1

SETTING UP YOUR PYTHON ENVIRONMENT

This is the least fun—but nevertheless critical—part of the book, where we walk through setting up an environment in which to write and test Python. We are going to do a crash course in setting up a Kali Linux virtual machine (VM) and installing a nice IDE so that you have everything you need to develop code. By the end of this chapter, you should be ready to tackle the exercises and code examples in the remainder of the book.

Before you get started, go ahead and download and install VMWare Player.[1] I also recommend that you have some Windows VMs at the ready as well, including Windows XP and Windows 7, preferably 32-bit in both cases.

1. You can download VMWare Player from *http://www.vmware.com/*.

Installing Kali Linux

Kali is the successor to the BackTrack Linux distribution, designed by Offensive Security from the ground up as a penetration testing operating system. It comes with a number of tools preinstalled and is based on Debian Linux, so you'll also be able to install a wide variety of additional tools and libraries beyond what's on the OS to start.

First, grab a Kali VM image from the following URL: *http://images.offensive-security.com/kali-linux-1.0.9-vm-i486.7z*.[2] Download and decompress the image, and then double-click it to make VMWare Player fire it up. The default username is *root* and the password is *toor*. This should get you into the full Kali desktop environment as shown in Figure 1-1.

Figure 1-1: The Kali Linux desktop

The first thing we are going to do is ensure that the correct version of Python is installed. This book will use Python 2.7 throughout. In the shell (**Applications ▸ Accessories ▸ Terminal**), execute the following:

```
root@kali:~# python --version
Python 2.7.3
root@kali:~#
```

2. For a "clickable" list of the links in this chapter, visit *http://nostarch.com/blackhatpython/*.

If you downloaded the exact image that I recommended above, Python 2.7 will be automatically installed. Please note that using a different version of Python might break some of the code examples in this book. You have been warned.

Now let's add some useful pieces of Python package management in the form of `easy_install` and `pip`. These are much like the `apt` package manager because they allow you to directly install Python libraries, without having to manually download, unpack, and install them. Let's install both of these package managers by issuing the following commands:

```
root@kali:~#: apt-get install python-setuptools python-pip
```

When the packages are installed, we can do a quick test and install the module that we'll use in Chapter 7 to build a GitHub-based trojan. Enter the following into your terminal:

```
root@kali:~#: pip install github3.py
```

You should see output in your terminal indicating that the library is being downloaded and installed.

Then drop into a Python shell and validate that it was installed correctly:

```
root@kali:~#: python
Python 2.7.3 (default, Mar 14 2014, 11:57:14)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import github3
>>> exit()
```

If your results are not identical to these, then there is a "misconfiguration" in your Python environment and you have brought great shame to our Python dojo! In this case, make sure that you followed all the steps above and that you have the correct version of Kali.

Keep in mind that for most examples throughout this book, you can develop your code in a variety of environments, including Mac, Linux, and Windows. There are some chapters that are Windows-specific, and I'll make sure to let you know at the beginning of the chapter.

Now that we have our hacking virtual machine set up, let's install a Python IDE for development.

WingIDE

While I typically don't advocate commercial software products, WingIDE is the best IDE that I've used in the past seven years at Immunity. WingIDE provides all the basic IDE functionality like auto-completion and explanation of function parameters, but its debugging capabilities are what set it

apart from other IDEs. I will give you a quick rundown of the commercial version of WingIDE, but of course you should choose whichever version is best for you.[3]

You can grab WingIDE from *http://www.wingware.com/*, and I recommend that you install the trial so that you can experience firsthand some of the features available in the commercial version.

You can do your development on any platform you wish, but it might be best to install WingIDE on your Kali VM at least to get started. If you've followed along with my instructions so far, make sure that you download the 32-bit `.deb` package for WingIDE, and save it to your user directory. Then drop into a terminal and run the following:

```
root@kali:~# dpkg -i wingide5_5.0.9-1_i386.deb
```

This should install WingIDE as planned. If you get any installation errors, there might be unmet dependencies. In this case, simply run:

```
root@kali:~# apt-get -f install
```

This should fix any missing dependencies and install WingIDE. To verify that you've installed it properly, make sure you can access it as shown in Figure 1-2.

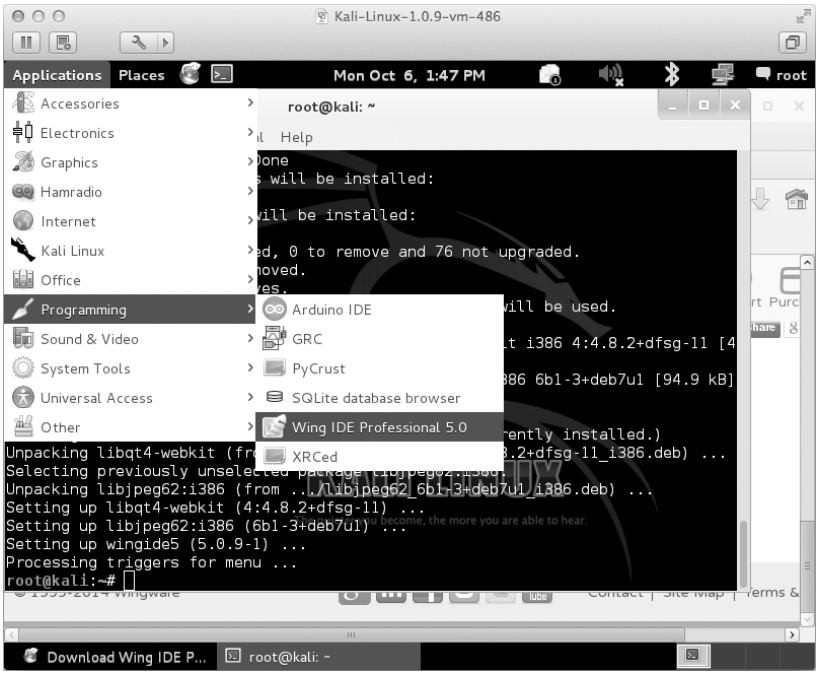

Figure 1-2: Accessing WingIDE from the Kali desktop

3. For a comparison of features among versions, visit *https://wingware.com/wingide/features/*.

Fire up WingIDE and open a new, blank Python file. Then follow along as I give you a quick rundown of some useful features. For starters, your screen should look like Figure 1-3, with your main code editing area in the top left and a set of tabs on the bottom.

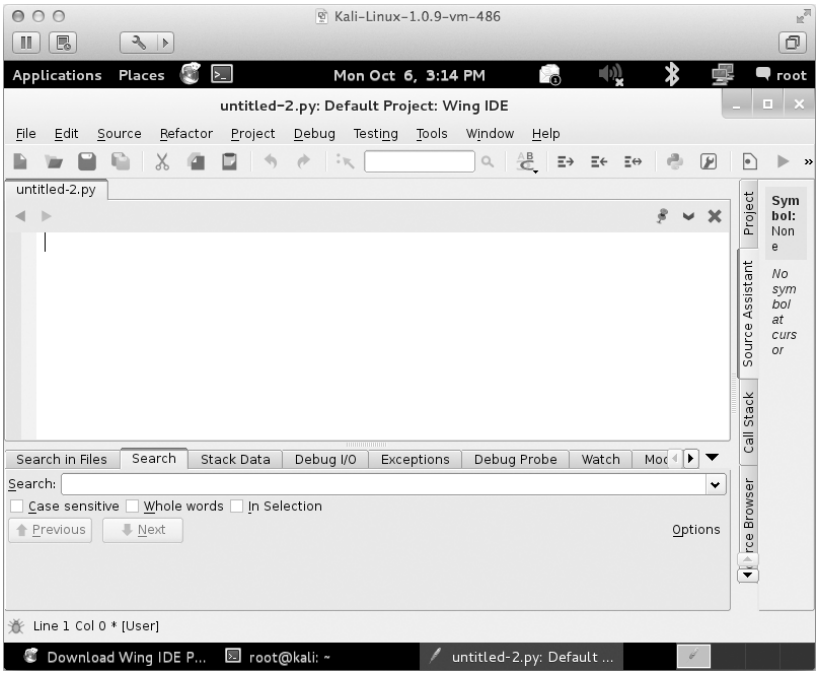

Figure 1-3: Main WingIDE window layout

Let's write some simple code to illustrate some of the useful functions of WingIDE, including the Debug Probe and Stack Data tabs. Punch the following code into the editor:

```
def sum(number_one,number_two):
    number_one_int = convert_integer(number_one)
    number_two_int = convert_integer(number_two)

    result = number_one_int + number_two_int

    return result

def convert_integer(number_string):

    converted_integer = int(number_string)
    return converted_integer

answer = sum("1","2")
```

This is a very contrived example, but it is an excellent demonstration of how to make your life easy with WingIDE. Save it with any filename you want, click the **Debug** menu item, and select the **Select Current as Main Debug File** option, as shown in Figure 1-4.

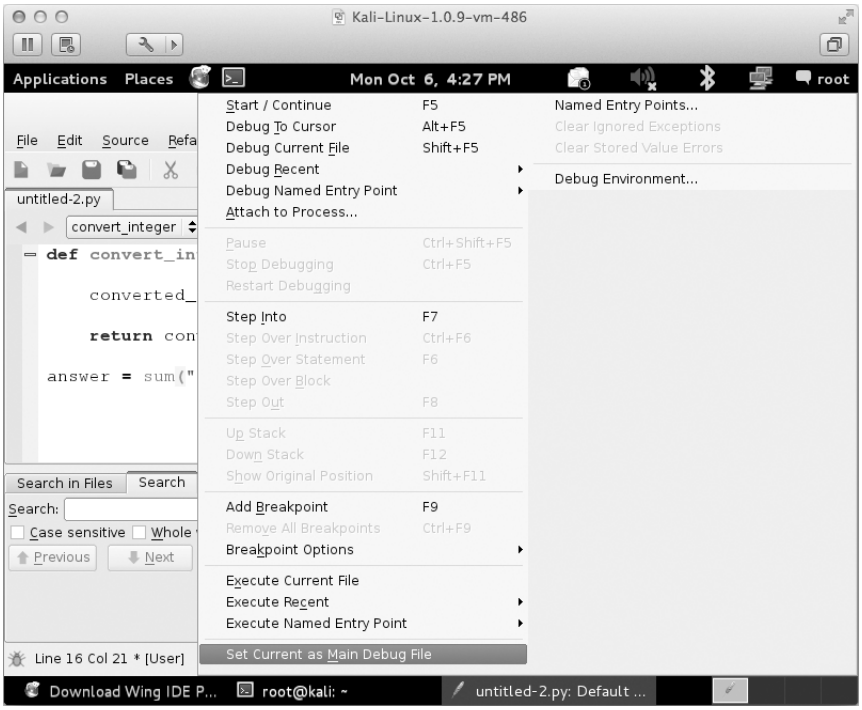

Figure 1-4: Setting the current Python script for debugging

Now set a breakpoint on the line of code that says:

```
return converted_integer
```

You can do this by clicking in the left margin or by hitting the F9 key. You should see a little red dot appear in the margin. Now run the script by pressing F5, and execution should halt at your breakpoint. Click the **Stack Data** tab and you should see a screen like the one in Figure 1-5.

The Stack Data tab is going to show us some useful information such as the state of any local and global variables at the moment that our breakpoint was hit. This allows you to debug more advanced code where you need to inspect variables during execution to track down bugs. If you click the drop-down bar, you can also see the current call stack, which tells you which function called the function you are currently inside. Have a look at Figure 1-6 to see the stack trace.

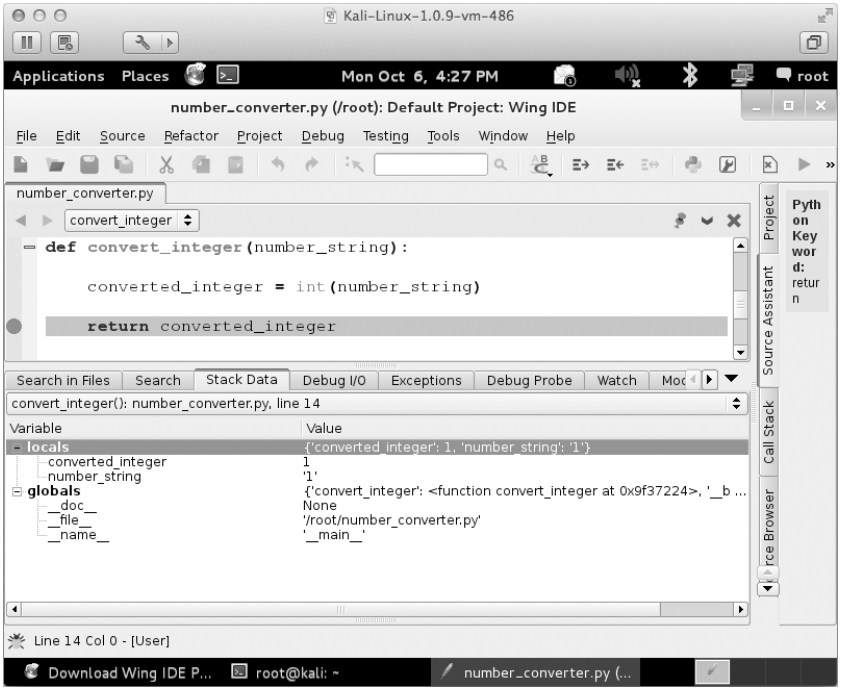

Figure 1-5: Viewing stack data after a breakpoint hit

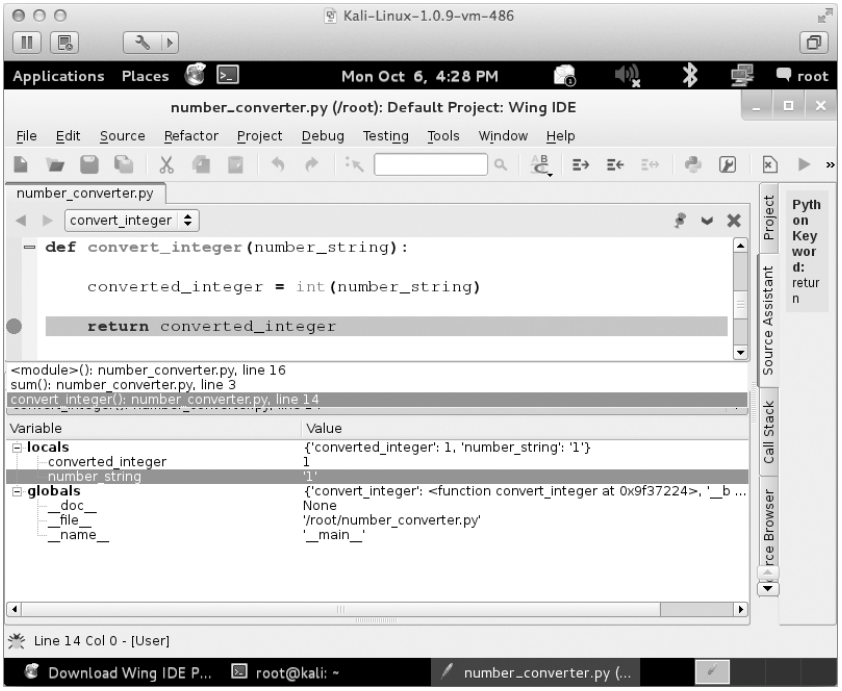

Figure 1-6: Viewing the current stack trace

We can see that `convert_integer` was called from the `sum` function on line 3 of our Python script. This becomes very useful if you have recursive function calls or a function that is called from many potential places. Using the Stack Data tab will come in very handy in your Python developing career!

The next major feature is the Debug Probe tab. This tab enables you to drop into a Python shell that is executing within the current context of the exact moment your breakpoint was hit. This lets you inspect and modify variables, as well as write little snippets of test code to try out new ideas or to troubleshoot. Figure 1-7 demonstrates how to inspect the `converted_integer` variable and change its value.

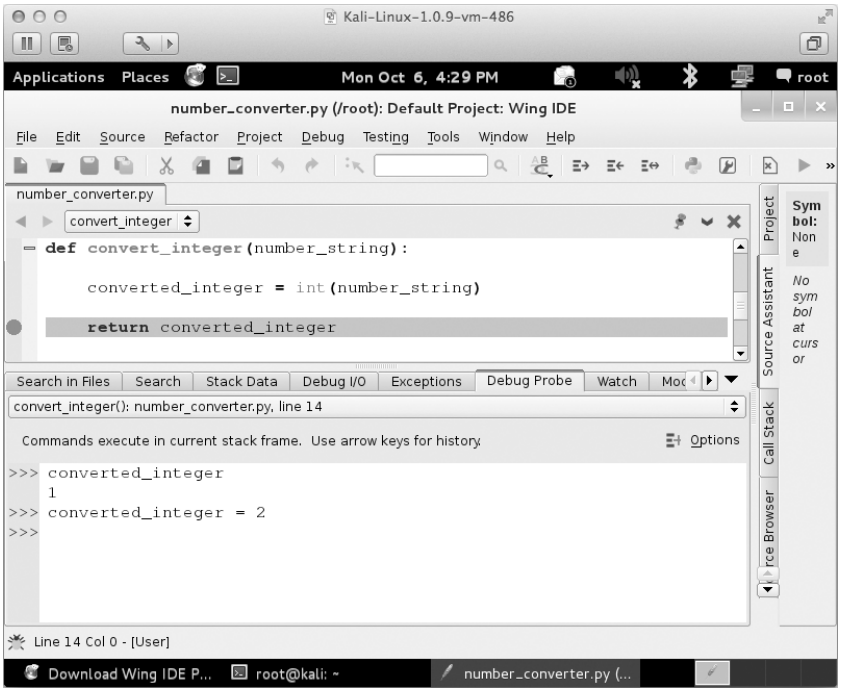

Figure 1-7: Using Debug Probe to inspect and modify local variables

After you make some modifications, you can resume execution of the script by pressing F5.

Even though this is a very simple example, it demonstrates some of the most useful features of WingIDE for developing and debugging Python scripts.[4]

That's all we need in order to begin developing code for the rest of this book. Don't forget about making virtual machines ready as target machines for the Windows-specific chapters, but of course using native hardware should not present any issues.

Now let's get into some actual fun!

4. If you already use an IDE that has comparable features to WingIDE, please send me an email or a tweet because I would love to hear about it!