Reversing: Secrets of Reverse Engineering

Eldad Eilam

Wiley Publishing, Inc.

Reversing: Secrets of Reverse Engineering
Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published simultaneously in Canada

Library of Congress Control Number: 2005921595

ISBN-10: 0-7645-7481-7
ISBN-13: 978-0-7645-7481-8

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/QR/QU/QV/IN

Credits

Executive Editor
Robert Elliott

Development Editor
Eileen Bien Calabro

Copy Editor
Foxxe Editorial Services

Editorial Manager
Mary Beth Wakefield

Vice President & Executive Group Publisher
Richard Swadley

Vice President and Publisher
Joseph B. Wikert

Project Editor
Pamela Hanley

Project Coordinator
Ryan Steffen

Graphics and Production Specialists
Denny Hager
Jennifer Heleine
Lynsey Osborn
Mary Gillot Virgin

Quality Control Technician
Leeann Harney

Proofreading and Indexing
TECHBOOKS Production Services

Cover Designer
Michael Trent

This page intentionally left blank

Foreword

It is amazing, and rather disconcerting, to realize how much software we run without knowing for sure what it does. We buy software off the shelf in shrink-wrapped packages. We run setup utilities that install numerous files, change system settings, delete or disable older versions and superceded utilities, and modify critical registry files. Every time we access a Web site, we may invoke or interact with dozens of programs and code segments that are necessary to give us the intended look, feel, and behavior. We purchase CDs with hundreds of games and utilities or download them as shareware. We exchange useful programs with colleagues and friends when we have tried only a fraction of each program's features.

Then, we download updates and install patches, trusting that the vendors are sure that the changes are correct and complete. We blindly hope that the latest change to each program keeps it compatible with all of the rest of the programs on our system. We rely on much software that we do not understand and do not know very well at all.

I refer to a lot more than our desktop or laptop personal computers. The concept of ubiquitous computing, or "software everywhere," is rapidly putting software control and interconnection in devices throughout our environment. The average automobile now has more lines of software code in its engine controls than were required to land the Apollo astronauts on the Moon.

Today's software has become so complex and interconnected that the developer often does not know all the features and repercussions of what has been created in an application. It is frequently too expensive and time-consuming to test all control paths of a program and all groupings of user options. Now, with multiple architecture layers and an explosion of networked platforms that the software will run on or interact with, it has become literally impossible for all

combinations to be examined and tested. Like the problems of detecting drug interactions in advance, many software systems are fielded with issues unknown and unpredictable.

Reverse engineering is a critical set of techniques and tools for understanding what software is really all about. Formally, it is "the process of analyzing a subject system to identify the system's components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction"(IEEE 1990). This allows us to visualize the software's structure, its ways of operation, and the features that drive its behavior. The techniques of analysis, and the application of automated tools for software examination, give us a reasonable way to comprehend the complexity of the software and to uncover its truth.

Reverse engineering has been with us a long time. The conceptual Reversing process occurs every time someone looks at someone else's code. But, it also occurs when a developer looks at his or her own code several days after it was written. Reverse engineering is a discovery process. When we take a fresh look at code, whether developed by ourselves or others, we examine and we learn and we see things we may not expect.

While it had been the topic of some sessions at conferences and computer user groups, reverse engineering of software came of age in 1990. Recognition in the engineering community came through the publication of a taxonomy on reverse engineering and design recovery concepts in *IEEE Software* magazine. Since then, there has been a broad and growing body of research on Reversing techniques, software visualization, program understanding, data reverse engineering, software analysis, and related tools and approaches. Research forums, such as the annual international Working Conference on Reverse Engineering (WCRE), explore, amplify, and expand the value of available techniques. There is now increasing interest in binary Reversing, the principal focus of this book, to support platform migration, interoperability, malware detection, and problem determination.

As a management and information technology consultant, I have often been asked: "How can you possibly condone reverse engineering?" This is soon followed by: "You've developed and sold software. Don't you want others to respect and protect your copyrights and intellectual property?" This discussion usually starts from the negative connotation of the term reverse engineering, particularly in software license agreements. However, reverse engineering technologies are of value in many ways to producers and consumers of software along the supply chain.

A stethoscope could be used by a burglar to listen to the lock mechanism of a safe as the tumblers fall in place. But the same stethoscope could be used by your family doctor to detect breathing or heart problems. Or, it could be used by a computer technician to listen closely to the operating sounds of a sealed disk drive to diagnose a problem without exposing the drive to

potentially-damaging dust and pollen. The tool is not inherently good or bad. The issue is the use to which the tool is put.

In the early 1980s, IBM decided that it would no longer release to its customers the source code for its mainframe computer operating systems. Mainframe customers had always relied on the source code for reference in problem solving and to tailor, modify, and extend the IBM operating system products. I still have my button from the IBM user group Share that reads: "If SOURCE is outlawed, only outlaws will have SOURCE," a word play on a famous argument by opponents of gun-control laws. Applied to current software, this points out that hackers and developers of malicious code know many techniques for deciphering others' software. It is useful for the good guys to know these techniques, too.

Reverse engineering is particularly useful in modern software analysis for a wide variety of purposes:

- Finding malicious code. Many virus and malware detection techniques use reverse engineering to understand how abhorrent code is structured and functions. Through Reversing, recognizable patterns emerge that can be used as signatures to drive economical detectors and code scanners.
- Discovering unexpected flaws and faults. Even the most well-designed system can have holes that result from the nature of our "forward engineering" development techniques. Reverse engineering can help identify flaws and faults before they become mission-critical software failures.
- Finding the use of others' code. In supporting the cognizant use of intellectual property, it is important to understand where protected code or techniques are used in applications. Reverse engineering techniques can be used to detect the presence or absence of software elements of concern.
- Finding the use of shareware and open source code where it was not intended to be used. In the opposite of the infringing code concern, if a product is intended for security or proprietary use, the presence of publicly available code can be of concern. Reverse engineering enables the detection of code replication issues.
- Learning from others' products of a different domain or purpose. Reverse engineering techniques can enable the study of advanced software approaches and allow new students to explore the products of masters. This can be a very useful way to learn and to build on a growing body of code knowledge. Many Web sites have been built by seeing what other Web sites have done. Many Web developers learned HTML and Web programming techniques by viewing the source of other sites.

- Discovering features or opportunities that the original developers did not realize. Code complexity can foster new innovation. Existing techniques can be reused in new contexts. Reverse engineering can lead to new discoveries about software and new opportunities for innovation.

In the application of computer-aided software engineering (CASE) approaches and automated code generation, in both new system development and software maintenance, I have long contended that any system we build should be immediately run through a suite of reverse engineering tools. The holes and issues that are uncovered would save users, customers, and support staff many hours of effort in problem detection and solution. The savings industry-wide from better code understanding could be enormous.

I've been involved in research and applications of software reverse engineering for 30 years, on mainframes, mid-range systems and PCs, from program language statements, binary modules, data files, and job control streams. In that time, I have heard many approaches explained and seen many techniques tried. Even with that background, I have learned much from this book and its perspective on reversing techniques. I am sure that you will too.

Elliot Chikofsky
Engineering Management and Integration (Herndon, VA)
Chair, Reengineering Forum
Executive Secretary, IEEE Technical Council on Software Engineering

Acknowledgments

First I would like to thank my beloved Odelya ("Oosa") Buganim for her constant support and encouragement—I couldn't have done it without you!

I would like to thank my family for their patience and support: my grandparents, Yosef and Pnina Vertzberger, my parents, Avraham and Nava Eilam-Amzallag, and my brother, Yaron Eilam.

I'd like to thank my editors at Wiley: My executive editor, Bob Elliott, for giving me the opportunity to write this book and to work with him, and my development editor, Eileen Bien Calabro, for being patient and forgiving with a first-time author whose understanding of the word deadline comes from years of working in the software business.

Many talented people have invested a lot of time and energy in reviewing this book and helping me make sure that it is accurate and enjoyable to read. I'd like to give special thanks to David Sleeper for spending all of those long hours reviewing the entire manuscript, and to Alex Ben-Ari for all of his useful input and valuable insights. Thanks to George E. Kalb for his review of Part III, to Mike Van Emmerik for his review of the decompilation chapter, and to Dr. Roger Kingsley for his detailed review and input. Finally, I'd like to acknowledge Peter S. Canelias who reviewed the legal aspects of this book.

This book would probably never exist if it wasn't for Avner ("Sabi") Zangvil, who originally suggested the idea of writing a book about reverse engineering and encouraged me to actually write it.

I'd like to acknowledge my good friends, Adar Cohen and Ori Weitz for their friendship and support.

Last, but not least, this book would not have been the same without Bookey, our charming cat who rested and purred on my lap for many hours while I was writing this book.

This page intentionally left blank

Contents