radare2 & gophers

Analysis of Go Binaries with radare2

hexes and punks

- hex0punk / Alex Useche
- Security Engineer @ Trail of Bits
- Previously worked as an application security penetration tester, focusing on IoT and Mobile
- Software dev before that
- Go aficionado, Rust fan
- Messes with binaries, code
- Father and husband

what is this about?

What makes a Go binary different that a C or C++ binary

Identifying techniques for conducting binary analysis of go binaries

Identifying common concurrency patterns in go binaries

Using go tools to make your r2 analysis more effective

Some knowledge of go is assumed

the go assembler

Based on input style of the Plan 9 assembler

Works on a semi-abstract instruction set

Output may not always be a direct representation of the machine code

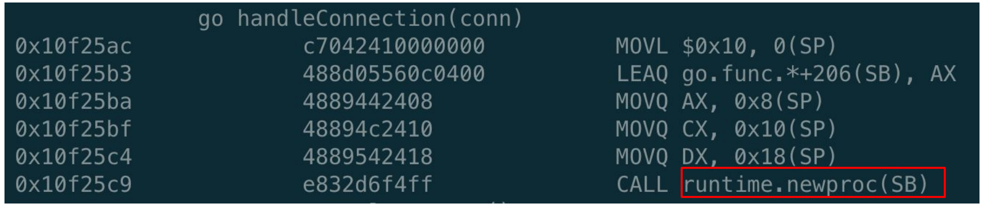
For instance, a MOV may actually be a LD

It introduces a set of pseudo registers

Introduces a set of directives used by the garbage collector

go pseudo-registers

FP -> Frame Pointer (arguments and locals)
PC -> Program Counter
SB -> Static Base pointer (used to name global functions or data based on memory origin)
SP (RSP) -> Stack Pointer (top of stack)
BP (RBP) -> Base Pointer
X0 -> A full 16 byte register



```
                go handleConnection(conn)
0x10f25ac              c7042410000000          MOVL $0x10, 0(SP)
0x10f25b3              488d05560c0400          LEAQ go.func.*+206(SB), AX
0x10f25ba              4889442408              MOVQ AX, 0x8(SP)
0x10f25bf              48894c2410              MOVQ CX, 0x10(SP)
0x10f25c4              4889542418              MOVQ DX, 0x18(SP)
0x10f25c9              e832d6f4ff              CALL runtime.newproc(SB)
```

`runtime.newproc(SB)` is the function name
`runtime.newproc` as an address in memory

tooling (other than r2, of course)

Complement your RE work with `go tool objdump`

Use `-S` to print go code (line numbers)

Use `-s` to specify a function

Helpful for grouping a set of statements to LoC

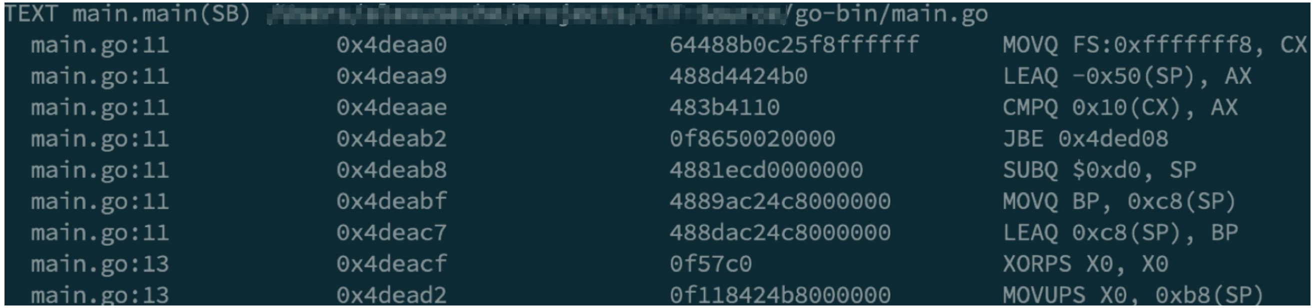
Prints left to right instructions

Not an option with stripped ELF binaries

Still works with stripped Mach-O binaries

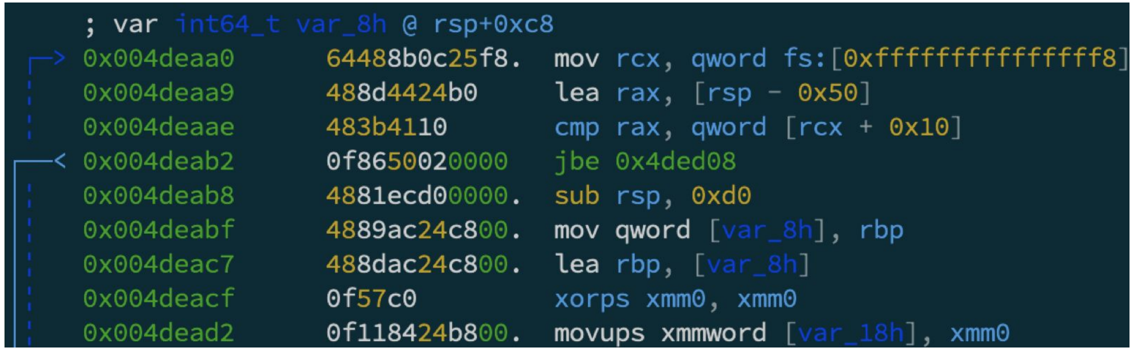
a match made in hacker heaven

```
go tool objdump -s main.main -S go-bin
```



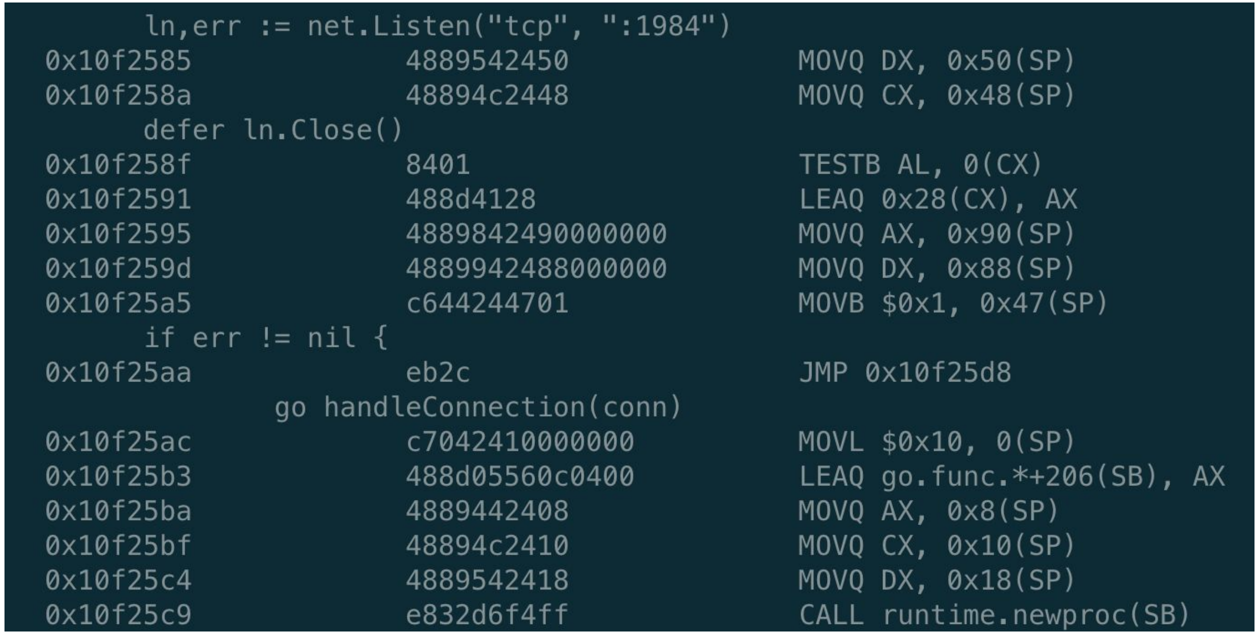
```
TEXT main.main(SB) [redacted]/go-bin/main.go
  main.go:11          0x4deaa0          64488b0c25f8ffffff      MOVQ FS:0xfffffff8, CX
  main.go:11          0x4deaa9          488d4424b0              LEAQ -0x50(SP), AX
  main.go:11          0x4deaae          483b4110                CMPQ 0x10(CX), AX
  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)
```

```
pdf@sym.main.main
```



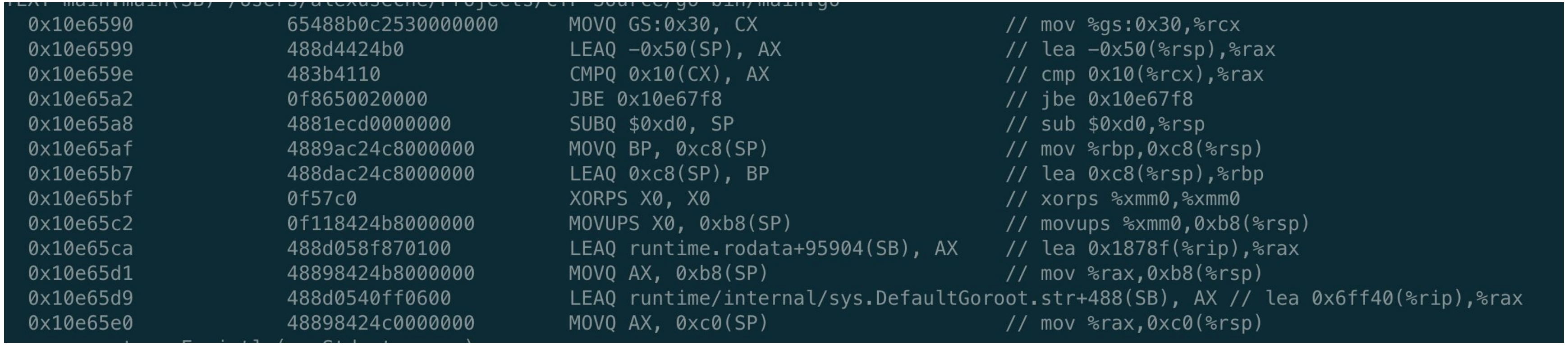
```
     ; var int64_t var_8h @ rsp+0xc8
 ┌─> 0x004deaa0      64488b0c25f8.  mov rcx, qword fs:[0xfffffffffffffff8]
 ╎   0x004deaa9      488d4424b0     lea rax, [rsp - 0x50]
 ╎   0x004deaae      483b4110       cmp rax, qword [rcx + 0x10]
┌──< 0x004deab2      0f8650020000   jbe 0x4ded08
│╎   0x004deab8      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
```

objdump in go 1.15



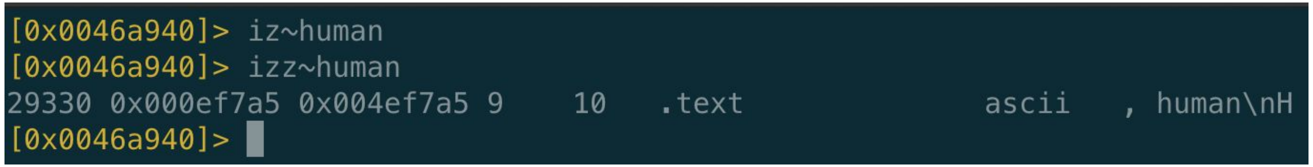
```
        ln,err := net.Listen("tcp", ":1984")
  0x10f2585             4889542450              MOVQ DX, 0x50(SP)
  0x10f258a             48894c2448              MOVQ CX, 0x48(SP)
        defer ln.Close()
  0x10f258f             8401                    TESTB AL, 0(CX)
  0x10f2591             488d4128                LEAQ 0x28(CX), AX
  0x10f2595             4889842490000000        MOVQ AX, 0x90(SP)
  0x10f259d             4889942488000000        MOVQ DX, 0x88(SP)
  0x10f25a5             c644244701              MOVB $0x1, 0x47(SP)
        if err != nil {
  0x10f25aa             eb2c                    JMP 0x10f25d8
                go handleConnection(conn)
  0x10f25ac             c7042410000000          MOVL $0x10, 0(SP)
  0x10f25b3             488d05560c0400          LEAQ go.func.*+206(SB), AX
  0x10f25ba             4889442408              MOVQ AX, 0x8(SP)
  0x10f25bf             48894c2410              MOVQ CX, 0x10(SP)
  0x10f25c4             4889542418              MOVQ DX, 0x18(SP)
  0x10f25c9             e832d6f4ff              CALL runtime.newproc(SB)
```

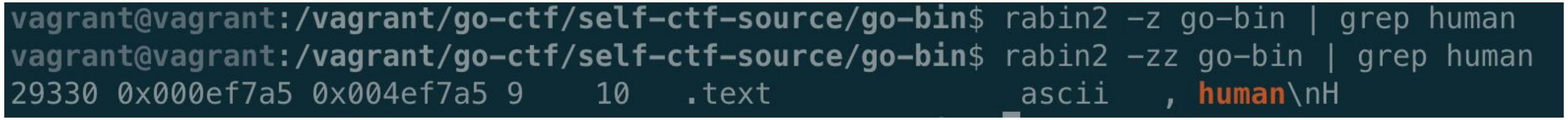
objdump in go 1.15

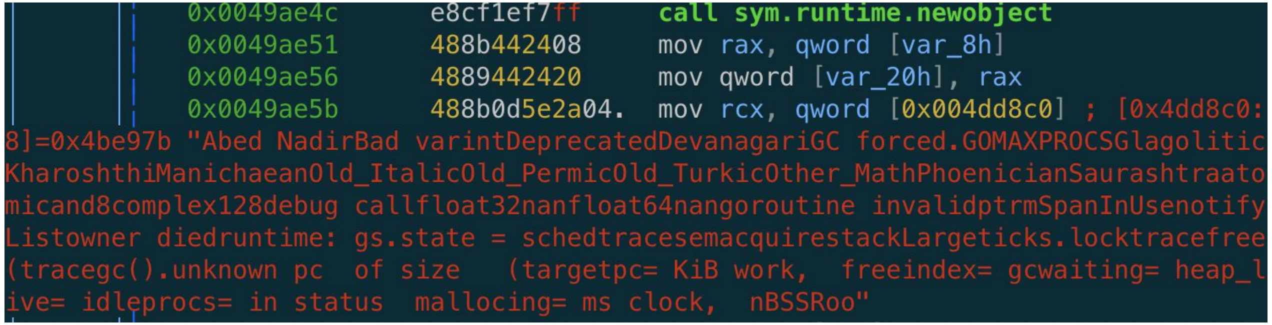


```
  0x10e6590           65488b0c2530000000      MOVQ GS:0x30, CX                            // mov %gs:0x30,%rcx
  0x10e6599           488d4424b0              LEAQ -0x50(SP), AX                          // lea -0x50(%rsp),%rax
  0x10e659e           483b4110                CMPQ 0x10(CX), AX                           // cmp 0x10(%rcx),%rax
  0x10e65a2           0f8650020000            JBE 0x10e67f8                               // jbe 0x10e67f8
  0x10e65a8           4881ecd0000000          SUBQ $0xd0, SP                              // sub $0xd0,%rsp
  0x10e65af           4889ac24c8000000        MOVQ BP, 0xc8(SP)                           // mov %rbp,0xc8(%rsp)
  0x10e65b7           488dac24c8000000        LEAQ 0xc8(SP), BP                           // lea 0xc8(%rsp),%rbp
  0x10e65bf           0f57c0                  XORPS X0, X0                                // xorps %xmm0,%xmm0
  0x10e65c2           0f118424b8000000        MOVUPS X0, 0xb8(SP)                         // movups %xmm0,0xb8(%rsp)
  0x10e65ca           488d058f870100          LEAQ runtime.rodata+95904(SB), AX           // lea 0x1878f(%rip),%rax
  0x10e65d1           48898424b8000000        MOVQ AX, 0xb8(SP)                           // mov %rax,0xb8(%rsp)
  0x10e65d9           488d0540ff0600          LEAQ runtime/internal/sys.DefaultGoroot.str+488(SB), AX // lea 0x6ff40(%rip),%rax
  0x10e65e0           48898424c0000000        MOVQ AX, 0xc0(SP)                           // mov %rax,0xc0(%rsp)
```

strings


```
[0x0046a940]> iz~human
[0x0046a940]> izz~human
29330 0x000ef7a5 0x004ef7a5 9    10   .text                  ascii   , human\nH
[0x0046a940]> 
```


```
vagrant@vagrant:/vagrant/go-ctf/self-ctf-source/go-bin$ rabin2 -z go-bin | grep human
vagrant@vagrant:/vagrant/go-ctf/self-ctf-source/go-bin$ rabin2 -zz go-bin | grep human
29330 0x000ef7a5 0x004ef7a5 9    10   .text                  ascii   , human\nH
```


```
           0x0049ae4c      e8cf1ef7ff     call sym.runtime.newobject
           0x0049ae51      488b442408     mov rax, qword [var_8h]
           0x0049ae56      4889442420     mov qword [var_20h], rax
           0x0049ae5b      488b0d5e2a04.  mov rcx, qword [0x004dd8c0] ; [0x4dd8c0:
8]=0x4be97b "Abed NadirBad varintDeprecatedDevanagariGC forced.GOMAXPROCSGlagolitic
KharoshthiManichaeanOld_ItalicOld_PermicOld_TurkicOther_MathPhoenicianSaurashtraato
micand8complex128debug callfloat32nanfloat64nangoroutine invalidptrmSpanInUsenotify
Listowner diedruntime: gs.state = schedtracesemacquirestackLargeticks.locktracefree
(tracegc().unknown pc  of size   (targetpc= KiB work,  freeindex= gcwaiting= heap_l
ive= idleprocs= in status  mallocing= ms clock,  nBSSRoo"
```

searching for strings

Go does not store null terminated strings

Clumped together in .text

Go uses a separate table with string length information

Can be difficult to find XREFs

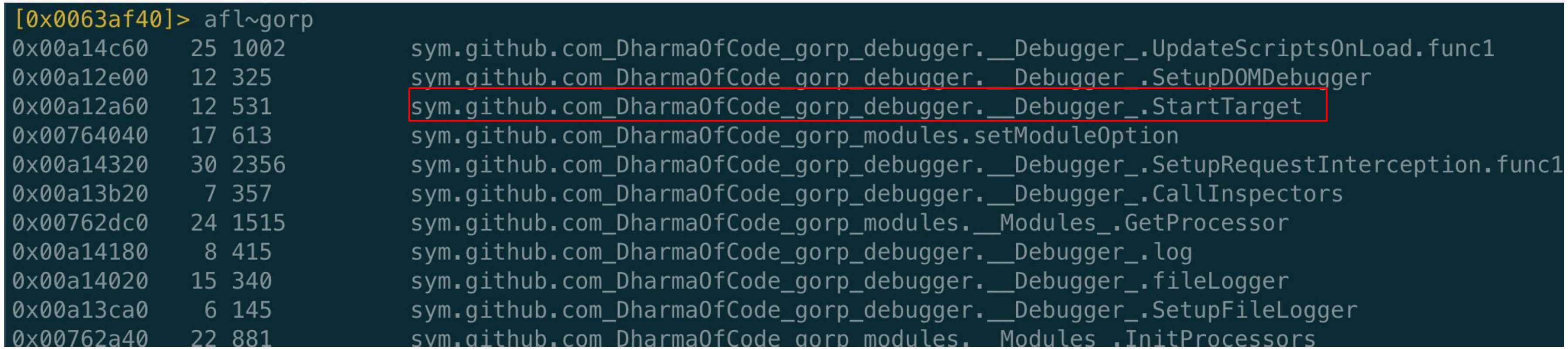
Can use https://github.com/CarveSystems/gostringsr2

Search for strings in entire binary, not just .text

functions

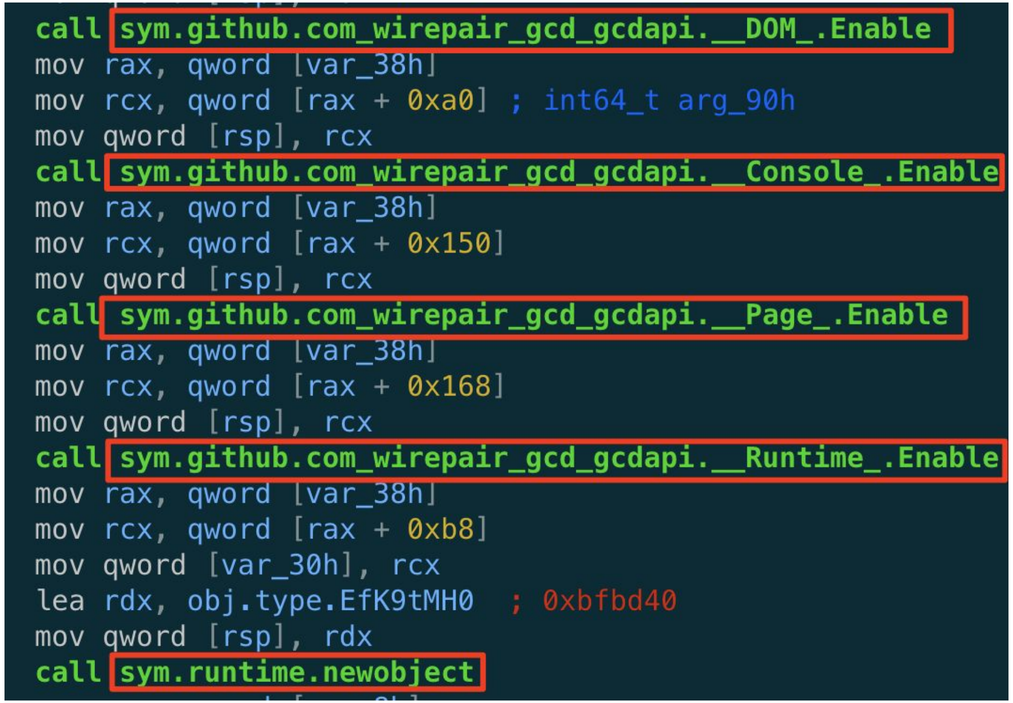
```
package.function

package.receiverStruct.method

package.__receiverStruct__.method
```

```go
package debugger

type Debugger struct {
    //..
}
func (d *Debugger) StartTarget(){
    //..
}
```



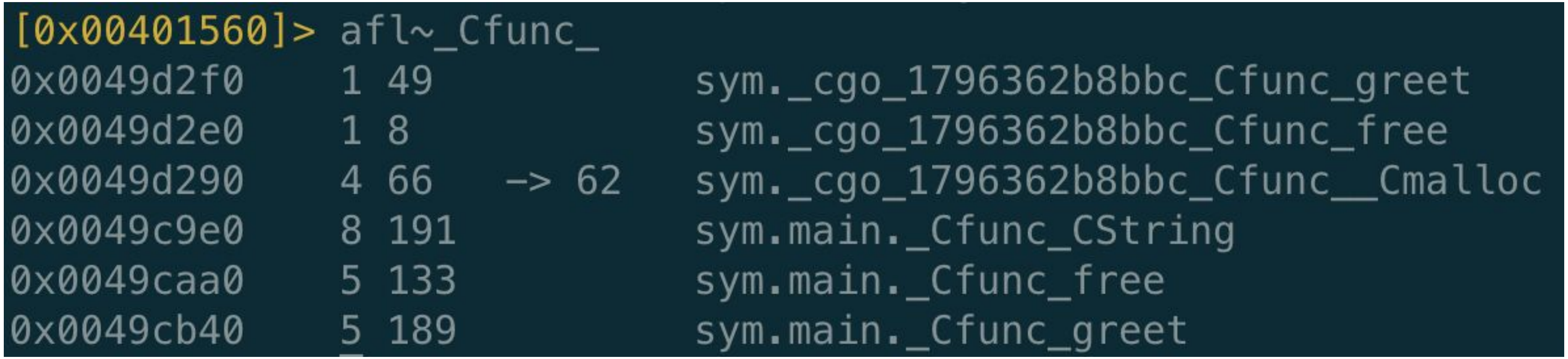
```
[0x0063af40]> afl~gorp
0x00a14c60   25 1002        sym.github.com_DharmaOfCode_gorp_debugger.__Debugger_.UpdateScriptsOnLoad.func1
0x00a12e00   12 325         sym.github.com_DharmaOfCode_gorp_debugger.__Debugger_.SetupDOMDebugger
0x00a12a60   12 531         sym.github.com_DharmaOfCode_gorp_debugger.__Debugger_.StartTarget
0x00764040   17 613         sym.github.com_DharmaOfCode_gorp_modules.setModuleOption
0x00a14320   30 2356        sym.github.com_DharmaOfCode_gorp_debugger.__Debugger_.SetupRequestInterception.func1
0x00a13b20    7 357         sym.github.com_DharmaOfCode_gorp_debugger.__Debugger_.CallInspectors
0x00762dc0   24 1515        sym.github.com_DharmaOfCode_gorp_modules.__Modules_.GetProcessor
0x00a14180    8 415         sym.github.com_DharmaOfCode_gorp_debugger.__Debugger_.log
0x00a14020   15 340         sym.github.com_DharmaOfCode_gorp_debugger.__Debugger_.fileLogger
0x00a13ca0    6 145         sym.github.com_DharmaOfCode_gorp_debugger.__Debugger_.SetupFileLogger
0x00762a40   22 881         sym.github.com_DharmaOfCode_gorp_modules.__Modules_.InitProcessors
```

function ID



```
call sym.github.com_wirepair_gcd_gcdapi.__DOM_.Enable
mov rax, qword [var_38h]
mov rcx, qword [rax + 0xa0] ; int64_t arg_90h
mov qword [rsp], rcx
call sym.github.com_wirepair_gcd_gcdapi.__Console_.Enable
mov rax, qword [var_38h]
mov rcx, qword [rax + 0x150]
mov qword [rsp], rcx
call sym.github.com_wirepair_gcd_gcdapi.__Page_.Enable
mov rax, qword [var_38h]
mov rcx, qword [rax + 0x168]
mov qword [rsp], rcx
call sym.github.com_wirepair_gcd_gcdapi.__Runtime_.Enable
mov rax, qword [var_38h]
mov rcx, qword [rax + 0xb8]
mov qword [var_30h], rcx
lea rdx, obj.type.EfK9tMH0   ; 0xbfbd40
mov qword [rsp], rdx
call sym.runtime.newobject
```

cgo functions



```
[0x00401560]> afl~_Cfunc_
0x0049d2f0    1 49              sym._cgo_1796362b8bbc_Cfunc_greet
0x0049d2e0    1 8               sym._cgo_1796362b8bbc_Cfunc_free
0x0049d290    4 66    -> 62     sym._cgo_1796362b8bbc_Cfunc__Cmalloc
0x0049c9e0    8 191             sym.main._Cfunc_CString
0x0049caa0    5 133             sym.main._Cfunc_free
0x0049cb40    5 189             sym.main._Cfunc_greet
```

* Output also includes non-custom C functions called from main

stacks & prologue

goroutines have small stacks by default (2048 bytes stack)

goroutines will call `morestack` to grow the stack as needed using stack copying

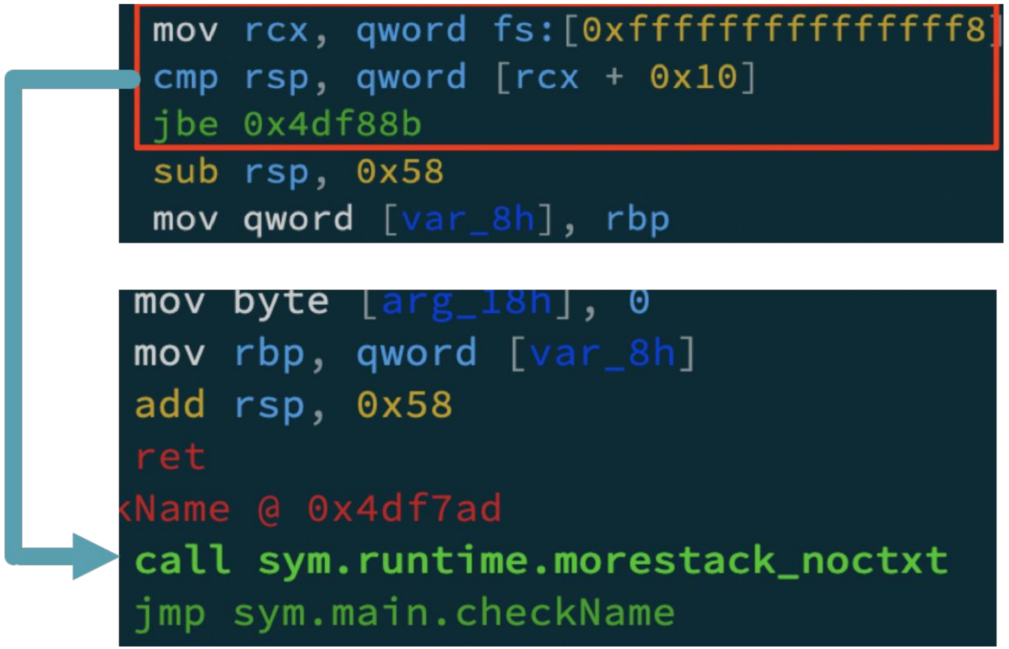
Go can’t be sure a function will outgrow the stack (i.e. recursive functions) given that goroutines are non-deterministic

Each function compares its stack pointer against `//g->stackguard` to check for overflow.

When this occurs, stack grows, pointers in the stack are updated.

Experiment with `//go:nosplit` pragma

```
// The minimum size of stack used by Go code
_StackMin = 2048
```



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

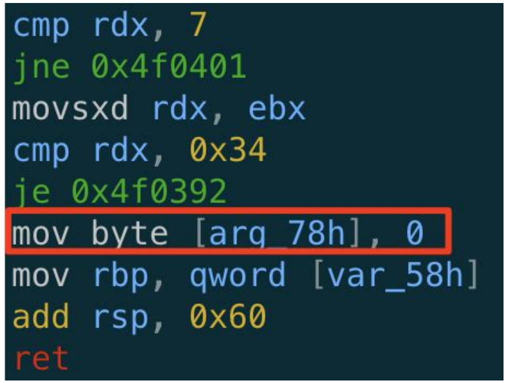
```
mov byte [arg_18h], 0
mov rbp, qword [var_8h]
add rsp, 0x58
ret
Name @ 0x4df7ad
call sym.runtime.morestack_noctxt
jmp sym.main.checkName
```

conventions

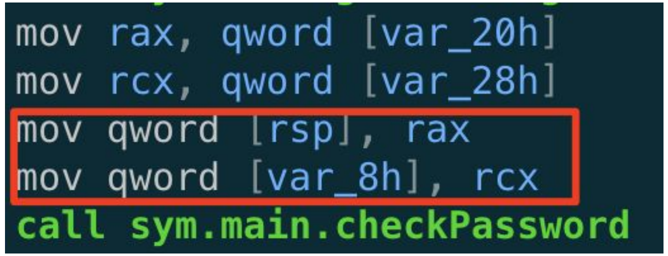
Return values are placed in the stack

As opposed to C where return values are placed in registers

Arguments are also moved to the stack rather than registers.



```
cmp rdx, 7
jne 0x4f0401
movsxd rdx, ebx
cmp rdx, 0x34
je 0x4f0392
mov byte [arg_78h], 0
mov rbp, qword [var_58h]
add rsp, 0x60
ret
```

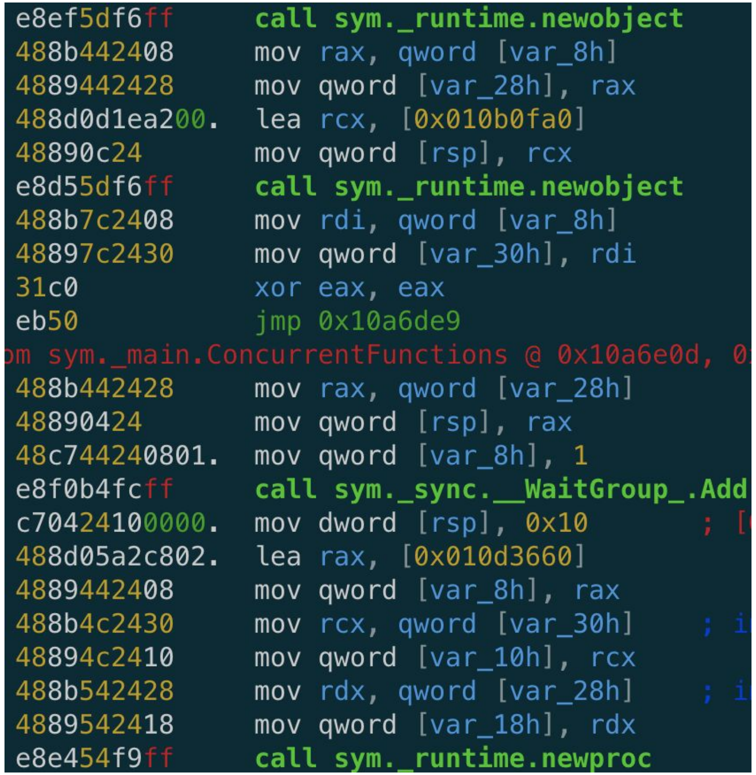


```
mov rax, qword [var_20h]
mov rcx, qword [var_28h]
mov qword [rsp], rax
mov qword [var_8h], rcx
call sym.main.checkPassword
```

identifying underlying language constructs

- Stripping function names is not easy (*)
- `go build -ldflags="-s -w"` only strips the DWARF symbols table
- It helps to become familiar with common functions from:
  - src/runtime/
  - src/io/
  - src/net/
  - src/os/
- This is key to reversing go, concurrent code and pointer operations in particular.

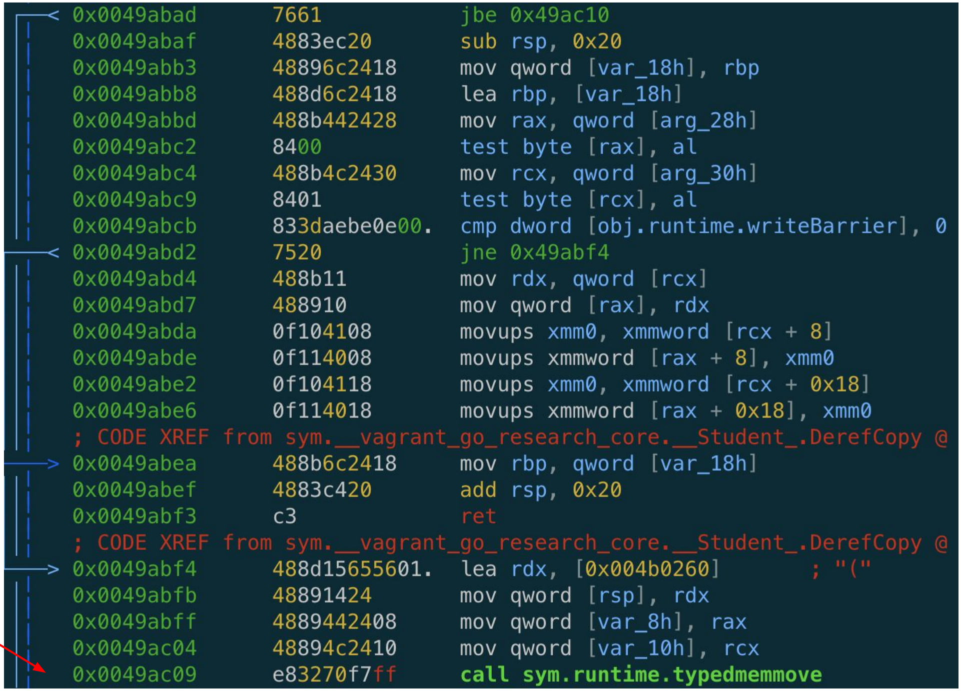
** Use a tool like UPX (https://upx.github.io/) to strip function names and reduce size*



```
e8ef5df6ff      call sym._runtime.newobject
488b442408      mov rax, qword [var_8h]
4889442428      mov qword [var_28h], rax
488d0d1ea200.   lea rcx, [0x010b0fa0]
48890c24        mov qword [rsp], rcx
e8d55df6ff      call sym._runtime.newobject
488b7c2408      mov rdi, qword [var_8h]
48897c2430      mov qword [var_30h], rdi
31c0            xor eax, eax
eb50            jmp 0x10a6de9
om sym._main.ConcurrentFunctions @ 0x10a6e0d, 0
488b442428      mov rax, qword [var_28h]
48890424        mov qword [rsp], rax
48c744240801.   mov qword [var_8h], 1
e8f0b4fcff      call sym._sync.__WaitGroup_.Add
c70424100000.   mov dword [rsp], 0x10        ; [
488d05a2c802.   lea rax, [0x010d3660]
4889442408      mov qword [var_8h], rax
488b4c2430      mov rcx, qword [var_30h]     ; i
48894c2410      mov qword [var_10h], rcx
488b542428      mov rdx, qword [var_28h]     ; i
4889542418      mov qword [var_18h], rdx
e8e454f9ff      call sym._runtime.newproc
```

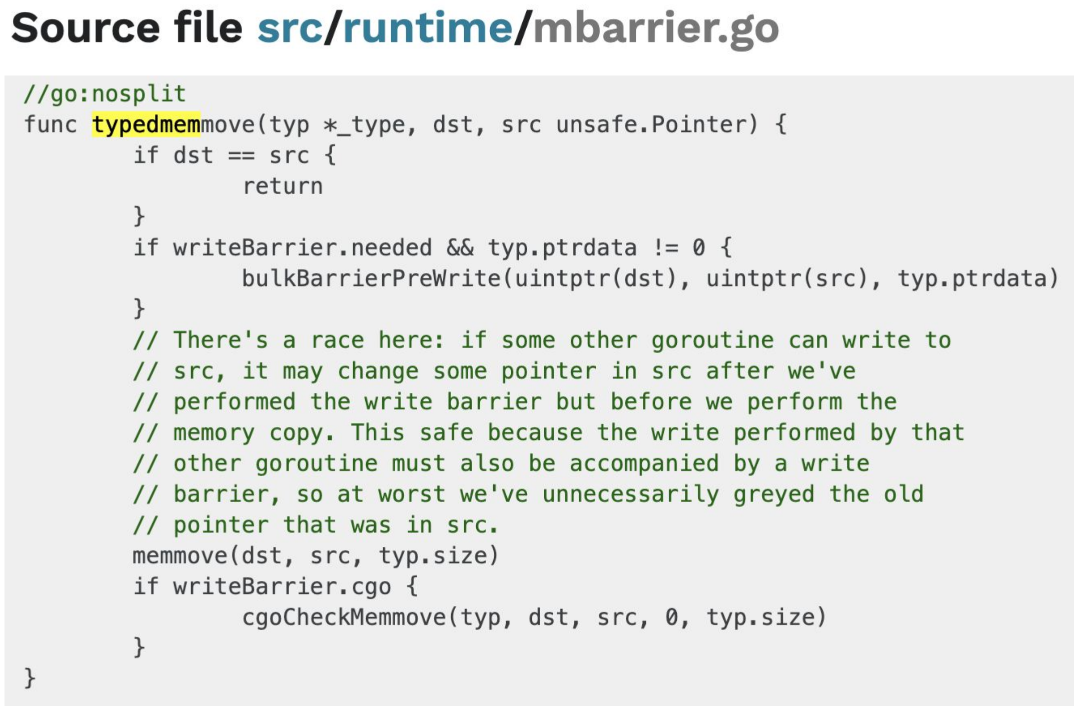
receiver dereference

```go
//go:noinline
func (s *Student) DerefCopy(student
*Student){
    *s = *student
}
```



```
0x0049abad      7661            jbe 0x49ac10
0x0049abaf      4883ec20        sub rsp, 0x20
0x0049abb3      48896c2418      mov qword [var_18h], rbp
0x0049abb8      488d6c2418      lea rbp, [var_18h]
0x0049abbd      488b442428      mov rax, qword [arg_28h]
0x0049abc2      8400            test byte [rax], al
0x0049abc4      488b4c2430      mov rcx, qword [arg_30h]
0x0049abc9      8401            test byte [rcx], al
0x0049abcb      833daebe0e00.   cmp dword [obj.runtime.writeBarrier], 0
0x0049abd2      7520            jne 0x49abf4
0x0049abd4      488b11          mov rdx, qword [rcx]
0x0049abd7      488910          mov qword [rax], rdx
0x0049abda      0f104108        movups xmm0, xmmword [rcx + 8]
0x0049abde      0f114008        movups xmmword [rax + 8], xmm0
0x0049abe2      0f104118        movups xmm0, xmmword [rcx + 0x18]
0x0049abe6      0f114018        movups xmmword [rax + 0x18], xmm0
; CODE XREF from sym.__vagrant_go_research_core.__Student_.DerefCopy @
0x0049abea      488b6c2418      mov rbp, qword [var_18h]
0x0049abef      4883c420        add rsp, 0x20
0x0049abf3      c3              ret
; CODE XREF from sym.__vagrant_go_research_core.__Student_.DerefCopy @
0x0049abf4      488d15655601.   lea rdx, [0x004b0260]        ; "("
0x0049abfb      48891424        mov qword [rsp], rdx
0x0049abff      4889442408      mov qword [var_8h], rax
0x0049ac04      48894c2410      mov qword [var_10h], rcx
0x0049ac09      e83270f7ff      call sym.runtime.typedmemmove
```

low level function

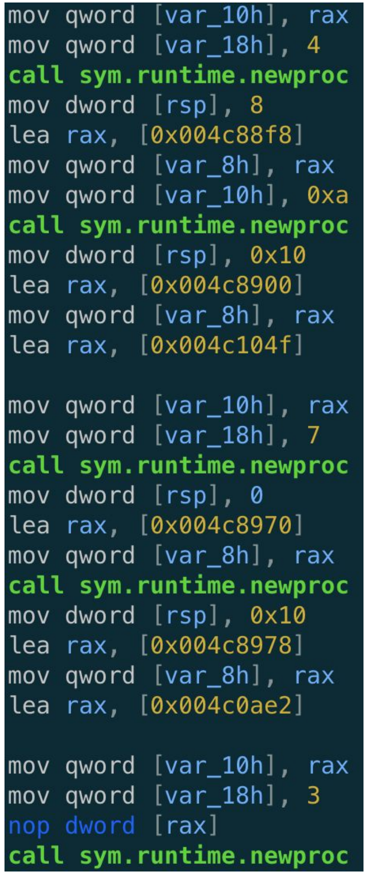


Source file src/runtime/mbarrier.go

```
//go:nosplit
func typedmemmove(typ *_type, dst, src unsafe.Pointer) {
	if dst == src {
		return
	}
	if writeBarrier.needed && typ.ptrdata != 0 {
		bulkBarrierPreWrite(uintptr(dst), uintptr(src), typ.ptrdata)
	}
	// 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)
	}
}
```

goroutines

```go
func main() {
    go core.Cpy_println("test")
    go core.ForLoop(10)
    go core.ForRange("gophers")
    go testRace()
    go testUnsafe("xyz")
}
```



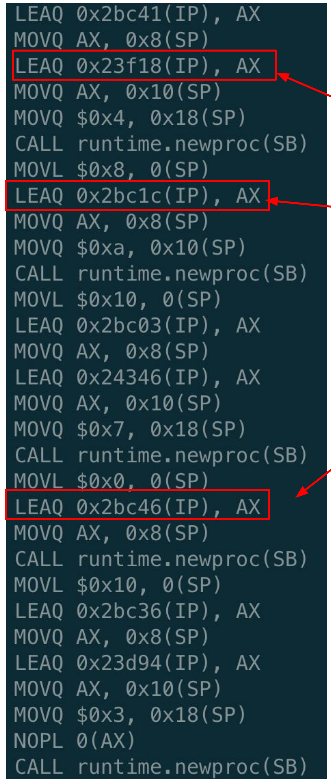
```
mov qword [var_10h], rax
mov qword [var_18h], 4
call sym.runtime.newproc
mov dword [rsp], 8
lea rax, [0x004c88f8]
mov qword [var_8h], rax
mov qword [var_10h], 0xa
call sym.runtime.newproc
mov dword [rsp], 0x10
lea rax, [0x004c8900]
mov qword [var_8h], rax
lea rax, [0x004c104f]

mov qword [var_10h], rax
mov qword [var_18h], 7
call sym.runtime.newproc
mov dword [rsp], 0
lea rax, [0x004c8970]
mov qword [var_8h], rax
call sym.runtime.newproc
mov dword [rsp], 0x10
lea rax, [0x004c8978]
mov qword [var_8h], rax
lea rax, [0x004c0ae2]

mov qword [var_10h], rax
mov qword [var_18h], 3
nop dword [rax]
call sym.runtime.newproc
```

func newproc

```go
func newproc(siz int32, fn *funcval)
```

Create a new g running fn with siz bytes of arguments. Put it on the queue of g's waiting to run. The compiler turns a go statement into a call to this.

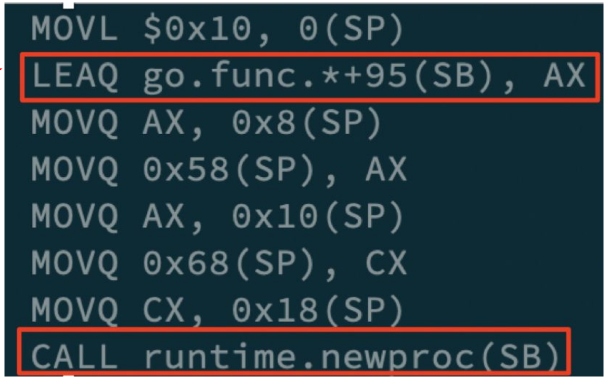
more goroutines with go tool objdump



```
LEAQ 0x2bc41(IP), AX
MOVQ AX, 0x8(SP)
LEAQ 0x23f18(IP), AX
MOVQ AX, 0x10(SP)
MOVQ $0x4, 0x18(SP)
CALL runtime.newproc(SB)
MOVL $0x8, 0(SP)
LEAQ 0x2bc1c(IP), AX
MOVQ AX, 0x8(SP)
MOVQ $0xa, 0x10(SP)
CALL runtime.newproc(SB)
MOVL $0x10, 0(SP)
LEAQ 0x2bc03(IP), AX
MOVQ AX, 0x8(SP)
LEAQ 0x24346(IP), AX
MOVQ AX, 0x10(SP)
MOVQ $0x7, 0x18(SP)
CALL runtime.newproc(SB)
MOVL $0x0, 0(SP)
LEAQ 0x2bc46(IP), AX
MOVQ AX, 0x8(SP)
CALL runtime.newproc(SB)
MOVL $0x10, 0(SP)
LEAQ 0x2bc36(IP), AX
MOVQ AX, 0x8(SP)
LEAQ 0x23d94(IP), AX
MOVQ AX, 0x10(SP)
MOVQ $0x3, 0x18(SP)
NOPL 0(AX)
CALL runtime.newproc(SB)
```

RIP relative

SB (Static Base) pointer relative



```
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)
```

what to look for

Concurrency

```
call sym.runtime.newobject
call sym.runtime.makechan
call sym._sync.__WaitGroup_.Add

call sym.runtime.deferprocStack
call sym.runtime.deferreturn
```

CGO Anything

```
call sym.main._cgo_cmalloc
call sym.main._Cfunc_CString
```

GC

```
call sym.runtime.gcWriteBarrier
call sym.runtime.typedmemmove
```

RTFM!

go error handling

Error handling is super clumsy and manual

When looking for bugs, it is worth taking the time to identify (missing) error handling logic

`error` is a go interface:

- A pointer to a vtable (which contains function pointers that point to the virtual functions)
- A value pointer

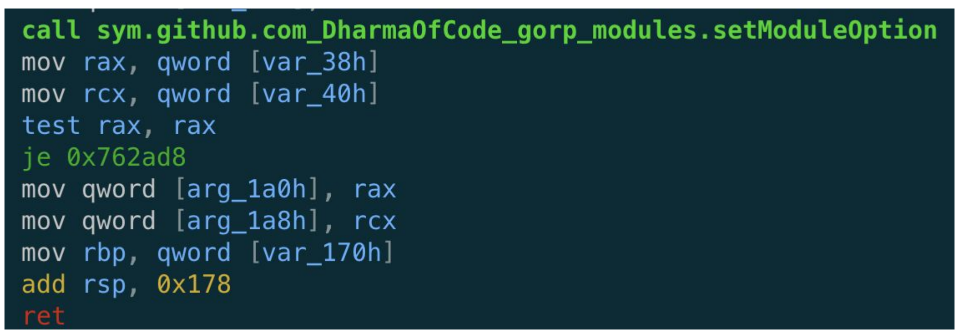
go error handling ID

When checking for `error != nil` we load the error vtable and error value.

Then we test if the value is nil

And branch or return depending on the result

```go
func (m *Modules) InitProcessors(mods
[]base.ModuleConfig) error {
   //..
   err := module.SetOption(option, value)
   if err != nil {
       return err
   }
   //..
}
```



```
call sym.github.com_DharmaOfCode_gorp_modules.setModuleOption
mov rax, qword [var_38h]
mov rcx, qword [var_40h]
test rax, rax
je 0x762ad8
mov qword [arg_1a0h], rax
mov qword [arg_1a8h], rcx
mov rbp, qword [var_170h]
add rsp, 0x178
ret
```

so you wanna research go?

Write go!

Play with pragmas:

```
//go:nosplit
//go:noinline
```

Set `stackDubug = 1` in /src/runtime/stack.go to monitor stack size

Use `go build -m` to print optimization decisions

Use `GODEBUG=gctrace` to better understand what the GC is doing

Use tools like go-fuzz and gosec to check for bugs

wrap up

- Combine go tool with r2 for easy RE of go binaries
- XREF strings is not fun
- Familiarize yourself with go internals
    - Scheduling
    - Garbage collector
- Looking for bugs
    - race condition bugs
    - unhandled errors
    - CGO

questions?

Use the google slides questions!