

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: Koninklijke Wöhrmann B.V., Zutphen, Niederlande

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 978-3-89864-633-8

1. Auflage 2009
Translation copyright für die deutschsprachige Ausgabe © 2009 dpunkt.verlag GmbH
Ringstraße 19 B
69115 Heidelberg

Copyright der amerikanischen Originalausgabe © 2009 by Justin Seitz.
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

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten.

Die Zusammenstellung der Software wurde nach bestem Wissen und Gewissen vorgenommen.
Bitte berücksichtigen Sie die jeweiligen Copyright-Hinweise, die bei den Programmen enthalten sind.
Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung
des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung,
Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware- Bezeichnungen sowie
Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-,
marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor
noch Verlag können jedoch für Schäden haftbar gemacht werden, die im Zusammenhang mit der
Verwendung dieses Buches stehen.

5 4 3 2 1 0

Inhaltsverzeichnis

	Einführung	1
1	Ihre Entwicklungsumgebung einrichten	3
1.1	Anforderungen an das Betriebssystem	3
1.2	Python 2.5 herunterladen und installieren	4
1.2.1	Python unter Windows installieren	4
1.2.2	Python unter Linux installieren	4
1.3	Einrichten von Eclipse und PyDev	6
1.3.1	Des Hackers bester Freund: ctypes	7
1.3.2	Dynamische Libraries nutzen	8
1.3.3	C-Datentypen konstruieren	10
1.3.4	Parameter per Referenz übergeben	12
1.3.5	Strukturen und Unions definieren	12
2	Debugger und Debugger-Design	15
2.1	Universal-CPU-Register	16
2.2	Der Stack	18
2.3	Debug-Events	20
2.4	Breakpunkte	21
2.4.1	Software-Breakpunkte	21
2.4.2	Hardware-Breakpunkte	24
2.4.3	Speicher-Breakpunkte	26
3	Entwicklung eines Windows-Debuggers	29
3.1	Prozess, wo bist Du?	29
3.2	Den Zustand der CPU-Register abrufen	37
3.2.1	Threads aufspüren	38
3.2.2	Alles zusammenfügen	39
3.3	Debug-Event-Handler implementieren	43

3.4	Der machtvolle Breakpunkt	47
3.4.1	Software-Breakpunkte	47
3.4.2	Hardware-Breakpunkte	52
3.4.3	Speicher-Breakpunkte	56
3.5	Fazit	60
4	PyDbg – ein reiner Python-Debugger für Windows	61
4.1	Breakpunkt-Handler erweitern	61
4.2	Handler für Zugriffsverletzungen	64
4.3	Prozess-Schnappschüsse	67
4.3.1	Prozess-Schnappschüsse erstellen	67
4.3.2	Alles zusammenfügen	69
5	Immunity Debugger – Das Beste beider Welten	73
5.1	Den Immunity Debugger installieren	73
5.2	Immunity Debugger – kurze Einführung	74
5.2.1	PyCommands	75
5.2.2	PyHooks	75
5.3	Entwicklung von Exploits	77
5.3.1	Exploit-freundliche Instruktionen finden	77
5.3.2	»Böse« Zeichen filtern	79
5.3.3	DEP unter Windows umgehen	82
5.4	Anti-Debugging-Routinen in Malware umgehen	86
5.4.1	IsDebuggerPresent	87
5.4.2	Prozessiteration unterbinden	87
6	Hooking	89
6.1	Soft Hooking mit PyDbg	89
6.2	Hard Hooking mit dem Immunity Debugger	94
7	DLL- und Code-Injection	101
7.1	Erzeugung entfernter Threads	101
7.1.1	DLL-Injection	103
7.1.2	Code-Injection	105
7.2	Zum Übeltäter werden	108
7.2.1	Dateien verstecken	108
7.2.2	Eine Hintertür codieren	109
7.2.3	Kompilieren mit py2exe	113

8	Fuzzing	117
8.1	Fehlerklassen	118
8.1.1	Pufferüberläufe	118
8.1.2	Integerüberläufe	119
8.1.3	Formatstring-Angriffe	121
8.2	Datei-Fuzzer	122
8.3	Weitere Überlegungen	128
8.3.1	Codendeckungsgrad (Code Coverage)	128
8.3.2	Automatisierte statische Analyse	129
9	Sulley	131
9.1	Sulley installieren	132
9.2	Sulley-Primitive	132
9.2.1	Strings	133
9.2.2	Trennsymbole	133
9.2.3	Statische und zufällige Primitive	133
9.2.4	Binäre Daten	134
9.2.5	Integerwerte	134
9.2.6	Blöcke und Gruppen	135
9.3	WarFTPD knacken mit Sulley	136
9.3.1	FTP – kurze Einführung	137
9.3.2	Das FTP-Protokollgerüst erstellen	138
9.3.3	Sulley-Sessions	139
9.3.4	Netzwerk- und Prozessüberwachung	140
9.3.5	Fuzzing und das Sulley-Webinterface	141
10	Fuzzing von Windows-Treibern	145
10.1	Treiberkommunikation	146
10.2	Treiber-Fuzzing mit dem Immunity Debugger	147
10.3	Driverlib – das statische Analysetool für Treiber	150
10.3.1	Gerätenamen aufspüren	151
10.3.2	Die IOCTL-Dispatch-Routine aufspüren	152
10.3.3	Unterstützte IOCTL-Codes aufspüren	154
10.4	Einen Treiber-Fuzzer entwickeln	156

11	IDAPython – Scripting für IDA Pro	161
11.1	IDAPython installieren	162
11.2	IDAPython-Funktionen	163
11.2.1	Utility-Funktionen	163
11.2.2	Segmente	163
11.2.3	Funktionen	164
11.2.4	Cross-Referenzen	164
11.2.5	Debugger-Hooks	165
11.3	Beispielskripten	166
11.3.1	Aufspüren von Cross-Referenzen auf gefährliche Funktionen	166
11.3.2	Codeabdeckung von Funktionen	168
11.3.3	Stackgröße berechnen	169
12	PyEmu – der skriptfähige Emulator	173
12.1	PyEmu installieren	173
12.2	PyEmu-Übersicht	174
12.2.1	PyCPU	174
12.2.2	PyMemory	175
12.2.3	PyEmu	175
12.2.4	Ausführung	175
12.2.5	Speicher- und Register-Modifier	175
12.2.6	Handler	176
12.2.6.1	Register-Handler	177
12.2.6.2	Library-Handler	177
12.2.6.3	Ausnahme-Handler	178
12.2.6.4	Instruktions-Handler	178
12.2.6.5	Opcode-Handler	179
12.2.6.6	Speicher-Handler	179
12.2.6.7	High-Level-Speicher-Handler	180
12.2.6.8	Programmzähler-Handler	181
12.3	IDAPyEmu	181
12.3.1	Funktionen emulieren	183
12.3.2	PEPyEmu	186
12.3.3	Packer für Executables	187
12.3.4	UPX-Packer	187
12.3.5	UPX mit PEPyEmu entpacken	188
	Index	193