Written by : Barak Weichselbaum, all rights reserved 2000-2009

Site: http://www.komodia.com/
Email : barak@komodia.com

Winsock LSP (Layered Service Provider) guide

Recommendation: this article assumes knowledge in winsock2 technology.

1. What is LSP?

LSP (Layered Service Provider) is a component that intercepts winsock2 calls, has the ability to manipulate them, and then has the option to pass them to the winsock2 provider.

There are two kinds of LSP frameworks:

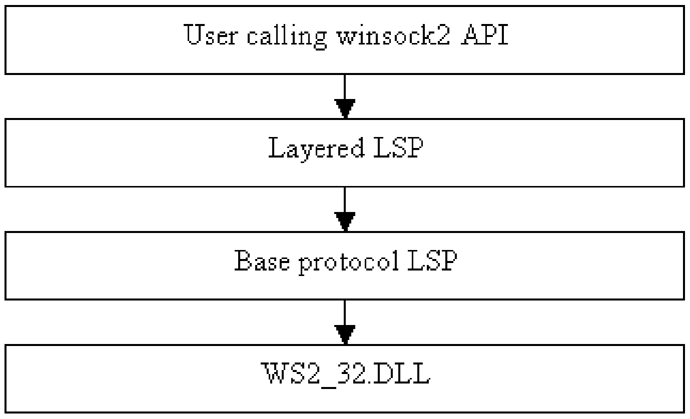
- Transport service provider
- Name space provider

2. LSP – Transport service provider

LSP is organized in a chain structure: the first call goes to the first LSP in the chain, the LSP processes the call, and then calls the next LSP in the chain, and so on, until the last LSP calls the actual winsock2 function.

Each "service" has a different LSP chain (i.e., LSP for TCP, UDP, raw sockets, netbios, etc.) These LSPs are called base providers because they call the actual winsock2 functions.

The LSPs in the chain that aren't base providers are called layered providers. (These implement only higher-level calls)



3. LSP – Name space provider (NSP)

This article focuses on transport provider LSPs. This section is a brief overview of name space provider LSPs.

A name space provider is a way to allow the protocol to handle its naming methods. For TCP/IP the name space provider will query the DNS and resolve a domain name into an IP address.

NSPs are mostly used to extend domains on the web. For example, you can sell domains that end in ".lsp". These are not official, of course, but if the resolving computer has NSP installed, you will be able to resolve the .lsp extension. (I am assuming that the NSP is made by the seller of the .lsp domain and acts to resolve the domain.)

Komodia has a module for intercepting DNS calls but are not using NSP for that.

4. LSP – where to start

Documentation on LSP is scarce, and the MSDN documentation will get you nowhere, or at least waste your valuable time with experiments.

Microsoft's default LSP samples have matured and are quite good. However, without proper guidance they can look pretty scary!

Part of this document uses my own LSP sample: I took Microsoft's LSP sample, and wrapped it in a VC6 project (I just hate makefiles, and open source projects that take ages to compile).

Writing LSP from scratch is time-consuming and basically not something most programmers will do. I have thought of writing such an LSP that would be pure C++ with lots of wonderful features and ideas, yet such a project demands more time than my other programming endeavors allow.

5. LSP – Let's get technical, and a WARNING

LSP is a DLL, and to get it working you must register it. Part of the sample is the installation software – the LSP Installer, again taken from Microsoft's sample.

I strongly recommend backing up the registry entry being modified, as this will later make removing the LSP easy and will not hamper the OS's normal operation. (Blindly tampering with LSPs can destroy the OS's normal networking ability, or even crash at boot time, so I cannot recommend backing up strongly enough.)

The LSP registry is :
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WinSock2

Back it up, together with all the sub-registry keys.

Also have a LSP remover handy, in case you damaged your stack you would not be able to download such tool (you will not have network).

6. LSP - loading sequence

Since LSP is a DLL, our first stop will be at the DllMain. The LSP initializes only its basic variables.
Be careful when adding code to the LSP. In the DllMain, you must perform a minimum of initializations, or it can cause unexpected results. I once tried to spawn threads/wait on events inside the DllMain. I then spent the night figuring out what went wrong. Special notice goes to CoInitialize and CoInitializeEx, two functions which the documentations state should not be called from the DllMain.

When the DLL is unloaded, the LSP will deallocate all of its resources.

The LSP is loaded in each process with winsock2 calls that use the base provider the LSP has layered (thus you may have many running instances of your LSP.)

The second important LSP function is WSPStartup (which is exported in the LSP's .def file)

This function is called when the user calls WSAStartup. Essentially, the LSP loads all the layered protocols beneath it in the chain, and initializes its own calling table (a dynamic table for calling its WSP functions.)

7. Installing the LSP

Every LSP has a unique GUID. Make sure you change the default GUID, found in provider.CPP, using the GUID variable, ProviderGuid. You can use GuidGen, supplied with the Visual Studio, to generate a new GUID.

The latest Microsoft LSP Installer sample can take the LSP file as a parameter and retrieve the GUID from an exported method called "GetLspGuid" – unlike the old LSP Installer, in which the GUID was embedded in the installer.

This guide shows how to use the default installer. However, for a 100% successful installation you'd need to write an extra 10,000 line of code. The good news is that we already did it for you. Check out *Komodia's advanced LSP installer* page at:
http://www.komodia.com/index.php?page=rlsp.html

8. Running the LSP Installer

The registrant (RegisterLSP) is a command-line based application.

Let's start with the most useful parameter, *RegisterLSP –p*
which lists all the LSPs configured on the machine. A sample list looks like this:

```
 - 1001MSAFD Tcpip [TCP/IP[
 - 1002MSAFD Tcpip [UDP/IP[
 - 1003MSAFD Tcpip [RAW/IP[
 - 1004RSVP UDP Service Provider
 - 1005RSVP TCP Service Provider
 - 1070MSAFD NetBIOS [\Device\NetBT_Tcpip_{F4754F49-E15B-43DC-B8D3-
1BA4BB08F224{
 [SEQPACKET 0
 - 1071MSAFD NetBIOS [\Device\NetBT_Tcpip_{F4754F49-E15B-43DC-B8D3-
1BA4BB08F224{
 [DATAGRAM 0
 - 1072MSAFD NetBIOS [\Device\NetBT_Tcpip_{0EC4D370-C932-4F6A-BE69-
7EE8A6653B7C{
 [SEQPACKET 1
 - 1073MSAFD NetBIOS [\Device\NetBT_Tcpip_{0EC4D370-C932-4F6A-BE69-
7EE8A6653B7C{
 [DATAGRAM 1
```

The number on the left is the LSP's ID, and the string on the right is the LSP's description.

Now that you have the list, you can layer your LSP on top of the base providers. The base provider's ID for TCP in this example is 1001, for UDP 1002, and for raw sockets 1003. However, this may vary from machine to machine, because on most laptops there is an IrDA (infrared) base provider installed with the ID 1001, which makes the ID for TCP 1002, and so on.

For most implementations you would normally want to install your LSP on top of the TCP, as follows: RegisterLSP –i –o 1001 –d LSPFilename

-i is for installing our LSP.
-o is the ID of the base provider you are layering on top of. To layer over multiple base providers, you must specify multiple –o parameters, e.g.: RegisterLSP –i –o 1001 –o 1002 –d LSPFilename.
-d is for the LSP file location.

Now that you've installed your LSP, you can view it with: RegisterLSP –p, or with: RegisterLSP –l, which displays layered protocols only.

To install your LSP over all base providers and all 3rd-party LSPs, use: RegisterLSP –a – d LSPFilename

For removing LSPs, I would simply restore the registry backup you made earlier, but it can also be done with: RegisterLSP –r

I do NOT recommend using: RegisterLSP –f, which removes all layered entries, as it may delete some useful LSPs.

9. Installation issues

All versions of Internet Explorer have a nice "feature" (not bug, of course...): they use the UDP protocol for internal communication, and send UDP sockets into the TCP functions (and vice versa). This is undesirable design, but apparently we are to blame. To fix this, whenever you layer over TCP, make sure to layer over UDP as well.

10. TCPIPv6

In Vista OS and above, you also have a TCPIPv6 base providers, if your LSP only supports TCPIPv4 and you don't care about intercepting WSPSelect then you should install only above TCPIPv4, if you do care about WSPSelect then you need to layer over TCPIPv6 as well.

If you layered over TCPIPv6 (but only supporting TCPIPv4), make sure to not run any custom code in case the protocols are v6 and not v4, also for WSPSelect you need to set some extra flags, which are covered in the tips you should receive in your emails.

11. Modifying the LSP

We have a compiling and working LSP, and now we want to add some features to it.

Every Winsock call prefixed by WSA has a WSP call in the LSP. WSARecv is mapped to WSPRecv, so if we want to log all the data that enters the system we can use:

```
int WSPAPI WSPRecv(
    SOCKET          s,
    LPWSABUF        lpBuffers,
    DWORD           dwBufferCount,
    LPDWORD         lpNumberOfBytesRecvd,
    LPDWORD         lpFlags,
    LPWSAOVERLAPPED lpOverlapped,
    LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine,
    LPWSATHREADID   lpThreadId,
    LPINT           lpErrno)
```

In the beginning of every WSP function that receives the socket handle as a parameter, there is a function for resolving the socket context (a structure that contains information about the socket.) If you wish to keep additional information about each socket, you must put it in this socket context structure.

After the socket context is found, the LSP checks if the operation is overlapped. If it is, the LSP function saves the overlapped data inside the overlap structure within the socket context, and queues this data for the function that handles the overlapped operation. OverlappedManagerThread dispatches the overlap and IntermediateCompletionRoutine performs the overlap operation itself.

If the operation is not overlapped, the LSP calls the next LSP after it in the chain.

Let’s say you wish to get the data that will end up inside the buffer after the lower-layered function finishes. For example, you wish to search a specific string in order to implement context checking for an IDS.

To do this, you need to intercept in two places. First, in the IntermediateCompletionRoutine, in case the receive call is overlapped, and second, after the call to the next LSP, if the call is overlapped (If you decide to go with IFS LSP you will not receive the call to the IntermediateCompletionRoutine, this will cause you not to get data for Chrome and any other application that uses overlapped sockets)

Don’t forget to check the error flag for errors: if there is an error, the buffer contains random data.

12. Blocking incoming connections

To do this, you need to modify WSPAccept:

```
SOCKET WSPAPI WSPAccept(
    SOCKET          s,
    struct sockaddr FAR * addr,
    LPINT           addrlen,
    LPCONDITIONPROC lpfnCondition,
    DWORD_PTR       dwCallbackData,
    LPINT           lpErrno)
```

You can find the peer address trying to connect by inspecting the variable *addr*.

If you decide not to allow this connection, simply return the value INVALID_SOCKET.

13. Storing additional socket data

To store your socket data, you must modify the socket context structure. This is defined in the Provider.h file, under SOCK_INFO.

Let's say we wish to add a running number to each socket. We will add an int variable, and the structure will look like this:

```
typedef struct _SOCK_INFO
{
    SOCKET ProviderSocket;      // lower provider socket
handle
    SOCKET LayeredSocket;       // app's socket handle
    DWORD  dwOutstandingAsync;  // count of outstanding
async operations
    BOOL   bClosing;            // has the app closed the
socket?
    DWORD  BytesSent;           // Byte counts
    DWORD  BytesRecv;
    HANDLE hIocp;               // associated with an
IOCP?

    HWND   hWnd;                // Window (if any)
associated with socket
    UINT   uMsg;                // Message for socket
events

    CRITICAL_SECTION   SockCritSec;
    struct _PROVIDER  *Provider;
    struct _SOCK_INFO *prev,
*                     next;
    int iRunningNumber;
} SOCK_INFO;
```

The running number variable is iRunningNumber.
A new socket context is created with CreateSockInfo:

```
SOCK_INFO *CreateSockInfo(PROVIDER *Provider, SOCKET
ProviderSocket, SOCK_INFO *Inherit)
```

A variable to look for is Inherit, which specifies if the data is taken from another SOCK_INFO structure (as when a socket is accepted.)

After all the copies and initializations we add:

```
//Original code
    NewInfo->ProviderSocket     = ProviderSocket;
    NewInfo->bClosing           = FALSE;
    NewInfo->dwOutstandingAsync = 0;
    NewInfo->BytesRecv          = 0;
    NewInfo->BytesSent          = 0;
    NewInfo->Provider           = Provider;
    NewInfo->hWnd               = (Inherit ? Inherit-
>hWnd : 0;(
    NewInfo->uMsg               = (Inherit ? Inherit-
>uMsg : 0;(

//Our new code

   static iID=1;
   NewInfo->iRunningNumber=iID++;
```

So that in each call, the context info will contain this data for our later use.

14. Communicating with the LSP

Since LSP is a DLL that doesn't run in our process, we must use IPC (Inter Process Communication) to communicate with it.

There are many ways to do this: mail slots, COM, named pipes, and more.

Every programmer has his or her own preference, but we'll save that topic for another article.

Something to consider about IPC is the new integrity feature introduced in Windows Vista, there are limitations on how you can use the IPC, for example you can no longer create a shared memory from a non service process.

In Komodia's Redirector we use COM to communicate with the LSP, keep in mind that you need to adjust the default privileges of the COM server to allow low integrity processes to communicate with it.

15. LSP vs other components

For a quick review of the technologies you can refer to Komodia's site: NDIS, TDI, WFP.

This is a quick comparison with other networking technology:

	LSP	TDI	NDIS	WFP	WinPCAP
OS	2000-Win7	2000-Win7, MS plans to drop support on Win8	2000-Win7	Vista SP1-Win7	2000-Win7
Can modify data	Yes	Yes	Yes	Yes	No
User/Kernel	User	Kernel	Kernel	Both	Kernel
Stream/Packet	Stream	Both	Packet	Both	Packet

* Comparing from Win2000, also Win7 also means Win2008
** NDIS means NDIS IM, WinPCAP also means NDIS Protocol.

16. The way forward.

Now that you understand LSP, the MSDN documentation hopefully looks less complicated, and you can spend your precious time on creating the functionality you need instead of on figuring out what went wrong. It took me two months to get the LSP up and running without any crashes.

17. Komodia's LSP templates

This guide features two samples: one is the original from Microsoft, and the other contains the same files wrapped inside a VS6 project. However, we at Komodia don't use these exact files for our projects. We have modified the LSP Installer, because: the MS LSP installer is limited in scope; it may be useful for test environments, but when used on live machines with existing LSPs installed, gives rise to a host of problems. We have added about 10,000 lines of code to handle all sorts of exceptions and scenarios, in order to pick up where Microsoft left off. Read more in section 7 of this article, or view our Installer page at:
http://www.komodia.com/products/komodias-advanced-lsp-installer/

18. Komodia's Redirector: ready-to-use LSP suite

Now that you've had an overview of LSP management, you'll probably want to develop your own LSP based application. As we've mentioned, this is an extremely time-intensive trial-and-error process.

The good news: Komodia has already done it for you.

Komodia's Redirector is a ready-to-use LSP suite that covers 95% of all possible LSP applications such as: Content inspection, traffic redirection, traffic modification, spam filtering and more. It also allows you to extend it easily. It will save you the endless hours of guesswork and debugging, and allow you to start developing your product right away.

We are developing it full time for three years, do you want to spend your time on trying to build the same?

For more details visit the products we page at:
http://www.komodia.com/products/komodia-redirector/