

# DESIGNING BSD ROOTKITS

AN INTRODUCTION TO KERNEL HACKING

JOSEPH KONG





---

## **DESIGNING BSD ROOTKITS**



---

# **DESIGNING BSD ROOTKITS**

**An Introduction to  
Kernel Hacking**

**by Joseph Kong**



**NO STARCH  
PRESS**

San Francisco

---

**DESIGNING BSD ROOTKITS.** Copyright © 2007 by Joseph Kong.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.



Printed on recycled paper in the United States of America

11 10 09 08 07 1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-142-5

ISBN-13: 978-1-59327-142-8

Publisher: William Pollock

Production Editor: Elizabeth Campbell

Cover and Interior Design: Octopod Studios

Developmental Editor: William Pollock

Technical Reviewer: John Baldwin

Copyeditor: Megan Dunchak

Compositors: Riley Hoffman and Megan Dunchak

Proofreader: Riley Hoffman

Indexer: Nancy Guenther

For information on book distributors or translations, please contact No Starch Press, Inc. directly:

No Starch Press, Inc.

555 De Haro Street, Suite 250, San Francisco, CA 94107

phone: 415.863.9900; fax: 415.863.9950; info@nostarch.com; www.nostarch.com

*Library of Congress Cataloging-in-Publication Data*

Kong, Joseph.

Designing BSD rootkits : an introduction to kernel hacking / Joseph Kong.

p. cm.

Includes index.

ISBN-13: 978-1-59327-142-8

ISBN-10: 1-59327-142-5

1. FreeBSD. 2. Free computer software. 3. Operating systems (Computers) I. Title.

QA76.76.063K649 2007

005.3--dc22

2007007644

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

---

To those who follow their dreams and specialize in the impossible.

---

## ACKNOWLEDGMENTS

Foremost, I am especially grateful to Bill Pollock for his belief in me and for his help in this book, as well as giving me so much creative control. His numerous reviews and suggestions show in the final result (and yes, the rumors are true, he does edit like a drill sergeant). I would also like to thank Elizabeth Campbell for, essentially, shepherding this entire book (and for remaining cheerful at all times, even when I rewrote an entire chapter, after it had been through copyedit). Thanks to Megan Dunchak for performing the copyedit and for improving the “style” of this book, and to Riley Hoffman for reviewing the entire manuscript for errors. Also, thanks to Patricia Witkin, Leigh Poehler, and Ellen Har for all of their work in marketing.

I would also like to thank John Baldwin, who served as this book’s technical reviewer, but went beyond the normal call of duty to provide a wealth of suggestions and insights; most of which became new sections in this book.

Also, I would like to thank my brother for proofreading the early drafts of this book, my dad for getting me into computers (he’s still the best hacker I know), and my mom for, pretty much, everything (especially her patience, because I was definitely a brat growing up).

Last but not least, I would like to thank the open-source software/hacker community for their innovation, creativity, and willingness to share.



---

## BRIEF CONTENTS

Foreword by John Baldwin .....	xiii
Introduction .....	xv
Chapter 1: Loadable Kernel Modules .....	1
Chapter 2: Hooking .....	23
Chapter 3: Direct Kernel Object Manipulation .....	37
Chapter 4: Kernel Object Hooking .....	59
Chapter 5: Run-Time Kernel Memory Patching .....	63
Chapter 6: Putting It All Together.....	91
Chapter 7: Detection .....	119
Closing Words .....	127
Bibliography.....	129
Index.....	131



---

# CONTENTS IN DETAIL

## FOREWORD by John Baldwin

xiii

## INTRODUCTION

xv

What Is a Rootkit? .....	xvi
Why FreeBSD? .....	xvi
The Goals of This Book .....	xvi
Who Should Read This Book? .....	xvi
Contents Overview .....	xvi
Conventions Used in This Book .....	xvii
Concluding Remarks .....	xvii

## 1

### LOADABLE KERNEL MODULES

1

1.1 Module Event Handler .....	2
1.2 The DECLARE_MODULE Macro .....	3
1.3 "Hello, world!" .....	4
1.4 System Call Modules .....	6
1.4.1 The System Call Function .....	6
1.4.2 The sysent Structure .....	7
1.4.3 The Offset Value .....	8
1.4.4 The SYSCALL_MODULE Macro .....	8
1.4.5 Example .....	9
1.4.6 The modfind Function .....	10
1.4.7 The modstat Function .....	10
1.4.8 The syscall Function .....	11
1.4.9 Executing the System Call .....	11
1.4.10 Executing the System Call Without C Code .....	12
1.5 Kernel/User Space Transitions .....	12
1.5.1 The copyin and copyinstr Functions .....	13
1.5.2 The copyout Function .....	13
1.5.3 The copystr Function .....	13
1.6 Character Device Modules .....	14
1.6.1 The cdevsw Structure .....	14
1.6.2 Character Device Functions .....	15
1.6.3 The Device Registration Routine .....	16
1.6.4 Example .....	17
1.6.5 Testing the Character Device .....	19
1.7 Linker Files and Modules .....	21
1.8 Concluding Remarks .....	22

---

## **2** **HOOKING** **23**

2.1	Hooking a System Call .....	24
2.2	Keystroke Logging .....	26
2.3	Kernel Process Tracing .....	28
2.4	Common System Call Hooks .....	29
2.5	Communication Protocols .....	30
2.5.1	The protosw Structure .....	30
2.5.2	The inetsw[] Switch Table .....	31
2.5.3	The mbuf Structure .....	32
2.6	Hooking a Communication Protocol .....	32
2.7	Concluding Remarks .....	35

## **3** **DIRECT KERNEL OBJECT MANIPULATION** **37**

3.1	Kernel Queue Data Structures .....	37
3.1.1	The LIST_HEAD Macro .....	38
3.1.2	The LIST_HEAD_INITIALIZER Macro .....	38
3.1.3	The LIST_ENTRY Macro .....	38
3.1.4	The LIST_FOREACH Macro .....	39
3.1.5	The LIST_REMOVE Macro .....	39
3.2	Synchronization Issues .....	39
3.2.1	The mtx_lock Function .....	40
3.2.2	The mtx_unlock Function .....	40
3.2.3	The sx_slock and sx_xlock Functions .....	40
3.2.4	The sx_sunlock and sx_xunlock Functions .....	41
3.3	Hiding a Running Process .....	41
3.3.1	The proc Structure .....	41
3.3.2	The allproc List .....	42
3.3.3	Example .....	43
3.4	Hiding a Running Process Redux .....	46
3.4.1	The hashinit Function .....	47
3.4.2	pidhashtbl .....	47
3.4.3	The pfind Function .....	48
3.4.4	Example .....	48
3.5	Hiding with DKOM .....	51
3.6	Hiding an Open TCP-based Port .....	52
3.6.1	The inpcb Structure .....	52
3.6.2	The tcbinfo.listhead List .....	53
3.6.3	Example .....	54
3.7	Corrupting Kernel Data .....	56
3.8	Concluding Remarks .....	57

## **4** **KERNEL OBJECT HOOKING** **59**

4.1	Hooking a Character Device .....	59
4.1.1	The cdevp_list and cdev_priv Structures .....	60
4.1.2	The devmtx Mutex .....	60
4.1.3	Example .....	60
4.2	Concluding Remarks .....	62

---

## **5**

### **RUN-TIME KERNEL MEMORY PATCHING** **63**

5.1	Kernel Data Access Library .....	63
5.1.1	The <code>kvm_openfiles</code> Function .....	64
5.1.2	The <code>kvm_nlist</code> Function .....	64
5.1.3	The <code>kvm_geterr</code> Function .....	65
5.1.4	The <code>kvm_read</code> Function .....	65
5.1.5	The <code>kvm_write</code> Function .....	65
5.1.6	The <code>kvm_close</code> Function .....	66
5.2	Patching Code Bytes .....	66
5.3	Understanding <code>x86</code> Call Statements .....	70
5.3.1	Patching Call Statements .....	70
5.4	Allocating Kernel Memory .....	73
5.4.1	The <code>malloc</code> Function .....	73
5.4.2	The <code>MALLOC</code> Macro .....	74
5.4.3	The <code>free</code> Function .....	74
5.4.4	The <code>FREE</code> Macro .....	74
5.4.5	Example .....	75
5.5	Allocating Kernel Memory from User Space .....	77
5.5.1	Example .....	77
5.6	Inline Function Hooking .....	81
5.6.1	Example .....	82
5.6.2	Gotchas .....	88
5.7	Cloaking System Call Hooks .....	88
5.8	Concluding Remarks .....	90

## **6**

### **PUTTING IT ALL TOGETHER** **91**

6.1	What <code>HIDSes</code> Do .....	91
6.2	Bypassing <code>HIDSes</code> .....	92
6.3	Execution Redirection .....	92
6.4	File Hiding .....	96
6.5	Hiding a <code>KLD</code> .....	101
6.5.1	The <code>linker_files</code> List .....	102
6.5.2	The <code>linker_file</code> Structure .....	102
6.5.3	The <code>modules</code> List .....	103
6.5.4	The <code>module</code> Structure .....	103
6.5.5	Example .....	104
6.6	Preventing Access, Modification, and Change Time Updates .....	107
6.6.1	Change Time .....	108
6.6.2	Example .....	112
6.7	Proof of Concept: Faking Out Tripwire .....	114
6.8	Concluding Remarks .....	117

## **7**

### **DETECTION** **119**

7.1	Detecting Call Hooks .....	120
7.1.1	Finding System Call Hooks .....	120

---

7.2	Detecting DKOM .....	123
7.2.1	Finding Hidden Processes .....	123
7.2.2	Finding Hidden Ports .....	125
7.3	Detecting Run-Time Kernel Memory Patching .....	125
7.3.1	Finding Inline Function Hooks .....	125
7.3.2	Finding Code Byte Patches .....	125
7.4	Concluding Remarks .....	126

<b>CLOSING WORDS</b>	<b>127</b>
----------------------	------------

<b>BIBLIOGRAPHY</b>	<b>129</b>
---------------------	------------

<b>INDEX</b>	<b>131</b>
--------------	------------

---

## FOREWORD

I have been working on various parts of the FreeBSD kernel for the past six years. During that time, my focus has always been on making FreeBSD more robust. This often means maintaining the existing stability of the system while adding new features or improving stability by fixing bugs and/or design flaws in the existing code. Prior to working on FreeBSD, I served as a system administrator for a few networks; my focus was on providing the desired services to users while protecting the network from any malicious actions. Thus, I have always been on the defensive “side” of the game when it comes to security.

Joseph Kong provides an intriguing look at the offensive side in *Designing BSD Rootkits*. He enumerates several of the tools used for constructing rootkits, explaining the concepts behind each tool and including working examples for many of the tools, as well. In addition, he examines some of the ways to detect rootkits.

Subverting a running system requires many of the same skills and techniques as building one. For example, both tasks require a focus on stability. A rootkit that reduces the stability of the system risks attracting the attention of a system administrator if the system crashes. Similarly, a system builder must

---

build a system that minimizes downtime and data loss that can result from system crashes. Rootkits must also confront some rather tricky problems, and the resulting solutions can be instructive (and sometimes entertaining) to system builders.

Finally, *Designing BSD Rootkits* can also be an eye-opening experience for system builders. One can always learn a lot from another's perspective. I cannot count the times I have seen a bug solved by a fresh pair of eyes because the developer who had been battling the bug was too familiar with the code. Similarly, system designers and builders are not always aware of the ways rootkits may be used to alter the behavior of their systems. Simply learning about some of the methods used by rootkits can change how they design and build their systems.

I have certainly found this book to be both engaging and informative, and I trust that you, the reader, will as well.

**John Baldwin**  
**Kernel Developer, FreeBSD**  
**Atlanta**



---

## INTRODUCTION



Welcome to *Designing BSD Rootkits!* This book will introduce you to the fundamentals of programming and developing kernel-mode rootkits under the FreeBSD operating system.

Through the “learn by example” method, I’ll detail the different techniques that a rootkit can employ so that you can learn what makes up rootkit code at its simplest level. It should be noted that this book does not contain or diagnose any “full-fledged” rootkit code. In fact, most of this book concentrates on *how* to employ a technique, rather than *what* to do with it.

Note that this book has nothing to do with exploit writing or how to gain root access to a system; rather, it is about maintaining root access long after a successful break-in.

---

## What Is a Rootkit?

While there are a few varied definitions of what constitutes a rootkit, for the purpose of this book, a *rootkit* is a set of code that allows someone to control certain aspects of the host operating system without revealing his or her presence. Fundamentally, that's what makes a rootkit—evasion of end user knowledge.

Put more simply, a rootkit is a “kit” that allows a user to maintain “root” access.

## Why FreeBSD?

FreeBSD is an advanced, open source operating system; with FreeBSD, you have full, uninhibited access to the kernel source, making it easier to learn systems programming—which is, essentially, what you'll be doing throughout this book.

## The Goals of This Book

The primary goal of this book is to expose you to rootkits and rootkit writing. By the time you finish this book, you should “theoretically” be able to rewrite the entire operating system, on the fly. You should also understand the theory and practicality behind rootkit detection and removal.

The secondary goal of this book is to provide you with a practical, hands-on look at parts of the FreeBSD kernel, with the extended goal of inspiring you to explore and hack the rest of it on your own. After all, getting your hands dirty is always the best way to learn.

## Who Should Read This Book?

This book is aimed at programmers with an interest in introductory kernel hacking. As such, experience writing kernel code is not required or expected.

To get the most out of this book, you should have a good grasp of the C programming language (i.e., you understand pointers) as well as x86 Assembly (AT&T Syntax). You'll also need to have a decent understanding of operating system theory (i.e., you know the difference between a process and a thread).

## Contents Overview

This book is (unofficially) divided into three sections. The first section (Chapter 1) is essentially a whirlwind tour of kernel hacking, designed to bring a novice up to speed. The next section (Chapters 2 through 6) covers the gamut of current, popular rootkit techniques (i.e., what you would find in “the wild”); while the last section (Chapter 7) focuses on rootkit detection and removal.

---

## Conventions Used in This Book

Throughout this book, I have used a boldface font in code listings to indicate commands or other text that I have typed in, unless otherwise specifically noted.

## Concluding Remarks

Although this book concentrates on the FreeBSD operating system, most (if not all) of the concepts can be applied to other OSes, such as Linux or Windows. In fact, I learned half of the techniques in this book on those very systems.

**NOTE** *All of the code examples in this book were tested on an IA-32-based computer running FreeBSD 6.0-STABLE.*



---

# 1

## LOADABLE KERNEL MODULES



The simplest way to introduce code into a running kernel is through a *loadable kernel module (LKM)*, which is a kernel subsystem that can be loaded and unloaded after bootup, allowing a system administrator to dynamically add and remove functionality from a live system. This makes LKMs an ideal platform for kernel-mode rootkits. In fact, the vast majority of modern rootkits are simply LKMs.

**NOTE** *In FreeBSD 3.0, substantial changes were made to the kernel module subsystem, and the LKM Facility was renamed the Dynamic Kernel Linker (KLD) Facility. Subsequently, the term KLD is commonly used to describe LKMs under FreeBSD.*

In this chapter we'll discuss LKM (that is, KLD) programming within FreeBSD for programmers new to kernel hacking.

---

**NOTE** Throughout this book, the terms device driver, KLD, LKM, loadable module, and module are all used interchangeably.

## 1.1 Module Event Handler

Whenever a KLD is loaded into or unloaded from the kernel, a function known as the *module event handler* is called. This function handles the initialization and shutdown routines for the KLD. Every KLD must include an event handler.<sup>1</sup> The prototype for the event handler function is defined in the `<sys/module.h>` header as follows:

---

```
typedef int (*modeventhand_t)(module_t, int /* modeventtype_t */, void *);
```

---

where `module_t` is a pointer to a module structure and `modeventtype_t` is defined in the `<sys/module.h>` header as follows:

---

```
typedef enum modeventtype {
    MOD_LOAD,      /* Set when module is loaded. */
    MOD_UNLOAD,    /* Set when module is unloaded. */
    MOD_SHUTDOWN, /* Set on shutdown. */
    MOD_QUIESCE   /* Set on quiesce. */
} modeventtype_t;
```

---

Here is an example of an event handler function:

---

```
static int
load(struct module *module, int cmd, void *arg)
{
    int error = 0;

    switch (cmd) {
    case MOD_LOAD:
        uprintf("Hello, world!\n");
        break;

    case MOD_UNLOAD:
        uprintf("Good-bye, cruel world!\n");
        break;

    default:
        error = EOPNOTSUPP;
        break;
    }

    return(error);
}
```

---

<sup>1</sup> Actually, this isn't entirely true. You can have a KLD that just includes a `sysctl`. You can also dispense with module handlers if you wish and just use `SYSINIT` and `SYSUNINIT` directly to register functions to be invoked on load and unload, respectively. You can't, however, indicate failure in those.

---

This function will print “Hello, world!” when the module loads, “Good-bye, cruel world!” when it unloads, and will return with an error (EOPNOTSUPP)<sup>2</sup> on shutdown and quiesce.

## 1.2 The DECLARE\_MODULE Macro

When a KLD is loaded (by the `kldload(8)` command, described in Section 1.3), it must link and register itself with the kernel. This can be easily accomplished by calling the `DECLARE_MODULE` macro, which is defined in the `<sys/module.h>` header as follows:

---

```
#define DECLARE_MODULE(name, data, sub, order)          \
    MODULE_METADATA(_md_##name, MDT_MODULE, &data, #name); \
    SYSINIT(name##module, sub, order, module_register_init, &data) \
    struct __hack
```

---

Here is a brief description of each parameter:

### name

This specifies the generic module name, which is passed as a character string.

### data

This parameter specifies the official module name and event handler function, which is passed as a `moduledata` structure. `struct moduledata` is defined in the `<sys/module.h>` header as follows:

---

```
typedef struct moduledata {
    const char    *name;           /* module name */
    modeventhand_t evhand;       /* event handler */
    void         *priv;          /* extra data */
} moduledata_t;
```

---

### sub

This specifies the system startup interface, which identifies the module type. Valid entries for this parameter can be found in the `<sys/kernel.h>` header within the `sysinit_sub_id` enumeration list.

For our purposes, we’ll always set this parameter to `SI_SUB_DRIVERS`, which is used when registering a device driver.

### order

This specifies the KLD’s order of initialization within the subsystem. You’ll find valid entries for this parameter in the `<sys/kernel.h>` header within the `sysinit_elem_order` enumeration list.

For our purposes, we’ll always set this parameter to `SI_ORDER_MIDDLE`, which will initialize the KLD somewhere in the middle.

---

<sup>2</sup> EOPNOTSUPP stands for *Error: Operation not supported*.

---

## 1.3 “Hello, world!”

You now know enough to write your first KLD. Listing 1-1 is a complete “Hello, world!” module.

---

```
#include <sys/param.h>
#include <sys/module.h>
#include <sys/kernel.h>
#include <sys/system.h>

/* The function called at load/unload. */
static int
load(struct module *module, int cmd, void *arg)
{
    int error = 0;

    switch (cmd) {
    case MOD_LOAD:
        uprintf("Hello, world!\n");
        break;

    case MOD_UNLOAD:
        uprintf("Good-bye, cruel world!\n");
        break;

    default:
        error = EOPNOTSUPP;
        break;
    }

    return(error);
}

/* The second argument of DECLARE_MODULE. */
static moduledata_t hello_mod = {
    "hello",      /* module name */
    load,        /* event handler */
    NULL         /* extra data */
};

DECLARE_MODULE(hello, hello_mod, SI_SUB_DRIVERS, SI_ORDER_MIDDLE);
```

---

*Listing 1-1: hello.c*

As you can see, this module is simply a combination of the sample event handler function from Section 1.1 and a filled-out `DECLARE_MODULE` macro.

To compile this module, you can use the system Makefile<sup>3</sup> `bsd.kmod.mk`. Listing 1-2 shows the complete Makefile for `hello.c`.

---

<sup>3</sup> A *Makefile* is used to simplify the process of converting a file or files from one form to another by describing the dependencies and build scripts for a given output. For more on Makefiles, see the `make(1)` manual page.



- [read The Vampire Book: The Encyclopedia of the Undead pdf](#)
- [download Building Homebrew Equipment \(Storey's Country Wisdom Bulletin A-186\)](#)
- [read Lyrical and Critical Essays](#)
- [Biomechanical Basis of Human Movement here](#)
- [download Discontinuity in Learning: Dewey, Herbart and Education as Transformation](#)
- [read online Understanding Symbolic Logic \(5th Edition\)](#)
  
- <http://conexdx.com/library/Taking-Sudoku-Seriously--The-Math-Behind-the-World-s-Most-Popular-Pencil-Puzzle.pdf>
- <http://www.celebritychat.in/?ebooks/Orion-and-King-Arthur.pdf>
- <http://monkeybubblemedia.com/lib/Lyrical-and-Critical-Essays.pdf>
- <http://anvilpr.com/library/The-New-Yorker--11-April-2016-.pdf>
- <http://www.1973vision.com/?library/I--Zombie.pdf>
- <http://www.netc-bd.com/ebooks/The-Black-Stallion-and-the-Girl--Black-Stallion--Book-19-.pdf>