1

Tools

This chapter stresses the importance of building and saving the tools required for rootkit development. Building a full-featured rootkit toolkit before you begin development enables you to research, design, develop, test, and package your rootkit without distraction. In addition, saving the tools, utilities, samples, scripts, and even the failed experiments enables you to pick up where you left off at any time. As an example, the rootkit presented in this book was originally developed and forgotten several years ago, but came to mind when I was contacted by Wiley, the publisher. Having the code, the scripts, the utilities, and a copy of the toolkit used to develop the rootkit, all in one convenient archive, turned an otherwise complex project into a delightful experience.

This chapter includes the following:

- What must go into a rootkit toolkit
- What should go into a rootkit toolkit
- How to verify the usefulness of your rootkit toolkit

How Do I Build a Rootkit?

Assembling a complete rootkit toolkit will take a lot of time. Fortunately, everything you need to get started can be downloaded from Microsoft (`http://msdn2.microsoft.com/en-us/default.aspx`). The three most important tools you need are the Microsoft Driver Development Kit (DDK), a C compiler, and the Windows Platform Software Development Kit (SDK). Fortunately, these can all be downloaded from Microsoft without cost.

Though the Visual C++ compiler and the Software Development Kit (SDK) can be downloaded directly, the Driver Development Kit (DDK) can only be downloaded as an ISO image (unless you happen to have a Microsoft MSDN subscription). At the time of this writing, you can get the ISO image from `www.microsoft.com/whdc/devtools/ddk/default.mspx`. This image can be

transferred to a CD using the "record a disk from a disk image" feature of your CD burning software. If you do not have the capability to burn a CD from an ISO image, and you don't have (or know someone who has) a Microsoft MSDN subscription, you can order the Windows Server 2003 SP1 DDK CD at no cost (other than a small shipping and handling fee) from `www.microsoft.com/whdc/devtools/ddk/orderddkcd.mspx`.

Currently, Microsoft Visual C++ 2005 Express is available for download, free of charge, from `http://msdn.microsoft.com/vstudio/express/visualc/download`. This development environment has everything needed to develop basic Windows applications. In addition, Visual C++ 2005 Express has a C compiler that will enable you to create the console programs needed to load, unload, and test the rootkits developed in this book.

The console programs you will be creating are native Win32 programs, so you will also need to download and install the Microsoft Windows Platform SDK separately. The SDK (`PSDK-x86.exe`) can currently be downloaded from `www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en#filelist`.

Combined, the Driver Development Kit, the Visual C++ compiler (or any Windows-compatible C compiler) and the Platform SDK will enable you to follow along with, compile, and run every example in this book. There are, however, several utilities that will make rootkit development much easier, the first of which is DebugView. This utility enables you to see debugging statements while executing rootkits. Though this is not technically a necessity for rootkit development, I can't imagine writing a rootkit without it. In addition to DebugView, the good folks at Sysinternals also provide Diskmon, Filemon, and Regmon, three utilities that enable you to monitor disk activity, file system activity, and registry activity, respectively. You'll want to have these in your toolkit as well. If you have downloaded the source code for this book, you will find individual archives for each of these utilities under "Chapter 1 Tools."

If you want to delve deeply into the technology behind rootkits, you will also want to get a copy of *IDA*. IDA is the reverse-engineering tool that will be used in Chapter 4 to pick apart the PGP encryption library. At the time of this writing, IDA cannot be downloaded from the creators, DataRescue. You can purchase IDA Pro from DataRescue, but you will need to perform an Internet search to find a download link for the free version of IDA. To the best of my knowledge, the last free version of IDA is 4.1, so entering **ida + "4.1 ida pro" download datarescue** should get you a list that contains at least one download link. Alternately, if you have downloaded the source code for this book, you will find the individual archive `IDA_4_1` under "Chapter 1 Tools."

Another tool for delving into the deepest layers of rootkit development is "Debugging Tools for Windows." This package contains four debuggers, one of which is a kernel-level debugger that can come in handy when your device driver isn't working as expected and debugging statements just aren't enough to figure out what's going on. This package includes the most recent DDKs, so you may already have it. If not, the package can be downloaded from `www.microsoft.com/whdc/devtools/debugging/installx86.mspx`. Kernel-level debugging isn't covered in this book, but "Debugging Tools for Windows" is nonetheless a valuable addition to any rootkit toolkit.

You will find that the kernel debugger mentioned above is of little value without the symbols for the operating system you are using. You can get these symbols from `www.microsoft.com/whdc/devtools/debugging/symbolpkg.mspx`. After downloading and installing the symbols, you need to tell your

kernel debugger where they are. From Start ➪ All Programs ➪ Debugging Tools for Windows ➪ WinDbg, select the menu option File ➪ Symbol File Path, and browse to the directory where symbols were installed. Selecting the Symbols directory will magically transform the kernel debugger into a fountain of information that can be used to both fix rootkits and investigate new rootkit technologies.

There is one additional development tool mentioned in this book that has yet to be covered: the Visual C# compiler used to create the rootkit controller developed in Chapter 11. This is another free development environment offered by Microsoft, and can be found at `http://msdn.microsoft.com/vstudio/express/visualcsharp`. The Visual Studio C# 2005 development environment will not interfere with the Visual Studio C++ 2005 development environment, so feel free to download and install both. The C# compiler also makes a good addition to any rootkit toolkit.

Once you have the Microsoft DDK, a Windows C/C++ compiler, the Microsoft Windows Platform SDK, Sysinternals' DebugView, RegMon, FileMon, and DiskMon, DataRescue's IDA, Debugging Tools for Windows, Kernel Debugging Symbols, and Visual Studio C# 2005 Express, you will be ready to tackle basic rootkit development. Remember that the toolkit you develop can be a valuable collection for years to come, so take a moment to zip and archive the components you've collected before jumping into rootkit development. Figure 1-1 shows a typical rootkit toolkit.

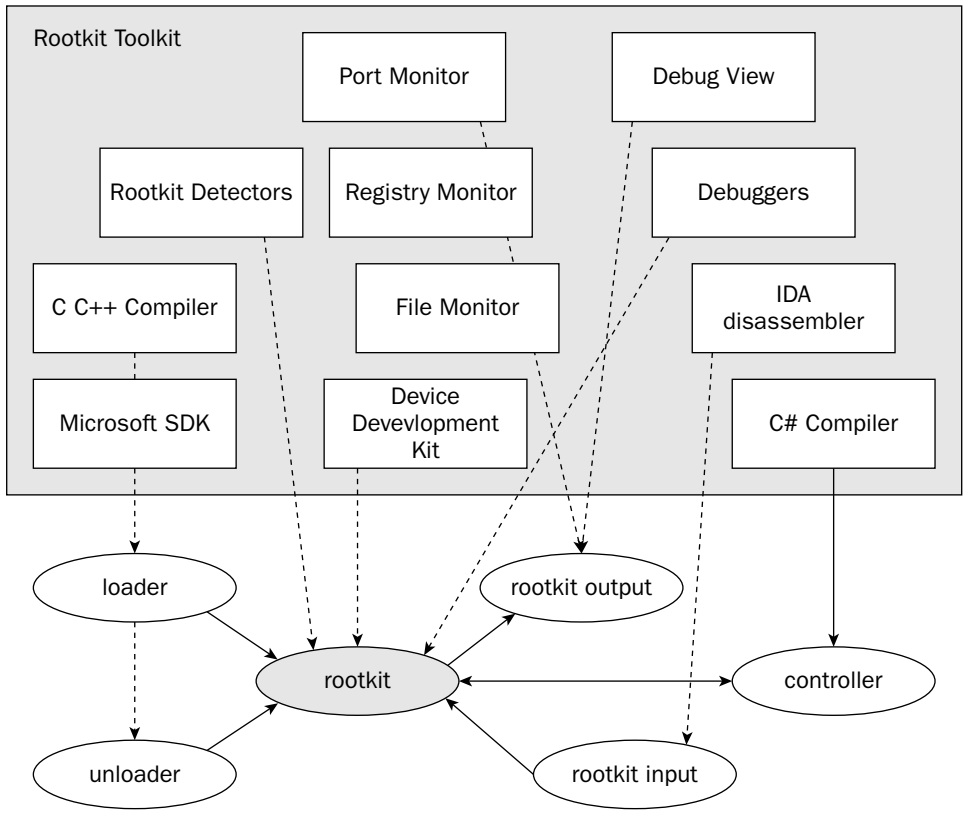

Figure 1-1

The Microsoft Driver Development Kit

The DDK installation wizard is as straightforward as any Microsoft installation; just double-click `setup.exe` and answer a few questions. However, you can do a few things to make rootkit development much easier. The single most time-saving installation recommendation is to use the default installation directory; and if you absolutely must use another directory, keep the path simple, with no spaces or long directory names. This will be especially important if you are using the older XP DDK. The second recommendation is to select every possible download option. Skipping a few samples or skipping documentation to save a few megabytes of disk space will not make your life easier. You can, however, skip the debuggers offered with the 2003 SP1 DDK if you've already installed "Debugging Tools for Windows," as these are the exact same debuggers.

Once you have installed the DDK, you can create two shortcuts to help with development. The shortcut examples that follow were developed for the Windows XP DDK, build number 2600; your target path may need to be altered depending upon your version of the DDK.

The first shortcut should use the following target:

```
%windir%\SYSTEM32\CMD.EXE /k C:\WINDDK\2600\bin\setenv.bat C:\WINDDK\2600 chk
```

For newer DDKs, use the following:

```
%windir%\SYSTEM32\CMD.EXE /k C:\NTDDK\bin\setenv.bat C:\NTDDK checked
```

For the 2003 SP1 DDK, use

```
%windir%\SYSTEM32\CMD.EXE /k C:\WINDDK\3790.1830\bin\setenv.bat C:\WINDDK\3790.1830
checked
```

and start in the `%windir%` directory.

This will be your "Checked DDK" icon.

The second shortcut should use this target:

```
%windir%\SYSTEM32\CMD.EXE /k C:\WINDDK\2600\bin\setenv.bat C:\WINDDK\2600 fre
```

For newer DDKs, use the following:

```
%windir%\SYSTEM32\CMD.EXE /k C:\NTDDK\bin\setenv.bat C:\NTDDK free
```

For the 2003 SP1 DDK, use

```
%windir%\SYSTEM32\CMD.EXE /k C:\WINDDK\3790.1830\bin\setenv.bat C:\WINDDK\3790.1830
free
```

and start in the same `%windir%` directory.

This will be your "Free DDK" icon.

The DDK uses the concept of Checked and Free driver development to differentiate between preliminary debug development and final release builds. The preceding shortcuts will set up the shells required for these two development environments. The projects covered in this book only use the Checked DDK shell, but you will eventually want to build a release version of one or all of the rootkits you will be creating.

Microsoft Visual VC++ 2005 Express

Unlike the DDK install, the Visual C++ 2005 Express installation might take a few minutes. For one thing, you have to use the Background Intelligent Transfer Service (BITS). If you get a message indicating that you must start this service, you will need to reconfigure the service to start automatically and then start it. This service might seem like a huge security hole just waiting to be exploited, but Microsoft has been pushing BITS for a long time, and there are currently no (publicly announced) known exploits taking advantage of this design, so you will need to go with the flow — at least until VC++ 2005 Express is fully installed.

As with the DDK, selecting the default installation path is recommended. Though an alternate path should not cause any problems, why ask for trouble? Unlike the DDK, there is no need to select all installation options. You may choose not to integrate the SQL Server. It isn't required for any project covered in this book, but there is always the chance that you will one day develop an application that requires a database, so add it if you have the room.

The only strong recommendation that can be made is to include MSDN. Yes, it's a large package that's available over the Internet, but you can use Google to overcome most obstacles and you will not find a better integrated resource for developing software under VC++ 2005 Express, so do yourself a favor and check the MSDN box during installation.

Microsoft Software Developers Kit

The Microsoft Platform SDK installation (`PSDK-x86.exe`) is also wizard driven. Just answer the questions, agree to the license agreement, and keep clicking Next. The default for the most recent SDK install selects a custom installation. You should keep the custom installation selection, but only to add "Register Environment Variables" to the installation options. This should make compiling a little easier. Once installed, the SDK features described in the MSDN help files will be available to your programs.

Sysinternals Freeware

DebugView is freely available from Sysinternals at `www.sysinternals.com/Utilities/DebugView.html`. Don't let the price fool you; DebugView is an invaluable tool that will make rootkit development much easier. Download and create a shortcut for this utility before going too much farther.

You can also download Diskmon, Filemon, and Regmon from Sysinternals. These utilities can monitor disk, file, and registry activity, respectively. Debugging statements won't always be able to tell you

what's happening, but these utilities will. Adding them to your toolkit and creating shortcuts to them will make development that much easier. Eventually things will get complicated, as they always do; and when that happens, you'll want all the help you can get.

IDA

As mentioned earlier, a Google search of **ida + "4.1 ida pro" download datarescue** should provide a list of web pages from which IDA can be downloaded. Once downloaded and installed, IDA can be used to look into Windows applications, libraries, and even device drivers. This tool provides an incredible wealth of information to the experienced user. Unfortunately, becoming an experienced user can be a daunting task. This book will walk you through basic IDA use, but if you intend to write rootkits, you will need to learn much more about IDA. If you already know that reverse engineering will be an important part of your future plans, I would recommend using this book as a primer and then replace the word "download" with the word "tutorial" in the Google search mentioned earlier.

Debugging Tools for Windows

The four debuggers provided in this package are exceptional tools. In particular, the Kernel Debugger can be invaluable for both fixing rootkits and investigating new rootkit technologies. Many of the difficulties encountered using undocumented Windows internals can also be overcome with this debugger. Unfortunately, this is also a complex utility that requires many hours to master. Fortunately, the Windows Debugger has a complete help system that can walk you through every step. If you are new to kernel debugging, I suggest you start with menu option Help ➪ Contents, and just keep reading.

Verification

To verify your Microsoft DDK installation, open a Checked shell (if you were following along you have an icon named "Checked DDK" on your desktop) and build one of the samples selected during the DDK installation. To build a sample, you need to traverse into a sample directory (any directory under the installation directory containing a "sources" file) and enter the command **build**. If you have installed properly, entering a build command from either the "Checked DDK" or "Free DDK" shell will initiate driver compilation and linking based on the "sources" file contained in that directory. Following the build, you can double-check your installation by searching for the newly created driver (`*.sys`) file in a directory beneath your build directory.

To verify your Microsoft VC++ 2005 Express installation, double-click the Microsoft VC++ 2005 Express icon. From the main menu, select File ➪ New ➪ Project. In the Project Types view, select Win32. From the Templates view, select Win32 Console Application. Enter the project name **myProject** and the solution name **MySolution,** and then press OK and Finish. Add the line **"printf("Hello World!\n");** just before the return in `_tmain`. You can now build the solution from the main menu by selecting Build ➪ Build Solution. If all is well, you should be able to open a command prompt, navigate to the solution directory defined during creation, and from the Debug directory, execute `myProject.exe`. If Microsoft VC++ 2005 Express was installed correctly, you should see Hello World! at the command prompt.

To verify IDA, double-click `idaw.exe` (or the shortcut you've already created) and click OK at the opening screen. Then use Windows Explorer to navigate to your `WINDOWS\System32` directory. From the `System32` directory, drag and drop any dynamic link library (`*.dll`) onto the IDA file selection dialog. Then press OK twice (you may also have to press OK a third time to truncate data from a large segment) to load and analyze the library. Once loaded, IDA should provide an assembly code listing of the contents of the file beginning with the public start entry point.

To verify Debugging Tools for Windows, click WinDbg from Start ➪ All Programs ➪ Debugging Tools for Windows. From the Windows Debugger, select the menu option File ➪ Symbol File Path. This path should have been set after downloading and installing the symbols for your specific operating system. Check the Reload check box and press OK. If you have a Windows XP or later operating system and you have never loaded symbols, this should bring up the Local Kernel Debugger. If you have previously loaded symbols, you might need to use menu option File ➪ Kernel Debug and click OK from the Local tab to bring up the Local Kernel Debugger. In either case, the Local Kernel Debugger window should show no errors after the lines "Loading Kernel Symbols" or "Loading User Symbols." To verify kernel debug operation, enter **!process 0 0** in the command box (after `lkd>`). You should see a detailed list of processes.

VCVARS32.BAT

After verifying the Visual C++ build environment, you will need to prepare for manual compiling and linking. Microsoft uses the convention `VCVARS32.BAT` as the filename used to prepare a Command Prompt window for manual compiling and linking. If you installed Microsoft VC++ 2005 Express to the default location, `VCVARS32.BAT` can be found in `C:\Program Files\Microsoft Visual Studio 8\VC\bin`. You need to copy this file to a convenient location and execute it before manually building the user-level programs found in this book. Alternately, you can create a shortcut to `cmd.exe` that executes `VCVARS32.BAT` from its default location. Once the setup file has been executed from a Command Prompt window, you will be able to manually compile and link source code from that window.

Other Tools to Consider

This might be a bit premature, but rootkit development also depends upon rootkit detection and prevention tools. Once you have a thorough understanding of these tools, you can design and develop rootkits that defeat these detection and prevention systems. Of course, the developers of these detection and prevention systems do not consider them to be tools at all; they are more likely considered to be "security applications," but to the rootkit developer, they are simply tools. See Chapter 13, "Detecting Rootkits," and Chapter 14, "Preventing Rootkits," for these tools.

What to Keep Out

Rootkits are often installed as payloads. A *payload* is the content section of an exploit. Exploits are the intrusions that take advantage of software vulnerabilities in order to add unintended software (payloads) to target machines. There are many types of payloads, and many exploits that can be used to deliver these payloads. This is one application detail of a rootkit that can also be applied to spyware,

viruses, and other malicious program types. Separating rootkit development from exploit development will provide an object-oriented environment in which any payload can be attached to any exploit. The advantage to this approach can be seen by using MetaSploit software (`www.metasploit.com`). MetaSploit enables the user to first select an exploit and then select the payload to insert using that exploit. Keeping these functions separated can be difficult if rootkit development is folded in with exploit development. Because rootkit development and exploit development require some of the same tools, it is easy to mix these development environments and end up with a rootkit that can only be compiled and linked in an exploit development environment that has changed since the last rootkit build.

Summary

At this point, you should have (as a minimum) the following:

- A Microsoft Windows Driver Development Kit (XP, 2000, or 2003)
- A C/C++ compiler (VC++ 2005 Express)
- The Microsoft Platform Software Development Kit

Also recommended are the following:

- MSDN
- A Kernel Debug Output Utility (DebugView from Sysinternals)
- IDA
- Debugging Tools for Windows

Once you have downloaded, installed, and verified the tools discussed in this chapter, you will be able to compile and run the code presented in this book. If you wish to follow along without compiling source code, binaries for each chapter can be downloaded from the Wrox website at `www.wrox.com` and run on any Windows 2000, XP, or 2003 operating system.