Foreword

Who can argue with testing things before you allow yourself to depend on them? No one *can* argue. No one *will* argue. Therefore, if testing is not done, the reasons have to be something other than a reasoned objection to testing. There seem to be exactly three: I can't afford it, I can get along without it, and I don't know how.

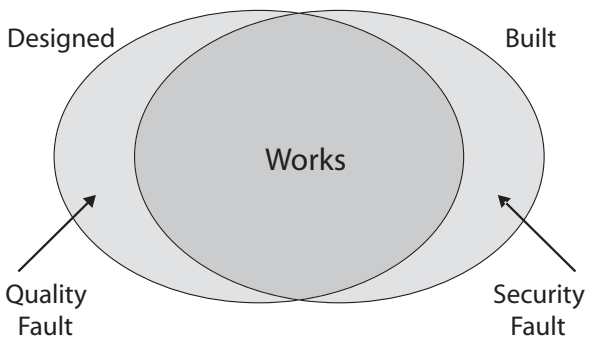
- **Not being able to afford it**—Allowing for economists to disagree over fine points, the cost of anything is the foregone alternative. If you do testing, what didn't you do? If it is to add yet another feature, perhaps you deserve congratulations on choosing a simpler product. Simpler products are in fact easier to test (and for good reason: the chief enemy of security is complexity, and nothing breeds complexity like creeping featuritis). If you didn't do testing, the usual reason given is to "get the product out on time." That reason is insufficient if not petulant. The sort of testing taught in this book is about the future even more than getting the product out on time is about the future. Only CEOs intoxicated on visions of wealth are immune to thinking about the future in ways that preclude testing. Testing is about controlling your future rather than allowing it to control you. Testing accelerates the inevitable future failure of products into the present. When William Gibson famously said, "The future is already here—it's just unevenly distributed," he wasn't thinking of testing as we mean it here. What you explicitly want is to unevenly distribute the future so that your product gets to see its future before your customers (and opponents) do. Since you are reading this paragraph, it's pretty likely you are of a testing frame of mind, so we'll drop the argument and move on.

- **Getting along without it**—Some products probably don't need much testing. They are not subject to innovation; they're nonperishable commodities, or something equally boring. That's not why we are here. We are here to protect security-sensitive products. Which products are those? A product is security-sensitive if, in its operation, it faces sentient opponents. If the only perils it faces are cluelessness ("Hey, watch this!") or random happenstance (alpha particles), the product may well not be security-sensitive. But with software and networks being as they are, nearly everything is security-sensitive because, if nothing else, every sociopath is your next-door neighbor. The burden of perfection is no longer on the criminal to commit the perfect crime but rather is on the defender to commit the perfect defense. Sure, you can get away with not testing, just as you can get away with never wearing protective gear while you band-saw aluminum, mountain-bike in Moab, or scrub down a P3 containment lab. There's always someone who has gotten away with that and more. That doesn't apply here. Why? Because the more successful and widespread your product is, the more those sociopaths, the more those sentient opponents, will adopt you as a special project. Just ask Microsoft. If you want to get widespread adoption, you will be tested. The only question is "Tested by whom?"
- **Not knowing how**—And so we come to the purpose of this book. You are ready, willing, and unable. Or you want to make sure that you're as up to date as your opponents. Or you need raw material for even more extreme sports than what is outlined here. You've come to a right place (there is no "the" right place). This is (let's be clear) a very right place. The authors are proven, and the techniques are current. Although techniques in security have the terrible beauty of never being "done," you won't do much better than these. If you can, there is an audience for your book. In the meantime, absorb what Chris Wysopal, Lucas Nelson, Dino Dai Zovi, and Elfriede Dustin have to teach you, and put it into practice. Skill sets like these do not grow on trees, and they don't stand still any more than the opposition stands still.

As you can see from the table of contents, testing is a way of thinking, not a button to press or a budget item to approve. You very nearly have to adopt this way of thinking—and nothing enforces a way of thinking as much as the regular use of tools and techniques that embody it. This is no joke. The outside attacker is skillful and increasingly professional and has tools and thought patterns. Malware—in particular, malware that turns good citizens into unintentionally bad citizens—has made true the long-standing supposition of security geeks: The real threat is the insider.

Question: What is an external attacker's first measure of success? Answer: Gaining an insider's credentials, access, and authority. If that attacker intends to do so by exploiting software the target insider runs, only your design and your testing stand in the way of the attacker's goals. As shown in the following figure, the idea is not "Does the product do what it is supposed to do?" but "Does the product not do what it supposed to not do?" That question is far harder than the quality assurance question because it is inherently open-ended. It cannot be fully handled by development per se. It has to be tested—preferably by informed testers not tangled up with the build process.

Quality vs. Security[1]



That, again, is where this book comes in. It tells you how to exert the kind of expert pressure that does accelerated failure time testing. You learn how to do so efficiently enough to be willing to do the testing and not think that you can get away without it. In other words, this is hunting in the bush. You can learn to do it by yourself, but following an expert tracker is a faster education than learning everything the hard way. Absorb everything that is here, and you'll either be a formidable hunter or you'll be in a position to be a tracker yourself. Remember, all skill is the result of practice. These authors are well practiced; it is your turn, and they have given you a leg up.

Daniel E. Geer, Jr., Sc.D.
24 July 2006

Endnote

1. Herbert H. Thompson and James A. Whittaker. Testing for Software Security. *Dr. Dobb's Journal*, 27(11): 24–32, November 2002.