

whaleCTF-30days-隐写【第二期】-彩虹糖-writeup

原创

sec小玖 于 2018-07-24 11:56:40 发布 789 收藏 1

分类专栏: [CTF 隐写](#) 文章标签: [CTF 隐写](#) [汇编capstone](#) [bmp像素](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/afuzong4267/article/details/81178405>

版权



[CTF 同时被 2 个专栏收录](#)

6 篇文章 0 订阅

订阅专栏



[隐写](#)

3 篇文章 0 订阅

订阅专栏

题目:

彩虹糖公司的机器出了问题, 不断的在他们的宣传海报里插入彩虹糖, 最后公司发现, 插入的竟然是公司机密, 请帮助他们找到所有机密。答案格式whaleCTF{xxx}

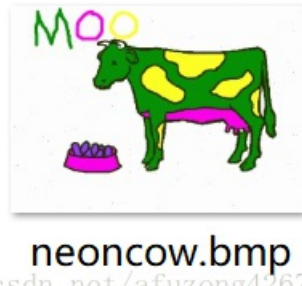
打开压缩包发现6张图片, 放大图片能看到很多彩色小点, 也就是题目中说的彩虹糖, 仔细观察其他图片也都有小点。



使用binwalk、stegsolve、winhex、strings等都没有发现有用的信息, 只发现如下内容:

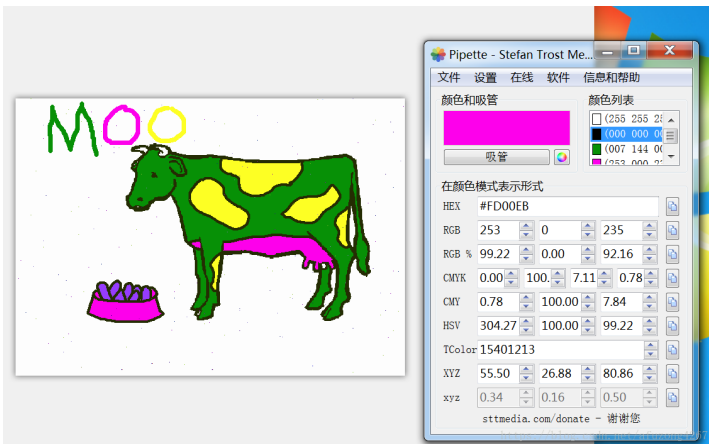
```
sec@LAPTOP-0988M64E:/mnt/f/whaleCTF/隐写【第二期】/彩虹糖【初级】$ strings neoncow.bmp
,$this is not the flag you're looking for, but keep looking!! :: this is not the flag you're looking for, b
```

已经没有思路了, 经大神指点, 考虑图片的像素, 使用PIL库将图片中的彩虹糖(也就是彩色的小点提取出来), 这里可以使用这两张像素颜色较少的图片进行提取:



<https://blog.csdn.net/afuzong4267>

我们使用neoncow.bmp这张图片，这张图片中包含了绿、黄、粉、紫、黑、白六种颜色，使用取色器依次将这六种颜色的16进制RGB值取出来。



接下来，使用python遍历图片，把这六种颜色去除：

```

from PIL import Image
import operator
import pprint

img = Image.open("neoncow.bmp")

colors = {}

pixels = img.load()
w,h = img.size

for y in range(h):
    for x in range(w):
        if pixels[x,y] not in [
            (0xfd, 0xfd, 0xfd),
            (0x07, 0x90, 0x06),
            (0xfd, 0x00, 0xeb),
            (0xfd, 0xff, 0x24),
            (0x96, 0x3c, 0xfd),
            (0x24, 0x2c, 0x03),
        ]:

            pixels_repr = ''.join(["%02x" % _ for _ in pixels[x,y]])
            if pixels_repr not in colors:
                colors[pixels_repr] = 0
            colors[pixels_repr] += 1

pprint.pprint(sorted(colors.items(), key=operator.itemgetter(1))[::-1])

```

运行程序即可提取像素内容：

```

Cmder
λ py2 count_neoncow.py
[('90c031', 47),
 ('9080cd', 47),
 ('9004b0', 46),
 ('900000', 3),
 ('022980', 3),
 ('012980', 3),
 ('310180', 2),
 ('90db31', 2),
 ('032980', 2),
 ('020180', 2),
 ('202980', 2),
 ('060180', 2),
 ('322980', 1),
 ('340180', 1),
 ('909059', 1),
 ('030180', 1),
 ('2c2980', 1),
 ('320180', 1),

```

仍然没有思路，小白表示很无助，还得大神指点，观察压缩包文件名，叫ximage.zip，学名是execimage，脑洞不够大呀，完全想不到，考虑这个一个可执行的图片，于是和机器代码联系起来，发现在红色通道出现多处90，在蓝色通道出现多处80，先通过一个[在线的反编译工具](#)对90c031进行测试：

```
"\x90\xC0\x31"
```

Array Literal:

```
{ 0x90, 0xC0, 0x31 }
```

Disassembly:

```
0:  90                nop
1:  c0                (bad)
2:  31                https://blog.byte0x31.com/4267
```

发现结果并不是很对，考虑会不会是倒序存储，尝试将RGB改为BGR，及对31c090进行反编译

```
"\x31\xC0\x90"
```

Array Literal:

```
{ 0x31, 0xC0, 0x90 }
```

Disassembly:

```
0:  31 c0            xor     eax, eax
2:  90                https://blog.byte0x31.com/4267
```

出现了异或运算，说明思路应该没有错。

使用python脚本capstone库，对每段代码进行反汇编：

```
from capstone import *

from PIL import Image

md = Cs(CS_ARCH_X86, CS_MODE_32)

img = Image.open("neoncow.bmp")

pixels = img.load()
w,h = img.size

s = ""
for y in range(h):
    for x in range(w):
        if pixels[x,y] not in [
            (0xfd, 0xfd, 0xfd),
            (0x07, 0x90, 0x06),
            (0xfd, 0x00, 0xeb),
            (0xfd, 0xff, 0x24),
            (0x96, 0x3c, 0xfd),
            (0x24, 0x2c, 0x03),
        ]:
            s += ''.join([chr(_) for _ in pixels[x,y][::-1]])

for inst in md.disasm(s, 0):
    print "0x%x:\t%s\t%s" % (inst.address, inst.mnemonic, inst.op_str)
```

发现代码中都是对ecx进行赋值和加减，并且每次计算结束都调用系统中断输出，最后输出了ecx的内容。

Cmdr	Cmdr
<pre>λ py2 cap-cal.py 0x0: xor ebx, ebx 0x2: nop 0x3: int 0x80 0x5: nop 0x6: xor eax, eax 0x8: nop 0x9: inc al 0xb: nop 0xc: int 0x80 0xe: nop 0xf: mov al, 4 0x11: nop 0x12: xor eax, eax 0x14: nop 0x15: int 0x80 0x17: nop 0x18: sub byte ptr [ecx], 0x20</pre>	<pre>0x21e: nop 0x21f: xor eax, eax 0x221: nop 0x222: add byte ptr [ecx], 0x2a 0x225: mov edx, 1 0x22a: nop 0x22b: xor edx, edx 0x22d: nop 0x22e: mov byte ptr [ecx], 0 0x231: pop ecx 0x232: nop 0x233: nop 0x234: mov ebx, 1 0x239: nop 0x23a: call 0x23f 0x23f: nop 0x240: xor ebx, ebx 0x242: nop</pre>

将输出结果重定向到sam.out文件，再对ecx的值进行计算，代码如下：

```
f = open("sam.out")

lines = f.readlines()

s = ""
ecx = 0

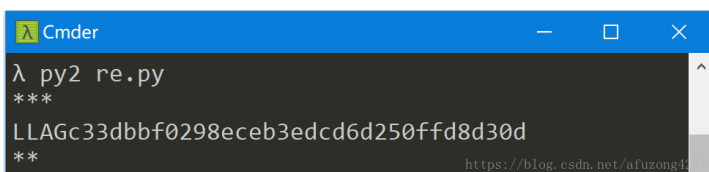
for l in lines[::-1]: # reverse the instructions
    if 'byte ptr [ecx]' in l:
        tmp = l.strip().split(' ')[1]
        if tmp.startswith("0x"):
            tmp = int(tmp, 16)
        else:
            tmp = int(tmp)

        if 'add' in l:
            ecx += tmp
        elif 'sub' in l:
            ecx -= tmp

    elif 'int\t0x80' in l:
        s += chr(ecx)

print s
```

运行代码后发现有点小问题。



仔细观察发现其他图片也有类似的彩虹糖小点，换一张图片试试，这次我们只需提取RGB中红色为0x90和蓝色为0x80的像素点即可：

```

from capstone import *

from PIL import Image

md = Cs(CS_ARCH_X86, CS_MODE_32)

img = Image.open("happycow.bmp")

pixels = img.load()
w,h = img.size

s = ""
for y in range(h):
    for x in range(w):
        if pixels[x,y][0]==0x90 or pixels[x,y][2]==0x80:
            s += ''.join([chr(_) for _ in pixels[x,y][::-1]])

for inst in md.disasm(s, 0):
    print "%x%x:\t%s\t%s" % (inst.address, inst.mnemonic, inst.op_str)

```

将输出结果重定向到nappycow.out，再对结果进行计算，发现了flag，但是提交并不对。尝试其他图片也都不对，ㄟ(▽ ㄟ)ㄒ，三次登门请教大神。

```

λ py2 re.py
***
FLAG:c3dbbf0288ceebeddc6d2505fddd00d
**
https://blog.csdn.net/afuzong4267

```

经提示，BPM图片的存储方式（像素点映射方式）是按照从左到右，从下到上！所以取出来的点都是反的，于是重新取点：

```

f = open("happycow.out")

lines = f.readlines()

s = ""
ecx = 0

for l in lines:
    if 'byte ptr [ecx]' in l:
        tmp = l.strip().split(' ')[1]
        if tmp.startswith("0x"):
            tmp = int(tmp, 16)
        else:
            tmp = int(tmp)

        if 'add' in l:
            ecx += tmp
        elif 'sub' in l:
            ecx -= tmp

    elif 'int\t0x80' in l:
        s += chr(ecx)

print s

```

得到了不一样的FLAG，提交发现flag正确。

```
λ py2 re-fanxiang.py
***
FLAG:c3dbbf0298eceb3edcd6d2505fd8d30d
***
https://blog.csdn.net/afuzong4267
```

再次对第一张图片进行计算验证，发现得到了相同的FLAG

```
λ py2 re-fanxiang.py
***
FLAG:c3dbbf0298eceb3edcd6d2505fd8d30d
***
https://blog.csdn.net/afuzong4267
```

对六张图片一次进行计算验证：

```
from capstone import *
import glob
from PIL import Image

md = Cs(CS_ARCH_X86, CS_MODE_32)

for fn in glob.glob("*.bmp"):

    img = Image.open(fn)
    pixels = img.load()
    w,h = img.size

    s = ""
    for y in range(h-1, -1, -1):
        for x in range(w):
            if pixels[x,y][0] == 0x90 or pixels[x,y][2] == 0x80:
                s += ''.join([chr(_) for _ in pixels[x,y][::-1]])

    lines = []
    for inst in md.disasm(s, 0):
        lines += ["0x%x:\t%s\t%s" % (inst.address, inst.mnemonic, inst.op_str)]

    print fn

    ecx = 0
    out = ""
    for l in lines:
        if 'byte ptr [ecx]' in l:
            tmp = l.strip().split(',')[1]
            if tmp.startswith("0x"):
                tmp = int(tmp, 16)
            else:
                tmp = int(tmp)

            if 'add' in l:
                ecx += tmp
            elif 'sub' in l:
                ecx -= tmp

            elif 'int\t0x80' in l:
                out += chr(ecx)

    print out
```

发现结果都一样，得到flag!

```
λ py2 cal-all.py
660px-San_Francisco_districts_map.bmp
***
FLAG:c3dbbf0298eceb3edcd6d2505fd8d30d
***

alcatraz.bmp
***
FLAG:c3dbbf0298eceb3edcd6d2505fd8d30d
***

angry_cow.bmp
***
FLAG:c3dbbf0298eceb3edcd6d2505fd8d30d
***

firescape.bmp
***
FLAG:c3dbbf0298eceb3edcd6d2505fd8d30d
***

happycow.bmp
***
FLAG:c3dbbf0298eceb3edcd6d2505fd8d30d
***

neoncow.bmp
***
FLAG:c3dbbf0298eceb3edcd6d2505fd8d30d
***
```

非常不容易，通过这道题，学到了很多，总结一下新的姿势，知识点如下：

Capstone:

做pwn题经常使用，功能强大的汇编处理库，本题用到的是将16进制转换为汇编程序。

```
from capstone import *
shellcode = b"\x31\xc0\x90"
md = Cs(CS_ARCH_X86, CS_MODE_32)
for i in md.disasm(shellcode, 0x00):
    print "0x%x:\t%s\t%s" % (i.address, i.mnemonic, i.op_str)
```

shellcode = b"\x31\xc0\x90": 定义16进制shellcode

md = Cs(CS_ARCH_X86, CS_MODE_32): 初始化类并给出两个参数（硬件架构和硬件模式），本题中，我们反汇编的是x86体系结构的32位代码

for i in md.disasm(shellcode,0x00): disasm 反汇编16进制代码，其参数是shellcode和起始地址。

print "0x%x: \ t% s \ t% s"% (i.address, i.mnemonic, i.op_str) : 输出地址，操作指令和操作数。