Hidden Internet connections

Part 1 - Windows and Internet connections

written by Steinowitz

November 1999

1 Introduction

It's often said that Microsoft writes certain hidden features in its software which automatically connects to Microsoft servers to transmit data about the user. All this incoming information is stored in huge databases which Microsoft subsequently uses for its merciless tactics in maintaining its world-dominating position...

How can we prove that Microsoft is doing this? And is Microsoft the only company doing this? Aren't there *many* other software companies doing exactly the same? How can we detect (hidden) Internet connections? How can we see what data is transmitted? And what could we do to prevent this? How could we modify data being transmitted? How could we prevent a program from connecting to the Internet?

I'll try to answer all these questions (and many more!) in a new series of essays called 'Hidden Internet connections'. And, of course, I won't be the only one who'll show you, many others will try to do exactly the same. And since you don't know which essays are better before you've read all of them, I highly recommend you to read all of them. Sounds logical, doesn't it?

2 Before we start

The essays in this series will all about Microsoft Windows 95/98. Actually, I'll use Windows 98, but there are only small differences between Windows 95 and Windows 98. If you want to be sure that everything is exactly the same, you should get yourself Windows 98 UK (version 4.10.1998).

Also, I assume that you have a very basic knowledge of reverse engineering. I am not going to explain what a disassembler does, neither am I going to explain what a debugger is. Some basic knowledge about networks (especially the Internet) would be very useful, but I I'll explain most of that. Some of you may think it's boring to read because you already know, but that's always the case with essays.

Tools I might use in this series are Numega SoftIce 3.24, Numega Boundschecker 5.02 (VC++ edition), Numega SmartCheck 6.0, URSoft W32Dasm 8.93, File Monitor, several registry monitors, Hex Workshop 2.0 and HIEW 5.92. The last tool I'd like to mention is very special one: WinTECH's SocketSpy/32.

More on this tool in part 3 of this series. I'll let you know when I use any other tools.

Alright, enough about this, let's start with the real thing.

3 Telnet

If we want to be able to detect hidden Internet connections, we should first learn how to detect Internet connections. And what would be a better way to learn this than to learn it by reversing a target from which we *already know* that it connects to the Internet and from which we also know *when* it connects to the Internet? Exactly! That's why I've chosen Microsoft's *Telnet* application as our first target.

We already know that Microsoft provides standard API's for all basic functions a programmer might want to use. Therefore, we assume that there also is a Windows DLL available which allows you to connect to other computers connected to the Internet. Imagine that we have no idea which DLL this could be, how would we find out?

3.1 Finding the DLL name

Launch W32Dasm (or IDA, if you prefer) and disassemble our first target: telnet.exe! This is a very small executable, which makes it very suitable for our objective. Since telnet.exe has to import all functions it would like to use, we should be able to find the DLL name we're looking for in the imported functions window. ADVAPI32, comdlg32, GDI32, no, we already know those DLL's, we're not looking for any of them. But when we scroll to the bottom of the imported functions, we'll see some interesting function names.

Among some others, we'll see: `WSOCK32.connect`, `WSOCK32.send`, `WSOCK32.recv`, `WSOCK32.closesocket`. Gotcha! Those are exactly the function names we're looking for. We now know the filename of the DLL we were looking for: *wsock32.dll*, which you'll find in your Windows system directory (e.g., *c:\windows\system*).

3.2 Breaking on `connect`

Sounds great, but we haven't got any evidence that these really are the functions used for Internet connections. We'll proof that our assumptions were right with a little help of SoftIce. Since WSOCK32 might be a DLL we never looked at before, we should first check the SoftIce exports in *winice.dat*. Add the line `EXP=C:\windows\system\wsock32.dll` if it's not yet there and restart your computer. (Or load the DLL using the Symbol Loader.)

We can now set breakpoints on WSOCK32 functions as if it is a function like `hmemcpy`, which we all know so well. Press Ctrl-D and type `bpx connect` to set a breakpoint on the `WSOCK32.connect` function. The next thing we do is launch *telnet.exe* and try to connect to some hosts to see if the `connect` function is called.

Let's try to connect to *212.212.212.212*. I'm currently offline, since I'll try to reduce the amount of online reversing as much as possible, because of the phone costs. Let's just see if Telnet calls `connect` anyway.

Wow! Our first break on a WSOCK32 API call! But, believe me, it won't be our last. We'll use this a lot more in this 'Hidden Internet connection' series.

The next thing we try is to connect to *www.microsoft.com*. Hmm, no break! So there's a difference between IP addresses like *212.212.212.212* and hostnames like *www.microsoft.com*. What could this difference be? What would happen if we would go online and try to connect to the Microsoft hostname? Hey, it does break on `connect` now!

If you want to know what the difference exactly is, you should read part 2 of this series, I'll explain you there. For now, it's enough to assume that there is a difference. If we want to break on that Microsoft connection attempt while we are offline, we should set a breakpoint on `gethostbyname`. After setting this breakpoint (and clearing any others), try to connect to *www.microsoft.com*. Also try to connect to an IP address while you've got this breakpoint set: it won't break!

4 Physical and virtual connections

It's very important that you understand the difference between a *physical connection* and what I'll call a *virtual connection*. Therefore, I'll explain what these are.

When your computer dials up to your Internet provider, it's establishing a physical connection between your computer and the computer you're dialing to. You could compare it to calling someone: you dial the number and when someone who hears the phone ringing picks it up, a physical connection has been established.

The person picking up the phone will probably say his or her name, but if you don't say anything after that and he/she also keeps quite, there's nothing but the physical connection we just established.

You could now say your name and you could ask if you could talk to the person you called for. This is comparable to establishing a virtual connection: your computer says 'Hi there, I'm computer X and I'd like to talk to computer Y now.'. And if the person (or the computer) who picked up the phone isn't very rude, you (or your computer) will soon be able to talk with the one you (or your computer) asked for.

That is what we call a virtual connection, although there are some differences between people calling and connected computers. The most important difference is that your computer is able to establish as many virtual connections as it wants. You probably don't want to talk to 10 persons at the same time, but your computer can handle communicating with 10 other computers connected to the Internet at the same time.

Now back to the `connect` function we learned about in the previous section. This function does not establish a physical connection: if there's is no physical connection, it won't be able to connect to the computer you wanted to connect to. `Connect` tries to establish a virtual connection to the target computer, in order to make it possible to communicate with that computer. And, just like you when you're calling someone, it uses the physical connection to achieve that.

As you probably already expected, there also is a DLL which provides an API you could use to establish the physical connection: *wininet.dll*. We might want to explore that API sooner or later, but we'll skip that one now.

5 Winsock

Let's take a closer look at our `WSOCK32` DLL. This DLL is really very important for us. Therefore, I'll dedicate a whole part of this series to this DLL later. Something you should know is that the common name for this DLL is 'Winsock'.

There are a couple of things we could do now. We could dive into the API documentation available, but with the knowledge we currently have, that would be rather difficult. Therefore, we'll use some other techniques.

The first thing we'd like to know is: which functions are provided by the Winsock API? Disassemble *wsock32.dll* and have a good look at the exported functions.

There are quite some exported functions, but we won't need all of them for our purposes. Near the bottom of the list, we see about 20 functions starting with `WSA`. One of them is `WSAStartup`. You probably already guessed what I was thinking? Should a program first initiate WinSock before establishing connections using `connect`? Let's try it!

Set a breakpoint on `WSAStartup` and start some programs which use the Internet. Telnet, break! mIRC, break! HotDog, about 4 breaks during startup! Netscape, break! WS_FTP Pro, break! Internet Explorer, no break... Hmm, let's try to access a website. Ah, break!

I think we've just found a nice way to see if a program uses Internet connections, don't you think so?

Assuming that programs need to import functions from our Winsock DLL, we could do a little search to see how many programs are able to establish and use Internet connections. Search all your harddrives for files containing the text 'wsock32', you'll be amazed how many DLL's and executables import functions from this DLL!

Until now, I haven't mentioned the fact that there are more Winsock DLL's: *wsock32.dll*, *wsock32n.dll*, *mswsock.dll*, *ws2_32.dll* and a 16-bits Winsock: *winsock.dll*. More about all these Winsock DLL's in part 3 of this series.

6 Last words

I hope you were able to follow this essay and that you learned something. For those who already knew all this: read the other essays of this series I'll write, since each new essay will proceed where the previous essay left.

Last, but not least, personal greetz go out to everyone I know... Special greetz go out to ~S~, you did a great job!

Steinowitz
switz@newmail.net
http://dread99.cjb.net and *http://www.s.url*