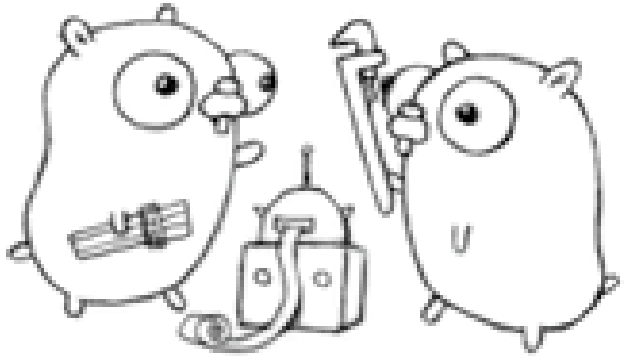
3, 2, 1 GO!

From Noon to Pro with Golang!
iDigitalFlame
BSides Delaware 2019



Warning! Learning Ahead!

- What is this workshop?
  - Intro to Golang for everyone!
- What will we do?
  - Learn the basics to the advanced in Golang
- Asking for Help?
  - Feel fee during the workshop!
- What is Expected?
  - Learning?
  - Fun?
  - New passion for programming?
  - New projects?

whoami

- @iDigitalFlame
- Malware Author, Gamer, Gadget Tinkerer and Programmer
- Works for Booz Allen’s Dark Labs
  - Research and Development
  - Red Teaming / APT Scenario Reenactments
- ProsVJoes Gold Team
- BSides Delaware Staff
- Loves Golang!

What is Golang?

- C-Style programming language invented by Google.
- Compiled language
- Written to be easy to use and powerful
  - Enforces a very “clean” writing style
- Influences by Python and C
  - You’ll see some very Python like syntax shortly!
- 10 years old! (As of this Weekend 11/9!)

Why Golang?

- Generates statically compiled binaries
  - No need for additional dependencies
- Memory management is “Simple”
  - No need to free memory
  - There are some caveats to this
- Advanced features are built in
- Ability to be cross compiled
  - Architecture
  - Operating System

Cool Capabilities & Features

- Packages like Python
- Simple control flow
- Lightweight
  - Can run on the web
- Great documentation
- Go is written in Go
- Can compile into C
  - Can also use C in Go code
  - Can be used in C libraries
- Additional features
  - TinyGo
  - Go Playground (https://play.golang.org)

Lab Setup

Prepare for Action!

How to setup your Lab!

- Editor
  - Visual Studio Codium (VSCodium)
    - http://l.idfla.me/vsc
  - Visual Studio Code is fine, but VSCodium is recommended
    - VSCodium is the “more free” and “less telemetry” version of VSCode.
  - Golang Plugin for VSCodium / Visual Studio Code
    - Extensions (Block Icon) -> Search “Go” -> Install -> Reload
- Windows 7 or 10 Host or VM (Optional for now)
- Golang
  - http://l.idfla.me/go
- Workshop
  - http://l.idfla.me/321go

First Steps!

- Building Go source is simple
  - **go build –o <output> <gofile>**
- Building our first Go program
  - Get a command prompt into the “lab0” directory.
  - Run “go build hello.go”
    - The output will be “hello.exe” (Just “hello” in Linux)

First Steps: Main

- “func main” is the primary execution point
  - Where your program begins.
- The “main function” is required for every Go build
  - Except libraries

First Steps: Declarations

- There are two ways to create (declare) variables in Go
  - Standard
  - Short / “Quick”
- Standard assignments are similar to C
  - var <name> <type>
  - Denotes the type of variable
  - Defaults to the default value if not assigned
- Standard assignments support setting
  - var <name> <type> = <value>
  - IDEs might complain about this.

First Steps: Declarations

- Short Assignments
  - Quick way to creating new variables
  - <name> := <value>
- Uses the ":=" operator (colon and equals)
- Assumes the value type
  - Defaults unless casted
  - Which can be dangerous!
- Supports the Python "_" variable
  - Discards the value

First Steps: Variables

- Variables in Go must be used!
  - Compiler complains about unused variables
  - Will not compile!
- Variables (and many other names) in Go must follow
  - Cannot start with a number
  - Should be in “camelCase” format
    - This is more of a recommendation
- Case of the first letter determines access
  - Uppercase = Public
  - Lowercase = private (more like protected)

Variable Types

- Non Numbers / Floating
  - bool – Simple true / false
  - float32 / float64 – Floating point number
    - 32 has less precision (only 4 bits), compared to 64
- Signed Numbers – Can be Negative
  - int, int64 – Takes up 64 bits
  - int8 – Takes up 8 bits
  - int16 – Takes up 16 bits
  - int32 – Takes up 32 bits

Variable Types

- Unsigned Numbers – Cannot be Negative
  - uint, uint64 – Takes up 64 bits
  - uint8 – Takes up 8 bits
  - uint16 – Takes up 16 bits
  - uint32 – Takes up 32 bits
- Unsigned Numbers can be larger than Signed
- int and uint are usually aliases to int64 and uint64
  - Smaller architectures may be only int32 and uint32!

Variable Types

- Signed:
  - int8 -128 – 128
  - int16 -32,768 – 32,768
  - int32 -2,147,483,648 – 2,147,483,648
  - int64 -9,223,372,036,854,775,808 – 9,223,372,036,854,775,808
- Unsigned:
  - uint8 0 – 255
  - uint16 0 – 65,535
  - uint32 0 – 4,294,967,295
  - uint64 0 – 18,446,744,073,709,551,615

Variable Types

- Strings!
  - string
- Aliases
  - byte = uint8
  - rune = uint32
    - This represents a “char” (character)

Variable Defaults

- If not specified, variables will be set to their default
  - Most cases are with Standard Assignment
- bool defaults to false
- Numbers / Floats
  - All default to 0 (zero)
- Strings default to empty (“”)
- There’s more than this, but well get to that soon..

Imports and Packages

- Imports are how to add extra functions to your code
- Act more like C imports
- Four types of imports
  - Internal: “fmt” or “os/exec”
  - External: “github.com/iDigitalFlame/logx/logx”
  - Named: log “github.com/iDigitalFlame/logx/logx”
  - Ignored: _ “github.com/iDigitalFlame/logx/logx”

Imports and Packages

- Multiple “import” statements can be used
  - Simpler format is in an “import” block
    - Surrounded by parenthesis “(“ “)”
- Stored at the top, under “package”
- Package
  - The “package” line must be the first line in a Go file
    - There is some exceptions to this
  - Denotes the package name
    - Usually same as the folder that contains the Go file
  - “main” is the default package

Common Imports

- fmt – Printing and Formatting
  - fmt.Printf and fmt.Println will be common
- os – OS / Device access (Opening Files)
- net - Networking
- strconv – Parsing or Printing strings
- strings – String manipulation and Buffers
- Bytes – Byte manipulation and Buffers
- io – Reading and Writing to / from resources
- ioutil – IO Utilities for ease of use

Printf and Println

- There are two ways of printing to console
- Printf vs Println
  - Printf takes a format string
    - Println does not
  - Println automatically adds a newline
    - Printf expects you to do this

Printf

- Printf takes a format string
  - The format string defines where and what is printed
- There are some specific printing types
  - %d, %u, %n - Numbers
  - %f – Floating Points
  - %s – Strings
  - %c – Characters
  - %p – Pointers
  - %v, %+v – Verbose
  - %w – Errors (Golang)

Lab 0 & Lab 1

Here’s where it gets Crazy!

Pointers

- Pointers are a “reference” to a variable’s address in memory
- **Not** the direct value
- Can be used to modify the value of the address it pointes to
- Useful to pass parameters to be changed
- Can sometimes save memory usage.

Pointers

- Pointers can be created using the “&” symbol
  - Returns the variable type pointer
- Declared with “*” in front of the type
- Accessing the value of the pointer is done with “*”
  - Assignments to a pointer with “*” will change the value
- Pointers do not need to be freed

Lab 2

Complex Variables

- Golang has other more complex variables
  - array
  - slice
  - map
- These can be “nil” (NULL in Golang)
  - This is also their default value (except for arrays!)
- Maps and Slices are initialized differently than other variables

Arrays

- Similar to a bookshelf of variables
- Size cannot be changed at runtime
  - Fixed size
- Arrays are created similar to standard variables
  - var <name> [size]<type>

Slices

- Similar to Arrays
- Contains three objects
  - Pointer to the underlying array
  - How many items are stored (length)
  - Size of the underlying array (capacity)
- Preferred over Arrays
- Can be resized at any time!
- Capacity can be specified to prevent unneeded memory allocation
- Has multiple methods of creation

Slice Creation

- Slices can be created using an explicit declaration
  - MySlice := []string{“one”, “two’’, “three”}
- Or use the built in “make” function
  - MySlice := make([]string, <length>)
  - MySlice := make([]string, <length>, <capacity>)
- The built in “append” function can add items to a slice
  - append can also create a slice if the source is nil!
  - MySlice = append(MySlice, “four”)

Maps

- Maps allow for referencing a value with a key
- Similar to slices, maps have two creation methods
  - Explicit: MyMap := map[string]string{“1”: “one”, “2”: “two”}
  - Using make: MyMap := make(map[string]string)
- Referencing values via key is done with “[]”
  - MyMap[“1”]
- Deleting values via key is done with the built in “delete”
  - delete(MyMap, “1”)
- Maps return the default value’s value if they do not contain an entry for that key

Maps

- What happens if a default value is a valid value?
  - 0 (zero) is a default value for Numbers

Maps

- Maps have a special syntax that can be used to check if an key exists
- Retrieving values can happen in two ways
  - value := MyMap[“key”]
  - Value, ok := MyMap[“key”]
- The second value is a bool
  - Set to true if the key exists in the map!
- Example that just checks for existence
  - _, ok := MyMap[“key”]

Lab 3 & Lab 4

Control Flow

- Logic and testing
  - If / Else
  - For
  - Switch
  - Select
- Golang does not have while!

Control Flow: If

- If statements allow for comparison
  - Do not require parenthesis “(“ “)”
- First brace must be in line with the if statement
- Expressions and variable declarations are allowed in if
  - Only available in that block
  - Require a semicolon if used
- Braces for Else must be in line also

Control Flow: For

- For in Golang is very flexible
- For have many types of configurations
  - Standard for loop
  - Forever loop
  - “While” loop
  - “Range” loop
  - “Foreach” loop
- Expressions are allowed inside for loop statements
- Braces must be in line!

Control Flow: Switch

- Similar to C or Java switches
- Multiple entries on each line
- Does not fallthrough
  - Can fallthrough using the “fallthrough” keyword
- Does not need break
- Expressions allowed in the switch statement