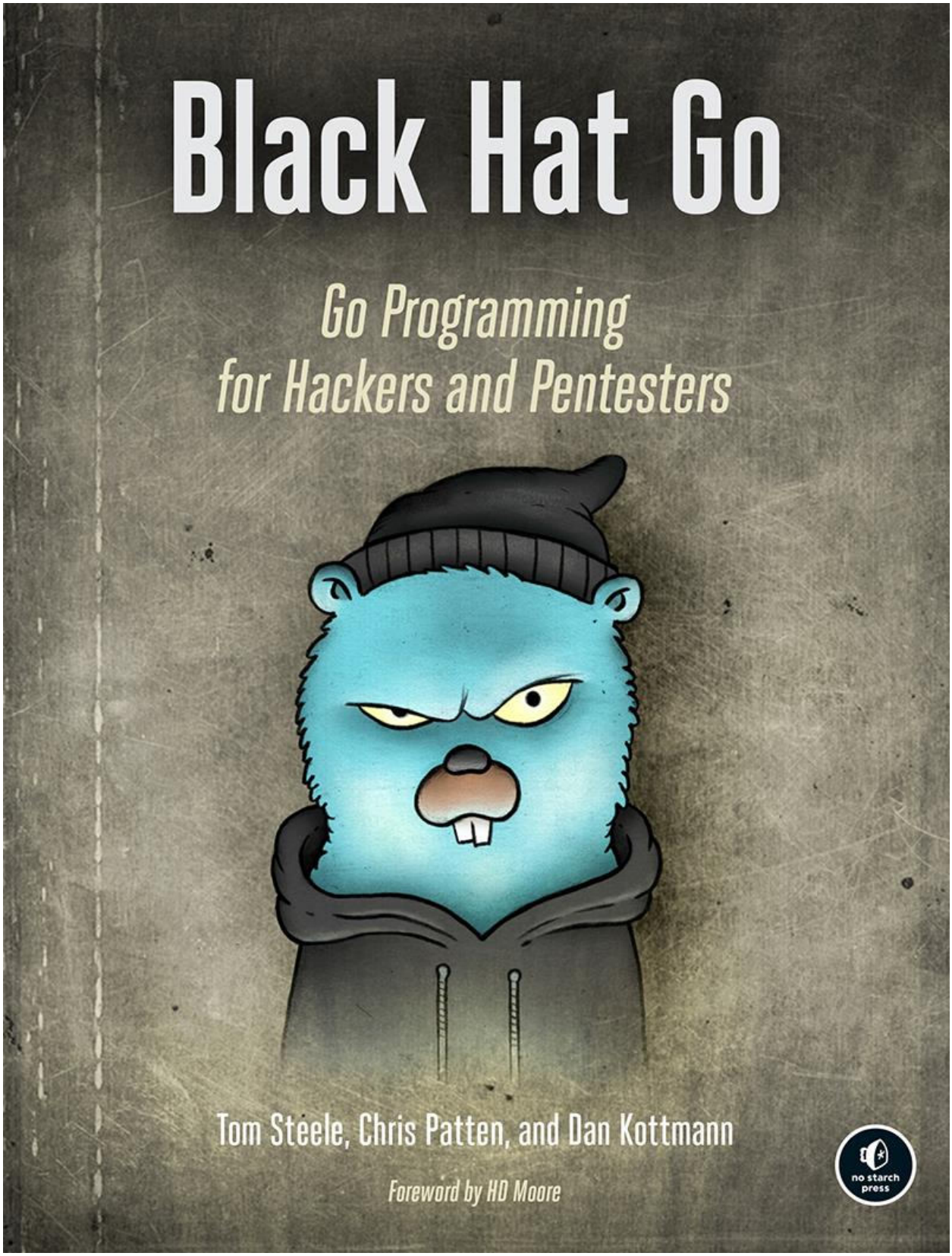
Black Hat Go
Go Programming
for Hackers and Pentesters
Tom Steele, Chris Patten, and Dan Kottmann
Foreword by HD Moore
no starch press

BLACK HAT GO

Go Programming for Hackers and Pentesters

by Tom Steele, Chris Patten, and Dan Kottmann

San Francisco

ISBN-10: 1-59327-865-9
ISBN-13: 978-1-59327-865-6

Publisher: William Pollock
Production Editor: Laurel Chun
Cover Illustration: Jonny Thomas
Interior Design: Octopod Studios
Developmental Editors: Frances Saux and Zach Lebowski
Technical Reviewer: Alex Harvey
Copyeditor: Sharon Wilkey
Compositor: Danielle Foster
Proofreader: Brooke Littrel
Indexer: Beth Nauman-Montana

For information on distribution, translations, or bulk sales, please contact No Starch Press, Inc. directly:
No Starch Press, Inc.
245 8th Street, San Francisco, CA 94103
phone: 1.415.863.9900; info@nostarch.com
www.nostarch.com

Library of Congress Cataloging-in-Publication Data

Names: Steele, Tom (Security Consultant), author. | Patten, Chris, author. | Kottmann, Dan, author.
Title: Black Hat Go : Go programming for hackers and pentesters / Tom Steele, Chris Patten, and Dan Kottmann.
Description: San Francisco : No Starch Press, 2020. | Includes bibliographical references and index. | Summary: "A guide to Go that begins by introducing fundamentals like data types, control structures, and error handling. Provides instruction on how to use Go for tasks such as sniffing and processing packets, creating HTTP clients, and writing exploits."-- Provided by publisher.
Identifiers: LCCN 2019041864 (print) | LCCN 2019041865 (ebook) | ISBN 9781593278656 | ISBN 9781593278663 (ebook)

Subjects: LCSH: Penetration testing (Computer security) | Go (Computer program language)
Classification: LCC QA76.9.A25 S739 2020 (print) | LCC QA76.9.A25 (ebook)
| DDC 005.8--dc23
LC record available at https://lccn.loc.gov/2019041864
LC ebook record available at https://lccn.loc.gov/2019041865

ABOUT THE AUTHORS

Tom Steele has been using Go since the version 1 release in 2012 and was one of the first in his field to leverage the language for offensive tooling. He is a managing principal research consultant at Atredis Partners with over 10 years of experience performing adversarial and research-based security assessments. Tom has presented and conducted training courses at numerous conferences, including Defcon, Black Hat, DerbyCon, and BSides. Outside of tech, Tom is also a Black Belt in Brazilian jiujitsu who competes regularly, both regionally and nationally. He owns and operates his own jiujitsu academy in Idaho.

Chris Patten is the founding partner and lead consultant of STACKTITAN, a specialized adversarial services security consultancy. Chris has been practicing in the security industry for more than 25 years in various capacities. He spent the last decade consulting for a number of commercial and government organizations on diverse security issues, including adversarial offensive techniques, threat hunting capabilities, and mitigation strategies. Chris spent his latest tenure leading one of North America's largest advanced adversarial teams.

Prior to formal consulting, Chris honorably served in the US Air Force, supporting the war-fighting effort. He actively served within the Department of Defense Special Operations Intelligence community at USSOCOM, consulting for Special Operations Groups on sensitive cyber warfare initiatives. Following Chris's military service, he held lead architect positions at numerous Fortune 500 telecommunication companies, working with partners in a research capacity.

Dan Kottmann is a founding partner and lead consultant of STACKTITAN. He has played an integral role in the growth and development of the largest North American adversarial consultancy, directly influencing technical tradecraft, process efficiency, customer experience, and delivery quality. With 15 years of experience, Dan has dedicated nearly the entirety of his professional career to cross-industry, customer-direct consulting and consultancy development, primarily focused on information security and application delivery.

Dan has presented at various national and regional security conferences, including Defcon, BlackHat Arsenal, DerbyCon, BSides, and more. He has a passion for software development and has created various open-source and proprietary applications, from simple command line tools to complex, three-tier, and cloud-based web applications.

ABOUT THE TECHNICAL REVIEWER

Alex Harvey has been working with technology his whole life and got his start with embedded systems, robotics, and programming. He moved into information security about 15 years ago, focusing on security testing and research. Never one to shy away from making a tool for the job, he started using the Go programming language and has not looked back.

BRIEF CONTENTS

CONTENTS IN DETAIL

FOREWORD

Programming languages have always had an impact on information security. The design constraints, standard libraries, and protocol implementations available within each language end up defining the attack surface of any application built on them. Security tooling is no different; the right language can simplify complex tasks and make the incredibly difficult ones trivial. Go's cross-platform support, single-binary output, concurrency features, and massive ecosystem make it an amazing choice for security tool development. Go is rewriting the rules for both secure application development and the creation of security tools, enabling faster, safer, and more portable tooling.

Over the 15 years that I worked on the Metasploit Framework, the project went through two full rewrites, changed languages from Perl to Ruby, and now supports a range of multilingual modules, extensions, and payloads. These changes reflect the constantly evolving nature of software development; in order to keep up in security, your tools need to adapt, and using the right language can save an enormous amount of time. But just like Ruby, Go didn't become ubiquitous overnight. It takes a leap of faith to build anything of value using a new language, given the uncertainties of the ecosystem and the sheer amount of effort needed to accomplish common tasks before the standard libraries catch up.

The authors of *Black Hat Go* are pioneers in Go security tool development, responsible for some of the earliest open source Go projects, including BlackSheepWall, Lair Framework, and sipbrute, among many others. These projects serve as excellent examples of what can be built using the language. The authors are just as comfortable building software as tearing it apart, and this book is a great example of their ability to combine these skills.

Black Hat Go provides everything necessary to get started with Go development in the security space without getting bogged down into the lesser-used language features. Want to write a ridiculous fast network scanner, evil HTTP proxy, or cross-platform command-and-control framework? This book is for you. If you are a seasoned programmer looking for insight into security tool development, this book will introduce the concepts and trade-offs that hackers of all stripes consider when writing tools. Veteran Go developers who are interested in security may learn a lot from the approaches taken here, as building tools to attack other software requires a different mindset than typical application development. Your design trade-offs will likely be substantially different when your goals include bypassing security controls and evading detection.

If you already work in offensive security, this book will help you build utilities that are light-years faster than existing solutions. If you work on the

defense side or in incident response, this book will give you an idea of how to analyze and defend against malware written in the Go language.

Happy hacking!

HD Moore
Founder of the Metasploit Project and the Critical Research Corporation
VP of Research and Development at Atredis Partners

ACKNOWLEDGMENTS

This book would not be possible had Robert Griesemer, Rob Pike, and Ken Thompson not created this awesome development language. These folks and the entire core Go development team consistently contribute useful updates upon each release. We would have never written this book had the language not been so easy and fun to learn and use.

The authors would also like to thank the team at No Starch Press: Laurel, Frances, Bill, Annie, Barbara, and everyone else with whom we interacted. You all guided us through the unchartered territory of writing our first book. Life happens—new families, new jobs—and all the while you've been patient but still pushed us to complete this book. The entire No Starch Press team has been a pleasure to work with on this project.

I would like to thank Jen for all her support, encouragement, and for keeping life moving forward while I was locked away in my office nights and weekends, working on this never-ending book. Jen, you helped me more than you know, and your constant words of encouragement helped make this a reality. I am sincerely grateful to have you in my life. I must thank "T" (my canine quadra-pet) for holding the floor down in my office while I hacked away and reminding me that "outside" is a real place I should visit. Lastly, and close to my heart, I want to dedicate this book to my pups, Luna and Annie, who passed while I was writing this book. You girls were and are everything to me and this book will always be a reminder of my love for you both.

Chris Patten

I would like to extend a sincere thank you to my wife and best friend, Katie, for your constant support, encouragement, and belief in me. Not a day goes by when I'm not grateful for everything you do for me and our family. I'd like to thank Brooks and Subs for giving me reason to work so hard. There is no better job than being your father. And to the best "Office Hounds" a guy could ask for—Leo (RIP), Arlo, Murphy, and even Howie (yes, Howie too)—you've systematically destroyed my house and periodically made me question my life choices, but your presence and companionship mean the world to me. I'll give each of you a signed copy of this book to chew on.

Dan Kottmann

Thank you to the love of my life, Jackie, for your love and encouragement; nothing I do would be possible without your support and everything you do

for our family. Thank you to my friends and colleagues at Atredis Partners and to anyone I've shared a shell with in the past. I am where I am because of you. Thank you to my mentors and friends who have believed in me since day one. There are too many of you to name; I am grateful for the incredible people in my life. Thank you, Mom, for putting me in computer classes (these were a thing). Looking back, those were a complete waste of time and I spent most of the time playing Myst, but it sparked an interest (I miss the 90s). Most importantly, thank you to my Savior, Jesus Christ.

Tom Steele

It was a long road to get here—almost three years. A lot has happened to get to this point, and here we are, finally. We sincerely appreciate the early feedback we received from friends, colleagues, family, and early-release readers. For your patience, dear reader, thank you so, so very much; we are truly grateful and hope you enjoy this book just as much as we enjoyed writing it. All the best to you! Now Go create some amazing code!

INTRODUCTION

For about six years, the three of us led one of North America's largest dedicated penetration-testing consulting practices. As principal consultants, we executed technical project work, including network penetration tests, on behalf of our clients—but we also spearheaded the development of better tools, processes, and methodology. And at some point, we adopted Go as one of our primary development languages.

Go provides the best features of other programming languages, striking a balance between performance, safety, and user-friendliness. Soon, we defaulted to it as our language of choice when developing tools. Eventually, we even found ourselves acting as advocates of the language, pushing for our colleagues in the security industry to try it. We felt the benefits of Go were at least worthy of consideration.

In this book, we'll take you on a journey through the Go programming language from the perspective of security practitioners and hackers. Unlike other hacking books, we won't just show you how to automate third-party or commercial tools (although we'll touch on that a little). Instead, we'll delve into practical and diverse topics that approach a specific problem, protocol, or tactic useful to adversaries. We'll cover TCP, HTTP, and DNS communications, interact with Metasploit and Shodan, search filesystems and databases, port exploits from other languages to Go, write the core functions of an SMB client, attack Windows, cross-compile binaries, mess with crypto, call C libraries, interact with the Windows API, and much, much more. It's ambitious! We'd better begin . . .

WHO THIS BOOK IS FOR

This book is for anyone who wants to learn how to develop their own hacking tools using Go. Throughout our professional careers, and particularly as consultants, we've advocated for programming as a fundamental skill for penetration testers and security professionals. Specifically, the ability to code

enhances your understanding of how software works and how it can be broken. Also, if you've walked in a developer's shoes, you'll gain a more holistic appreciation for the challenges they face in securing software, and you can use your personal experience to better recommend mitigations, eliminate false positives, and locate obscure vulnerabilities. Coding often forces you to interact with third-party libraries and various application stacks and frameworks. For many people (us included), it's hands-on experience and tinkering that leads to the greatest personal development.

To get the most out of this book, we encourage you to clone the book's official code repository so you have all the working examples we'll discuss. Find the examples at *https://github.com/blackhat-go/bhg/*.

WHAT THIS BOOK ISN'T

This book is not an introduction to Go programming in general but an introduction to using Go for developing security tools. We are hackers and then coders—in that order. None of us have ever been software engineers. This means that, as hackers, we put a premium on function over elegance. In many instances, we've opted to code as hackers do, disregarding some of the idioms or best practices of software design. As consultants, time is always scarce; developing simpler code is often faster and, therefore, preferable over elegance. When you need to quickly create a solution to a problem, style concerns come secondary.

This is bound to anger Go purists, who will likely tweet at us that we don't gracefully handle all error conditions, that our examples could be optimized, or that better constructs or methods are available to produce the desired results. We're not, in most cases, concerned with teaching you the best, the most elegant, or 100 percent idiomatic solutions, unless doing so will concretely benefit the end result. Although we'll briefly cover the language syntax, we do so purely to establish a baseline foundation upon which we can build. After all, this isn't *Learning to Program Elegantly with Go*—this is *Black Hat Go*.

WHY USE GO FOR HACKING?

Prior to Go, you could prioritize ease of use by using dynamically typed languages—such as Python, Ruby, or PHP—at the expense of performance and safety. Alternatively, you could choose a statically typed language, like C or C++, that offers high performance and safety but isn't very user-friendly. Go is stripped of much of the ugliness of C, its primary ancestor, making development more user-friendly. At the same time, it's a statically typed language that produces syntax errors at compile time, increasing your

assurance that your code will actually run safely. As it's compiled, it performs more optimally than interpreted languages and was designed with multicore computing considerations, making concurrent programming a breeze.

These reasons for using Go don't concern security practitioners specifically. However, many of the language's features are particularly useful for hackers and adversaries:

Clean package management system Go's package management solution is elegant and integrated directly with Go's tooling. Through the use of the `go` binary, you can easily download, compile, and install packages and dependencies, which makes consuming third-party libraries simple and generally free from conflict.

Cross-compilation One of the best features in Go is its ability to cross-compile executables. So long as your code doesn't interact with raw C, you can easily write code on your Linux or Mac system but compile the code in a Windows-friendly, Portable Executable format.

Rich standard library Time spent developing in other languages has helped us appreciate the extent of Go's standard library. Many modern languages lack the standard libraries required to perform many common tasks such as crypto, network communications, database connectivity, and data encoding (JSON, XML, Base64, hex). Go includes many of these critical functions and libraries as part of the language's standard packaging, reducing the effort necessary to correctly set up your development environment or to call the functions.

Concurrency Unlike languages that have been around longer, Go was released around the same time as the initial mainstream multicore processors became available. For this reason, Go's concurrency patterns and performance optimizations are tuned specifically to this model.

WHY YOU MIGHT NOT LOVE GO

We recognize that Go isn't a perfect solution to every problem. Here are some of the downsides of the language:

Binary size 'Nuff said. When you compile a binary in Go, the binary is likely to be multiple megabytes in size. Of course, you can strip debugging symbols and use a packer to help reduce the size, but these steps require attention. This can be a drawback, particularly for security practitioners who need to attach a binary to an email, host it on a shared filesystem, or transfer it over a network.

Verbosity While Go is less verbose than languages like C#, Java, or even C/C++, you still might find that the simplistic language construct forces you to be overly expressive for things like lists (called *slices* in Go), processing, looping, or error handling. A Python one-liner might easily become a three-liner in Go.

CHAPTER OVERVIEW

The first chapter of this book covers a basic overview of Go's syntax and philosophy. Next, we start to explore examples that you can leverage for tool development, including various common network protocols like HTTP, DNS, and SMB. We then dig into various tactics and problems that we've encountered as penetration testers, addressing topics including data pilfering, packet sniffing, and exploit development. Finally, we take a brief step back to talk about how you can create dynamic, pluggable tools before diving into crypto, attacking Microsoft Windows, and implementing steganography.

In many cases, there will be opportunities to extend the tools we show you to meet your specific objectives. Although we present robust examples throughout, our real intent is to provide you with the knowledge and foundation through which you can extend or rework the examples to meet your goals. We want to teach you to fish.

Before you continue with anything in this book, please note that we—the authors and publisher—have created this content for legal usage only. We won't accept any liability for the nefarious or illegal things you choose to do. All the content here is for educational purposes only; do not perform any penetration-testing activities against systems or applications without authorized consent.

The sections that follow provide a brief overview of each chapter.

Chapter 1: Go Fundamentals

The goal of this chapter is to introduce the fundamentals of the Go programming language and provide a foundation necessary for understanding the concepts within this book. This includes an abridged review of basic Go syntax and idioms. We discuss the Go ecosystem, including supporting tools, IDEs, dependency management, and more. Readers new to the programming language can expect to learn the bare necessities of Go, which will allow them to, hopefully, comprehend, implement, and extend the examples in later chapters.

Chapter 2: TCP, Scanners, and Proxies

This chapter introduces basic Go concepts and concurrency primitives and patterns, input/output (I/O), and the use of interfaces through practical TCP applications. We'll first walk you through creating a simple TCP port scanner that scans a list of ports using parsed command line options. This will highlight the simplicity of Go code compared to other languages and will develop your understanding of basic types, user input, and error handling. Next, we'll discuss how to improve the efficiency and speed of this port scanner by introducing concurrent functions. We'll then introduce I/O by building a TCP proxy—a port forwarder—starting with basic examples and refining our code to create a more reliable solution. Lastly, we'll re-create Netcat's "gaping security hole" feature in Go, teaching you how to run operating system commands while manipulating stdin and stdout and redirecting them over TCP.

Chapter 3: HTTP Clients and Remote Interaction with Tools

HTTP clients are a critical component to interacting with modern web server architectures. This chapter shows you how to create the HTTP clients necessary to perform a variety of common web interactions. You'll handle a variety of formats to interact with Shodan and Metasploit. We'll also demonstrate how to work with search engines, using them to scrape and parse document metadata so as to extract information useful for organizational profiling activities.

Chapter 4: HTTP Servers, Routing, and Middleware

This chapter introduces the concepts and conventions necessary for creating an HTTP server. We'll discuss common routing, middleware, and templating patterns, leveraging this knowledge to create a credential harvester and keylogger. Lastly, we'll demonstrate how to multiplex command-and-control (C2) connections by building a reverse HTTP proxy.

Chapter 5: Exploiting DNS

This chapter introduces you to basic DNS concepts using Go. First, we'll perform client operations, including how to look for particular domain records. Then we'll show you how to write a custom DNS server and DNS proxy, both of which are useful for C2 operations.

Chapter 6: Interacting with SMB and NTLM

We'll explore the SMB and NTLM protocols, using them as a basis for a discussion of protocol implementations in Go. Using a partial implementation of the SMB protocol, we'll discuss the marshaling and unmarshaling of data, the usage of custom field tags, and more. We'll discuss and demonstrate how to use this implementation to retrieve the SMB-signing policy, as well as perform password-guessing attacks.

Chapter 7: Abusing Databases and Filesystems

Pillaging data is a critical aspect of adversarial testing. Data lives in numerous resources, including databases and filesystems. This chapter introduces basic ways to connect to and interact with databases across a variety of common SQL and NoSQL platforms. You'll learn the basics of connecting to SQL databases and running queries. We'll show you how to search databases and tables for sensitive information, a common technique used during post-exploitation. We'll also show how to walk filesystems and inspect files for sensitive information.

Chapter 8: Raw Packet Processing

We'll show you how to sniff and process network packets by using the gopacket library, which uses libpcap. You'll learn how to identify available network devices, use packet filters, and process those packets. We will then develop a port scanner that can scan reliably through various protection mechanisms, including syn-flood and syn-cookies, which cause normal port scans to show excessive false positives.

Chapter 9: Writing and Porting Exploit Code

This chapter focuses almost solely on creating exploits. It begins with creating a fuzzer to discover different types of vulnerabilities. The second half of the chapter discusses how to port existing exploits to Go from other languages. This discussion includes a port of a Java deserialization exploit and the Dirty COW privilege escalation exploit. We conclude the chapter with a discussion on creating and transforming shellcode for use within your Go programs.

Chapter 10: Go Plugins and Extendable Tools

We'll introduce two separate methods for creating extendable tools. The first method, introduced in Go version 1.8, uses Go's native plug-in mechanism. We'll discuss the use cases for this approach and discuss a second approach that leverages Lua to create extensible tools. We'll demonstrate practical

examples showing how to adopt either approach to perform a common security task.

Chapter 11: Implementing and Attacking Cryptography

This chapter covers the fundamental concepts of symmetric and asymmetric cryptography using Go. This information focuses on using and understanding cryptography through the standard Go package. Go is one of the few languages that, instead of using a third-party library for encryption, uses a native implementation within the language. This makes the code easy to navigate, modify, and understand.

We'll explore the standard library by examining common use cases and creating tools. The chapter will show you how to perform hashing, message authentication, and encryption. Lastly, we'll demonstrate how to brute-force decrypt an RC2-encrypted ciphertext.

Chapter 12: Windows System Interaction and Analysis

In our discussion on attacking Windows, we'll demonstrate methods of interacting with the Windows native API, explore the `syscall` package in order to perform process injection, and learn how to build a Portable Executable (PE) binary parser. The chapter will conclude with a discussion of calling native C libraries through Go's C interoperability mechanisms.

Chapter 13: Hiding Data with Steganography

Steganography is the concealment of a message or file within another file. This chapter introduces one variation of steganography: hiding arbitrary data within a PNG image file's contents. These techniques can be useful for exfiltrating information, creating obfuscated C2 messages, and bypassing detective or preventative controls.

Chapter 14: Building a Command-and-Control RAT

The final chapter discusses practical implementations of command-and-control (C2) implants and servers in Go. We'll leverage the wisdom and knowledge gained in previous chapters to build a C2 channel. The C2 client/server implementation will, by nature of being custom-made, avoid signature-based security controls and attempt to circumvent heuristics and network-based egress controls.

1

GO FUNDAMENTALS

This chapter will guide you through the process of setting up your Go development environment and introduce you to the language's syntax. People have written entire books on the fundamental mechanics of the language; this chapter covers the most basic concepts you'll need in order to work through the code examples in the following chapters. We'll cover everything from primitive data types to implementing concurrency. For readers who are already well versed in the language, you'll find much of this chapter to be a review.

SETTING UP A DEVELOPMENT ENVIRONMENT

To get started with Go, you'll need a functional development environment. In this section, we'll walk you through the steps to download Go and set up your workspace and environment variables. We'll discuss various options for your integrated development environment and some of the standard tooling that comes with Go.

Downloading and Installing Go

Start by downloading the Go binary release most appropriate to your operating system and architecture from *https://golang.org/dl/*. Binaries exist for Windows, Linux, and macOS. If you're using a system that doesn't have an available precompiled binary, you can download the Go source code from that link.

Execute the binary and follow the prompts, which will be minimal, in order to install the entire set of Go core packages. *Packages*, called *libraries* in most other languages, contain useful code you can use in your Go programs.

Setting GOROOT to Define the Go Binary Location

Next, the operating system needs to know how to find the Go installation. In most instances, if you've installed Go in the default path, such as */usr/local/go* on a *Nix/BSD-based system, you don't have to take any action here. However, in the event that you've chosen to install Go in a nonstandard path or are installing Go on Windows, you'll need to tell the operating system where to find the Go binary.

You can do this from your command line by setting the reserved GOROOT environment variable to the location of your binary. Setting environment variables is operating-system specific. On Linux or macOS, you can add this to your *~/.profile*:

```
set GOROOT=/path/to/go
```

On Windows, you can add this environment variable through the System (Control Panel), by clicking the **Environment Variables** button.

Setting GOPATH to Determine the Location of Your Go Workspace

Unlike setting your GOROOT, which is necessary in only certain installation scenarios, you must always define an environment variable named GOPATH to instruct the Go toolset where your source code, third-party libraries, and compiled programs will exist. This can be any location of your choosing. Once you've chosen or created this base workspace directory, create the following three subdirectories within: *bin*, *pkg*, and *src* (more on these directories shortly). Then, set an environment variable named GOPATH that points to your base workspace directory. For example, if you want to place your projects in a directory called *gocode* located within your home directory on Linux, you set GOPATH to the following:

```
GOPATH=$HOME/gocode
```

The *bin* directory will contain your compiled and installed Go executable binaries. Binaries that are built and installed will be automatically placed into this location. The *pkg* directory stores various package objects, including third-party Go dependencies that your code might rely on. For example, perhaps you want to use another developer's code that more elegantly

handles HTTP routing. The *pkg* directory will contain the binary artifacts necessary to consume their implementation in your code. Finally, the *src* directory will contain all the evil source code you'll write.

The location of your workspace is arbitrary, but the directories within must match this naming convention and structure. The compilation, build, and package management commands you'll learn about later in this chapter all rely on this common directory structure. Without this important setup, Go projects won't compile or be able to locate any of their necessary dependencies!

After configuring the necessary GOROOT and GOPATH environment variables, confirm that they're properly set. You can do this on Linux and Windows via the `set` command. Also, check that your system can locate the binary and that you've installed the expected Go version with the `go version` command:

```
$ go version

go version go1.11.5 linux/amd64
```

This command should return the version of the binary you installed.

Choosing an Integrated Development Environment

Next, you'll probably want to select an integrated development environment (IDE) in which to write your code. Although an IDE isn't required, many have features that help reduce errors in your code, add version-control shortcuts, aid in package management, and more. As Go is still a fairly young language, there may not be as many mature IDEs as for other languages.

Fortunately, advancements over the last few years leave you with several, full-featured options. We'll review some of them in this chapter. For a more complete list of IDE or editor options, check out the Go wiki page at *https://github.com/golang/go/wiki/IDEsAndTextEditorPlugins/*. This book is IDE/editor agnostic, meaning we won't force you into any one solution.

Vim Editor

The *Vim* text editor, available in many operating-system distributions, provides a versatile, extensible, and completely open source development environment. One appealing feature of Vim is that it lets users run everything from their terminal without fancy GUIs getting in the way.

Vim contains a vast ecosystem of plug-ins through which you can customize themes, add version control, define snippets, add layout and code-navigation features, include autocomplete, perform syntax highlighting and linting, and

much, much more. Vim's most common plug-in management systems include Vundle and Pathogen.

To use Vim for Go, install the vim-go plug-in (*https://github.com/fatih/vim-go/*) shown in Figure 1-1.

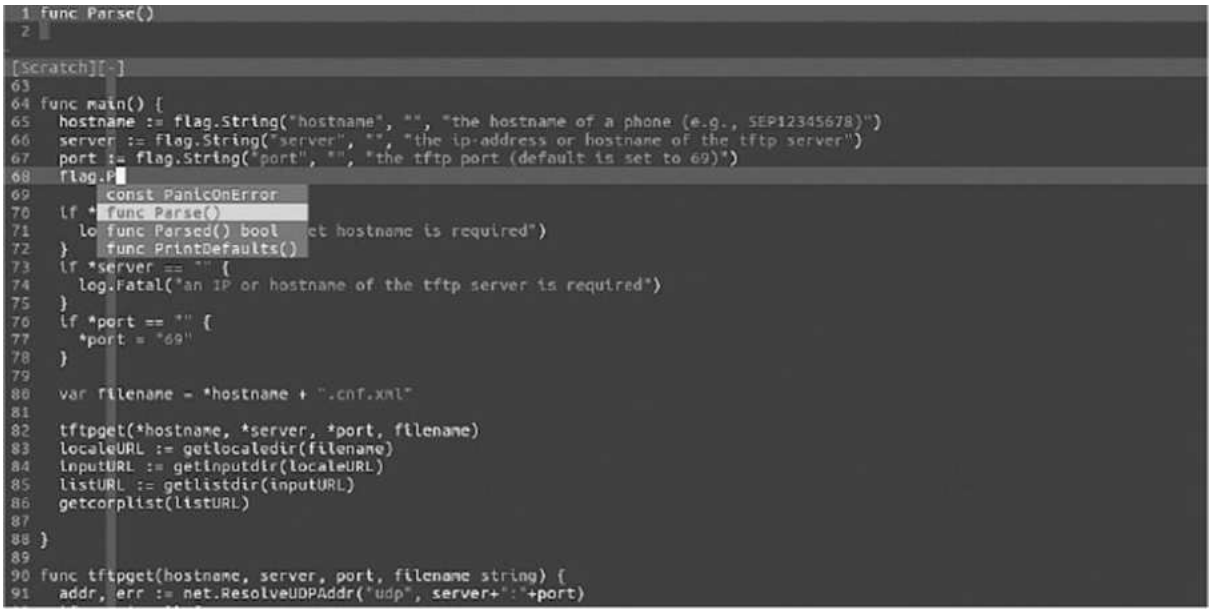

Figure 1-1: The vim-go plug-in

Of course, to use Vim for Go development, you'll have to become comfortable with Vim. Further, customizing your development environment with all the features you desire might be a frustrating process. If you use Vim, which is free, you'll likely need to sacrifice some of the conveniences of commercial IDEs.

GitHub Atom

GitHub's IDE, called *Atom* (*https://atom.io/*), is a hackable text editor with a large offering of community-driven packages. Unlike Vim, Atom provides a dedicated IDE application rather than an in-terminal solution, as shown in Figure 1-2.