FOREWORD

Software has one goal: simplify. If there's a workflow that can be optimized or automated, data that can be stored or processed more efficiently, software steps in to fill the job. While simplifying, software must not introduce undo complexity, and therefore should install with minimal user interaction, seamlessly integrate services and data from other applications and multiple sources, and be resilient to changes in its software and hardware environment. For the most part, software magically just works.

However, while software strives to simplify the experiences of end users and administrators, it has become more and more complex. Whether it's the amount of the data they work with, the number of applications with which they communicate, their degree of internal parallelism, or the APIs they import directly and indirectly from the software stack upon which they run, most of software's apparent simplicity hides a world of subtle timings, dependencies, and assumptions that run between layers of software, often across different applications and even computers. Just determining which component is at fault—much less why, for a problem that surfaces as a crash in a library, a meaningless error message, or a hang—is often daunting.

The reason you're reading this book is that you develop, test, or support software, and therefore face breakdowns in software's myriad moving parts that you are charged with investigating through to a root cause and maybe fixing. Success in this endeavor means identifying the source of a problem as quickly and efficiently as possible, which requires knowing what to look with, where to look, and how to look. In other words, succeeding means knowing what tools are at your disposal, which ones are the most effective for a class of failures, and how to apply the tool's features and functionality to quickly narrow in on the source of a problem.

Learning how to troubleshoot and debug Windows applications on the job has, for the most part, been the only option, but when you debug an application failure, knowing about that one obscure tool or scenario-specific debugger command can mean the difference between instantly understanding a problem and spending hours or even days hunting it without success. That's why a book like this pays for itself many times over.

Advanced Windows Debugging takes the combined knowledge and years of hands-on experience of not just Mario and Daniel, but also the Microsoft Customer

Support Services and the Windows product and tools development teams and puts it at your fingertips. There's no more authoritative place to learn about how the Windows heap manager influences the behavior of buffer overflows or what debugger extension command you should use to troubleshoot DCOM hangs, for example. I've been debugging my own Windows applications and device drivers for over 10 years, but when I reviewed the manuscript, I learned about new techniques, tools, and debugger commands that I'd never come across and that I've already found use for.

We all earn our pay and reputations not by how we debug, but by how quickly and accurately we do it. Whether you've been debugging Windows applications for years or are just getting started, Mario and Daniel equip you well for your bug hunting expeditions. Happy hunting!

Mark Russinovich
Technical Fellow, Platform and Services Division
Microsoft Corporation