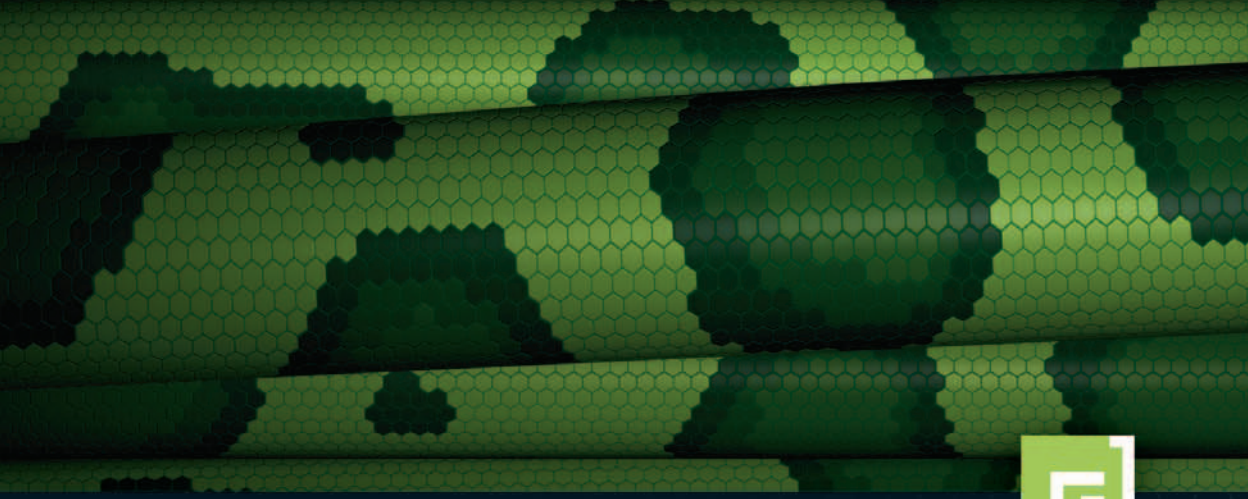

Justin Seitz

Hacking mit Python

Fehlersuche, Programmanalyse, Reverse Engineering

dpunkt.verlag

Justin Seitz ist als Sicherheitsingenieur bei der Firma Immunity, Inc., beschäftigt, wo er einen Großteil seiner Arbeitszeit für Fehlersuche, Reverse Engineering, Python-Entwicklung und Analyse von Malware aufwendet.

Justin Seitz

Hacking mit Python

Fehlersuche, Programmanalyse, Reverse Engineering

dpunkt.verlag

Übersetzung: Peter Klicman, Köln
Copy-Editing: Ursula Zimpfer, Herrenberg
Lektorat: Dr. Michael Barabas
Herstellung: Birgit Bäuerlein
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, Paderborn

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

ISBN 978-3-89864-981-0

1. Auflage 2009

Title of American original: Gray Hat Python: Python Programming for Hackers and Reverse Engineers.
No Starch Press, Inc., San Francisco · www.nostarch.com
ISBN 978-1-59327-192-3

5 4 3 2 1 0

Vorwort

Der Satz, den ich bei Immunity am häufigsten höre, ist wohl: »Ist es schon fertig?« Typische Unterhaltungen laufen etwa so ab: »Ich beginne mit der Arbeit an dem neuen ELF-Importer für den Immunity Debugger.« Kurze Pause. »Ist er schon fertig?« oder »Ich habe gerade einen Bug im Internet Explorer gefunden«! Und dann: »Ist der Exploit schon fertig?« Es ist dieses enorme Tempo bei Entwicklung und Modifikation, das Python zur perfekten Wahl für das nächste Sicherheitsprojekt macht, sei es die Entwicklung eines speziellen Decompilers oder eines vollständigen Debuggers.

Wenn ich manchmal Ace Hardware hier in South Beach besuche, wird mir ganz schwindlig, wenn ich den Gang mit den Hammern entlanggehe. Es werden etwa 50 verschiedene Modelle ausgestellt, schön in Reih und Glied sortiert. Jeder weist einen kleinen, aber extrem wichtigen Unterschied zu den anderen auf. Ich bin nicht Handwerker genug, um den idealen Einsatzbereich für jedes Werkzeug zu erkennen, aber das gleiche Prinzip gilt auch bei der Entwicklung von Sicherheitstools. Insbesondere bei der Arbeit an Webanwendungen oder vom Kunden selbst entwickelten Programmen verlangt jedes Assessment einen ganz speziellen »Hammer«. In der Lage zu sein, etwas zusammenzubauen, das sich in die SQL-API einklinkt, hat ein Immunity-Team bei mehr als einer Gelegenheit gerettet. Aber natürlich gilt das nicht nur für Assessments. Sobald man sich in die SQL-API einklinken kann, ist es einfach, ein Tool zu entwickeln, das SQL-Queries auf Anomalien untersucht und so Ihre Organisation mit einem schnellen Bugfix vor einem hartnäckigen Angreifer schützt.

Jeder weiß, dass es sehr schwierig ist, Entwickler von Sicherheitsmaßnahmen dazu zu bewegen, als Teil eines Teams zu arbeiten. Wenn man sie mit irgendeinem Problem konfrontiert, neigen die meisten von ihnen dazu, zuerst die Bibliothek neu zu schreiben, mit der sie das Problem angehen wollen. Nehmen wir an, es geht um eine Sicherheitslücke in irgendeinem SSL-Daemon. Es ist sehr wahrscheinlich, dass Ihr Mitarbeiter zuerst einen SSL-Client von Grund auf neu entwickeln will, weil »die vorhandene SSL-Bibliothek so *furchtbar*« ist.

Sie müssen das auf jeden Fall verhindern. Die SSL-Bibliothek ist nämlich in Wahrheit nicht furchtbar – sie wurde nur nicht in dem Stil geschrieben, den der fragliche Mitarbeiter bevorzugt. Die Fähigkeit, in einen großen Codeblock abzutauchen, ein

Problem zu finden und es zu beheben, ist der Schlüssel, um mit einer funktionierenden SSL-Bibliothek einen Exploit zu entwickeln, der noch relevant ist. Und in der Lage zu sein, die Mitarbeiter als Team arbeiten zu lassen, ist entscheidend, um die notwendigen Fortschritte zu erzielen. Ein mit Python vertrauter Entwickler ist viel Wert, ebenso wie ein Ruby-Spezialist. Der Unterschied ist die Fähigkeit der »Pythonistas« zusammenzuarbeiten, alten Quellcode zu nutzen, ohne ihn neu zu schreiben, und darüber hinaus als eine Art Superorganismus zu funktionieren. Die Ameisenkolonie in Ihrer Küche hat in etwa die gleiche Masse wie ein Tintenfisch, aber es ist wesentlich aufwendiger, sie loszuwerden!

Und genau an diesem Punkt hilft Ihnen dieses Buch. Sie besitzen wahrscheinlich schon Tools für einige der von Ihnen zu erledigenden Arbeiten. Sie sagen: »Ich habe Visual Studio. Es beinhaltet einen Debugger. Ich muss keinen eigenen spezialisierten Debugger entwickeln.« Oder: »WinDbg besitzt doch eine Plug-in-Schnittstelle?« Und die Antwort ist natürlich »Ja«. WinDbg besitzt eine Plug-in-Schnittstelle und Sie können diese API nutzen, um sich mit der Zeit etwas Nützliches aufzubauen. Doch eines Tages werden Sie sagen: »Mann, es wäre doch wesentlich besser, wenn ich das mit 5.000 anderen WinDbg-Nutzern verbinden könnte, damit wir unsere Ergebnisse abgleichen können.« Und wenn Sie Python nutzen, brauchen Sie zusammen gerade einmal 100 Zeilen Code sowohl für einen XML-RPC-Client als auch für einen Server, und schon ist jeder synchronisiert und arbeitet auf der gleichen Wellenlänge.

Hacking ist nicht das Gleiche wie Reverse Engineering – Ihr Ziel besteht *nicht* darin, den Originalquellcode der Anwendung ans Licht zu bringen. Ihr Ziel ist es, das Programm oder System besser zu verstehen, als die Leute, die es entwickelt haben. Sobald Sie dieses Verständnis besitzen, egal in welcher Form, sind Sie in der Lage, das Programm anzugreifen und die darin versteckten Exploits an die Oberfläche zu bringen. Das bedeutet, dass Sie zu einem Experten in Sachen Visualisierung, entfernter Synchronisation, Graphentheorie, der Lösung linearer Gleichungen, Techniken der statischen Analyse und einer ganzen Reihe anderer Dinge werden müssen. Immunitys Entscheidung in dieser Richtung war die komplette Standardisierung mittels Python, sodass wir jedes Mal, wenn wir einen Graphenalgorithmus entwickelt haben, diesen über all unsere Tools hinweg nutzen konnten.

In Kapitel 6 zeigt Justin, wie man einen schnellen Hook für Firefox entwickelt, der Benutzernamen und Passwörter abfängt. Das ist einerseits eine Sache, die Malware-Entwickler machen würden – und die Berichte zeigen, dass Malware-Entwickler Hochsprachen für genau diese Art von Dingen nutzen (*http://philosecurity.org/2009/01/12/interview-with-an-adware-author*). Andererseits ist es etwas, das man in 15 Minuten zusammenschustern kann, um den Entwicklern diejenigen Annahmen über ihre Software vorzuführen, die schlicht und einfach falsch sind. Softwareunternehmen investieren viel in den Schutz ihrer internen Daten. Angeblich geschieht das aus Sicherheitsgründen, aber häufig geht es doch eher um Kopierschutz und Digital Rights Management (DRM).

Was also bringt Ihnen dieses Buch? Die Fähigkeit, sehr schnell Softwaretools zur Manipulation anderer Anwendungen zu entwickeln. Und Sie werden dies in einer Art und Weise tun, die es Ihnen oder einem Team erlaubt, auf Ihren Erfolgen aufzubauen. Das ist die Zukunft der Sicherheitstools: schnell implementiert, schnell modifiziert, schnell verbunden. Bleibt wohl nur noch eine Frage übrig: »Ist es schon fertig?«

Dave Aitel
Miami Beach, Florida
February 2009

Danksagungen

Ich möchte meiner Familie dafür danken, dass sie mich während der ganzen Zeit ertragen hat, während der ich an diesem Buch geschrieben habe. Meine vier wundervollen Kinder Emily, Carter, Cohen und Brady gaben mir einen Grund, weiter an diesem Buch zu schreiben. Ich liebe euch dafür, die großartigen Kinder zu sein, die ihr seid. Meinen Brüdern und meiner Schwester möchte ich einen Dank sagen für die Ermutigung während des Entstehungsprozesses. Ihr habt selbst schon einige dicke Schinken geschrieben und es war immer hilfreich, jemanden zu haben, der die Sorgfalt versteht, die notwendig ist, um jede Art technischer Arbeit abzuliefern – ich liebe euch. Meinem Vater, dessen Sinn für Humor mir durch die vielen Tage half, an denen mir nicht nach Schreiben war – ich liebe dich, Harold. Hör nicht damit auf, jeden um dich herum zum Lachen zu bringen.

All denen, die diesem frischgebackenen Entwickler von Sicherheitsmaßnahmen halfen – Jared DeMott, Pedram Amini, Cody Pierce, Thomas Heller (der Über-Python-Mann), Charlie Miller –, schulde ich ein großes Dankeschön. Immunity-Team, ohne Frage wart ihr mir beim Schreiben dieses Buches eine unglaubliche Hilfe. Und ihr habt mir sehr dabei geholfen, nicht nur als Python-Mann zu wachsen, sondern auch als Entwickler und Forscher. Ein großes Dankeschön an Nico und Dami für die Zeit, die ihr damit verbracht habt, mir auszuhelfen. Dave Aitel, mein Lektor, half mir dabei, die Dinge zu einem Ende zu bringen, und stellte sicher, dass alles einen Sinn ergibt und lesbar ist. Dankeschön Dave. Dem anderen Dave, Dave Falloon, gilt ein großer Dank für das Korrekturlesen dieses Buches. Er ließ mich über meine eigenen Fehler lachen, rettete mein Notebook auf der CanSecWest und ist ein wandelndes Lexikon des Netzwerk-Wissens.

Abschließend, und ich weiß, dass sie immer zuletzt kommen, möchte ich dem Team von No Starch Press danken. Tyler blieb während des gesamten Buchprojekts an meiner Seite (glauben Sie mir, Tyler ist der geduldigste Mensch, dem ich je begegnet bin). Dank an Bill für den großartigen Perl-Becher und die Worte des Zuspruchs, Megan für ihre Hilfe, dieses Buch so schmerzlos wie möglich fertigzustellen, und an die restliche Crew, die hinter den Kulissen dafür sorgt, dass diese großartigen Bücher auch erscheinen. Ein großes Dankeschön an euch alle. Ich weiß zu schätzen, was ihr alles für mich getan habt.

Nachdem die Danksagungen so lange gedauert haben wie eine Grammy-Dankesrede, möchte ich mich noch bei denen bedanken, die mir geholfen haben und die ich in dieser Liste vergessen habe.

Inhaltsverzeichnis

Einführung

Ich habe Python nur wegen des Hackings gelernt – und ich behaupte, dass das auch für eine ganze Reihe anderer Leute gilt. Ich habe viel Zeit damit verbracht, eine Sprache zu suchen, die sich für das Hacking und das Reverse Engineering eignet, und vor ein paar Jahren wurde mir klar, dass sich Python zum Anführer in der Liga der Hacking-Programmiersprachen entwickeln würde. Das Problematische an der Sache war, dass es keine Anleitung gab, wie man Python für eine Vielzahl von Hacking-Aufgaben nutzen konnte. Man musste sich durch Forum-Postings und Manpages kämpfen und viel Zeit damit verbringen, den Code immer wieder durchzugehen, damit die Dinge richtig funktionierten. Dieses Buch möchte diese Lücke schließen und Ihnen zeigen, wie man Python auf unterschiedlichste Art und Weise für das Hacking und das Reverse Engineering nutzen kann.

Das Buch vermittelt Ihnen die Theorie zu vielen Hacking-Tools und -Techniken, wie Debuggern, Hintertüren, Fuzzern, Emulatoren und Code-Injection, und erläutert ihnen gleichzeitig, wie Sie sich vorgefertigte Python-Tools zunutze machen können, wenn eigene Lösungen unnötig sind. Und Sie lernen nicht nur, wie man Python-basierte Tools nutzt, sondern wie man solche Tools in Python *entwickelt*. Doch seien Sie gewarnt: Dies ist keine allumfassende Referenz! Es gibt sehr viele in Python geschriebene Infosec-Tools (»Information Security«), die hier nicht behandelt werden. Dennoch erlaubt es Ihnen dieses Buch, Ihr Wissen auf viele weitere Anwendungen zu übertragen, sodass Sie jedes Python-Tool Ihrer Wahl nutzen, debuggen, erweitern und anpassen können.

Sie können dieses Buch auf verschiedene Arten durcharbeiten. Wenn Sie mit Python oder der Entwicklung von Hacking-Tools noch nicht vertraut sind, sollten Sie es vom Anfang bis zum Ende lesen. Sie werden die erforderliche Theorie kennen lernen, Unmengen an Python-Code entwickeln und ein solides Verständnis dafür erwerben, wie eine Vielzahl von Hacking- und Reverse-Engineering-Aufgaben anzupacken sind. Wenn Sie bereits mit Python vertraut sind und die Python-Bibliothek ctypes schon kennen, dann können Sie direkt mit Kapitel 2 loslegen. Diejenigen, die sich schon länger mit der Materie beschäftigen, können sich nach Gutdünken innerhalb des

Buches bewegen und Codefragmente oder bestimmte Abschnitte so nutzen, wie es ihre tägliche Arbeit erfordert.

Ich widme einen Großteil der Zeit den Debuggern, angefangen mit der Debugger-Theorie in Kapitel 2 bis hin zum Immunity Debugger in Kapitel 5. Debugger sind ein wichtiges Werkzeug für jeden Hacker und ich betone ausdrücklich, dass ich sie umfassend behandle. Danach lernen Sie in den Kapiteln 6 und 7 einige Hooking- und Injection-Techniken, die einen Teil der Debugging-Konzepte zur Programmkontrolle und Speicherverarbeitung erweitern.

Der nächste Abschnitt des Buches zeigt, wie man Anwendungen mithilfe sogenannter Fuzzer knackt. In Kapitel 8 führen wir in das Fuzzing ein, und Sie werden einen eigenen einfachen Datei-Fuzzer entwickeln. In Kapitel 9 nutzen wir das leistungsfähige Sulley Fuzzing-Framework, um einen echten FTP-Daemon zu knacken, und in Kapitel 10 werden Sie lernen, wie man einen Fuzzer entwickelt, der Windows-Treiber knackt.

In Kapitel 11 schließlich lernen Sie, wie man Aufgaben der statischen Analyse unter IDA Pro (einem populären Tool zur binären statischen Analyse) automatisiert. Wir runden das Buch in Kapitel 12 mit PyEmu, dem Python-basierten Emulator, ab.

Ich habe versucht, die Codelistings kurz zu halten, wobei ich an bestimmten Stellen umfangreiche Kommentare eingefügt habe, die die Funktionsweise des Codes erläutern. Ein Teil des Erlernens einer neuen Sprache oder einer neuen Bibliothek besteht in der schweißtreibenden Arbeit, den Code tatsächlich »herunterzuschreiben« und die eigenen Fehler zu entdecken. Ich empfehle Ihnen, den Code wirklich einzugeben, auch wenn der gesamte Quellcode auf *www.dpunkt.de/python-hacking* abgelegt ist, damit Sie ihn bequem herunterladen können.

Lassen Sie uns beginnen!

1 Ihre Entwicklungsumgebung einrichten

Bevor Sie sich der Kunst der Hacking-orientierten Python-Programmierung widmen können, müssen Sie sich zuerst mit dem am wenigsten unterhaltsamen Teil dieses Buches auseinandersetzen: der Einrichtung Ihrer Entwicklungsumgebung. Eine solide Entwicklungsumgebung ist von besonderer Bedeutung, weil sie es Ihnen erlaubt, sich auf die interessanten Informationen in diesem Buch zu konzentrieren, statt Ihre Zeit darauf zu verschwenden, den Code irgendwie zum Laufen zu bringen.

Dieses Kapitel behandelt die Installation von Python 2.5, die Konfiguration Ihrer Eclipse-Entwicklungsumgebung und die Grundlagen der Entwicklung C-kompatiblen Codes mit Python. Sobald Sie Ihre Umgebung eingerichtet und die Grundlagen verstanden haben, liegt Ihnen die Welt zu Füßen. Dieses Buch zeigt Ihnen, was diese Welt zu bieten hat.

1.1 Anforderungen an das Betriebssystem

Ich setze voraus, dass Sie für einen Großteil Ihrer Entwicklungen eine 32-Bit-Windows-Plattform verwenden. Windows verfügt über die größte Anzahl von Tools und eignet sich selbst sehr gut für die Python-Entwicklung. Alle Kapitel in diesem Buch sind Windows-spezifisch, und die meisten Beispiele funktionieren nur mit einem Windows-Betriebssystem.

Allerdings gibt es auch einige Beispiele, die Sie unter einer Linux-Distribution ausführen können. Für die Linux-Entwicklung empfehle ich den Download einer 32-Bit-Linux-Distribution als VMware-Appliance. VMwares Appliance-Player ist kostenlos verfügbar und ermöglicht es Ihnen, Dateien von Ihrem Entwicklungssystem schnell auf Ihre virtuelle Linux-Maschine zu kopieren. Wenn Sie eine Maschine übrig haben, können Sie natürlich auch eine komplette Distribution darauf installieren. Im Hinblick auf dieses Buch sollten Sie eine Red Hat-basierte Distribution wie Fedora Core 7 oder Centos 5 verwenden. Alternativ können Sie natürlich auch mit Linux arbeiten und Windows emulieren. Es liegt ganz bei Ihnen.

Freie VMware-Images

VMware stellt auf seiner Website ein Verzeichnis mit freien Anwendungen bereit. Diese ermöglichen es dem Reverse Engineer oder dem Entwickler, der nach Sicherheitslücken sucht, Malware oder Anwendungen zu Analysezwecken innerhalb einer virtuellen Maschine auszuführen. Das minimiert das Risiko für die physikalische Infrastruktur und schafft einen isolierten Bereich, in dem man gefahrlos arbeiten kann. Besuchen Sie den »virtuellen Marktplatz« unter *http://www.vmware.com/appliances/* und laden Sie den Player unter *http://www.vmware.com/ products/player/*.

1.2 Python 2.5 herunterladen und installieren

Die Python-Installation erfolgt sowohl unter Linux als auch unter Windows schnell und problemlos. Windows-Anwendern steht ein Installer zur Verfügung, der sich um das gesamte Setup kümmert. Unter Linux erfolgt die Installation hingegen über den Quellcode.

1.2.1 Python unter Windows installieren

Windows-Anwender können den Installer von der Python-Homepage herunterladen: *http://python.org/ftp/python/2.5.1/python-2.5.1.msi*. Klicken Sie den Installer doppelt an und folgen Sie den einzelnen Installationsschritten. Der Installer legt ein Verzeichnis namens *C:/Python25/* an, in dem der Interpreter *python.exe* zu finden ist sowie alle standardmäßig installierten Bibliotheken.

Hinweis Alternativ können Sie den Immunity Debugger installieren, der nicht nur den Debugger selbst umfasst, sondern auch einen Installer für Python 2.5. In späteren Kapiteln werden wir den Immunity Debugger für viele Aufgaben nutzen, d.h., Sie können gleich zwei Fliegen mit einer Klappe schlagen. Um den Immunity Debugger herunterzuladen und zu installieren, besuchen Sie *http://debugger.immunityinc.com/*.

1.2.2 Python unter Linux installieren

Um Python 2.5 unter Linux zu installieren, laden Sie den Quellcode herunter und kompilieren ihn. Dadurch bekommen Sie die vollständige Kontrolle über die Installation, während Sie gleichzeitig die Python-Installation behalten, die auf einem Red Hat-basierten System vorhanden ist. Die Installation geht davon aus, dass Sie alle nachfolgenden Befehle als Benutzer *root* ausführen.

Der erste Schritt besteht im Herunterladen und Entpacken des Python 2.5-Quellcodes. In einer Terminal-Session geben Sie die folgenden Kommandozeilenbefehle ein:

```
# cd /usr/local/
# wget http://python.org/ftp/python/2.5.1/Python-2.5.1.tgz
# tar –zxvf Python-2.5.1.tgz
# mv Python-2.5.1 Python25
# cd Python25
```

Sie haben nun den Quellcode heruntergeladen und nach */usr/local/Python25* entpackt. Der nächste Schritt besteht im Kompilieren des Quellcodes und natürlich müssen Sie sicherstellen, dass der Python-Interpreter auch funktioniert:

```
# ./configure --prefix=/usr/local/Python25
# make && make install
# pwd
/usr/local/Python25
# python
Python 2.5.1 (r251:54863, Mar 14 2012, 07:39:18)
[GCC 3.4.6 20060404 (Red Hat 3.4.6-8)] on Linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Sie befinden sich nun in der interaktiven Python-Shell, die Ihnen den vollständigen Zugriff auf den Python-Interpreter und alle verfügbaren Bibliotheken erlaubt. Ein schneller Test wird zeigen, ob Befehle korrekt interpretiert werden:

```
>>> print "Hallo, Welt!"
Hallo, Welt!
>>> exit()
#
```

Ausgezeichnet! Alles funktioniert so, wie Sie es sich wünschen. Um sicherzustellen, dass die Benutzerumgebung automatisch weiß, wo sie den Python-Interpreter findet, müssen Sie die Datei */root/.bashrc* bearbeiten. Ich persönlich verwende `nano` für die gesamte Textbearbeitung, aber Sie können natürlich jeden Ihnen vertrauten Editor verwenden. Öffnen Sie die Datei */root/.bashrc* und fügen Sie am Ende der Datei die folgende Zeile hinzu:

```
export PATH=/usr/local/Python25/:$PATH
```

Diese Zeile teilt der Linux-Umgebung mit, dass der Benutzer root auf den Python-Interpreter zugreifen kann, ohne den vollständigen Pfad verwenden zu müssen. Nachdem Sie sich als root aus- und wieder eingeloggt haben, können Sie in der Kommandozeile jederzeit `python` eingeben und landen direkt im Python-Interpreter.

Nachdem Sie nun einen voll funktionsfähigen Python-Interpreter sowohl für Windows als auch für Linux besitzen, wird es Zeit, Ihre integrierte Entwicklungsumgebung

(*Integrated Development Environment, kurz IDE*) einzurichten. Wenn Sie bereits eine eigene IDE verwenden, können Sie den folgenden Abschnitt überspringen.

1.3 Einrichten von Eclipse und PyDev

Um Python-Anwendungen schnell entwickeln und debuggen zu können, ist der Einsatz einer guten IDE unverzichtbar. Die Verbindung der populären Eclipse-Entwicklungsumgebung mit einem Modul namens *PyDev* stellt Ihnen eine Vielzahl leistungsfähiger Features zur Verfügung, die die meisten anderen IDEs nicht zu bieten haben. Eclipse läuft unter Windows, Linux und Mac und der Support durch die Community ist ausgezeichnet. Sehen wir uns im Folgenden an, wie man Eclipse und PyDev installiert und konfiguriert:

1. Laden sie das Paket Eclipse Classic von *http://www.eclipse.org/downloads/* herunter.
2. Entpacken Sie es nach *C:\Eclipse*.
3. Führen Sie `C:\Eclipse\eclipse.exe` aus.
4. Beim ersten Start werden Sie gefragt, wo Ihr Workspace gespeichert werden soll. Akzeptieren Sie die Voreinstellung und aktivieren Sie die Checkbox **Use this as default and do not ask again**, um nicht wieder nach dem Workspace gefragt zu werden. Klicken Sie dann auf **OK**.
5. Sobald Eclipse gestartet wurde, wählen Sie **Help ▸ Software Updates ▸ Find and Install**.
6. Aktivieren Sie den Radiobutton namens **Search for new features to install** und klicken Sie auf **Next**.
7. Auf der nächsten Seite klicken Sie auf **New Remote Site**.
8. Geben Sie nun im Feld Name einen beschreibenden Text wie **PyDev Update** ein. Stellen Sie sicher, dass das URL-Feld den Link *http://pydev.sourceforge.net/updates/* enthält und klicken Sie auf **OK**. Schließen Sie den Vorgang mit einem Klick auf **Finish** ab, um den Eclipse-Updater zu starten.
9. Nach kurzer Zeit erscheint der Update-Dialog. Klappen Sie den Punkt **PyDev Update** auf und aktivieren Sie das **PyDev**-Element. Klicken Sie auf **Next**.
10. Lesen Sie sich nun die Lizenzvereinbarung für PyDev durch. Wenn Sie mit den Bedingungen einverstanden sind, klicken Sie den Radiobutton **I accept the terms in the license agreement** an.
11. Klicken Sie nun **Next** und dann **Finish** an. Eclipse beginnt nun damit, die PyDev-Erweiterung herunterzuladen. Nachdem das geschehen ist, klicken Sie auf **Install All**.
12. Der letzte Schritt besteht darin, den **Yes**-Button in der Dialogbox anzuklicken, die nach der Installation von PyDev erscheint. Nach dem Neustart von Eclipse steht Ihnen PyDev zur Verfügung.

Der nächste Schritt bei der Eclipse-Konfiguration besteht darin sicherzustellen, dass PyDev den richtigen Python-Interpreter findet, wenn Sie Skripten innerhalb von PyDev ausführen wollen:

1. In Eclipse wählen Sie **Window ▸ Preferences**.
2. Klappen Sie den **PyDev**-Baum auf und wählen Sie **Interpreter – Python**.
3. Im Abschnitt Python-Interpreter zu Beginn des Dialogs klicken Sie auf **New**.
4. Wählen Sie *C:\Python25\python.exe* aus und klicken Sie auf **Open**.
5. Der nächste Dialog zeigt eine Liste der verfügbaren Bibliotheken für den Interpreter. Behalten Sie alle Einstellungen bei und klicken Sie einfach auf **OK**.
6. Klicken Sie noch einmal auf **OK**, um das Interpreter-Setup abzuschließen.

Sie besitzen nun eine funktionierende PyDev-Installation, die für den Einsatz Ihres frisch installierten Python 2.5-Interpreters eingerichtet ist. Bevor Sie mit dem Programmieren beginnen können, müssen Sie ein neues PyDev-Projekt anlegen. Dieses Projekt wird alle Quelldateien enthalten, die wir im Verlauf dieses Buches verwenden. Um ein neues Projekt einzurichten, sind folgende Schritte notwendig:

1. Wählen Sie **File ▸ New ▸ Project**.
2. Klappen Sie den **PyDev**-Baum auf, wählen Sie **PyDev Project** und klicken Sie dann auf **Next**.
3. Nennen Sie das Projekt *Gray Hat Python* und klicken Sie auf **Finish**.

Sie werden bemerken, dass sich Ihr Eclipse-Layout selbst neu anordnet, und Sie sollten das Gray Hat Python-Projekt oben links auf dem Bildschirm sehen. Klicken Sie den Ordner **src** mit der rechten Maustaste an und wählen Sie **New ▸ PyDev Module**. Im Feld **Name** geben Sie **chapter1-test** ein und klicken auf **Finish**. Sie werden bemerken, dass die Projektleiste aktualisiert und die Datei *chapter1-test.py* in die Liste eingefügt wurde.

Um Python-Skripten unter Eclipse auszuführen, klicken Sie einfach den **Run As**-Button (den grünen Kreis mit dem weißen Pfeil) in der Werkzeugleiste an. Um das zuletzt von Ihnen ausgeführte Skript erneut auszuführen, drücken Sie CTRL-F11. Wenn Sie ein Skript innerhalb von Eclipse ausführen, erscheint die Ausgabe nicht in einem Kommandozeilenfenster, sondern in einem Bereich namens *Console* am unteren Rand des Eclipse-Bildschirms. Alle Ausgaben Ihres Skripts erscheinen in diesem Console-Bereich. Sie werden bemerken, dass der Editor die Datei *chapter1-test.py* geöffnet hat und auf etwas süßen Python-Nektar wartet.

1.3.1 Des Hackers bester Freund: ctypes

Das Python-Modul ctypes ist die mit Abstand mächtigste Bibliothek, die dem Python-Entwickler zur Verfügung steht. Die ctypes-Library ermöglicht es Ihnen, Funktionen in DLLs (Dynamic Link Libraries) aufzurufen, und besitzt weitreichende Möglichkeiten zum Aufbau komplexer C-Datentypen sowie Hilfsfunktionen zur Manipulation des

(Arbeits-)Speichers auf niedriger Ebene. Es ist wichtig, dass Sie die Grundlagen der Verwendung der ctypes-Library verstehen, da Sie diese im Verlauf des gesamten Buches häufig nutzen werden.

1.3.2 Dynamische Libraries nutzen

Um sich ctypes zunutze machen zu können, müssen Sie zunächst verstehen, wie man Funktionen in DLLs auflöst und auf sie zugreift. Eine *DLL* ist ein kompiliertes Binary, das zur Laufzeit des Hauptprozesses dynamisch eingebunden wird. Unter Windows heißen diese Binaries *Dynamic Link Libraries (DLL)*, unter Linux werden sie *Shared Objects (SO)* genannt. In beiden Fällen stellen diese Binaries Funktionen über exportierte Namen zur Verfügung, die im Speicher in reale Adressen aufgelöst werden. Normalerweise müssen Sie zur Laufzeit diese Funktionsadressen auflösen, um die Funktionen aufrufen zu können. Mit ctypes ist die Schmutzarbeit aber bereits erledigt.

Es gibt drei verschiedene Wege, dynamische Libraries mit ctypes zu laden: `cdll()`, `windll()` und `oledll()`. Die Unterschiede zwischen diesen Dreien liegen in der Art und Weise, wie die Funktionen innerhalb der Libraries aufgerufen werden, und in den resultierenden Rückgabewerten. Die Methode `cdll()` wird für das Laden von Libraries verwendet, die Funktionen nach der Standardaufrufkonvention *cdecl* exportieren. Die Methode `windll()` lädt Bibliotheken, die nach der Aufrufkonvention *stdcall* (einer nativen Konvention der Microsoft Win32-API) exportieren. Die Methode `oledll()` funktioniert genau wie die `windll()`-Methode, erwartet aber, dass die exportierten Funktionen einen Windows *HRESULT*-Fehlercode zurückgeben, der speziell für Fehlermeldungen verwendet wird, die von Funktionen des Microsoft *Component Object Model (COM)* zurückgegeben werden.

Als kurzes Beispiel wollen wir die `printf()`-Funktion der C-Runtime sowohl unter Windows als auch unter Linux auflösen und diese verwenden, um eine Testnachricht auszugeben. Bei Windows heißt die C-Runtime *msvcrt.dll* und ist unter *C:\WINDOWS\system32* zu finden. Bei Linux heißt sie *libc.so.6* und liegt standardmäßig in */lib/*. Legen Sie (entweder in Eclipse oder in Ihrem normalen Python-Arbeitsverzeichnis) ein Skript namens *chapter1-printf.py* an und geben Sie den folgenden Code ein.

chapter1-printf.py-Code für Windows

```
from ctypes import *

msvcrt = cdll.msvcrt
message_string = "Hallo, Welt!\n"
msvcrt.printf("Test:: %s", message_string)
```

Hier die Ausgabe dieses Skripts:

```
C:\Python25> python chapter1-printf.py
Test: Hallo, Welt!
C:\Python25>
```

Unter Linux sieht dieses Beispiel etwas anders aus, führt aber zum gleichen Ergebnis. Wechseln Sie zu Ihrer Linux-Installation und legen Sie die Datei *chapter1-printf.py* in Ihrem */root/*-Verzeichnis an.

chapter1-printf.py-Code für Linux

```
from ctypes import *

libc = CDLL("libc.so.6")
message_string = "Hallo, Welt!\n"
libc.printf("Test: %s", message_string)
```

Hier die Ausgabe der Linux-Version Ihres Skripts:

```
# python /root/chapter1-printf.py
Test: Hallo, Welt!
#
```

Sie sehen, wie einfach die Einbindung einer dynamischen Library und die Verwendung einer von ihr exportierten Funktion ist. Sie werden diese Technik in diesem Buch sehr häufig verwenden, weshalb es wichtig ist, die Funktionsweise zu verstehen.

Aufrufkonventionen

Eine *Aufrufkonvention* beschreibt, wie eine bestimmte Funktion korrekt aufzurufen ist. Das umfasst die Reihenfolge der Allozierung der Funktionsparameter, welche Parameter auf dem Stack abgelegt oder in Registern übergeben werden und wie der Stack geleert wird, wenn die Funktion zurückkehrt. Sie müssen zwei Aufrufkonventionen verstehen: *cdecl* und *stdcall*. Bei der cdecl-Konvention werden Parameter von rechts nach links auf den Stack geschoben und die Funktion ist dafür verantwortlich, dass die Argumente vom Stack entfernt werden. Sie wird von den meisten C-Systemen für die x86-Architektur verwendet.

Hier ein Beispiel für einen cdecl-Funktionsaufruf:

In C

```
int python_rocks(reason_one, reason_two, reason_three);
```

→

In x86-Assembler

```
push reason_three
push reason_two
push reason_one
call python_rocks
add esp, 12
```

Sie können deutlich erkennen, wie die Argumente übergeben werden und dass die letzte Zeile den Stackpointer um 12 Bytes inkrementiert (die Funktion besitzt 3 Parameter und jeder Stackparameter ist 4 Byte groß, daher 12 Byte), wodurch die Parameter vom Stack entfernt werden.

Ein Beispiel für die stdcall-Konvention, die von der Win32-API verwendet wird, sehen Sie hier:

In C

```
int my_socks(color_one, color_two, color_three);
```

In x86-Assembler

```
push color_three
push color_two
push color_one
call my_socks
```

Wie Sie sehen, ist die Reihenfolge der Parameter identisch, aber das Aufräumen des Stacks erfolgt nicht durch den Aufrufer. Vielmehr muss das die Funktion `my_socks` erledigen, bevor sie zurückkehrt.

Für beide Konventionen gilt als wichtiger Winweis, dass die Rückgabewerte im EAX-Register gespeichert werden.

1.3.3 C-Datentypen konstruieren

Auch die Möglichkeit, einen C-Datentypen mit Python zu erzeugen, ist auf eine schräge Art recht sexy. Dank dieses Features können Sie in C und C++ geschriebene Komponenten vollständig integrieren, was die Leistungsfähigkeit von Python deutlich erhöht. Die Tabelle 1–1 zeigt, wie die Datentypen zwischen C, Python und dem resultierenden ctypes-Typ abgebildet werden.

C-Typ	Python-Typ	ctypes-Typ
char	1-character string	c_char
wchar_t	1-character Unicode string	c_wchar
char	int/long	c_byte
char	int/long	c_ubyte
short	int/long	c_short
unsigned short	int/long	c_ushort
int	int/long	C_int
unsigned int	int/long	c_uint
long	int/long	c_long
unsigned long	int/long	c_ulong
long long	int/long	c_longlong
unsigned long long	int/long	c_ulonglong
float	float	c_float
double	float	c_double
char * (NULL terminated)	string or none	c_char_p
wchar_t * (NULL terminated)	unicode or none	c_wchar_p
void *	int/long or none	c_void_p

Tab. 1–1 *Abbildung von Python- in C-Datentypen*

Sehen Sie, wie schön die Datentypen untereinander konvertiert werden können? Halten Sie diese Tabelle griffbereit, für den Fall, dass Sie dieses Mapping vergessen. Die ctypes-Typen können mit einem Wert initialisiert werden, dieser muss aber vom richtigten Typ und der richtigen Größe sein. Um das zu demonstrieren, öffnen Sie die Python-Shell und geben Sie die folgenden Beispiele ein:

```
C:\Python25> python.exe
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from ctypes import *
>>> c_int()
c_long(0)
>>> c_char_p("Hallo, Welt!")
c_char_p('Hallo, Welt!')
>>> c_ushort(-5)
c_ushort(65531)
>>>
>>> seitz = c_char_p("loves the python")
>>> print seitz
c_char_p('loves the python')
>>> print seitz.value
loves the python
>>> exit()
```