start client implemenation within server ? : ");
scanf("%d",&j);

LPVOID threadParam = &srvCtx;
DWORD threadId;

//PROCESS_INFORMATION Pi;
//STARTUPINFO Si;
//HANDLE hTokenNew = NULL, hTokenDup = NULL;

//ZeroMemory( &Pi,sizeof(Pi));
//ZeroMemory( &Si, sizeof( STARTUPINFO ) );
//Si.cb = sizeof( STARTUPINFO );
// Si.lpDesktop = "winsta0\\default";

//GetUserName( buf, &bufsiz );
//rc = (pf->ImpersonateSecurityContext)( &srvCtx );

//rc = GetLastError();
//GetUserName( buf, &bufsiz );

//rc = CreateProcess("C:\\WINDOWS\\system32\\notepad.exe","C:\\WINDOWS
```

SSPI Kerberos for delegation

```
\\system32\notepad.exe", NULL, NULL, FALSE, NORMAL_PRIORITY_CLASS,  
NULL, NULL, &Si, &Pi);
```

```
rc = GetLastError();
```

```
//threadRet = CreateThread( NULL, 0, (LPTHREAD_START_ROUTINE)  
Ganesh_Client_Impl, threadParam, 0, &threadId);
```

```
j = Ganesh_Client_Impl(srvCtx);
```

```
WaitForSingleObject(threadRet,INFINITE);  
CloseHandle( threadRet );
```

```
FreeLibrary( hSecLib );
```

```
return 0;  
}
```

```
*****
```

```
.
```