ANATOMY OF A GOPHER || BINARY ANALYSIS OF GO BINARIES

HEX0PUNK / ALEX USECHE

WHO IS THIS GUY?

- hex0punk
- Father & husband
- Senior AppSec Consultant at nVisium
- Fan of Go and Rust
- Likes playing with binaries

WHAT ARE WE DOING?

- Learning about what makes go binaries different than C and C++ binaries
- Identifying techniques for recognizing and conducting analysis of go binaries.
- Tips for finding vulnerabilities in go binaries.
- Identifying common patterns found in go binaries
- Learning about protections that can be added to go binaries

ONCE UPON A TIME

- Found go binaries during an IoT assessment
- Wait, this isn't C or C++!
- This is Go!
- Now what?

THE GO ASSEMBLER

- The go compiler is based on the plan9 compiler
- Semi-abstract instruction set
- Pseudo-Assembly
- Not a direct representation of the underlying machine (i.e. a MOV may be a LD)
- It also introduces a set of pseudo registers

GO TOOL OBJDUMP

```
go tool objdump -s <func> <bin>
```

```
TEXT main.main(SB) /Users/alexuseche/Projects/CTF-Source/go-bin/main.go
  main.go:11        0x4deaa0        64488b0c25f8ffffff      MOVQ FS:0xfffffff8, CX
  main.go:11        0x4deaa9        488d4424b0              LEAQ -0x50(SP), AX
  main.go:11        0x4deaae        483b4110                CMPQ 0x10(CX), AX
  main.go:11        0x4deab2        0f8650020000            JBE 0x4ded08
  main.go:11        0x4deab8        4881ecd0000000          SUBQ $0xd0, SP
  main.go:11        0x4deabf        4889ac24c8000000        MOVQ BP, 0xc8(SP)
  main.go:11        0x4deac7        488dac24c8000000        LEAQ 0xc8(SP), BP
  main.go:13        0x4deacf        0f57c0                  XORPS X0, X0
  main.go:13        0x4dead2        0f118424b8000000        MOVUPS X0, 0xb8(SP)
```

RADARE2

```
      ; var int64_t var_8h @ rsp+0xc8
 ┌─> 0x004deaa0      64488b0c25f8.   mov rcx, qword fs:[0xfffffffffffffff8]
 ╎   0x004deaa9      488d4424b0      lea rax, [rsp - 0x50]
 ╎   0x004deaae      483b4110        cmp rax, qword [rcx + 0x10]
┌──< 0x004deab2      0f8650020000    jbe 0x4ded08
│╎   0x004deab8      4881ecd00000.   sub rsp, 0xd0
│╎   0x004deabf      4889ac24c800.   mov qword [var_8h], rbp
│╎   0x004deac7      488dac24c800.   lea rbp, [var_8h]
│╎   0x004deacf      0f57c0          xorps xmm0, xmm0
│╎   0x004dead2      0f118424b800.   movups xmmword [var_18h], xmm0
```

go tool objdump vs. radare2

- Use both!
- Differences are easy to understand:
- A big advantage of using `go tool objdump` is that you get line numbers in code. This can be help you group instructions by operations

THE GO ASSEMBLER

DEFAULT PROTECTIONS

```
canary   false
class    ELF64
crypto   false
endian   little
havecode true
laddr    0x0
lang     go
linenum  true
lsyms    true
machine  AMD x86-
maxopsz  16
minopsz  1
nx       true
os       linux
pcalign  0
pic      false
relocs   true
rpath    NONE
sanitiz  false
static   true
stripped false
subsys   linux
```

true for MachO

Got ROP?

- Stack Protection: `export CGO_LDFLAGS='-fstack-protector'`
- Stripped: `GOOS=linux go build -ldflags="-s -w"`
- PIE: `export GOFLAGS='-buildmode=pie'`
- Use a tool like UPX (https://upx.github.io/) to strip function names and reduce size

SEARCHING FOR STRINGS

```
rabin2 -zz go-bin-goo | grep 1.__TEXT.__rodata | grep intruder
```

```
alex.useche@Alexs-MacBook-Pro-7  ~/Projects/CTF-Source/go-bin   master ●  rabin2
-zz go-bin-goo | grep 1.__TEXT.__rodata | grep intruder

33526 0x00135e7f 0x01135e7f 2040 2040 1.__TEXT.__rodata              ascii   ffdirecto
ry not emptydisc quota exceededfile already closedfile already existsfile does not exi
stm not found in allmmarking free objectmarkroot: bad indexmspan.sweep: state=no STREA
M resourcesnotesleep not on g0nwait > work.nprocsoperation timed outpanic during mallo
cpanic during panic\npanic holding lockspanicwrap: no ( in panicwrap: no ) in previous
 owner diedreflect.Value.Fieldreflect.Value.Floatreflect.Value.Indexreflect.Value.IsNi
lreflect.Value.Sliceruntime: insert t= runtime: pcdata is runtime: preempt g0semaRoot
rotateLeftskip this directorystopm holding lockstoo many open filesunknown wait reason
unsupported messagezero length segment markroot jobs done\n to unallocated span, Recur
sionDesired: /usr/share/zoneinfo/37252902984619140625EMULTIHOP (Reserved)Egyptian_Hier
oglyphsEnter the password: IDS_Trinary_OperatorIntruder! intruder!\nMeroitic_Hieroglyp
hsSIGALRM: alarm clockSIGTERM: terminationSTREAM ioctl timeoutSeek: invalid offsetSeek
: invalid whenceTerminal_Punctuationauthentication errorbad defer size classbad system
 page sizebad use of bucket.bpbad use of bucket.mpchan send (nil chan)close of nil cha
nnelfloating point errorforcegc: phase errorgc_trigger underflowgo of nil func valuego
park: bad g statusinvalid DNS responsemSpanList.insertBackmalloc during signalnon-empt
y swept listnotetsleep not on g0p mcache not flushedpacer: assist ratio=preempt off re
ason: reflect.makeFuncStubruntime: casgstatus runtime: double waitruntime: unknown pc
semaRoot rotateRightsysctl kern.hostnametime: invalid numbertrace: out of memoryunexpe
cted IP lengthunexpected network: wirep: already in goworkbuf is not emptywrite of Go
pointer  gp.gcscanvalid=true\n of unexported method previous allocCount=18626451492309
5703125931322574615478515625Anatolian_HieroglyphsInscriptional_PahlaviOther_Grapheme_E
xtend[+] Checking password_cgo_unsetenv missingbad type in compare: block device requi
redbufio: negative countcheckdead: runnable gconcurrent map writesdefer on system stac
kdevice not configuredfindrunnable: wrong pillegal byte s
```

SEARCHING FOR STRINGS

- Go does not store null terminated strings.
- Strings are clumped together, while keeping a separate table with length information.
- This can make it difficult to look for string cross-references
- We can use a projects like https://github.com/carvesystems/gostringsr2 to parse strings
- When working with MachO binaries, you'd have to list strings from .rodata or entire binary
  - `rabin2 -zz <binary>`
  - `izz` using r2

SEARCHING FOR FUNCTIONS

- Most of the times, functions are easy to find, even in stripped binaries.
- Functions follow with the following naming conventions:

```
package.function
package.receiver struct.function
Package.__receiver struct__.function
```

```
func (m *Modules) InitProcessors(mods []base.ModuleConfig) error
```

```
sym.local.runtime.modulesinit
sym.local.runtime.moduledataverify
sym.local.runtime.moduledataverify1
sym.local.runtime.findmoduledatap
sym.local.github.com_DharmaOfCode_gorp_modules.__Modules_.InitProcessors
sym.local.github.com_DharmaOfCode_gorp_modules.setModuleOption
sym.local.github.com_DharmaOfCode_gorp_modules.printOptions
sym.local.github.com_DharmaOfCode_gorp_modules.__Modules_.GetProcessor
sym.local.github.com_DharmaOfCode_gorp_modules.__Modules_.InitInspectors
```

THE EASY PART

FINDING MAIN

```
nm go-bin-elf | grep main
 5f6ec0 D main..inittask
 4dee40 T main.authenticate
 54fa60 R main.authenticate.stkobj
 4dfae0 T main.check
 54cd30 R main.check.stkobj
 4df7a0 T main.checkName
 54cd50 R main.checkName.stkobj
 4df8a0 T main.checkPassword
 54cd70 R main.checkPassword.stkobj
 4ded20 T main.handleConnection
 54cd90 R main.handleConnection.stkobj
 4df620 T main.jackpot
 4deaa0 T main.main
```

```
main.main
main.handleConnection
main.authenticate
main.jackpot
main.checkName
main.checkPassword
main.check
```

GO STACKS

GO FUNCTION PROLOGUE

- Go routines have small stacks by default (2 kibibyte = 1024 bytes stack)
- Many goroutines will calls `morestack` to grow the stack (in powers of 2) as needed using **stack copying**
- **This is called because go can't be sure the function will outgrow the stack (i.e. recursive functions) given non-deterministic goroutines**
- When this occurs, stack grows, pointers in the stack are updated.
- Additionally, each function compares its stack pointer against `g->stackguard` to check for overflow.

```
mov rcx, qword fs:[0xfffffffffffffff8]
cmp rsp, qword [rcx + 0x10]
jbe 0x4df88b
sub rsp, 0x58
mov qword [var_8h], rbp
```

```
mov byte [arg_18h], 0
mov rbp, qword [var_8h]
add rsp, 0x58
ret
Name @ 0x4df7ad
call sym.runtime.morestack_noctxt
jmp sym.main.checkName
```

GO FUNCTION PROLOGUE

OR FINDING OPPORTUNITIES TO OVERFLOW THE STACK

- Developers call tell go to skip this check to test performance issues with pragma `//go:nosplit`
- Some library functions may call `//go:nosplit`
- Look for opportunities to corrupt the stack
- Callers pass arguments in the stack

```go
//go:nosplit
func checkName(name string) bool{
    fmt.Println( a: "[+] Checking name")

    if len(name) != 12{
        return false
    }
```

```
sym.main.checkName (int64_t arg_8h, int64_t arg_10h, int64_t
    ; var int64_t var_50h @ rsp+0x8
    ; var int64_t var_48h @ rsp+0x10
    ; var int64_t var_40h @ rsp+0x18
    ; var int64_t var_38h @ rsp+0x20
    ; var int64_t var_18h @ rsp+0x40
    ; var int64_t var_10h @ rsp+0x48
    ; var int64_t var_8h @ rsp+0x50
    ; arg int64_t arg_8h @ rsp+0x60
    ; arg int64_t arg_10h @ rsp+0x68
    ; arg int64_t arg_18h @ rsp+0x70
    0x010e72a0      4883ec58      sub rsp, 0x58
    0x010e72a4      48896c2450    mov qword [var_8h], rbp
    0x010e72a9      488d6c2450    lea rbp, [var_8h]
```

CONVENTIONS

ARGUMENTS AND RETURN VALUES

- Return values are placed in the stack
- A opposed to C where return values are placed in registers
- Arguments are also moved to the stack rather than registers.

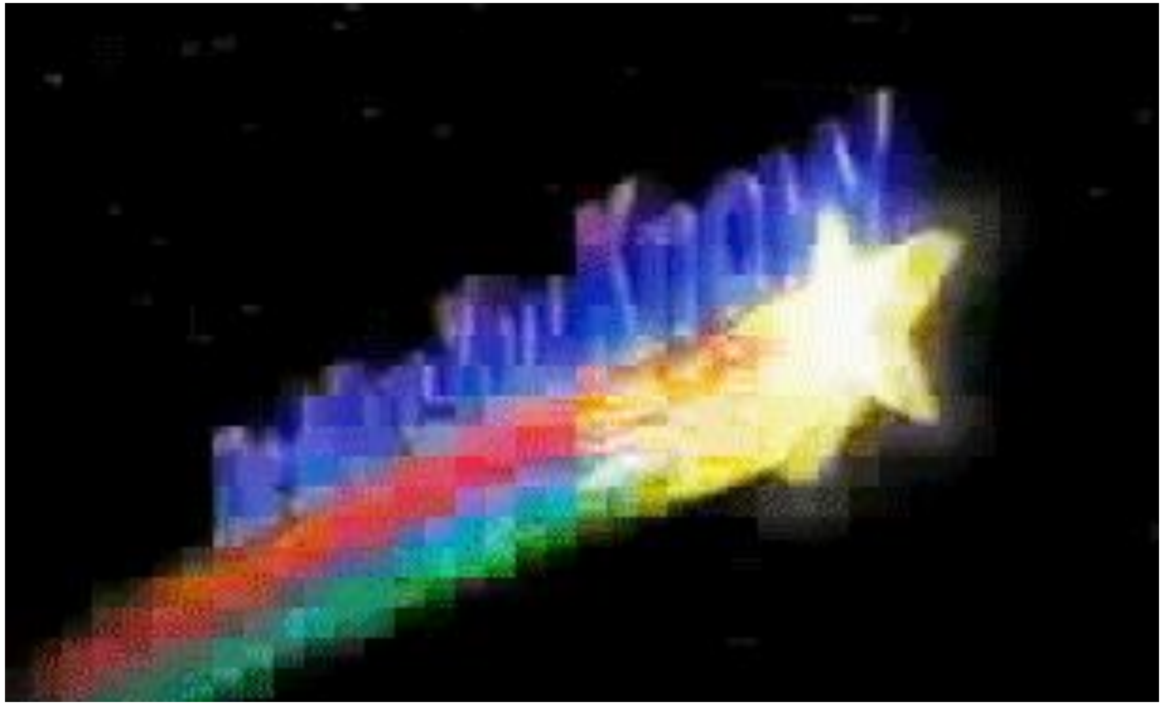
Placing return values in the stack

```
mov byte [arg_18h], 0
mov rbp, qword [var_8h]
add rsp, 0x58
ret
```

Passing arguments in the stack

```
e8026dfcff        call sym.strings.TrimRight
488b442420        mov rax, qword [var_240h]
488b4c2428        mov rcx, qword [var_238h]
48890424          mov qword [rsp], rax
48894c2408        mov qword [var_258h], rcx
e89a070000        call sym.main.checkName
```

IT'S ALL ABOUT THE GO LIBRARY FUNCTIONS



USE GO LIBRARIES AS A GUIDE TO UNDERSTAND THE BINARY

- Understanding go internal libraries can significantly help us understand what is going on the assembly code
- Read the go docs!

CALLING A GOROUTINE

```go
go student.DerefCopyMethod(&students[0])
core.RefCopy(&student, students)
```

```
MOVL $0x10, 0(SP)
LEAQ go.func.*+95(SB), AX
MOVQ AX, 0x8(SP)
MOVQ 0x58(SP), AX
MOVQ AX, 0x10(SP)
MOVQ 0x68(SP), CX
MOVQ CX, 0x18(SP)
CALL runtime.newproc(SB)
```

DEREFENCING RECEIVER

```go
func (s *Student) DerefCopyMethod(student *Student){
    *s = *student
}
```

```go
//go:nosplit
func typedmemmove(typ *_type, dst, src unsafe.Pointer) {
	if dst == src {
		return
	}
	if typ.ptrdata != 0 {
		bulkBarrierPreWrite(uintptr(dst), uintptr(src), typ.size)
	}
	// There's a race here: if some other goroutine can write to
	// src, it may change some pointer in src after we've
	// performed the write barrier but before we perform the
	// memory copy. This safe because the write performed by that
	// other goroutine must also be accompanied by a write
	// barrier, so at worst we've unnecessarily greyed the old
	// pointer that was in src.
	memmove(dst, src, typ.size)
	if writeBarrier.cgo {
		cgoCheckMemmove(typ, dst, src, 0, typ.size)
	}
}
```

```
MOVQ 0x30(SP), CX
TESTB AL, 0(CX)
CMPL $0x0, runtime.writeBarrier(SB)
JNE 0x109d584
MOVQ 0(CX), DX
MOVQ DX, 0(AX)
MOVUPS 0x8(CX), X0
MOVUPS X0, 0x8(AX)
MOVUPS 0x18(CX), X0
MOVUPS X0, 0x18(AX)
MOVUPS 0x28(CX), X0
MOVUPS X0, 0x28(AX)
MOVUPS 0x38(CX), X0
MOVUPS X0, 0x38(AX)
MOVQ 0x18(SP), BP
ADDQ $0x20, SP
RET
LEAQ runtime.rodata+159488(SB), DX
MOVQ DX, 0(SP)
MOVQ AX, 0x8(SP)
MOVQ CX, 0x10(SP)
CALL runtime.typedmemmove(SB)
```

SO WHAT?

- We have identified a race condition
- Note `//go:nosplit` for `typedmemmove`


```
//go:nosplit
func typedmemmove(typ *_type, dst, src unsafe.Pointer) {
	if dst == src {
		return
	}
	if typ.ptrdata != 0 {
		bulkBarrierPreWrite(uintptr(dst), uintptr(src), typ.size)
	}
	// There's a race here: if some other goroutine can write to
	// src, it may change some pointer in src after we've
	// performed the write barrier but before we perform the
	// memory copy. This safe because the write performed by that
	// other goroutine must also be accompanied by a write
	// barrier, so at worst we've unnecessarily greyed the old
	// pointer that was in src.
	memmove(dst, src, typ.size)
	if writeBarrier.cgo {
		cgoCheckMemmove(typ, dst, src, 0, typ.size)
	}
}
```

DEFER

- Function responsible for concurrency are easy to locate
- In the case of a defer call:
  - Functions are registered with `deferprocStack`
  - Check whether a panic has been caught
  - Prepare return values and call deferred functions before exiting function
  - Assure to defer if a panic is caught

```
lea rax, sym.os.__File_.Close.f ;
mov qword [var_80h], rax
mov qword [var_68h], rcx
lea rax, [var_98h]
mov qword [rsp], rax
call sym.runtime.deferprocStack
test eax, eax
jne 0xc0f90
```

```
xorps xmm0, xmm0
movups xmmword [arg_58h], xmm0
nop
call sym.runtime.deferreturn
mov rbp, qword [var_8h]
add rsp, 0x1c8
ret
```

```
call sym.runtime.deferreturn
mov rbp, qword [var_8h]
add rsp, 0x1c8
ret
```

GO INTERFACES

- Interfaces consist of:
  - A pointer to a **vtable** (which contains function pointers that point to the virtual functions)
  - A value pointer

GO ERROR HANDLING

- `error` is an an interface
- Error handling in clumsy in go
- Bugs due to unhandled errors are common
- When checking for `error != nil` we load the error vtable and error value.
- Then we test if the value is nil
- And branch depending on the result

```
s, err := ioutil.ReadFile( filename:
if err != nil{
    panic(err)
}
```

```
call sym.io_ioutil.ReadFile
mov rax, qword [var_48h]
mov rcx, qword [var_40h]
mov rdx, qword [var_60h]
mov rbx, qword [var_58h]
test rax, rax
jne 0x4df789
```

SO WHAT? WELL…

CVE-2018-1002105 Detail

Current Description

In all Kubernetes versions prior to v1.10.11, v1.11.5, and v1.12.3, $\boxed{\text{incorrect handling of error responses}}$ to proxied upgrade requests in the kube-apiserver allowed specially crafted requests to establish a connection through the Kubernetes API server to backend servers, then send arbitrary requests over the same connection directly to the backend, authenticated with the Kubernetes API server's TLS credentials used to establish the backend connection.

WHAT ELSE SHOULD I LOOK FOR?

```
LEAQ 0x30(SP), AX
MOVQ AX, 0x18(SP)
MOVQ main._cgo_8c2b83f3b113_Cfunc_memcpy(SB), AX
MOVQ AX, 0(SP)
```

- cgo functions!
- Hacking like its 1995(98)(2000)

EL FIN

- Go is everywhere
- We are likely to see see more go in IoT with projects like TinyGo
- Fancy, yet exploitable under the right conditions
- Look for programmer errors (i.e. error handling)
- Understand stack handling
- Look for cgo usage
- Leverage go tools
- RTFM

- For developers:
  - Test your code with gosec, govet
  - Fuzz it with go-fuzz

QUESTIONS?