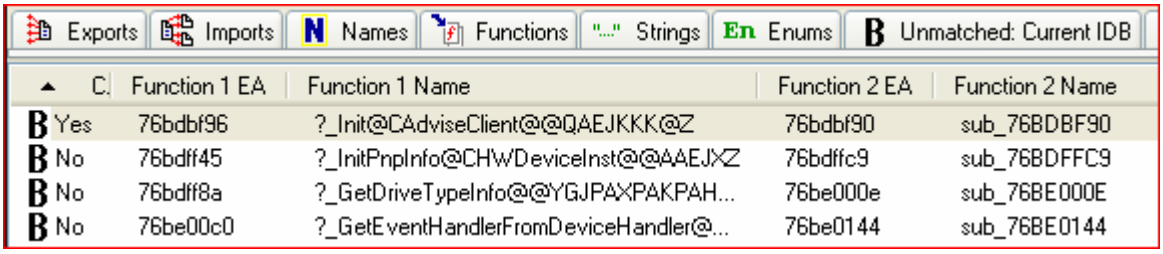
Analysis of Windows Shell Privilege Escalation Vulnerability, CVE-2007-0211

What we know:

1) Privilege escalation in Windows shell (hardware detection)
2) The patch installs new versions of shell32.dll and shsvcs.dll

Per BinDiff, by Sabre:



C.	Function 1 EA	Function 1 Name	Function 2 EA	Function 2 Name
Yes	76bdbf96	?_Init@CAdviseClient@@QAEJKKK@Z	76bdbf90	sub_76BDBF90
No	76bdff45	?_InitPnpInfo@CHWDeviceInst@@AAEJXZ	76bdffc9	sub_76BDFFC9
No	76bdff8a	?_GetDriveTypeInfo@@YGJPAXPAKPAH...	76be000e	sub_76BE000E
No	76be00c0	?_GetEventHandlerFromDeviceHandler@...	76be0144	sub_76BE0144

The only changed function is `Init()`, a method of `CAdviseClient` class. Here is a fragment of the class, based on the sub-function(s) that `Init()` calls, and the structure(s) (well, class members) seen in the database:

```
class CAdviseClient {
public:
	int Init(DWORD, HANDLE, int);
	long Cleanup(void);
	HANDLE hSourceProcessHandle;
	HANDLE hTargetHandle;
}
```

Inside the `Init()` function are `OpenProcess()` and `DuplicateHandle()`. Basically, it accepts a client process id (the process requesting help from the hardware detection server), and an open handle within that process' context. The handle can be an access token, communications device (hmm…), thread, process, mutex, or whatever.

The function creates a duplicate of that open handle for usage within its own context. Now, the problem is, that shsvcs.dll is loaded by svchost.exe which we know is going to be executing with full admin privileges. Also, when the function calls `DuplicateHandle()`, it does so by selecting itself, with full admin rights, for the target process. Understand the `DuplicateHandle()` a little better:

```
DuplicateHandle(
	//Handle to process that owns handle to dup
	HANDLE hSourceProcessHandle,
	//Handle to dup
	HANDLE hSourceHandle,
	//Handle to process to obtain dup'ed handle
	HANDLE hTargetProcessHandle,
	//Pointer to var to receive the dup'ed handle. Handle is then
	//valid in the context of the target process.
	LPHANDLE lpTargetHandle,
	[...]
);
```

So basically it is taking a handle from an unprivileged process context and preparing it for usage in a privileged process context. Say, for example, that the handle in question is a handle to one of the source process' threads. After the `DuplicateHandle()` call, if shsvcs.dll resumes the thread, it will then execute with the privileges of shsvcs.dll. So, it would be possible for a client to supply a thread that it cannot execute itself, but that it can pass off to shsvcs.dll for help executing within a privileged environment.

Let's now see how the patch prevents privilege escalation possibilities. It's pretty simple actually. Before calling `OpenProcess()` and `DuplicateHandle()`, the server (shsvcs.dll) calls `CoImpersonateClient()`. This kind of throws a mask over the server's permissions for a temporary period of time, and when the client's handle is duplicated, it is duplicated with the client's permissions and not the server's. When the server is done messing around, it calls `CoRevertToSelf()` to switch back into its own context.

The following code shows the two relevant `CAdviseClient` methods (the main function and its sub-function). Comments mark the code that was inserted by the patch to prevent the privilege escalation.

```
int CAdviseClient::Init(DWORD dwProcessId, HANDLE hSourceHandle, int
iDunno)
{
	bool duphandle=false;
	Cleanup();

	//__Start Patch
	HRESULT hRes=CoImpersonateClient();
	if (hRes!=S_OK) {
		CoRevertToSelf();
		return(E_FAIL);
	}
	//__End Patch

	hSourceProcessHandle=OpenProcess(
		SYNCHRONIZE | PROCESS_DUP_HANDLE |
		PROCESS_VM_WRITE | PROCESS_VM_OPERATION,
		false,
		dwProcessId
	);

	if (hSourceProcessHandle==NULL)
		return(E_FAIL);

	duphandle=DuplicateHandle(
		hSourceProcessHandle,
		hSourceHandle,
		GetCurrentProcess(),
		&hTargetHandle,
		PROCESS_ALL_ACCESS, //Kinda (0x1F0FFF vs 0x1F03FF)
		false, 0
	);
```

```
	if (!duphandle) {
		CoRevertToSelf();	//Added by patch
		return(E_FAIL);
	}

	return(0);
}

long CAdviseClient::Cleanup(void)
{
	//Close any open handles before reusing them...
	if (hSourceProcessHandle!=NULL)
	{
		CloseHandle(hSourceProcessHandle);
		hSoureProcessHandle=NULL;
	}
	if (hTargetHandle!=NULL)
	{
		CloseHandle(hTargetHandle);
		hTargetHandle=NULL;
	}
	return(0);
}
```

More reading here:

MSDN: Impersonating a Client
http://msdn2.microsoft.com/en-us/library/aa390870.aspx

MSDN: Duplicate Handle
http://msdn2.microsoft.com/en-us/library/ms724251.aspx

MSDN: Handle Inheritance
http://msdn2.microsoft.com/en-us/library/ms724466.aspx