Low-Level GoLang

To reverse engineer Go binaries

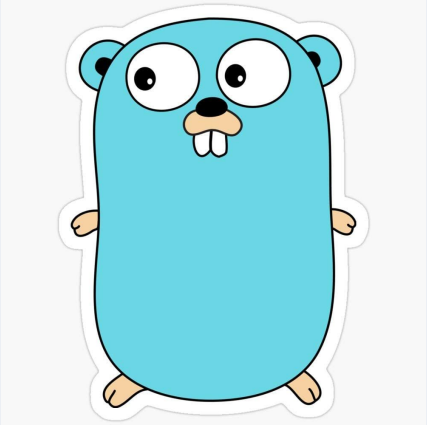
Motivation

Some backstory

What is GoLang?

+ Developed by a team at Google
+ Since 2009

++ Compiled language
++ Statically linked



HelloWorld.go

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, 世界")
}
```

Go Malware

+ Compiles to all platforms (Windows, MacOS, Linux)
+ Can run anywhere (because statically linked)
+ Large file size (antivirus cannot scan)
+ Rich library ecosystem

Go Malware

According to research by someone in Palo Alto Networks in 2019,

++ 8025 Go malware samples
++ 53 malware families

Go Malware

According to research by someone in Palo Alto Networks in 2019,

++ 8025 Go malware samples
++ 53 malware families

Apparently this is not a lot.

Go in low-level

Assembly code, structs, symbols, ...

Go support in GDB

When program is compiled with debugging information (the default setting) and loaded in GDB, GDB will load runtime-gdb.py which contains pretty-printers and convenience functions.

```
"""GDB Pretty printers and convenience functions for Go's runtime structures.

This script is loaded by GDB when it finds a .debug_gdb_scripts
section in the compiled binary. The [68]l linkers emit this with a
path to this file based on the path to the runtime package.
"""
```

Go support in GDB

If compiled without debugging information, DIY

```
❯ echo "add-auto-load-safe-path /usr/local/go/src/runtime/runtime-gdb.py" >> ~/.gdbinit
```

But this is actually only useful when there is debugging information.

0.main.main

```go
package main

import "fmt"

func main() {
  fmt.Println("Hello, 世界")
}
```

0.main.main

30 lines of asm for a HelloWorld program:

```go
package main

import "fmt"

func main() {
  fmt.Println("Hello, 世界")
}
```

```
❯ objdump -d -Mintel main | grep -A 30 _main.main:
00000000010a6e80 _main.main:
 10a6e80: 65 48 8b 0c 25 30 00 00 00    mov    rcx, qword ptr gs:[48]
 10a6e89: 48 3b 61 10                   cmp    rsp, qword ptr [rcx + 16]
 10a6e8d: 76 71                         jbe    113 <_main.main+0×80>
...
 10a6e8f: 48 83 ec 58                   call   -25877 <_fmt.Fprintln>
 10a6ef5: 48 8b 6c 24 50                mov    rbp, qword ptr [rsp + 80]
 10a6efa: 48 83 c4 58                   add    rsp, 88
 10a6efe: c3                            ret
 10a6eff: 90                            nop
 10a6f00: e8 7b 9f fb ff                call   -286853 <_runtime.morestack_noctxt>
 10a6f05: e9 76 ff ff ff                jmp    -138 <_main.main>
```

0.main.main

```
❯ objdump -d -Mintel main | grep -A 30 _main.main:
0000000010a6e80 _main.main:
 10a6e80: 65 48 8b 0c 25 30 00 00 00    mov    rcx, qword ptr gs:[48]
 10a6e89: 48 3b 61 10                   cmp    rsp, qword ptr [rcx + 16]
 10a6e8d: 76 71                         jbe    113 <_main.main+0×80>
...
 10a6e8f: 48 83 ec 58                   call   -25877 <_fmt.Fprintln>
 10a6ef5: 48 8b 6c 24 50                mov    rbp, qword ptr [rsp + 80]
 10a6efa: 48 83 c4 58                   add    rsp, 88
 10a6efe: c3                            ret
 10a6eff: 90                            nop
 10a6f00: e8 7b 9f fb ff                call   -286853 <_runtime.morestack_noctxt>
 10a6f05: e9 76 ff ff ff                jmp    -138 <_main.main>
```

Is there enough space for stack?

O.main.main

```
❯ objdump -d -Mintel main | grep -A 30 _main.main:
0000000010a6e80 _main.main:
 10a6e80: 65 48 8b 0c 25 30 00 00 00    mov    rcx, qword ptr gs:[48]
 10a6e89: 48 3b 61 10                   cmp    rsp, qword ptr [rcx + 16]
 10a6e8d: 76 71                         jbe    113 <_main.main+0×80>
...
 10a6e8f: 48 83 ec 58                   call   -25877 <_fmt.Fprintln>
 10a6ef5: 48 8b 6c 24 50                mov    rbp, qword ptr [rsp + 80]
 10a6efa: 48 83 c4 58                   add    rsp, 88
 10a6efe: c3                            ret
 10a6eff: 90                            nop
 10a6f00: e8 7b 9f fb ff                call   -286853 <_runtime.morestack_noctxt>
 10a6f05: e9 76 ff ff ff                jmp    -138 <_main.main>
```

Is there enough space for stack?

Make stack bigger
Go back to start of function

0.main.main

```
❯ objdump -d -Mintel main | grep -A 30 _main.main:
0000000010a6e80 _main.main:
 10a6e80: 65 48 8b 0c 25 30 00 00 00    mov    rcx, qword ptr gs:[48]
 10a6e89: 48 3b 61 10                   cmp    rsp, qword ptr [rcx + 16]
 10a6e8d: 76 71                         jbe    113 <_main.main+0×80>
...
 10a6e8f: 48 83 ec 58                   call   -25877 <_fmt.Fprintln>
 10a6ef5: 48 8b 6c 24 50                mov    rbp, qword ptr [rsp + 80]
 10a6efa: 48 83 c4 58                   add    rsp, 88
 10a6efe: c3                            ret
 10a6eff: 90                            nop
 10a6f00: e8 7b 9f fb ff                call   -286853 <_runtime.morestack_noctxt>
 10a6f05: e9 76 ff ff ff                jmp    -138 <_main.main>
```

Is there enough space for stack?

```
fmt.Println("Hello, 世界")
```

Make stack bigger
Go back to start of function

0.main.main

Only 6 lines of asm with the C equivalent:

```c
int main()
{
    puts("Hello World");
}
```

```
❯ objdump -d -Mintel hello-c | grep -A7  \<main\>:
000000000000063a <main>:
 63a:  55                       push   rbp
 63b:  48 89 e5                 mov    rbp,rsp
 63e:  48 8d 3d 9f 00 00 00     lea    rdi,[rip+0x9f]
 645:  e8 c6 fe ff ff           call   510 <puts@plt>
 64a:  b8 00 00 00 00           mov    eax,0x0
 64f:  5d                       pop    rbp
 650:  c3                       ret
```

1. Calling convention

```go
package main

import "fmt"

func foo(a, b, c int) int {
  return a + b - c
}

func main() {
  fmt.Printf("%d\n", foo(1, 2, 3))
}
```

1. Calling convention

Arguments are placed on the stack (instead of into registers like C programs)

```go
package main

import "fmt"

func foo(a, b, c int) int {
  return a + b - c
}

func main() {
  fmt.Printf("%d\n", foo(1, 2, 3))
}
```

```
❯ objdump -d -Mintel calling | grep -A6 _main.foo:
000000000105c8a0 _main.foo:
 105c8a0: 48 c7 44 24 20 00 00 00 00    mov    qword ptr [rsp + 32], 0
 105c8a9: 48 8b 44 24 08                mov    rax, qword ptr [rsp + 8]
 105c8ae: 48 03 44 24 10                add    rax, qword ptr [rsp + 16]
 105c8b3: 48 2b 44 24 18                sub    rax, qword ptr [rsp + 24]
 105c8b8: 48 89 44 24 20                mov    qword ptr [rsp + 32], rax
 105c8bd: c3                            ret

❯ objdump -d -Mintel calling | grep -A2 -B3 \<_main.foo\>
 10a87af: 48 c7 04 24 01 00 00 00       mov    qword ptr [rsp], 1
 10a87b7: 48 c7 44 24 08 02 00 00 00    mov    qword ptr [rsp + 8], 2
 10a87c0: 48 c7 44 24 10 03 00 00 00    mov    qword ptr [rsp + 16], 3
 10a87c9: e8 92 ff ff ff                call   -110 <_main.foo>
 10a87ce: 48 8b 44 24 18                mov    rax, qword ptr [rsp + 24]
 10a87d3: 48 89 44 24 40                mov    qword ptr [rsp + 64], rax
```

1. Calling convention

Arguments are placed on the stack

```
❯ objdump -d -Mintel calling | grep -A6 _main.foo:
000000000105c8a0 _main.foo:
 105c8a0: 48 c7 44 24 20 00 00 00 00    mov    qword ptr [rsp + 32], 0
 105c8a9: 48 8b 44 24 08                mov    rax, qword ptr [rsp + 8]
 105c8ae: 48 03 44 24 10                add    rax, qword ptr [rsp + 16]
 105c8b3: 48 2b 44 24 18                sub    rax, qword ptr [rsp + 24]
 105c8b8: 48 89 44 24 20                mov    qword ptr [rsp + 32], rax
 105c8bd: c3                            ret

❯ objdump -d -Mintel calling | grep -A2 -B3 \<_main.foo\>
 10a87af: 48 c7 04 24 01 00 00 00       mov    qword ptr [rsp], 1
 10a87b7: 48 c7 44 24 08 02 00 00 00    mov    qword ptr [rsp + 8], 2
 10a87c0: 48 c7 44 24 10 03 00 00 00    mov    qword ptr [rsp + 16], 3
 10a87c9: e8 92 ff ff ff                call   -110 <_main.foo>
 10a87ce: 48 8b 44 24 18                mov    rax, qword ptr [rsp + 24]
 10a87d3: 48 89 44 24 40                mov    qword ptr [rsp + 64], rax
```

```
func foo(a, b, c int) int {
  return a + b - c
}
```

```
foo(1, 2, 3)
```

1. Calling convention

Arguments are placed on the stack

```
❯ objdump -d -Mintel calling | grep -A6 _main.foo:
000000000105c8a0 _main.foo:
 105c8a0: 48 c7 44 24 20 00 00 00 00    mov    qword ptr [rsp + 32], 0
 105c8a9: 48 8b 44 24 08                mov    rax, qword ptr [rsp + 8]
 105c8ae: 48 03 44 24 10                add    rax, qword ptr [rsp + 16]
 105c8b3: 48 2b 44 24 18                sub    rax, qword ptr [rsp + 24]
 105c8b8: 48 89 44 24 20                mov    qword ptr [rsp + 32], rax
 105c8bd: c3                            ret

❯ objdump -d -Mintel calling | grep -A2 -B3 \<_main.foo\>
 10a87af: 48 c7 04 24 01 00 00 00       mov    qword ptr [rsp], 1
 10a87b7: 48 c7 44 24 08 02 00 00 00    mov    qword ptr [rsp + 8], 2
 10a87c0: 48 c7 44 24 10 03 00 00 00    mov    qword ptr [rsp + 16], 3
 10a87c9: e8 92 ff ff ff                call   -110 <_main.foo>
 10a87ce: 48 8b 44 24 18                mov    rax, qword ptr [rsp + 24]
 10a87d3: 48 89 44 24 40                mov    qword ptr [rsp + 64], rax
```

```
func foo(a, b, c int) int
```

rsp	return address
rsp+8	a
rsp+16	b
rsp+24	c
rsp+32	a + b + c

```
foo(1, 2, 3)
```

1. Calling convention

Arguments are placed on the stack

```
❯ objdump -d -Mintel calling | grep -A6 _main.foo:
000000000105c8a0 _main.foo:
 105c8a0: 48 c7 44 24 20 00 00 00 00    mov    qword ptr [rsp + 32], 0
 105c8a9: 48 8b 44 24 08                mov    rax, qword ptr [rsp + 8]
 105c8ae: 48 03 44 24 10                add    rax, qword ptr [rsp + 16]
 105c8b3: 48 2b 44 24 18                sub    rax, qword ptr [rsp + 24]
 105c8b8: 48 89 44 24 20                mov    qword ptr [rsp + 32], rax
 105c8bd: c3                            ret

❯ objdump -d -Mintel calling | grep -A2 -B3 \<_main.foo\>
 10a87af: 48 c7 04 24 01 00 00 00       mov    qword ptr [rsp], 1
 10a87b7: 48 c7 44 24 08 02 00 00 00    mov    qword ptr [rsp + 8], 2
 10a87c0: 48 c7 44 24 10 03 00 00 00    mov    qword ptr [rsp + 16], 3
 10a87c9: e8 92 ff ff ff                call   -110 <_main.foo>
 10a87ce: 48 8b 44 24 18                mov    rax, qword ptr [rsp + 24]
 10a87d3: 48 89 44 24 40                mov    qword ptr [rsp + 64], rax
```

func foo(a, b, c int) int	
rsp	return address
rsp+8	a | 1
rsp+16	b | 2
rsp+24	c | 3
rsp+32	a + b - c | 1 + 2 - 3

foo(1, 2, 3)	
rsp	1
rsp+8	2
rsp+16	3
rsp+24	1 + 2 - 3

2. Stack frame

Earlier we looked at a couple of functions.
Sometimes they didn’t start with

```
mov rbp, rsp
```

and neither ending with

```
leave/pop rbp
```

like C programs normally do.

2. Stack frame

If a function doesn’t call another function, Go will omit this, to save instruction/space on the stack.

2. Stack frame

Example:

+ `goo` doesn't call any functions
+ `rbp` and `rsp` are not touched

```go
package main

func foo(i int) int { return goo(i) }

func goo(i int) int { return i + 1 }

func main() {
	foo(1)
}
```

```
❯ objdump -d -Mintel stack-frame | grep -A6 main.goo\>:
000000000045dce0 <main.goo>:
  45dce0:       48 c7 44 24 10 00 00    mov    QWORD PTR [rsp+0x10],0x0
  45dce7:       00 00
  45dce9:       48 8b 44 24 08          mov    rax,QWORD PTR [rsp+0x8]
  45dcee:       48 ff c0                inc    rax
  45dcf1:       48 89 44 24 10          mov    QWORD PTR [rsp+0x10],rax
  45dcf6:       c3                      ret
```

2. Stack frame

Example:

- `foo` calls `goo`
- `rbp` and `rsp` are modified to adjust the stack frame

```go
package main

func foo(i int) int { return goo(i) }

func goo(i int) int { return i + 1 }

func main() {
	foo(1)
}
```

```
❯ objdump -d -Mintel stack-frame | grep -A18 main.foo\>:
000000000045dc80 <main.foo>:
...
  45dc8f:	48 83 ec 20 		sub    rsp,0x20
  45dc93:	48 89 6c 24 18 		mov    QWORD PTR [rsp+0x18],rbp
  45dc98:	48 8d 6c 24 18 		lea    rbp,[rsp+0x18]

... < asm for return i + 1 > ...

  45dcc3:	48 8b 6c 24 18 		mov    rbp,QWORD PTR [rsp+0x18]
  45dcc8:	48 83 c4 20 		add    rsp,0x20
  45dccc:	c3 		ret
```

3. Strings

Strings in C:

```
char* str = "Hello";
char* str = {'H', 'e', 'l', 'l', 'o', 0};
```

3. Strings

Strings in Go:

```
var str = "Hello";
```

3. Strings

Reference to strings in a C program:

```c
int main()
{
    puts("Hello World");
}
```

```
❯ objdump -d -Mintel hello-c | grep -A7  \<main\>:
000000000000063a <main>:
 63a:  55                       push   rbp
 63b:  48 89 e5                 mov    rbp,rsp
 63e:  48 8d 3d 9f 00 00 00     lea    rdi,[rip+0x9f]
 645:  e8 c6 fe ff ff           call   510 <puts@plt>
 64a:  b8 00 00 00 00           mov    eax,0x0
 64f:  5d                       pop    rbp
 650:  c3                       ret
```

3. Strings

Reference to strings in a C program:

```
❯ objdump -d -Mintel hello-c | grep -A7  \<main\>:
000000000000063a <main>:
 63a:  55                       push   rbp
 63b:  48 89 e5                 mov    rbp,rsp
 63e:  48 8d 3d 9f 00 00 00     lea    rdi,[rip+0x9f] #6e4
 645:  e8 c6 fe ff ff           call   510 <puts@plt>
 64a:  b8 00 00 00 00           mov    eax,0x0
 64f:  5d                       pop    rbp
 650:  c3                       ret
```

3. Strings

Reference to strings in a C program:

+ `char[]` is directly referenced

```
❯ objdump -d -Mintel hello-c | grep -A7  \<main\>:
000000000000063a <main>:
 63a:  55                       push   rbp
 63b:  48 89 e5                 mov    rbp,rsp
 63e:  48 8d 3d 9f 00 00 00     lea    rdi,[rip+0x9f] #6e4
 645:  e8 c6 fe ff ff           call   510 <puts@plt>
 64a:  b8 00 00 00 00           mov    eax,0x0
 64f:  5d                       pop    rbp
 650:  c3                       ret
```

rdi	Pointer to "You"

3. Strings

Reference to strings in a C program:

+ `char[]` is directly referenced

```
❯ objdump -d -Mintel hello-c | grep -A7  \<main\>:
000000000000063a <main>:
 63a:  55                       push   rbp
 63b:  48 89 e5                 mov    rbp,rsp
 63e:  48 8d 3d 9f 00 00 00     lea    rdi,[rip+0x9f] #6e4
 645:  e8 c6 fe ff ff           call   510 <puts@plt>
 64a:  b8 00 00 00 00           mov    eax,0x0
 64f:  5d                       pop    rbp
 650:  c3                       ret
```

rdi	Pointer to "You"

```
gef➢  hexdump byte $_base()+0x6e4 L16
0x00005555555546e4     48 65 6c 6c 6f 20 57 6f 72 6c 64 00 01 1b 03 3b    Hello World....;
```

3. Strings

Reference to strings in a Go program:

```go
package main

import "fmt"

func main() {
  fmt.Println("Hello, 世界")
}
```

```
❯ objdump -d -Mintel main | grep -A30 \<main.main\>:
0000000000499080 <main.main>:
 ...
  4990a5:    48 8d 05 f4 b7 00 00      lea    rax,[rip+0×b7f4]        # 4a48a0
  4990ac:    48 89 44 24 40            mov    QWORD PTR [rsp+0×40],rax
  4990b1:    48 8d 05 38 1a 04 00      lea    rax,[rip+0×41a38]        # 4daaf0
  4990b8:    48 89 44 24 48            mov    QWORD PTR [rsp+0×48],rax
…
  4990d4:    48 8d 44 24 40            lea    rax,[rsp+0×40]
  4990d9:    48 89 44 24 10            mov    QWORD PTR [rsp+0×10],rax
 ...
  4990f0:    e8 eb 9a ff ff            call   492be0 <fmt.Fprintln>
 ...
```

3. Strings

Reference to strings in a Go program:

- `string` is stored in `[rsp+0×40]`
- and a pointer to this location is stored in `[rsp+0×10]` as an argument for `fmt.Fprintln`

```
❯ objdump -d -Mintel main | grep -A30 \<main.main\>:
0000000000499080 <main.main>:
...
  4990a5:    48 8d 05 f4 b7 00 00      lea    rax,[rip+0×b7f4]        # 4a48a0
  4990ac:    48 89 44 24 40            mov    QWORD PTR [rsp+0×40],rax
  4990b1:    48 8d 05 38 1a 04 00      lea    rax,[rip+0×41a38]        # 4daaf0
  4990b8:    48 89 44 24 48            mov    QWORD PTR [rsp+0×48],rax
…
  4990d4:    48 8d 44 24 40            lea    rax,[rsp+0×40]
  4990d9:    48 89 44 24 10            mov    QWORD PTR [rsp+0×10],rax
...
  4990f0:    e8 eb 9a ff ff            call   492be0 <fmt.Fprintln>
```

3. Strings

Reference to strings in a Go program:

- `string` is stored in `[rsp+0×40]`
- and a pointer to this location is stored in `[rsp+0×10]` as an argument for `fmt.Fprintln`

```
❯ objdump -d -Mintel main | grep -A30 \<main.main\>:
0000000000499080 <main.main>:
...
  4990a5:     48 8d 05 f4 b7 00 00       lea    rax,[rip+0×b7f4]        # 4a48a0
  4990ac:     48 89 44 24 40             mov    QWORD PTR [rsp+0×40],rax
  4990b1:     48 8d 05 38 1a 04 00       lea    rax,[rip+0×41a38]        # 4daaf0
  4990b8:     48 89 44 24 48             mov    QWORD PTR [rsp+0×48],rax
…
  4990d4:     48 8d 44 24 40             lea    rax,[rsp+0×40]
  4990d9:     48 89 44 24 10             mov    QWORD PTR [rsp+0×10],rax
...
  4990f0:     e8 eb 9a ff ff             call   492be0 <fmt.Fprintln>
```

```
fmt.Fprintln( ... , "You", ... )
```

3. Strings

Reference to strings in a Go program:

- `string` is stored in `[rsp+0×40]`
- and a pointer to this location is stored in `[rsp+0×10]` as an argument for `fmt.Fprintln`

```
❯ objdump -d -Mintel main | grep -A30 \<main.main\>:
0000000000499080 <main.main>:
...
  4990a5:    48 8d 05 f4 b7 00 00      lea    rax,[rip+0×b7f4]        # 4a48a0
  4990ac:    48 89 44 24 40            mov    QWORD PTR [rsp+0×40],rax
  4990b1:    48 8d 05 38 1a 04 00      lea    rax,[rip+0×41a38]        # 4daaf0
  4990b8:    48 89 44 24 48            mov    QWORD PTR [rsp+0×48],rax
…
  4990d4:    48 8d 44 24 40            lea    rax,[rsp+0×40]
  4990d9:    48 89 44 24 10            mov    QWORD PTR [rsp+0×10],rax
...
  4990f0:    e8 eb 9a ff ff            call   492be0 <fmt.Fprintln>
```

1. Pointer to `[rsp+0×40]` loaded into `rax`
2. Then passed as an argument to `fmt.Fprintln`

rax	Pointer to "You"

```
fmt.Fprintln( ... , "You", ... )
```

3. Strings

Reference to strings in a Go program:

- `[rsp+0×40]` contains 16 bytes of contents

```
4990a5:    48 8d 05 f4 b7 00 00      lea    rax,[rip+0×b7f4]              # 4a48a0
4990ac:    48 89 44 24 40            mov    QWORD PTR [rsp+0×40],rax
4990b1:    48 8d 05 38 1a 04 00      lea    rax,[rip+0×41a38]             # 4daaf0
4990b8:    48 89 44 24 48            mov    QWORD PTR [rsp+0×48],rax
```

```
gef➢  deref $rsp+0×40
0×000000c000118f68│+0×0000: 0×00000000004a48a0  →  0×0000000000000010
0×000000c000118f70│+0×0008: 0×00000000004daaf0  →  0×00000000004be0b0  →  0×e4202c6f6c6c6548
```

3. Strings

Reference to strings in a Go program:

- `[rsp+0×40]` contains 16 bytes of contents

```
4990a5:    48 8d 05 f4 b7 00 00      lea    rax,[rip+0×b7f4]              # 4a48a0
4990ac:    48 89 44 24 40            mov    QWORD PTR [rsp+0×40],rax
4990b1:    48 8d 05 38 1a 04 00      lea    rax,[rip+0×41a38]             # 4daaf0
4990b8:    48 89 44 24 48            mov    QWORD PTR [rsp+0×48],rax
```

&string	
rsp+0×40	0×4a48a0
rsp+0×48	0×4daaf0

```
gef➢  deref $rsp+0×40
0×000000c000118f68│+0×0000: 0×00000000004a48a0  →  0×0000000000000010
0×000000c000118f70│+0×0008: 0×00000000004daaf0  →  0×00000000004be0b0  →  0×e4202c6f6c6c6548
```

3. Strings

Reference to strings in a Go program:

+ `*[rsp+0×40] (0×4a48a0)` contains the interface information. Not very interesting.

```
 4990a5:  48 8d 05 f4 b7 00 00  lea    rax,[rip+0×b7f4]          # 4a48a0
 4990ac:  48 89 44 24 40        mov    QWORD PTR [rsp+0×40],rax
 4990b1:  48 8d 05 38 1a 04 00  lea    rax,[rip+0×41a38]         # 4daaf0
 4990b8:  48 89 44 24 48        mov    QWORD PTR [rsp+0×48],rax
```

```
gef➢  deref 0×00000000004a48a0
0×00000000004a48a0│+0×0000: 0×0000000000000010
0×00000000004a48a8│+0×0008: 0×0000000000000008
0×00000000004a48b0│+0×0010: 0×18080807e0ff5cb4
0×00000000004a48b8│+0×0018: 0×00000000004c4e30  →  0×0000000000403420  →  <runtime.strequal+0> mov rcx, QWORD PTR
fs:0×fffffffffffffff8
0×00000000004a48c0│+0×0020: 0×00000000004d7fc0  →  0×0807060504030201
0×00000000004a48c8│+0×0028: 0×000084000000113e
0×00000000004a48d0│+0×0030: 0×0000000000000000
```

3. Strings

Reference to strings in a Go program:

- `*[rsp+0×48] (0×4daaf0)` contains the string

```
4990a5:    48 8d 05 f4 b7 00 00     lea    rax,[rip+0×b7f4]              # 4a48a0
4990ac:    48 89 44 24 40           mov    QWORD PTR [rsp+0×40],rax
4990b1:    48 8d 05 38 1a 04 00     lea    rax,[rip+0×41a38]             # 4daaf0
4990b8:    48 89 44 24 48           mov    QWORD PTR [rsp+0×48],rax
```

&string	
rsp+0×40	interface
rsp+0×48	0×4daaf0

3. Strings

Reference to strings in a Go program:

- `*[rsp+0×48] (0×4daaf0)` contains the string

```
4990a5:    48 8d 05 f4 b7 00 00     lea    rax,[rip+0×b7f4]              # 4a48a0
4990ac:    48 89 44 24 40           mov    QWORD PTR [rsp+0×40],rax
4990b1:    48 8d 05 38 1a 04 00     lea    rax,[rip+0×41a38]             # 4daaf0
4990b8:    48 89 44 24 48           mov    QWORD PTR [rsp+0×48],rax
```

&string	
rsp+0×40	interface
rsp+0×48	0×4daaf0

```
gef➢  deref 0×00000000004daaf0
0×00000000004daaf0 | +0×0000: 0×00000000004be0b0  →  0×e4202c6f6c6c6548    # string contents
0×00000000004daaf8 | +0×0008: 0×000000000000000d                            # string length
```

0×4daaf0:	
0×4daaf0	Pointer to string contents
0×4daaf8	Length of string

```
gef➢  hexdump byte 0×00000000004be0b0 L0×d
0×00000000004be0b0 <string.*+14c0>    48 65 6c 6c 6f 2c 20 e4 b8 96 e7 95 8c    Hello, 世界
```

3. Strings

Strings in Go are not null-terminated

```
gef➢  x/s 0x00000000004be0b0
0x4be0b0:    "Hello, 世界Masaram_GondiMende_KikakuiOld_HungarianSIGKILL: killSIGQUIT: quitbad flushGen bad map
statedebugCall2048exchange fullfatal error: level 3 resetload64 failedmin too largenil stackbaseout of memorysrmount
errortimer expiredtraceStackTabtriggerRatio=value method xadd64 failedxchg64 failed}\n\tsched={pc: but progSize
nmidlelocked= on zero Value out of range  to finalizer  untyped args -thread
limit\n/proc/self/exe190734863281259536743164062 5GC assist waitGC worker initMB; allocated
Other_ID_StartPattern_SyntaxQuotation_MarkSIGABRT: abortallocfreetracebad allocCountbad span statebad stack sizefile
too largefinalizer waitgcstoptheworldinvalid syntaxis a directorylevel 2 haltedlevel 3 haltednil elem type!no module
datano such devicepollCache.lockprotocol errorruntime: full=s.allocCount= semaRoot queuestack overflowstopm
spinningstore64 failedsync.Cond.Waittext file busytoo many linkstoo many usersunexpected EOFunknown methodunreachable:
unsafe.Pointerwork.full ≠ 0  with GC prog\n476837158203125<invalid
Value>ASCII_Hex_DigitHanifi_RohingyaOther_LowercaseOther_UppercasePsalter_Pahlavi]\n\tmorebuf={pc:advertise
errorasyncpreemptoffforce gc (idle)key has expiredmalloc deadlockmisaligned maskmissing mcache?ms: gomaxprocs=network
is downno medium foundno such processnot a directoryrecovery failedruntime error: runtime: frame runtime: max =
runtime: min = runtimer: bad pscan missed a gstartm: m has pstopm holding p already; errno= mheap.sweepgen= not in
<and more>
```

3. Strings

Strings in Go are not null-terminated

+ All the strings are packed together
+ Makes it really hard to analyse

```
❯ strings hello | grep Hello
entersyscallgcBitsArenasgcpacertracehost is downillegal seekinvalid
slotlfstack.pushmadvdontneedmheapSpecialmspanSpecialnot pollableraceFiniLockreleasep: m=runtime: gp=runtime:
sp=short bufferspanSetSpinesweepWaiterstraceStringsuname failedwirep: p→m= ≠ sweepgen  MB) workers= called from
failed with  flushedWork  heap_marked= idlethreads= is nil, not  nStackRoots= s.spanclass= span.base()=
syscalltick= work.nproc=  work.nwait= , gp→status=, not pointer-byte block (3814697265625GC sweep
waitGunjala_GondiHello,
```

++ Make your own scripts to find strings/offset of strings

3. Strings/Slices

Strings are actually very simple: they are just read-only slices of bytes with a bit of extra syntactic support from the language.

- https://blog.golang.org/slices

3. Strings/Slices

A slice is not an array. A slice describes a piece of an array.

e.g.

```
var slice []byte = buffer[100:150]
```

- https://blog.golang.org/slices

3. Strings/Slices

- Arrays are an important building block in Go, but like the foundation of a building they are often hidden below more visible components.
- Arrays are not often seen in Go programs because the size of an array is part of its type, which limits its expressive power.

e.g.

```
var buffer [256]byte
```

- https://blog.golang.org/slices

3. Strings/Slices

In short,

- Strings are slices
- Slices describe a piece of an array
- Strings describe a piece of a `char/byte/rune` array

4. Structs

Structs in C:

```
struct Person
{
    char* name;
    int age;
} person;

struct Person person;
person.name = "You";
person.age = 100;
```

4. Structs

Structs in Go:

```
type Person struct {
     name string
     age int
}

Person{"You", 100}
```

4. Structs

Reference to strings in a C program:

```c
struct Person
{
    char* name;
    int age;
} person;

void foo(struct Person* person)
{}

int main()
{
    struct Person person;
    person.name = "You";
    person.age = 100;
    foo(&person);
}
```

```
❯ objdump -d -Mintel struct-c | grep -A14 \<main\>:
0000000000000675 <main>:
...
 68c:  48 8d 05 c1 00 00 00      lea    rax,[rip+0x c1]        # 754 "You"
 693:  48 89 45 e0               mov    QWORD PTR [rbp-0x20],rax
 697:  c7 45 e8 64 00 00 00      mov    DWORD PTR [rbp-0x18],0x64
 69e:  48 8d 45 e0               lea    rax,[rbp-0x20]
 6a2:  48 89 c7                  mov    rdi,rax
 6a5:  e8 c0 ff ff ff            call   66a <foo>
...
```

4. Structs

Reference to strings in a C program:

+ Values are placed next to each other on the stack (starting from `[rbp-0×10]`)

```
61a:  48 8d 05 b3 00 00 00      lea    rax,[rip+0×b3]                  # 6d4 "You"
621:  48 89 45 f0               mov    QWORD PTR [rbp-0×10],rax
625:  c7 45 f8 64 00 00 00      mov    DWORD PTR [rbp-0×8],0×64
```

```
gef➢  deref $rbp-0×10
0×00007fffffffe3e0│+0×0000: 0×00005555555546d4  →  0×3b031b0100756f59 ("You"?)   ← $rsp
0×00007fffffffe3e8│+0×0008: 0×0000000000000064 ("d"?)
```

4. Structs

Reference to strings in a C program:

+ Values are placed next to each other on the stack (starting from `[rbp-0×10]`)

```
61a:  48 8d 05 b3 00 00 00      lea    rax,[rip+0×b3]                  # 6d4 "You"
621:  48 89 45 f0               mov    QWORD PTR [rbp-0×10],rax
625:  c7 45 f8 64 00 00 00      mov    DWORD PTR [rbp-0×8],0×64
```

```
gef➢  deref $rbp-0×10
0×00007fffffffe3e0│+0×0000: 0×00005555555546d4  →  0×3b031b0100756f59 ("You"?)   ← $rsp
0×00007fffffffe3e8│+0×0008: 0×0000000000000064 ("d"?)
```

struct	
rbp-0×10	String
rsp-0×8	Age

4. Structs

Reference to structs in a Go program:

```go
package main

type Person struct {
    name string
    age int
}

func foo(person *Person) {}

func main() {
    foo(&Person{"You", 100})
}
```

```
❯ objdump -d -Mintel struct | grep -A44 \<main.main\>:
0000000000499080 <main.main>:
...
  45dcce:    48 8d 44 24 10          lea    rax,[rsp+0×10]
  45dcd3:    48 89 44 24 08          mov    QWORD PTR [rsp+0×8],rax
  45dcd8:    84 00                   test   BYTE PTR [rax],al
  45dcda:    48 c7 44 24 18 03 00    mov    QWORD PTR [rsp+0×18],0×3
  45dce1:    00 00
  45dce3:    48 8d 0d 20 62 01 00    lea    rcx,[rip+0×16220]    # 473f0a
  45dcea:    48 89 4c 24 10          mov    QWORD PTR [rsp+0×10],rcx
  45dcef:    84 00                   test   BYTE PTR [rax],al
  45dcf1:    48 c7 44 24 20 64 00    mov    QWORD PTR [rsp+0×20],0×64
  45dcf8:    00 00
  45dcfa:    48 89 04 24             mov    QWORD PTR [rsp],rax
  45dcfe:    66 90                   xchg   ax,ax
  45dd00:    e8 7b ff ff ff          call   45dc80 <main.foo>
...
```

4. Structs

Reference to strings in a Go program:

+ Values are placed next to each other on the stack (similar to C program)

```
...
 45dcda:    48 c7 44 24 18 03 00      mov    QWORD PTR [rsp+0×18],0×3
 45dce1:    00 00
 45dce3:    48 8d 0d 20 62 01 00      lea    rcx,[rip+0×16220]    # 473f0a "You"
 45dcea:    48 89 4c 24 10            mov    QWORD PTR [rsp+0×10],rcx
 45dcef:    84 00                     test   BYTE PTR [rax],al
 45dcf1:    48 c7 44 24 20 64 00      mov    QWORD PTR [rsp+0×20],0×64
 45dcf8:    00 00
 45dcfa:    48 89 04 24               mov    QWORD PTR [rsp],rax
...
 45dd00:    e8 7b ff ff ff            call   45dc80 <main.foo>
```

4. Structs

Reference to strings in a Go program:

+ Values are placed next to each other on the stack (similar to C program)

```
...
  45dcda:    48 c7 44 24 18 03 00      mov    QWORD PTR [rsp+0x18],0x3
  45dce1:    00 00
  45dce3:    48 8d 0d 20 62 01 00      lea    rcx,[rip+0x16220]    # 473f0a "You"
  45dcea:    48 89 4c 24 10            mov    QWORD PTR [rsp+0x10],rcx
  45dcef:    84 00                     test   BYTE PTR [rax],al
  45dcf1:    48 c7 44 24 20 64 00      mov    QWORD PTR [rsp+0x20],0x64
  45dcf8:    00 00
  45dcfa:    48 89 04 24               mov    QWORD PTR [rsp],rax
...
  45dd00:    e8 7b ff ff ff            call   45dc80 <main.foo>
```

```
gef➢  deref $rsp
0x000000c00002c750│+0x0000: 0x000000c00002c760  →  0x0000000000473f0a  →  "You]: ... "    ← $rsp
```

4. Structs

Reference to strings in a Go program:

+ Values are placed next to each other on the stack (similar to C program)

```
gef➢  deref $rsp
0x000000c00002c750│+0x0000: 0x000000c00002c760  →  0x0000000000473f0a  →  "You]: ... "    ← $rsp
```

```
gef➢  deref 0x000000c00002c760
0x000000c00002c760│+0x0000: 0x0000000000473f0a  →  "You ... "  ← $rax      # name string
0x000000c00002c768│+0x0008: 0x0000000000000003                             # name length
0x000000c00002c770│+0x0010: 0x0000000000000064 ("d"?)                      # age
```

4. Structs

Reference to strings in a Go program:

+ Values are placed next to each other on the stack (similar to C program)

```
gef➢  deref $rsp
0x000000c00002c750│+0x0000: 0x000000c00002c760  →  0x0000000000473f0a  →  "You]: ... "    ← $rsp
```

```
gef➢  deref 0x000000c00002c760
0x000000c00002c760│+0x0000: 0x0000000000473f0a  →  "You ... "  ← $rax      # name string
0x000000c00002c768│+0x0008: 0x0000000000000003                              # name length
0x000000c00002c770│+0x0010: 0x0000000000000064 ("d"?)                       # age
```

struct	
+0x0:	Pointer to string contents
+0x8:	Length of string
+0x10:	Age

5. Goroutines

A goroutine is a lightweight thread managed by the Go runtime.

```
go f(x, y, z)
```

starts a new goroutine running

```
f(x, y, z)
```

The evaluation of f, x, y, and z happens in the current goroutine and the execution of f happens in the new goroutine.

5. Goroutines

Goroutines run in the same address space as the rest of the program.

5. Goroutines

Goroutines are instantiated with `runtime.newproc`

```go
package main

import (
    "fmt"
    "time"
)

func say(s string) {
    for i := 0; i < 5; i++ {
        time.Sleep(100 * time.Millisecond)
        fmt.Println(s)
    }
}

func main() {
    go say("world")
    say("hello")
}
```

```
hello
world
hello
hello
world
world
hello
hello
world
world
```

5. Goroutines

Goroutines are instantiated with `runtime.newproc`

```go
package main

import (
    "fmt"
    "time"
)

func say(s string) {
    for i := 0; i < 5; i++ {
        time.Sleep(100 * time.Millisecond)
        fmt.Println(s)
    }
}

func main() {
    go say("world")
    say("hello")
}
```

```
❯ objdump -d -Mintel goroutine | grep -A29 \<main.main\>:
00000000004994a0 <main.main>:
...
  4994c4:    48 8d 05 ed b5 02 00      lea    rax,[rip+0x2b5ed]
# 4c4ab8 <go.func.*+0x6f>

  4994cb:    48 89 44 24 08            mov    QWORD PTR [rsp+0x8],rax
  4994d0:    48 8d 05 5f 3a 02 00      lea    rax,[rip+0x23a5f]
# 4bcf36 <go.string.*+0x346>

  4994d7:    48 89 44 24 10            mov    QWORD PTR [rsp+0x10],rax
  4994dc:    48 c7 44 24 18 05 00      mov    QWORD PTR [rsp+0x18],0x5
  4994e3:    00 00
  4994e5:    e8 76 38 fa ff            call   43cd60 <runtime.newproc>
...
```

5. Goroutines

Goroutines are instantiated with `runtime.newproc`

+ The goroutine function and its arguments are passed to `runtime.newproc`

```
❯ objdump -d -Mintel goroutine | grep -A29 \<main.main\>:
00000000004994a0 <main.main>:
...
  4994c4:    48 8d 05 ed b5 02 00      lea    rax,[rip+0x2b5ed]        # 4c4ab8 <go.func.*+0x6f>

  4994cb:    48 89 44 24 08            mov    QWORD PTR [rsp+0x8],rax
  4994d0:    48 8d 05 5f 3a 02 00      lea    rax,[rip+0x23a5f]        # 4bcf36 <go.string.*+0x346>

  4994d7:    48 89 44 24 10            mov    QWORD PTR [rsp+0x10],rax
  4994dc:    48 c7 44 24 18 05 00      mov    QWORD PTR [rsp+0x18],0x5
  4994e3:    00 00
  4994e5:    e8 76 38 fa ff            call   43cd60 <runtime.newproc>
...
```

5. Goroutines

Goroutines are instantiated with `runtime.newproc`

+ The goroutine function and its arguments are passed to `runtime.newproc`

```
❯ objdump -d -Mintel goroutine | grep -A29 \<main.main\>:
00000000004994a0 <main.main>:
...
  4994c4:     48 8d 05 ed b5 02 00       lea    rax,[rip+0x2b5ed]        # 4c4ab8 <go.func.*+0x6f>
...
```

5. Goroutines

Goroutines are instantiated with `runtime.newproc`

+ The goroutine function and its arguments are passed to `runtime.newproc`

```
❯ objdump -d -Mintel goroutine | grep -A29 \<main.main\>:
00000000004994a0 <main.main>:
...
  4994c4:     48 8d 05 ed b5 02 00       lea    rax,[rip+0x2b5ed]        # 4c4ab8 <go.func.*+0x6f>
...
```

```
gef➢  deref 0x4c4ab8
0x00000000004c4ab8│+0x0000: 0x0000000000499380  →  <main.say+0> mov rcx, QWORD PTR fs:0xfffffffffffffff8
```

5. Goroutines

Goroutines are instantiated with `runtime.newproc`

+ The goroutine function and its arguments are passed to `runtime.newproc`

```
❯ objdump -d -Mintel goroutine | grep -A29 \<main.main\>:
00000000004994a0 <main.main>:
...
  4994c4:     48 8d 05 ed b5 02 00       lea    rax,[rip+0x2b5ed]        # 4c4ab8 <go.func.*+0x6f>
...
```

```
gef➢  deref 0x4c4ab8
0x00000000004c4ab8│+0x0000: 0x0000000000499380  →  <main.say+0> mov rcx, QWORD PTR fs:0xfffffffffffffff8
```

```
❯ objdump -d -Mintel goroutine | grep main.say\>:
0000000000499380 <main.say>:
```

5. Goroutines

Convenience functions in GDB

```
gef➢  info goroutines
* 0×1 running  runtime.asyncPreempt2
  0×2 waiting  runtime.gopark
  0×11 waiting  runtime.gopark
  0×12 waiting  runtime.gopark
  0×21 waiting  runtime.gopark
  0×22 waiting  runtime.gopark
```

```
gef➢  goroutine 0×22 bt
#0  runtime.gopark (unlockf={void (runtime.g *, void *, bool *)} 0×c000067f10, lock=0×c00005a050,
reason=0×13, traceEv=0×13, traceskip=0×1) at /usr/local/go/src/runtime/proc.go:307
#1  0×000000000046071f in time.Sleep (ns=0×5f5e100) at /usr/local/go/src/runtime/time.go:188
#2  0×00000000004993c6 in main.say (s="world") at /tmp/goroutine.go:10
#3  0×0000000000463461 in runtime.goexit () at /usr/local/go/src/runtime/asm_amd64.s:1374
#4  0×00000000004bcf36 in string.* ()
#5  0×0000000000000005 in ?? ()
#6  0×0000000000000000 in ?? ()
```

6. Symbols

Ask the compiler to strip symbols

+ All symbols should be removed

```go
package main

func main() {
	panic("Hello, 世界")
}
```

```
❯ go build panic.go
❯ file panic
panic: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped

❯ go build -o panic -ldflags "-s -w" panic.go
❯ file panic
panic: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, stripped
```

6. Symbols

Symbols were stripped

+ But the panic stack trace shows `main.main`

```go
package main

func main() {
	panic("Hello, 世界")
}
```

```
❯ file panic
panic: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, stripped

❯ ./panic
panic: Hello, 世界

goroutine 1 [running]:
main.main()
	/tmp/panic.go:4 +0x39
```

6. Symbols

Symbols are stored in `.pclntab`

+ Binary is not completely stripped of symbols

```go
package main

func main() {
	panic("Hello, 世界")
}
```

```
❯ file panic
panic: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, stripped

❯ ./panic
panic: Hello, 世界

goroutine 1 [running]:
main.main()
	/tmp/panic.go:4 +0×39
```

6. Symbols

Symbols are stored in `.pclntab`

+ We can inspect the contents

```
gef➢  info file
Symbols from "/tmp/hello".
Native process:
      Using the running image of child LWP 2336.
      While running this, GDB does not access memory from ...
Local exec file:
      `/tmp/hello', file type elf64-x86-64.
      Entry point: 0x464820

...
      0x00000000004de680 - 0x000000000053dfab is .gopclntab
...

gef➢  set $pclntab=0x00000000004de680
```

6. Symbols

Symbols are stored in `.pclntab`

- We can inspect the contents
- Each entry is 16 bytes wide

```
gef➢  deref $pclntab
0x00000000004de680│+0x0000: 0x08010000fffffffb       # Section header
0x00000000004de688│+0x0008: 0x00000000000006e9       # Section size

0x00000000004de690│+0x0010: 0x0000000000401000  →  <internal/cpu.Initialize+0> mov rcx, ...
0x00000000004de698│+0x0018: 0x0000000000006eb0

0x00000000004de6a0│+0x0020: 0x0000000000401060  →  <internal/cpu.processOptions+0> mov rcx, ...
0x00000000004de6a8│+0x0028: 0x0000000000006f38

0x00000000004de6b0│+0x0030: 0x00000000004017c0  →  <internal/cpu.indexByte+0> mov rax, QWORD PTR [rsp+0x10]
0x00000000004de6b8│+0x0038: 0x0000000000007078

0x00000000004de6c0│+0x0040: 0x0000000000401800  →  <internal/cpu.doinit+0> mov rcx, ...
0x00000000004de6c8│+0x0048: 0x00000000000070e8
```

6. Symbols

Symbols are stored in `.pclntab`

+ Each entry is 16 bytes wide

```
gef➢  deref $pclntab
...
0x00000000004de690│+0x0010: 0x0000000000401000  →  <internal/cpu.Initialize+0> mov rcx, ...
0x00000000004de698│+0x0018: 0x0000000000006eb0       # offset
...
```

6. Symbols

Symbols are stored in `.pclntab`

+ Each entry is 16 bytes wide

```
gef➢  deref $pclntab
...
0x00000000004de690│+0x0010: 0x0000000000401000  →  <internal/cpu.Initialize+0> mov rcx, ...
0x00000000004de698│+0x0018: 0x0000000000006eb0       # offset
...
```

```
gef➢  deref $pclntab+0x0000000000006eb0
0x00000000004e5530│+0x0000: 0x0000000000401000  →  <internal/cpu.Initialize+0>
0x00000000004e5538│+0x0008: 0x0000001000006ef0
```

6. Symbols

Symbols are stored in `.pclntab`

+ Each entry is 16 bytes wide

```
gef➢  deref $pclntab
...
0x00000000004de690 | +0x0010: 0x0000000000401000  →  <internal/cpu.Initialize+0> mov rcx, ...
0x00000000004de698 | +0x0018: 0x0000000000006eb0       # offset
...
```

```
gef➢  deref $pclntab+0x0000000000006eb0
0x00000000004e5530 | +0x0000: 0x0000000000401000  →  <internal/cpu.Initialize+0>
0x00000000004e5538 | +0x0008: 0x0000001000006ef0

gef➢  hexdump byte $pclntab+0x6ef0
0x00000000004e5570 <runtime.pclntab+6ef0>    69 6e 74 65 72 6e 61 6c 2f 63 70 75 2e 49 6e 69    internal/cpu.Ini
0x00000000004e5580 <runtime.pclntab+6f00>    74 69 61 6c 69 7a 65 00 02 13 30 33 2f 08 00 04    tialize ... 03/ ...
```

6. Symbols

With `.pclntab`, we can easily find all symbols in stripped binaries.

Tools:

+ **IDA -** https://github.com/danigargu/ida-scripts/blob/master/go_stripped_helper.py
+ **Ghidra -** https://github.com/ghidraninja/ghidra_scripts/blob/master/golang_renamer.py
+ **r2/rz -** https://github.com/zlowram/radare2-scripts/tree/master/go_helpers

7. Types/Interfaces

With `.typelink`, we can also easily find type information in stripped binaries.

Tools:

+ **IDA -** https://github.com/danigargu/ida-scripts/blob/master/go_stripped_helper.py
+ **r2/rz -** https://github.com/zlowram/radare2-scripts/tree/master/go_helpers

Takeaway

+ **Fallback to `brain.exe` when decompiler doesn’t do well**
+ **Identify common patterns of assembly code**

References/Further readings

+ https://unit42.paloaltonetworks.com/the-gopher-in-the-room-analysis-of-golang-malware-in-the-wild/
+ https://www.youtube.com/watch?v=KFitDf_XPYE
+ https://tkmr.hatenablog.com/entry/2016/06/07/113641
+ https://golang.org/src/runtime/runtime-gdb.py
+ https://dr-knz.net/go-calling-convention-x86-64.html#strings-and-slices-use-two-and-three-words

Credits

Govtech for Zoom

Cher Boon (@Gladiator) for coordinating