

Book Review: Coding for Penetration Testers

Review by Andrew Johnson CISSP, GPEN, eCPPT, OSWP et al

With a title as ambitious as Coding for Penetration Testers, it's important to set expectations properly at the onset. In this context, coding is synonymous with scripting, and the content primarily focuses on Bash Scripting, Python, PERL, Ruby, PHP, SQL, PowerShell, and scripting related to various scanners such as Nmap and Nessus. Compiled languages such as Assembly, Java, and the C variants are not within the content's scope.

This Syngress published book by EH-Net Columnists Jason Andress and Ryan Linn strives to remove the mystery surrounding the development of security tools and scripts by presenting dozens of easy-to-follow examples. The ultimate goal is to alleviate the reliance on pre-built security tools and create more versatile and effective penetration testers. With this resource, readers will gain the knowledge to start such a journey that will likely have numerous, "That's all there is to it!" epiphanies as they progress through the book.

Discuss in Forums {mos_smf_discuss:Book Reviews}

- Foreword by Ed Skoudis
- Chapter 0: Introduction
- Chapter 1: Introduction to Command Shell Scripting
- Chapter 2: Introduction to Python
- Chapter 3: Introduction to Perl
- Chapter 4: Introduction to Ruby
- Chapter 5: Introduction to Web Scripting with PHP
- Chapter 6: Manipulating Windows with PowerShell
- Chapter 7: Scanner Scripting
- Chapter 8: Information Gathering
- Chapter 9: Exploitation Scripting
- Chapter 10: Post-Exploitation Scripting
- Appendix: Subnetting and CIDR Addresses

At approximately 300 pages in length, this breadth of topics will obviously not be covered comprehensively, and the authors emphasize this point from the very beginning. The purpose of this book is not to teach the aforementioned languages in their entirety, but rather to demonstrate their use in practical, real-world penetration testing activities such as password cracking, network scanning, and exploit development.

The number of languages covered may seem intimidating to those with limited programming experience, but the barrier to entry is likely lower than most would expect. Common programming activities such as defining variables, using loops, and creating conditional statements are introduced as the book progresses. While it is certainly possible for novice programmers to make it through the book by simply following along with the examples, readers will undoubtedly find the content easier to grasp if they have an existing understanding of basic programming concepts. For readers with absolutely no programming experience, it is recommended to spend some time with an introductory resource like the Google Two-Day Python Course in advance.

Despite its breadth, the chapters are not shallow. A single chapter may start off defining classes that perform basic file operations and then quickly transition into banner grabbing, SNMP scanning, database development, and the creation of home-grown file transfer protocols. Network programming, a Scapy introduction, and the creation of a small vulnerable web application are other noteworthy examples of content. The book's variety is one of its most impressive attributes. Even if a reader is familiar with the material in one or two chapters, a wealth of other useful information is presented throughout the rest of the book.

Each chapter can stand on its own and be read in isolation, but the content is arranged in a manner that flows naturally

when read from cover to cover. The structure of each chapter typically consists of highlighting the unique characteristics and benefits of a given language, introducing new concepts that will be implemented in the upcoming code, and then creating several practical, functional tools. The chapters conclude with suggestions for increasing the tools' functionality, and additional references are provided for further independent research.

The topics presented progress in difficulty as each additional language is introduced. For example, the Perl chapter is more advanced than the Python chapter, and the Ruby chapter is subsequently more advanced than the Perl chapter. This progression also loosely mirrors an actual penetration test and transitions through stages of information gathering, scanning, exploitation, and post-exploitation.

This book succeeds at packing a great deal of information into a limited space by keeping the amount of repetition to a minimum. After constructs such as loops and conditional statements are initially introduced, they are usually only mentioned again to make note of any interesting or unique characteristics of a given language. Additionally, the concepts learned in later chapters can often easily be applied to a language covered in an earlier chapter. For example, classes aren't introduced until the Ruby chapter, but the general concept could easily be translated to Python with a minimal amount of research and effort. The fact that such similar languages were presented so uniquely, without space wasted on repetition, is truly a testament to the authors' and editor's foresight.

The chapter on exploit development walks the reader through creating a standard proof-of-concept exploit in Python and then porting it over to Ruby for use in Metasploit. Readers unfamiliar with exploit development and/or the Metasploit Framework will have some difficulty fully understanding what's going on behind-the-scenes. However, these demonstrations successfully provide a general overview of the exploitation process and may pique readers' interest in further studies in exploit development. This chapter also contains a variety of web-based exploitation techniques that readers may find more accessible.

The concluding chapter picks up where exploitation left off and focuses on post-exploitation techniques that demonstrate the importance of continuing a penetration test once shell or root access is obtained. Post-exploitation automation is demonstrated with OS and Meterpreter scripting, and some Microsoft SQL Server manipulation is demonstrated as well.

Most exercises in the book require nothing more than a text editor and a terminal. However, some effort must be put forth by the reader in order to fully follow along with every exercise. This effort is rather minimal for tasks like setting up a web server for the web application exercises. On the other hand, having an available Windows XP SP0 system for exploit development, or MS SQL Server for database attacks, may prove to be more challenging.

The writing is extremely polished, and it's quite apparent that the authors and editor are masters of their craft. The few errors that were identified were obvious and trivial to correct, such as auto-capitalization changing the case of a variable after a newline. Even though errors rarely occur, it would be beneficial for the publisher to create an errata webpage that details these corrections. Downloadable code examples would be another worthwhile online addition.

In conclusion, Coding for Penetration Testers is a must-read for any novice or aspiring penetration tester. More experienced penetration testers who are familiar with tools and techniques, but have never done any significant amount of coding, will likely benefit greatly from this book as well. This is the best single resource to begin a journey into the upper-echelons of penetration testing.

Andrew Johnson (CISSP, GWAPT, GPEN, GCIH, GSEC, CEH, eCPPT, OSWP, CWSP, CCNA:S, MCSE:S, et al) has over a decade of experience in information technology and security. He has provided information security services, including penetration testing, social engineering, and risk management, to over a hundred financial institutions, businesses, and other organizations across the country. He currently manages information security for the US operations of a financial services company. Andrew is a perpetual learner and enjoys sharing knowledge with others. He is a SANS Mentor, Advisory Board Member, and Exam Question Writer. His personal security blog is at www.infosiege.net, and he can be followed @infosiege.