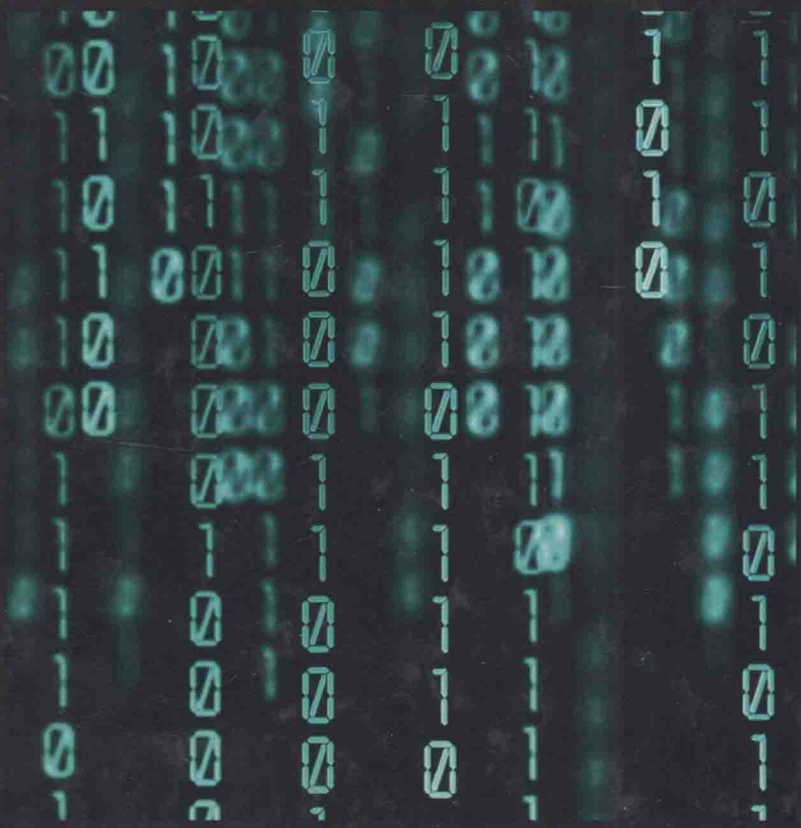
ARTECH HOUSE

INFORMATION SECURITY AND PRIVACY SERIES

FUZZING

for Software Security Testing and Quality Assurance

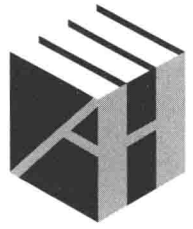


ARI TAKANEN • JARED D. DEMOTT

CHARLES MILLER

Fuzzing for Software Security Testing and Quality Assurance

Ari Takanen
Jared DeMott
Charlie Miller



ARTECH
HOUSE
BOSTON | LONDON
artechhouse.com

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library.

ISBN 13: 978-1-59693-214-2

Cover design by Igor Valdman

685 Canton Street
Norwood, MA 02062

10 9 8 7 6 5 4 3 2 1

Fuzzing for Software Security Testing and Quality Assurance

This book is dedicated to our families and friends . . .

. . . and also to all quality assurance specialists and security experts who are willing to share their knowledge and expertise to enable others to learn and improve their skills.

Foreword

It was a dark and stormy night. Really.

Sitting in my apartment in Madison in the Fall of 1988, there was a wild midwest thunderstorm pouring rain and lighting up the late night sky. That night, I was logged on to the Unix systems in my office via a dial-up phone line over a 1200 baud modem. With the heavy rain, there was noise on the line and that noise was interfering with my ability to type sensible commands to the shell and programs that I was running. It was a race to type an input line before the noise overwhelmed the command.

This fighting with the noisy phone line was not surprising. What did surprise me was the fact that the noise seemed to be causing programs to crash. And more surprising to me was the programs that were crashing—common Unix utilities that we all use everyday.

The scientist in me said that we need to make a systematic investigation to try to understand the extent of the problem and the cause.

That semester, I was teaching the graduate Advanced Operating Systems course at the University of Wisconsin. Each semester in this course, we hand out a list of suggested topics for the students to explore for their course project. I added this testing project to the list.

In the process of writing the description, I needed to give this kind of testing a name. I wanted a name that would evoke the feeling of random, unstructured data. After trying out several ideas, I settled on the term "fuzz."

Three groups attempted the fuzz project that semester and two failed to achieve any crash results. Lars Fredriksen and Bryan So formed the third group, and were more talented programmers and most careful experiments; they succeeded well beyond my expectations. As reported in the first fuzz paper [cite], they could crash or hang between 25–33% of the utility programs on the seven Unix variants that they tested.

However, the fuzz testing project was more than a quick way to find program failures. Finding the cause of each failure and categorizing these failures gave the results deeper meaning and more lasting impact. The source code for the tools and scripts, the raw test results, and the suggested bug fixes were all made public. Trust and repeatability were crucial underlying principles for this work.

In the following years, we repeated these tests on more and varied Unix systems for a larger set of command-line utility programs and expanded our testing to GUI programs based on the then-new X-window system [cite fuzz 1995]. Windows followed several years later [cite fuzz 2000] and, most recently, MacOS [cite fuzz 2006]. In each case, over the span of the years, we found a lot of bugs and, in each case, we diagnosed those bugs and published all of our results.

In our more recent research, as we have expanded to more GUI-based application testing, we discovered that classic 1983 testing tool, "The Monkey" used on the earlier Macintosh computers [cite Hertzfeld book]. Clearly a group ahead of their time.

In the process of writing our early fuzz papers, we came across strong resistance from the testing and software engineering community. The lack of a formal model and methodology and undisciplined approach to testing often offended experienced practioners in the field. In fact, I still frequently come across hostile attitudes to this type of "stone axes and bear skins" (my apologies to Mr. Spock) approach to testing.

My response was always simple: "We're just trying to find bugs." As I have said many times, fuzz testing is not meant to supplant more systematic testing. It is just one more tool, albeit, and an extremely easy one to use, in the tester's toolkit.

As an aside, note that the fuzz testing has not ever been a funded research effort for me; it is a research advocation rather than a vocation. All the hard work has been done by a series of talented and motivated graduate students in our Computer Sciences Department. This is how we have fun.

Fuzz testing has grown into a major subfield of research and engineering, with new results taking it far beyond our simple and initial work. As reliability is the foundation of security, so has it become a crucial tool in security evaluation of software. Thus, the topic of this book is both timely and extremely important. Every practitioner who aspires to write safe and secure software needs to add these techniques to their bag of tricks.

Barton Miller
Madison, Wisconsin
April 2008

Operating System Utility Program Reliability—The Fuzz Generator

The goal of this project is to evaluate the robustness of various Unix utility programs, given an unpredictable input stream. This project has two parts. First, you will build a "fuzz" generator. This is a program that will output a random character stream. Second, you will take the fuzz generator and use it to attack as many Unix utilities as possible, with the goal of trying to break them. For the utilities that break, you will try to determine what type of input caused the break.

The Program

The fuzz generator will generate an output stream of random characters. It will need several options to give you flexibility to test different programs. Below is the start for a list of options for features that fuzz will support. It is important when writing this program to use good C and Unix style, and good structure, as we hope to distribute this program to others.

–p	only the printable ASCII characters
–a	all ASCII characters
–0	include the null (0 byte) character
–l	generate random length lines (\\n terminated strings)
–f name	record characters in file "name"
–d nnn	delay nnn seconds following each character
–r name	replay characters in file "name" to output

The Testing

The fuzz program should be used to test various Unix utilities. These utilities include programs like vi, mail, cc, make, sed, awk, sort, etc. The goal is to first see if the program will break and second to understand what type of input is responsible for the break.

Preface

Still today, most software fails with negative testing, or fuzzing, as it is known by security people. I (Ari) have never seen a piece of software or a network device that passes all fuzz tests thrown at it. Still, things have hopefully improved a bit from 1996 when we started developing our first fuzzers, and at least from the 1970s when Dr. Boris Beizer and his team built their fuzzer-like test automation scripts. The key driver for the change is the adaptation of these tools and techniques, and availability of the technical details on how this type of testing can be conducted. Fortunately there has been enormous development in the fuzzer market, as can be seen from the wide range of available open source and commercial tools for this test purpose.

The idea for this book came up in 2001, around the same time when we completed the PROTOS Classic project on our grammar-based fuzzers. Unfortunately we were distracted by other projects. Back then, as a result of the PROTOS project, we spawned a number of related security "spin-offs." One of them was the commercial company Codenomicon, which took over all technical development from PROTOS Classic, and launched the first commercial fuzzers in early 2002 (those were for SIP, TLS, and GTP protocols if you are interested). Another was the PROTOS Genome project, which started looking at the next steps in fuzzing and automated protocol reverse-engineering, from a completely clean table (first publicly available tests were for various compression formats, released in March 2008). And the third was FRONTIER, which later spun-out a company doing next-gen network analysis tools and was called Clarified Networks. At the same time we kept our focus on fuzzer research and teaching on all areas of secure programming at the University of Oulu. And all this was in a small town of about two hundred thousand people, so you could say that one out of a thousand people were experts in fuzzing in this far-north location. But, unfortunately, the book just did not fit into our plans at that time.

The idea for the book re-emerged in 2005 when I reviewed a paper Jared DeMott wrote for the Blackhat conference. For the first time since all the published and some unpublished works at PROTOS, I saw something new and unique in that paper. I immediately wrote to Jared to propose that he would co-author this fuzzer book project with me, and later also flew in to discuss with him to get to know him better. We had completely opposite experiences and thoughts on fuzzing, and therefore it felt like a good fit, and so finally this book was started. Fortunately I had a dialog going on with Artech House for some time already, and we got to start the project almost immediately.

We wanted everything in the book to be product independent, and also technology independent. With our combined experiences, this seemed to be natural for the book. But something was still missing. As a last desperate action in our constant struggle to get this book completed by end of 2007, we reached out to Charlie Miller. The

main reason for contacting him was that we wanted to have a completely independent comparison of various fuzzer technologies, and did not want to write that ourselves as both of us had strong opinions in various, conflicting, directions. I, for instance, have always been a strong believer in syntax testing-based negative testing (some call this model-based fuzzing), with no random component to the tests. Jared on the other hand was working on evolutionary fuzzers. Charlie accepted to write a chapter, but later actually got more deeply involved in the project and ended up writing almost one third of the book (Charlie should definitely do more traveling, as he claims he wrote all that in an airplane).

Our goal was to write something that would be used as a course book at universities, but also as a useful reference for both quality assurance engineers and security specialists. And I think we succeeded quite well. The problem with other available books was that they were targeted to either security people, or to quality assurance, or on very rare occasions to the management level. But fuzzing is not only about security, as fuzzers are used in many closed environments where there are no security threats. It is also not only about software quality. Fuzzing is a convergence of security practices into quality assurance practices, or sometimes the other way around. In all 100+ global customers of Codenomicon fuzzing tools (in late 2007), from all possible industry verticals, the same logic is always apparent in deploying fuzzers: Fuzzing is a team effort between security people and quality assurance people.

There are many things that were left out of this edition of the book, but hopefully that will motivate you to buy enough books so that the publisher will give us an opportunity to improve. This book will never be complete. For example in 2007 and early 2008 there were a number of completely new techniques launched around fuzzing. One example is the recent release of the PROTOS Genome. Also, commercial companies constantly continue to develop their offerings, such as the rumors of the Wurldtech "Achilles Inside" (whatever that will be), and the launch of the "fifth generation" Codenomicon Defensics 3.0 fuzzing framework, both of which were not covered in this book. Academics and security experts have also released new frameworks and tools. One example that you definitely should check out is the FuzzGuru, available through OWASP. I am also expecting to see something completely different from the number of academics working with fuzzing, such as the techniques developed by the Madynes team in France.

We promise to track those projects now and in the future, and update not only this book, but also our web site dedicated to fuzzing-related topics (www.fuzz-test.com.) For that, please contact us with your comments, whether they are positive or negative, and together we will make this a resource that will take software development a giant leap forward, into an era where software is reliable and dependable.

Ari, Jared, and Charlie

Acknowledgments

From Ari Takanen

There have been several people who have paved the way toward the writing of this book. First of all, I want to give big hugs and thanks to my home team. My family has been supportive in this project even if it has meant 24-hour workdays away from family activities. Combining a couple of book projects with running a security company, traveling 50% of the year around the globe attending various conferences, and meeting with global customers can take a bit of time from the family. I keep making promises about dedicating more time for the family, but always fail those promises.

I am forever grateful to both Marko Laakso and Prof. Juha Röning from University of Oulu for showing me how everything is broken in communication technologies. Everything. And showing that there is no silver bullet to fix that. That was really eye-opening. To me, my years as a researcher in the OUSPG enabled me to learn everything there was to learn about communications security.

Enormous thanks to all my colleagues at Codenomicon, for taking the OUSPG work even further through commercializing the research results, and for making it possible for me to write this book although it took time from my CTO tasks. Special thanks to Heikki and Rauli. Thank you to everyone who has used either the Codenomicon robustness testing tools, or the PROTOS test-suites, and especially to everyone who came back to us and told us of their experiences with our tools and performing security testing with them. Although you might not want to say it out loud, you certainly know how broken everything is. Special thanks to Sven Weizenegger who provided valuable insight into how fuzzers are used and deployed in real-life penetration testing assignments.

I would like to thank everyone involved at Artech House, and all the other people who patiently helped with all the editing and reviewing, and impatiently reminded about all the missed deadlines during the process. Special thanks to Dr. Boris Beizer for the useful dialog on how syntax testing (and fuzzing) was done in the early 70s, and to Michael Howard for the review comments.

Finally, thanks Jared and Charlie for joining me in this project. Although it was slow and painful at times, it certainly was more fun than anything else.

From Jared DeMott

Jared would like to thank God and those he's known. The Lord formed me from dust and my family, friends, co-workers, classmates, and students have shaped me from there. Special thanks to Ari, Charlie, and Artech for working hard to keep the book project on track. Thanks to my beautiful wife and two energetic boys for supporting

all of my career endeavors, and thanks to our parents for giving us much needed breaks and support.

Our goal is that readers of this book will receive a well-rounded view of computing, security, software development, and of course an in-depth knowledge of the art and science of this evolving branch of dynamic software testing known as fuzzing.

From Charlie Miller

I'd like to thank my family for their love and support. They make everything worth while. I'd also like to thank JRN, RS, JT, OB, and EC for teaching me this stuff. Finally, thanks to Michael Howard for his insightful comments while editing.

Recent Related Artech House Titles

Achieving Software Quality Through Teamwork, Isabel Evans

Agile Software Development, Evaluating the Methods for Your Organization, Alan S. Koch

Agile Systems with Reusable Patterns of Business Knowledge: A Component-Based Approach, Amit Mitra and Amar Gupta

Discovering Real Business Requirements for Software Project Success, Robin F. Goldsmith

Engineering Wireless-Based Software Systems and Applications, Jerry Zeyu Gao, Simon Shim, Xiao Su, and Hsin Mei

Enterprise Architecture for Integration: Rapid Delivery Methods and Technologies, Clive Finkelstein

Fuzzing for Software Security Testing and Quality Assurance, Ari Takanen, Jared DeMott, and Charlie Miller

Handbook of Software Quality Assurance, Fourth Edition, G. Gordon Schulmeyer

Implementing the ISO/IEC 27001 Information Security Management Standard, Edward Humphreys

Open Systems and Standards for Software Product Development, P. A. Dargan

Practical Insight into CMMI®, Tim Kasse

A Practitioner's Guide to Software Test Design, Lee Copeland

Role-Based Access Control, Second Edition, David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli

Software Configuration Management, Second Edition, Alexis Leon

Utility Computing Technologies, Standards, and Strategies, Alfredo Mendoza

Workflow Modeling: Tools for Process Improvement and Application Development, Alec Sharp and Patrick McDermott

For further information on these and other Artech House titles, including previously considered out-of-print books now available through our In-Print-Forever® (IPF®) program, contact:

Artech House
685 Canton Street
Norwood, MA 02062
Phone: 781-769-9750
Fax: 781-769-6334
e-mail: artech@artechhouse.com

Artech House
46 Gillingham Street
London SW1V 1AH UK
Phone: +44 (0)20 7596-8750
Fax: +44 (0)20 7630-0166
e-mail: artech-uk@artechhouse.com

Find us on the World Wide Web at: www.artechhouse.com

Contents