

DirectSound Capture Using Deviare

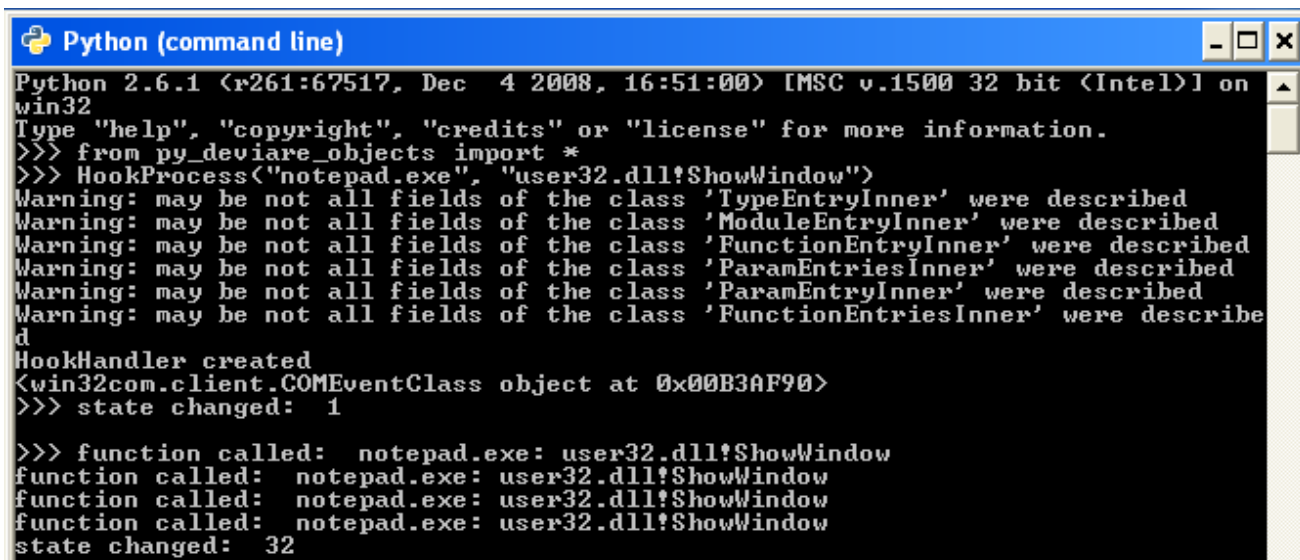
Contents

- [Introduction](#)
- [Research](#)
 - [Direct Sound Capturing](#)
 - [Monitoring Skype Conversations](#)
- [Implementation](#)
 - [Deviare Python Wrappers](#)
 - [Wave Tools](#)
 - [COM Type Libraries](#)
 - [Virtual Table Finder](#)
 - [Hooking Direct Sound](#)
- [Running Sound Capture](#)
 - [Easy Steps](#)
 - [Registration](#)
- [What's next](#)
 - [Optimizations](#)
 - [Wave API Hooking](#)
 - [Hook DirectSoundCapture And Listen To Full Conversations](#)
 - [Inspect More COM Interfaces](#)

Introduction

Today we are going to see how easy it can be to capture audio with Deviare. From players like Windows Media Player, instant messaging applications like Skype & Windows Live Messenger, to any application using DirectSound. The wave output will get captured by us.

Deviare is indeed a powerful framework. Built to resolve most complex tasks in the simplest way. With a few lines of python, all are hooking is done and running. Today performance is extremely important, yet Deviare proves itself as the best. It allows you to also take advantage of the full power of Python!



```
Python (command line)
Python 2.6.1 (r261:67517, Dec 4 2008, 16:51:00) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from py_deviare_objects import *
>>> HookProcess("notepad.exe", "user32.dll!ShowWindow")
Warning: may be not all fields of the class 'TypeEntryInner' were described
Warning: may be not all fields of the class 'ModuleEntryInner' were described
Warning: may be not all fields of the class 'FunctionEntryInner' were described
Warning: may be not all fields of the class 'ParamEntriesInner' were described
Warning: may be not all fields of the class 'ParamEntryInner' were described
Warning: may be not all fields of the class 'FunctionEntriesInner' were describe
d
HookHandler created
<win32com.client.COMEventClass object at 0x00B3AF90>
>>> state changed: 1

>>> function called: notepad.exe: user32.dll!ShowWindow
function called: notepad.exe: user32.dll!ShowWindow
function called: notepad.exe: user32.dll!ShowWindow
function called: notepad.exe: user32.dll!ShowWindow
state changed: 32
```

Research

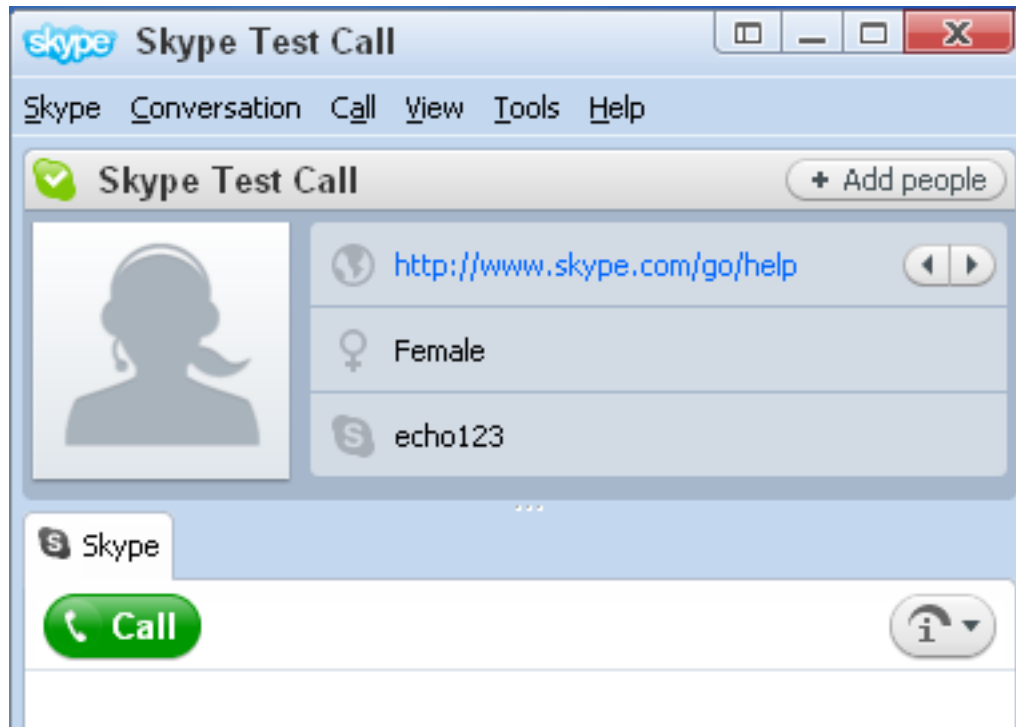
Direct Sound Capturing

I must be honest, I've never used *DirectX* API on my life, so I was a bit uncertain of how difficult this could be. I started by looking at MSDN documentation on [IDirectSound](#) and [IDirectSoundBuffer](#). The first goal was to find a *safe place* to read its sound buffers. I found out that [IDirectSoundBuffer::Unlock](#) could serve my intentions well. At this point, the user is telling DirectSound that he has finished writing his wave output and the locks may be released. So, if we step in between, we can safely read the buffer. The user is no longer writing to it, and DirectSound has not yet taken control of it.

I tested it on many applications and it turned to be the right choice. It works perfectly for WMP, Windows Live Messenger, and many others. No problems showed up until I stepped with Skype...

Monitoring Skype Conversations

This might be the way many applications handle their sound output, but it was the first application I have seen and I named the case after it. Later, I found a few [articles](#) describing it in detail.



So, to my surprise, I was not seeing any data being written to the sound buffers after the unlock was called. How the hell is it writing its wave, and how am I supposed to read it?!. It kept me thinking for a while, until I noticed an interesting and constant call to `IDirectSoundBuffer::GetCurrentPosition`. Then I realized that this writing method depends on constant reading of the *play* and *write* buffer pointers. That's because DirectSound, as most stream based implementations, works with a [circular buffer](#).

Capturing its wave output requires that we keep track of changes in the *write* pointer. Once we know it has moved forward in the buffer, we can read its steps. Since here we don't know how much the user has actually written to it, we must know the full size and location of the buffer. Unless we want to read garbage, but I'm sure that's not the case ;).

Implementation

Deviare Python wrappers

Before we get our hands dirty, let me introduce you to something new in Deviare: *Python Wrappers*.

As you already know Deviare is exposed through a series of COM interfaces. To save ourselves from the work of writing a whole new set of bindings, we used the well known project [PyWin32](#). It's very friendly to be used directly, as you may see in `py_deviare_objects.py`, just not enough to me. So I

built these wrappers on top of it and made them as transparent as possible. You'll find the use of the interface very similar to the way it's done in our C# examples and in compliance with the python way of life, of course.

```
#Handler for DHookEvents
class HookHandler(object):
    def __init__(self):
        print "HookHandler created"

    #private handlers
    def OnStateChanged(self, proc, newState, oldState):
        _proc_ = win32com.client.Dispatch(proc)
        self.PyStateChanged(_proc_, newState, oldState)

    def OnFunctionCalled(self, proc, callInfo, rCall):
        _proc_ = win32com.client.Dispatch(proc)
        _ci_ = win32com.client.Dispatch(callInfo)
        _rc_ = win32com.client.Dispatch(rCall)
        self.PyFunctionCalled(_proc_, _ci_, _rc_)
```

Wave Tools

I wrote this tools to help me write down the captured wave data. This may be obvious to people working in audio projects, but for me I cannot believe there is no native support in Windows to read-write Wav files! Yes, there is native support to write RICH content but come on!

Luckily for me, I found a small sample C++ class inside the DirectX SDK. This was good enough for me to write my own in Python. As you may see, my WaveFile class only supports the write operation. Though, adding a read member should be easy enough for you :). I have also added a lock to it, to ensure our data does not get corrupted by multiple thread operations. You may use it safely.

The structures used were defined exactly as found in DirectSound and WinMM headers. Some of them are used by DirectX to specify the format of the wave content.

COM Type Libraries

By default, DirectX installations do not register their library types. Since we need that information, so Deviare can hook them, I created my own definitions with the interfaces we are interested in. To prevent any collision with previous installations, I used a different GUID. There is a python script that takes care of its registration and it's automatically ran on demand by our example. Again, definitions are exactly as found on DirectX SDK.

```

[
    odl,
    uuid(0F0F0F0F-113A-4832-9104-000000000002),
    version(1.0),
    helpstring("IDirectSound")
]
interface IDirectSound : IUnknown {
    HRESULT _stdcall QueryInterface(
        GUID* __MIDL_0011,
        void** __MIDL_0012);
    unsigned long _stdcall AddRef();
    unsigned long _stdcall Release();
    HRESULT _stdcall CreateSoundBuffer(
        _DSBUFFERDESC* pcDSBufferDesc,
        IDirectSoundBuffer** ppDSBuffer,
        IUnknown* pUnkOuter);
    HRESULT _stdcall GetCaps(void* pDSCaps);
    HRESULT _stdcall DuplicateSoundBuffer(
        IDirectSoundBuffer* pDSBufferOriginal,
        IDirectSoundBuffer** ppDSBufferDuplicate);
    HRESULT _stdcall SetCooperativeLevel(
        wireHWND hwnd,
        unsigned long dwLevel);
    HRESULT _stdcall Compact();
    HRESULT _stdcall GetSpeakerConfig(unsigned long* pdwSpeakerConfig);
    HRESULT _stdcall SetSpeakerConfig(unsigned long dwSpeakerConfig);
    HRESULT _stdcall Initialize(GUID* pcGuidDevice);
};

```

Virtual Table Finder

To obtain the virtual tables for the interfaces, we basically have two options. Either we wait for its instantiation by the target process, or we find them ourselves on our own. Our first option is known to work for sure, yet we delay our installations until these events rise. This may also place us in a race and we may not capture all the output. The second one allows us to hook our targets immediately. Yet, in this case, we depend on the library (*dsound.dll*) being loaded in the same address space of our target. I have placed the two options in our example. If the current one is not working for you, uncomment the other at *py_deviare_directsound.py*.

Hooking Direct Sound

The first thing we need, is to know every time a sound buffer is created. For that we are going to intercept calls to [IDirectSound::CreateSoundBuffer](#). If the calls succeed, we look-up the table location inside the returned instance.

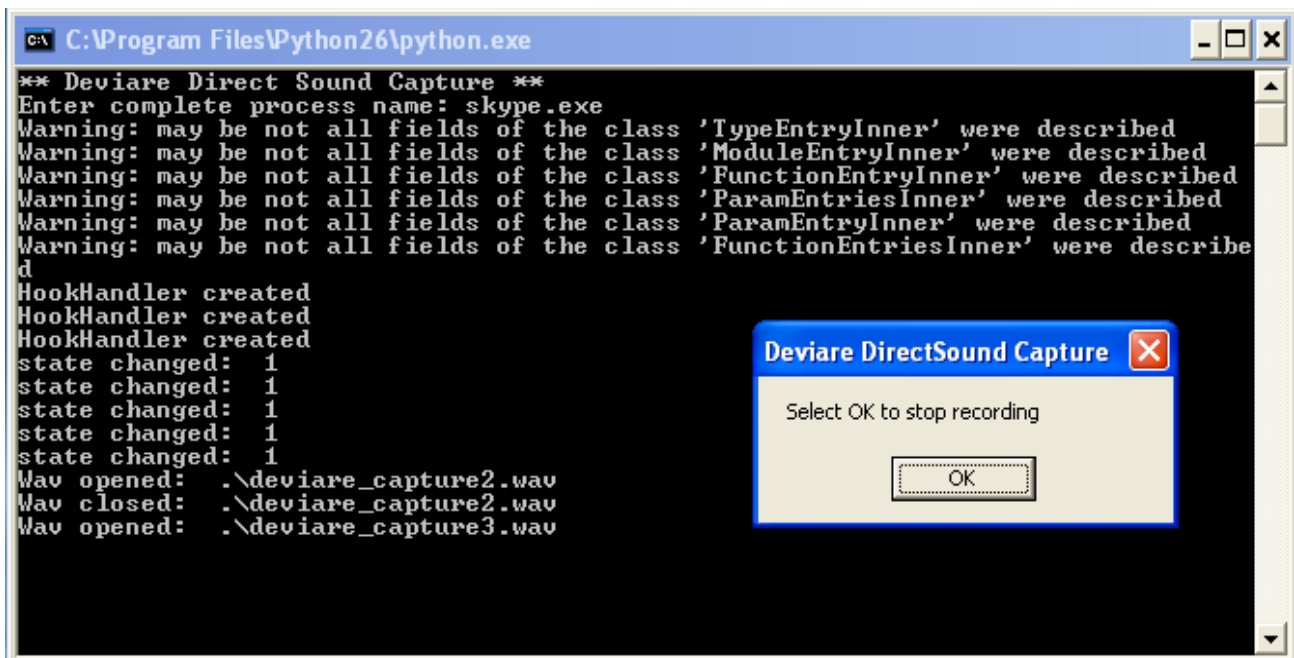
From there we are going to hook four members of [IDirectSoundBuffer](#): [Initialize](#), [SetFormat](#), [Unlock](#) and [GetCurrentPosition](#). The first two, are used to obtain the wave format that the user is writing to the buffer. We also need to watch details from the call that creates the sound buffer, in case it is specified there.

The [Unlock](#) member, as our research told us, is used in most applications to notify DirectX that the buffer is written and ready to be played. So we read the buffer pointers and size, to use [Deviare's](#) memory interface to copy all content. We need to be careful, and see if the call actually succeeded. Only then can we save the wave data, else we must discard our buffers.

With applications that keep track of the *play* and *write* cursors, we are going to monitor their calls to [GetCurrentPosition](#). As explained earlier, with this method, I need to know the full size of the buffer and its location. So I save it from the first call to `Unlock`. Then I virtually divided my buffer in `N` segments, and filled it with the wave data as the *write* cursor moves forward in the buffer. Once my buffer contains enough data, I write it down to the wave file. To prevent false positives, in the creation of sound buffers, I delay the creation of my file until I have real data stored.

In case we are monitoring the creation of `IDirectSound` in the target process, we also need to hook [DirectSoundCreate](#) and [DirectSoundCreate8](#) from `dsound.dll`. There we can obtain the virtual table for `IDirectSound`, and follow our quest.

Running Sound Capture



```
C:\Program Files\Python26\python.exe
** Deviare Direct Sound Capture **
Enter complete process name: skype.exe
Warning: may be not all fields of the class 'TypeEntryInner' were described
Warning: may be not all fields of the class 'ModuleEntryInner' were described
Warning: may be not all fields of the class 'FunctionEntryInner' were described
Warning: may be not all fields of the class 'ParamEntriesInner' were described
Warning: may be not all fields of the class 'ParamEntryInner' were described
Warning: may be not all fields of the class 'FunctionEntriesInner' were describe
d
HookHandler created
HookHandler created
HookHandler created
state changed: 1
state changed: 1
state changed: 1
state changed: 1
state changed: 1
Wav opened: .\deviare_capture2.wav
Wav closed: .\deviare_capture2.wav
Wav opened: .\deviare_capture3.wav
```

Easy Steps

Execute the `run_me.py` located among the deployed files, and you'll be prompted with a window to type the complete name of the process you want to start monitoring. For example: `Skype.exe`. Once the program starts capturing, the wave files will be written in the same folder.

Once you are done, click OK on the dialog box to stop recording. Now you can open the `.wav` files generated, and listen the capture. Do not open them before closing the example as the data may not be readable by then.

Registration

The first execution of our example, will automatically register its interfaces and data types. It

will also generate a file labeled *.deviare_types_registered* to prevent registration on the following executions. You can safely remove the file at any time you want the registration to be run again.

What's Next

Optimizations

At any point of our handling, performance is essential. Any delay is highly punished by the sound output. So we must be careful about any operation we do inside the function call. This example tries to cache enough data before doing a write operation to disk. In case you need to improve its performance, you should read the data and release the call as soon as possible. Then in a different thread, or in a non punitive call, flush our data to the wave file.

Wave API hooking

This example could be very easily adapted to capture wave data from applications using WinMM API. Most browsers, Flash, and Google Talk use it to throw their sound output.

Hook DirectSoundCapture And Listen To Full Conversations

You should have noticed, when capturing from Skype, that your own voice is not heard. That's because the application is not echoing its capture from the microphone. To get that too, it is necessary to hook [IDirectSoundCaptureBuffer](#) and proceed the same way to read its buffer.

Inspect More COM Interfaces

If you want to discover a lot more about the internals of DirectSound, then Deviare will be very valuable for you. Inspecting COM object is very easy indeed. Simply define one if its interfaces (if its not already registered in the system) and hook them the simpler way.

If you are wondering what other interfaces may be useful for you, try our [Deviare COM Console](#) to discover them. It comes with source code, and you are free to adapt it to your needs!

And That's All Folks, hope you find it useful, enjoy!