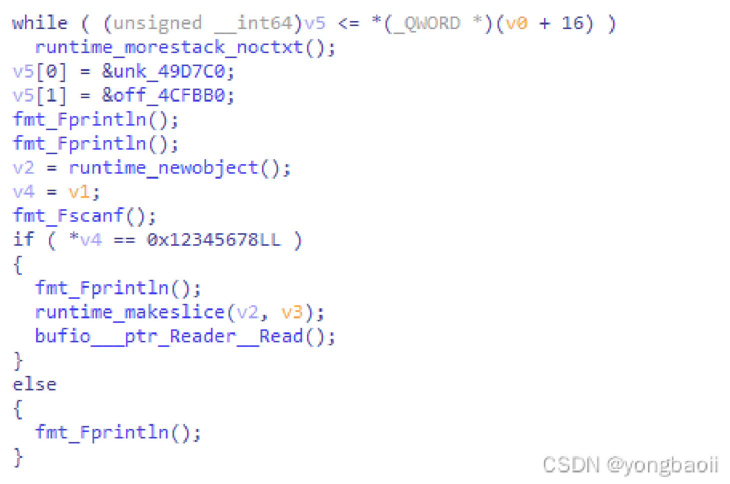
go pwn 2022 虎符 gogogo

原创

yongbaoii 于 2022-04-09 20:37:22 发布 4099 收藏
分类专栏： CTF 文章标签： 网络安全

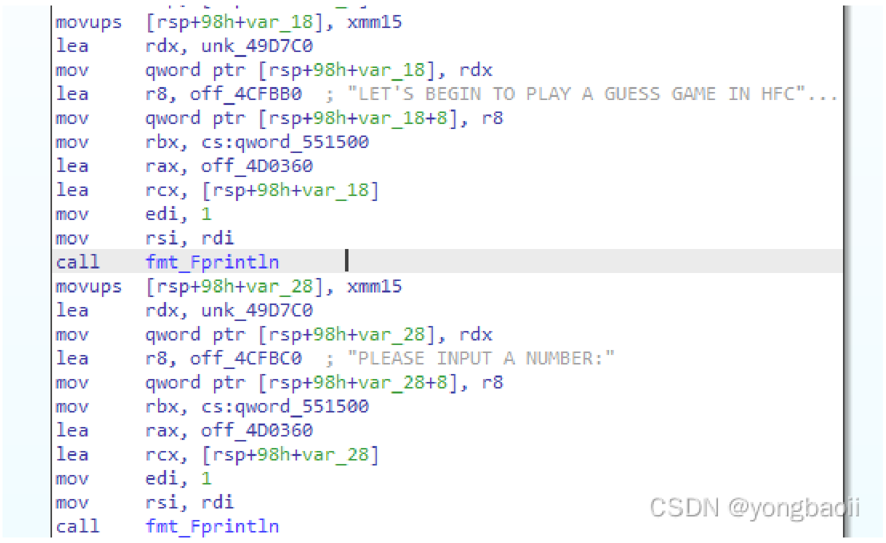
CTF 专栏收录该内容
213 篇文章 7 订阅
订阅专栏

刚刚看了17年seccon的baby_stack
看着这个感觉好亲切。



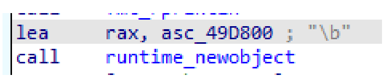
```
while ( (unsigned __int64)v5 <= *(_QWORD *)(v0 + 16) )
  runtime_morestack_noctxt();
v5[0] = &unk_49D7C0;
v5[1] = &off_4CFBB0;
fmt_Fprintln();
fmt_Fprintln();
v2 = runtime_newobject();
v4 = v1;
fmt_Fscanf();
if ( *v4 == 0x12345678LL )
{
  fmt_Fprintln();
  runtime_makeslice(v2, v3);
  bufio___ptr_Reader__Read();
}
else
{
  fmt_Fprintln();
}
```

就这么点逻辑。

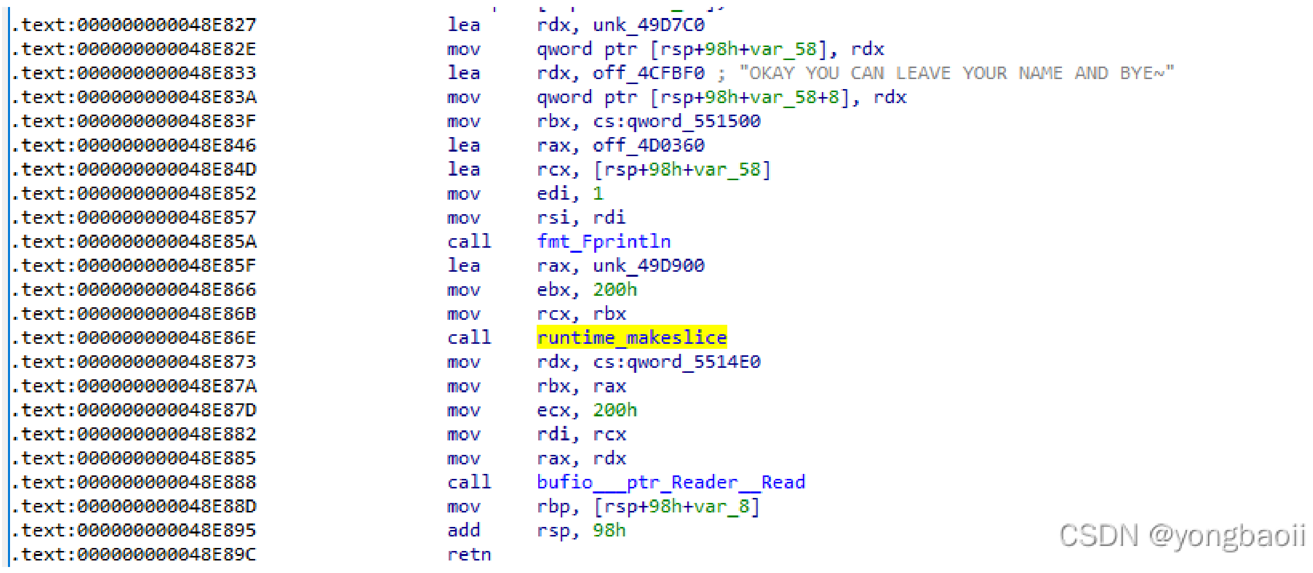


```
movups  [rsp+98h+var_18], xmm15
lea     rdx, unk_49D7C0
mov     qword ptr [rsp+98h+var_18], rdx
lea     r8, off_4CFBB0  ; "LET'S BEGIN TO PLAY A GUESS GAME IN HFC"...
mov     qword ptr [rsp+98h+var_18+8], r8
mov     rbx, cs:qword_551500
lea     rax, off_4D0360
lea     rcx, [rsp+98h+var_18]
mov     edi, 1
mov     rsi, rdi
call    fmt_Fprintln
movups  [rsp+98h+var_28], xmm15
lea     rdx, unk_49D7C0
mov     qword ptr [rsp+98h+var_28], rdx
lea     r8, off_4CFBC0  ; "PLEASE INPUT A NUMBER:"
mov     qword ptr [rsp+98h+var_28+8], r8
mov     rbx, cs:qword_551500
lea     rax, off_4D0360
lea     rcx, [rsp+98h+var_28]
mov     edi, 1
mov     rsi, rdi
call    fmt_Fprintln
```

两个输出。


```
lea     rax, asc_49D800 ; "\b"
call    runtime_newobject
```

这个地址当参数建了个结构体。


```
.text:000000000048E827                 lea     rdx, unk_49D7C0
.text:000000000048E82E                 mov     qword ptr [rsp+98h+var_58], rdx
.text:000000000048E833                 lea     rdx, off_4CFBF0 ; "OKAY YOU CAN LEAVE YOUR NAME AND BYE~"
.text:000000000048E83A                 mov     qword ptr [rsp+98h+var_58+8], rdx
.text:000000000048E83F                 mov     rbx, cs:qword_551500
.text:000000000048E846                 lea     rax, off_4D0360
.text:000000000048E84D                 lea     rcx, [rsp+98h+var_58]
.text:000000000048E852                 mov     edi, 1
.text:000000000048E857                 mov     rsi, rdi
.text:000000000048E85A                 call    fmt_Fprintln
.text:000000000048E85F                 lea     rax, unk_49D900
.text:000000000048E866                 mov     ebx, 200h
.text:000000000048E86B                 mov     rcx, rbx
.text:000000000048E86E                 call    runtime_makeslice
.text:000000000048E873                 mov     rdx, cs:qword_5514E0
.text:000000000048E87A                 mov     rbx, rax
.text:000000000048E87D                 mov     ecx, 200h
.text:000000000048E882                 mov     rdi, rcx
.text:000000000048E885                 mov     rax, rdx
.text:000000000048E888                 call    bufio___ptr_Reader__Read
.text:000000000048E88D                 mov     rbp, [rsp+98h+var_8]
.text:000000000048E895                 add     rsp, 98h
.text:000000000048E89C                 retn
```

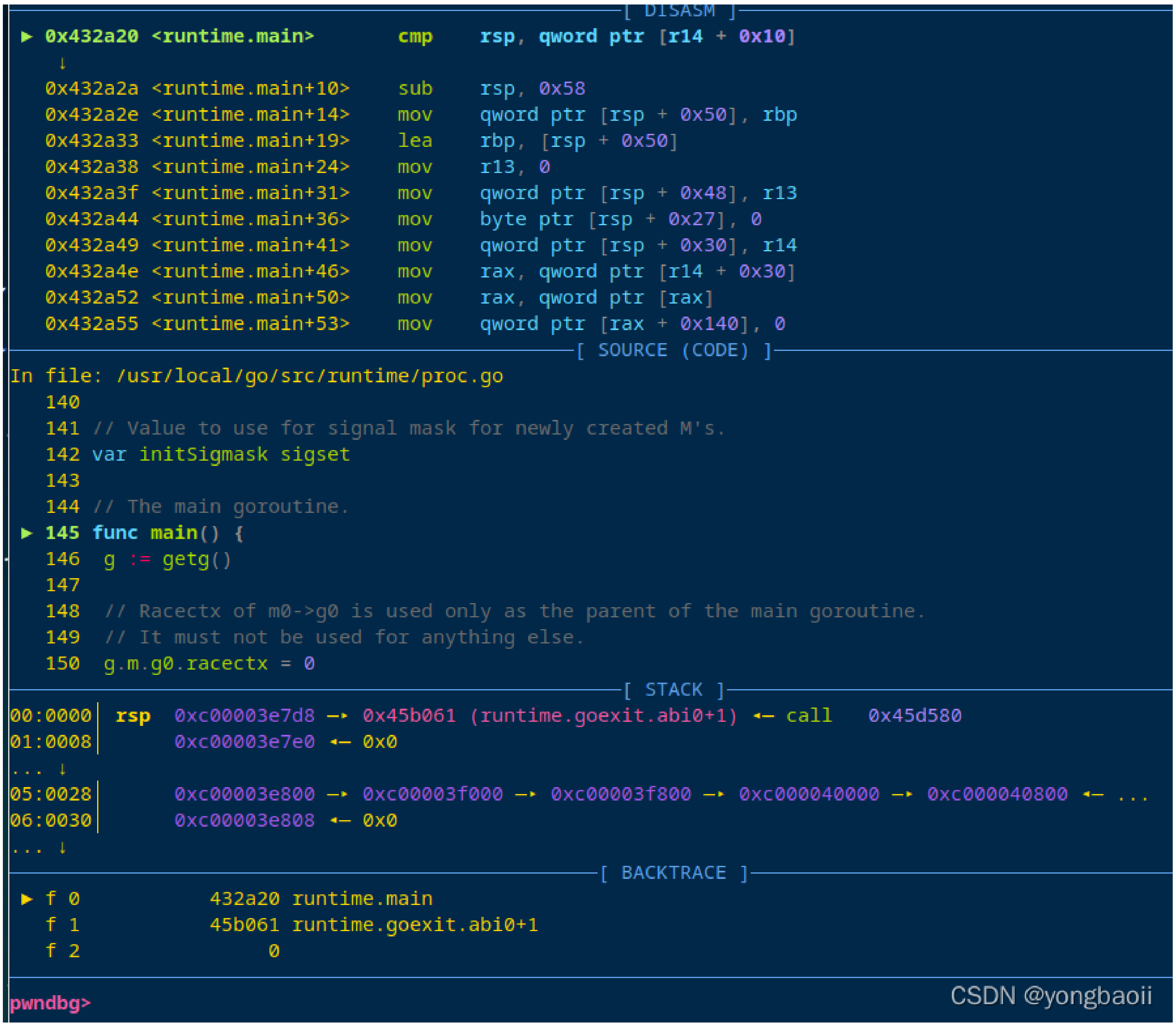
然后创建了个切片大小0x200
读0x200
一片祥和与美好…

gdb跑一下发下断不下来…
它只会在一些奇奇怪怪的函数里面去跑
那首先怀疑是改了go的符号表。

go的符号表一般都在gopclntab段
可以去看大佬系列文章

那就只能老老实实动调来瞅了嘛。
上网了解到go语言的main函数是通过runtime_main函数创建线程然后调用的
下断点下在那里之后还是不大好调
不知道啥时候进主函数。

我们直接掏出我们之前在本地写的逆向小程序
gdb在runtime.main打个断点
不仅可以知道函数在源码哪个位置
还可以直接调
非常好用。



```
────────────────────────────────[ DISASM ]────────────────────────────────
 ► 0x432a20 <runtime.main>       cmp    rsp, qword ptr [r14 + 0x10]
    ↓
   0x432a2a <runtime.main+10>    sub    rsp, 0x58
   0x432a2e <runtime.main+14>    mov    qword ptr [rsp + 0x50], rbp
   0x432a33 <runtime.main+19>    lea    rbp, [rsp + 0x50]
   0x432a38 <runtime.main+24>    mov    r13, 0
   0x432a3f <runtime.main+31>    mov    qword ptr [rsp + 0x48], r13
   0x432a44 <runtime.main+36>    mov    byte ptr [rsp + 0x27], 0
   0x432a49 <runtime.main+41>    mov    qword ptr [rsp + 0x30], r14
   0x432a4e <runtime.main+46>    mov    rax, qword ptr [r14 + 0x30]
   0x432a52 <runtime.main+50>    mov    rax, qword ptr [rax]
   0x432a55 <runtime.main+53>    mov    qword ptr [rax + 0x140], 0
─────────────────────────────[ SOURCE (CODE) ]─────────────────────────────
In file: /usr/local/go/src/runtime/proc.go
   140
   141 // Value to use for signal mask for newly created M's.
   142 var initSigmask sigset
   143
   144 // The main goroutine.
 ► 145 func main() {
   146  g := getg()
   147
   148  // Racectx of m0->g0 is used only as the parent of the main goroutine.
   149  // It must not be used for anything else.
   150  g.m.g0.racectx = 0
────────────────────────────────[ STACK ]────────────────────────────────
00:0000│ rsp  0xc00003e7d8 —▸ 0x45b061 (runtime.goexit.abi0+1) ◂— call   0x45d580
01:0008│      0xc00003e7e0 ◂— 0x0
... ↓
05:0028│      0xc00003e800 —▸ 0xc00003f000 —▸ 0xc00003f800 —▸ 0xc000040000 —▸ 0xc000040800 ◂— ...
06:0030│      0xc00003e808 ◂— 0x0
... ↓
──────────────────────────────[ BACKTRACE ]──────────────────────────────
 ► f 0         432a20 runtime.main
   f 1         45b061 runtime.goexit.abi0+1
   f 2              0
─────────────────────────────────────────────────────────────────────────
pwndbg>
```

CSDN @yongbaoii

就能把源码掏出来。

```go
func main() {
 g := getg()

 // Racectx of m0->g0 is used only as the parent of the main goroutine.
 // It must not be used for anything else.
 g.m.g0.racectx = 0

 // Max stack size is 1 GB on 64-bit, 250 MB on 32-bit.
 // Using decimal instead of binary GB and MB because
 // they look nicer in the stack overflow failure message.
 if goarch.PtrSize == 8 {
  maxstacksize = 1000000000
 } else {
  maxstacksize = 250000000
```

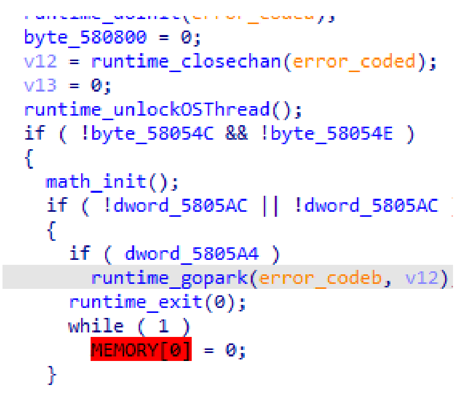
```
}

// An upper limit for max stack size. Used to avoid random crashes
// after calling SetMaxStack and trying to allocate a stack that is too big,
// since stackalloc works with 32-bit sizes.
maxstackceiling = 2 * maxstacksize

// Allow newproc to start new Ms.
mainStarted = true

if GOARCH != "wasm" { // no threads on wasm yet, so no sysmon
 systemstack(func() {
  newm(sysmon, nil, -1)
 })
}

// Lock the main goroutine onto this, the main OS thread,
// during initialization. Most programs won't care, but a few
// do require certain calls to be made by the main thread.
// Those can arrange for main.main to run in the main thread
// by calling runtime.LockOSThread during initialization
// to preserve the lock.
lockOSThread()

if g.m != &m0 {
 throw("runtime.main not on m0")
}

// Record when the world started.
// Must be before doInit for tracing init.
runtimeInitTime = nanotime()
if runtimeInitTime == 0 {
 throw("nanotime returning zero")
}

if debug.inittrace != 0 {
 inittrace.id = getg().goid
 inittrace.active = true
}

doInit(&runtime_inittask) // Must be before defer.

// Defer unlock so that runtime.Goexit during init does the unlock too.
needUnlock := true
defer func() {
 if needUnlock {
  unlockOSThread()
 }
}()

gcenable()

main_init_done = make(chan bool)
if iscgo {
 if _cgo_thread_start == nil {
  throw("_cgo_thread_start missing")
 }
 if GOOS != "windows" {
  if _cgo_setenv == nil {
    throw("_cgo_setenv missing")
   }
   if _cgo_unsetenv == nil {
    throw("_cgo_unsetenv missing")
   }
  }
  if _cgo_notify_runtime_init_done == nil {
   throw("_cgo_notify_runtime_init_done missing")
  }
  // Start the template thread in case we enter Go from
  // a C-created thread and need to create a new thread.
  startTemplateThread()
  cgocall(_cgo_notify_runtime_init_done, nil)
 }

 doInit(&main_inittask)

 // Disable init tracing after main init done to avoid overhead
 // of collecting statistics in malloc and newproc
 inittrace.active = false

 close(main_init_done)

 needUnlock = false
 unlockOSThread()

 if isarchive || islibrary {
  // A program compiled with -buildmode=c-archive or c-shared
  // has a main, but it is not executed.
  return
 }
 fn := main_main // make an indirect call, as the linker doesn't know the address of the main package when layin
g down the runtime
 fn()
 if raceenabled {
  racefini()
 }

 // Make racy client program work: if panicking on
 // another goroutine at the same time as main returns,
 // let the other goroutine finish printing the panic trace.
 // Once it does, it will exit. See issues 3934 and 20018.
 if atomic.Load(&runningPanicDefers) != 0 {
  // Running deferred functions should not take long.
  for c := 0; c < 1000; c++ {
   if atomic.Load(&runningPanicDefers) == 0 {
    break
   }
   Gosched()
  }
 }
 if atomic.Load(&panicking) != 0 {
  gopark(nil, nil, waitReasonPanicWait, traceEvGoStop, 1)
 }

 exit(0)
 for {
  var x *int32
  *x = 0
 }
```

```
}
```

```
if isarchive || islibrary {
  // A program compiled with -buildmode=c-archive or c-shared
  // has a main, but it is not executed.
  return
 }
 fn := main_main // make an indirect call, as the linker doesn't know the address of the main package when laying down the runtime
 fn()
 if raceenabled {
  racefini()
 }
```

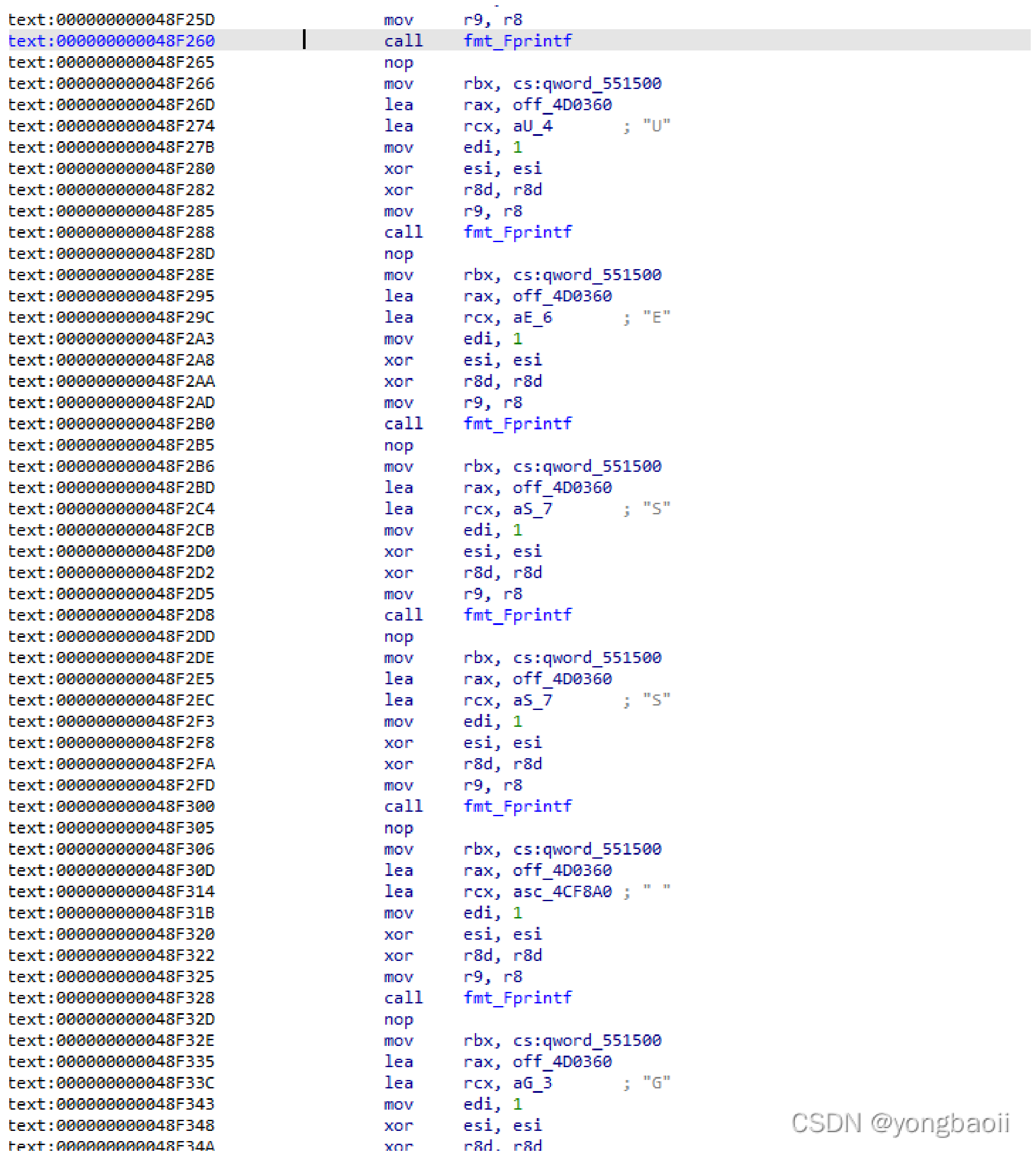
然后在这里发现了main_main

对标这道题



```
  byte_580800 = 0;
  v12 = runtime_closechan(error_coded);
  v13 = 0;
  runtime_unlockOSThread();
  if ( !byte_58054C && !byte_58054E )
  {
    math_init();
    if ( !dword_5805AC || !dword_5805AC
    {
      if ( dword_5805A4 )
        runtime_gopark(error_codeb, v12)
      runtime_exit(0);
      while ( 1 )
        MEMORY[0] = 0;
    }
```

就会发现它把主函数的名改成了math_init
非常可恶。



```
text:000000000048F25D                 mov     r9, r8
text:000000000048F260                 call    fmt_Fprintf
text:000000000048F265                 nop
text:000000000048F266                 mov     rbx, cs:qword_551500
text:000000000048F26D                 lea     rax, off_4D0360
text:000000000048F274                 lea     rcx, aU_4       ; "U"
text:000000000048F27B                 mov     edi, 1
text:000000000048F280                 xor     esi, esi
text:000000000048F282                 xor     r8d, r8d
text:000000000048F285                 mov     r9, r8
text:000000000048F288                 call    fmt_Fprintf
text:000000000048F28D                 nop
text:000000000048F28E                 mov     rbx, cs:qword_551500
text:000000000048F295                 lea     rax, off_4D0360
text:000000000048F29C                 lea     rcx, aE_6       ; "E"
text:000000000048F2A3                 mov     edi, 1
text:000000000048F2A8                 xor     esi, esi
text:000000000048F2AA                 xor     r8d, r8d
text:000000000048F2AD                 mov     r9, r8
text:000000000048F2B0                 call    fmt_Fprintf
text:000000000048F2B5                 nop
text:000000000048F2B6                 mov     rbx, cs:qword_551500
text:000000000048F2BD                 lea     rax, off_4D0360
text:000000000048F2C4                 lea     rcx, aS_7       ; "S"
text:000000000048F2CB                 mov     edi, 1
text:000000000048F2D0                 xor     esi, esi
text:000000000048F2D2                 xor     r8d, r8d
text:000000000048F2D5                 mov     r9, r8
text:000000000048F2D8                 call    fmt_Fprintf
text:000000000048F2DD                 nop
text:000000000048F2DE                 mov     rbx, cs:qword_551500
text:000000000048F2E5                 lea     rax, off_4D0360
text:000000000048F2EC                 lea     rcx, aS_7       ; "S"
text:000000000048F2F3                 mov     edi, 1
text:000000000048F2F8                 xor     esi, esi
text:000000000048F2FA                 xor     r8d, r8d
text:000000000048F2FD                 mov     r9, r8
text:000000000048F300                 call    fmt_Fprintf
text:000000000048F305                 nop
text:000000000048F306                 mov     rbx, cs:qword_551500
text:000000000048F30D                 lea     rax, off_4D0360
text:000000000048F314                 lea     rcx, asc_4CF8A0 ; " "
text:000000000048F31B                 mov     edi, 1
text:000000000048F320                 xor     esi, esi
text:000000000048F322                 xor     r8d, r8d
text:000000000048F325                 mov     r9, r8
text:000000000048F328                 call    fmt_Fprintf
text:000000000048F32D                 nop
text:000000000048F32E                 mov     rbx, cs:qword_551500
text:000000000048F335                 lea     rax, off_4D0360
text:000000000048F33C                 lea     rcx, aG_3       ; "G"
text:000000000048F343                 mov     edi, 1
text:000000000048F348                 xor     esi, esi
text:000000000048F34A                 xor     r8d, r8d
```

math_init里面输出字符都不用字符串的
就你拿字符串交叉引用也找不到的
好家伙。

下面的步骤呢就是寻常步骤
据说是个游戏
又得嘎嘎逆向
逆向完其实又是个栈溢出
没啥意思了就

重点我感觉这道题能学到的还是如何处理改过符号表这件事。游戏据说网上可以直接搜的到。
剩下的逆向劳动就不做了。
还有别的事要忙。
贴几个大佬exp
可以去大佬那里看看具体exp是啥。

虎符ctf2022
HFCTF（虎符）2022 Pwn gogogo WriteUp