



# OFFENSIVE GO

GOLANG FOR PENTESTERS AND RED TEAMERS

OWASP STAMMTISCH 28.03.2018

# AGENDA

- 0x00 Introduction
- 0x10 Basics
- 0x20 Network Programming
- 0x30 Web Hacking
- 0x40 Windows API and Post Exploitation
- 0x50 Wrapping up

The image features a dark blue background with white decorative elements resembling circuit board traces and nodes. These elements are located in the four corners of the page. The central text is '0X00 INTRODUCTION' in a white, bold, sans-serif font.

# 0X00 INTRODUCTION

# 0X01 CODING FOR PENTESTERS

- Current state of the art languages for pentesting
  - Python (sqlmap, OWASP OWTF, pwntools, pwndbg)
  - Ruby (Metasploit framework, beef,
  - Perl (enum4linux, fierce)
- Problems:
  - Dependencies
  - Cross-Platform Compatibility
  - Speed

## 0X02 WHY GO?

- easy to learn (easy-ish syntax)
- Static types + implicit types supported
- Compiles to native, statically linked binaries
- Built-in cross-compilation
- Concurrency is fairly straight forward
- Great toolchain
- Great Stdlib
- Low memory profile

## OX03 THE DOWNSIDES

- No immutable package repository
- Ecosystem not as mature as python's
- Large binaries
  - Can be solved by stripping / packing
- Very reliant on Github (and other VCS)

# 0X04 HOW TO LEARN GO

- Golang: <https://www.golang.org>
- Tour of Go: <https://tour.golang.org/welcome/1>
- Effective Go: [https://golang.org/doc/effective\\_go.html](https://golang.org/doc/effective_go.html)

# 0X05 OFFENSIVE TOOLS IN GO

- GoBot2 (<https://github.com/SaturnsVoid/GoBot2>)
- GoAT (<https://github.com/petercunha/GoAT>)
- Gobuster (<https://github.com/OJ/gobuster>)
- Cracklord (<https://github.com/jmmcatee/cracklord>)
- GoCrack (<https://github.com/fireeye/gocrack>)
- Bettercap 2.0 (<https://github.com/bettercap/bettercap>)
- Merlin (<https://github.com/Ne0nd0g/merlin>)
- Vuls (<https://github.com/future-architect/vuls>)
- ... many more (<https://github.com/topics/pentesting?l=go>)



# 0X06 WHAT'S MISSING

- Mostly libraries for network protocols
  - SOAP (esp. WSDL-parsers)
  - SMB
    - Impacket (python)

The image features a dark blue background with white decorative circuit board patterns in the corners. The top-left and bottom-left corners have more complex, branching patterns, while the top-right and bottom-right corners have simpler, more linear patterns. The text 'OX10 BASICS' is centered on the left side of the image.

# OX10 BASICS

# 0X11 HELLO WORLD

```
1 package main
2
3 import (
4     "fmt"
5     "strconv"
6     "strings"
7 )
8
9 func main() {
10     stringVar := "Hello World!"
11     intVar := 5
12     var (
13         intVar2    int
14         stringVar2 string
15     )
16     intVar2 = 10
17     stringVar2 = strings.Repeat(stringVar, intVar2)
18     fmt.Printf("intVar has value %d\n", intVar)
19     fmt.Printf("stringVar has value %s\n", stringVar)
20     fmt.Printf("stringVar2 has value %s\n", stringVar2)
21     var arrayVar = []string{
22         "Hello ",
23         "World",
24     }
25     arrayVar = append(arrayVar, "!")
26     for idx, element := range arrayVar {
27         fmt.Println(strconv.Itoa(idx) + " - " + element)
28     }
29     fmt.Println(strings.Join(arrayVar, ""))
30 }
```

Package declaration

Imports

Variable declarations

Repeating strings

Formatted printing

Array declaration / initialization

Appending to arrays

Looping over arrays

Joining strings

# 0X11 HELLO WORLD

- Filename: hello.go
- go run hello.go for “interpreted mode”
- go build hello.go to compile
- go get to install dependencies

The image features a dark blue background with white decorative elements resembling circuit board traces and nodes. These elements are located in the four corners of the frame. The central text is in a bold, white, sans-serif font.

# OX20 NETWORK PROGRAMMING

# 0X20 SIMPLE TCP SCANNER

- Basic network tool
- Full TCP Handshake
- Open connection to each port
- If a connection is established, the port is treated as open
- Concurrency can be added easily

# 0X21 EXECUTING SHELL COMMANDS

- Standard library: `os/exec`
- <https://golang.org/pkg/os/exec/>
- Commands are passed as array
- Arguments and values must be passed separately for commands to work correctly
- Environment variables can be passed via array `cmd.Env`

# 0X21 SIMPLE REVERSE SHELL

- Remote shell, that connects back to a server
- Runs `/bin/bash` on successful connection
- Provides remote access to compromised system
- Easier to bypass firewalls
- In Go:
  - Open socket
  - Execute `/bin/bash`
  - copy `stdout/stdin` of the shell to the socket



# 0X23 REMOTE BUFFER OVERFLOW EXPLOIT

- Buffer overflow are not that common today
- Still good for examples and demonstration
- Step by step walkthrough of exploiting a remote buffer overflow in vulnserver with Go
- vulnserver: <https://github.com/stephenbradshaw/vulnserver>

The image features a dark blue background with white, stylized circuit board traces in the corners. These traces consist of straight lines of varying lengths and angles, ending in small white circles, resembling a network or data flow diagram. The traces are located in the top-left, top-right, bottom-left, and bottom-right corners, framing the central text.

# 0X30 WEB HACKING WITH GO

# 0X31 HTTP CLIENT

```
1 package main
2
3 import (
4     "fmt"
5     "io/ioutil"
6     "log"
7     "net/http"
8 )
9
10 func main() {
11     url := "https://google.de"
12     res, err := http.Get(url)
13     if err != nil {
14         log.Fatal(err)
15     }
16     defer res.Body.Close()
17     body, err := ioutil.ReadAll(res.Body)
18     fmt.Println(string(body))
19 }
20
```



0X32 HTTP BASIC AUTH

DEMO



# 0X33 CLONING CEWL

- Commonly used tool to crawl websites
- Generates dictionaries for offline and online cracking
- Written in Ruby
- It's nice, but it's slow
- Latest version broken due to dependencies



0X33 CLONING CEWL

DEMO



The image features a dark blue background with white decorative circuit-like lines in the corners. These lines consist of straight paths that branch out and terminate in small circles, resembling a stylized PCB or network diagram. The lines are positioned in the top-left, top-right, bottom-left, and bottom-right corners, framing the central text.

# 0X40 WINDOWS API AND POST EXPLOITATION

# 0X41 ACCESSING THE WINDOWS API

- Standard library: `sys/windows`
- (Linux only) Must installed via `go get golang.org/x/sys/windows`
- Many syscalls are implemented as part of the library
- Can also load arbitrary DLLs to lookup functions
  - `(Must)LoadDLL`
  - `LazyDLL(System)`





# 0X41 ACCESSING THE WINDOWS API

DEMO



# 0X42 READING REGISTRY ENTRIES

- Standard library: `sys/windows/registry`
- (Linux only) Must installed via `go get`  
`golang.org/x/sys/windows/registry`
- Registry Keys are treated as files
- Perfect for post exploitation on windows systems
  - AlwaysInstallElevated
  - Service Binaries



# 0X42 READING REGISTRY ENTRIES

DEMO



# 0X43 USING WMI

- Not in the standard library, but available at:  
<https://github.com/StackExchange/wmi>
- Install `go get -u github.com/StackExchange/wmi`
- Interfaces with the local wmi service (currently no remote support)
- Can be used to script post-exploitation enumeration
- Alternative to powershell/python, as no dependencies are required on the target



0X43 USING WMI

DEMO



# 0X43 ENCRYPTED SHELLCODE INJECTOR

- AV Detection can be a massive “put back” during a pentest engagement
- Solutions exist, but evasion can be difficult
- Stubs are mostly known to AV vendors
- Solution is based on work from the veil framework
- Makes use of the win32-API to inject shellcode into the running process
- Includes server to deliver executables directly over http
- Planned features: migration / foreign process injection, process hollowing, user agent parsing
- Open Source (soon @ <https://github.com/kevin-ott/meeseeks>)

# 0X43 ENCRYPTED SHELLCODE INJECTOR

Build shellcode with  
msfvenom

Encrypt  
shellcode with  
AES256

Write  
encrypted  
shellcode to  
template

„go build“ the  
executable



# 0X43 ENCRYPTED SHELLCODE INJECTOR



## DEMO



The image features a dark blue background with white decorative elements resembling circuit board traces and nodes. These elements are located in the four corners of the frame. The central text is white and reads "OX50 WRAPPING UP".

# OX50 WRAPPING UP

# 0X51 TAKE AWAYS

- Go is a great language for pentesting and offensive tasks
- It's best suited for tools, not for PoCs
- It is not (yet) ready to replace Python, Ruby, Perl... in this domain
- Addition to the existing toolchain
- **Contribute!**

# 0X52 CODE

- Code

- <https://github.com/shellhunter/offensive-go> (soon™)
- <https://github.com/shellhunter/meeseeks> (soon™)
- <https://github.com/shellhunter/gocewl> (published)

# 0X53 FURTHER READING (BOOKS)

- The Go Programming Language
- Blackhat Go (Available for pre-order, August 2018)

The image features a dark blue background with white, stylized circuit board traces in the corners. These traces consist of straight lines of varying lengths and angles, ending in small white circles, resembling electronic components or nodes on a circuit. The traces are located in the top-left, top-right, bottom-left, and bottom-right corners, framing the central text.

QUESTIONS?

The background is a dark blue gradient. In the corners, there are decorative white line-art elements resembling circuit traces or a network diagram, with small circles at the end of the lines.

# THANKS!

TWITTER: [@KEVIN0X90](#)

GITHUB: [GITHUB.COM/SHELLHUNTER](#)