Foreword

Vulnerabilities are the life-blood of security research. Whether you are performing a penetration test, evaluating a new product, or auditing the source code of a critical component—vulnerabilities drive your decisions, provide justification for your time, and influence your choices for years to come.

Source code auditing is a white box testing technique that has long been a popular approach for uncovering vulnerabilities in software products. This method requires the auditor to know every programming concept and function used in the product, and to have a deep understanding of the product's operating environment. Source code auditing also has one obvious pitfall—the source code for the product must be available.

Thankfully, there are black box alternatives that do not require access to source code. One such alternative is a technique known as fuzzing, which has proven successful at discovering critical vulnerabilities in products that would not be feasible to audit any other way. Fuzzing is the process of sending intentionally invalid data to a product in the hopes of triggering an error condition or fault. These error conditions can lead to exploitable vulnerabilities.

There are no real rules for fuzzing. It is a technique where success is measured solely by the results of the test. For any given product, there may be an infinite number of inputs you can provide. Fuzzing is the process of predicting what types of programming errors may exist in the product and the inputs that will trigger those errors. For this reason, fuzzing is much more of an art than a science.

Fuzzing can be as simple as randomly hitting keys on a keyboard. The 3-year-old son of a friend once discovered a vulnerability in the Mac OS X operating system's screen

lock using this technique. My friend locked his screen and walked to the kitchen to get a drink. By the time he got back, his son had managed to disable the screen lock, and open a web browser, just by banging on the keyboard.

Over the last few years, I have used fuzzing tools and techniques to discover hundreds of vulnerabilities in an extensive range of software. In December 2003, I wrote a simple tool that sent a stream of random UDP packets to a remote service. This tool was used to discover two new vulnerabilities in the Microsoft WINS server. The same tool later aided in identifying serious flaws in a handful of other products. It turned out that a random stream of UDP packets was all that was needed to identify vulnerabilites in multiple Computer Associates products, the Norton Ghost management service, and a common service exposed on the Mac OS X operating system.

Fuzzers are useful for much more than network protocols. In the first quarter of 2006, I collaborated on three different browser fuzzing tools, leading to the discovery of dozens of flaws across a wide range of web browsers. In the second quarter of 2006, I wrote an ActiveX fuzzer (AxMan) that discovered over 100 unique flaws across Microsoft products alone. Many of these flaws were profiled during the Month of Browser Bugs project and led to the development of exploit modules for the Metasploit Framework. I am still finding new vulnerabilities with AxMan, nearly a year after it was originally developed. Fuzzers truly are the gift that keeps on giving.

This book is the first resource to do justice to fuzzing as a technique. The knowledge provided in this book is everything you need to start fuzzing new products and to build effective fuzzing tools of your own. The key to effective fuzzing is knowing what data to use for which products, and the tools needed to manipulate, monitor, and manage the fuzzing process. The authors of this book are pioneers in the area of fuzzing techniques and do an excellent job of covering the intricacies of the fuzzing process.

Happy bug hunting!

—HD