登录

【2018年 网鼎杯CTF 第四场】Crypto：APL

白菜 / 2018-08-29 21:54:25 / 浏览数 3311 安全技术 CTF 顶(0) 踩(0)

描述信息

> What's apl?
>
> flag格式：flag{xxxx}
>
> Download

简单google可以知道文件内容为APL程序，我们要读懂然后逆向得到flag

```
{■(~■)/('No_Please_continue')('Yes,This_is_flag')}(■(41(41)0+140)(■UCS('µ■»Õ■$#■$èáË■■■`âÞ■■#"!■"KE(©$#■$Q<k'))146){+/■≠33
```

APL

我不会APL，以下内容是从题目中推理的，不保证完全正确，如果有错误，请指正。

这里有个在线apl编程和apl文档
,文档有点难懂，可以一遍实验一遍推理出每个符号的意义。

apl是一个从右向左执行的语言。

APL的函数使用{}包裹，函数可以有一或两个参数，右边的参数用变量■表示，左边用变量■表示。

> Most verbs have two definitions, one for the monadic case (one argument), and one for the dyadic case (two arguments).

代码阅读

1. 代码结构分析

首先我们通过括号将代码分行，增加可读性

```
{
 ■(~■)/('No_Please_continue')('Yes,This_is_flag')
}
(■(41(41)0+140)
(■UCS('µ■»Õ■$#■$èáË■■■`âÞ■■#"!■"KE(©$#■$Q<k'))146)
{
 +/■≠33+2⊥(1(5)×8)■■
 {
   a≠8↑(1,a←(8■2)■■)
 }
 ¨
 2⊥8(+/■■(7*2)-■9.1■■'FlagIsWhat')
 ■
 10■■■
 (■4(■■88888)+16)
 ■
 (1+(|¯8)■1)■■UCS(■)
}
'YourFlagIsWhat?'
```

"格式化"后好看多了，我们可以猜测，这段程序是让我们输入flag，返回是否正确。

google一下或自己试一下可以知道UCS是将字符转成ascii码

```
■UCS('µ■»Õ■$#■$èáË■■■`âÞ■■#"!■"KE(©$#■$Q<k')

181 283 187 213 256 36 35 286 36 232 225 203 286 286 285 96 226 222 288 157 35 34 33 288 34 75 69 40 169 36 35 286 36 81 60 10
```

因此可以推测以上就是密文，我们需要逆向加密过程，还原明文。

根据函数中使用的变量，可以判断这段代码有两个函数。下面的函数有两个参数，计算结果作为上面函数的参数。

```
{
 ■(~■)/('No_Please_continue')('Yes,This_is_flag')
}

...
{
 +/■≠33+2⊥(1(5)×8)■■
 ...
 (1+(|¯8)■1)■■UCS(■)
}
'YourFlagIsWhat?'
```

2. 函数一

先看上面的函数

```
{
 ■(~■)/('No_Please_continue')('Yes,This_is_flag')
}
```

我们查阅文档知道，~表示对bool变量做非运算。

> Monad. ~ applies only to boolean arguments, and negates them: ~0 1 ⟷ 1 0 .

我们分别给参数0和1，发现第一个函数需要参数0，即第二个函数计算结果应为0

```
{   ■(~■)/('No_Please_continue')('Yes,This_is_flag') }1

■■■■■■■■■■■■■■■■■■■■■■
■No_Please_continue■
■■■■■■■■■■■■■■■■■■■■■■

{   ■(~■)/('No_Please_continue')('Yes,This_is_flag') }0

■■■■■■■■■■■■■■■■■■■■
■Yes,This_is_flag■
```

3. 函数二

3.1 **(1+(|¯8)■1)■■UCS(■)**

我们接着看第二个函数，先看加密过程，一行一行往上看

```
...
{
 ...
 // ■■■■■■■■2 2 2 2 2 2 2 2■
 (1+(|¯8)■1)
 ■
 //■■■■■■■■ascii
 ■UCS(■)
}
'YourFlagIsWhat?'
```

a ■ b表示将b填充至长度a。8 ■ 'abcd'即为abcdabcd

■和⊥互为逆运算

> For simple cases, ⍁ is inverse to the base ⊥ .

这里我们不用详细看文档，实验可以知道这里是将ascii转成二进制

```
{■UCS(■)}'YourFlagIsWhat?'

89 111 117 114 70 108 97 103 73 115 87 104 97 116 63

(1+(|¯8)■1)

2 2 2 2 2 2 2 2

{(1+(|¯8)■1)■■UCS(■)}'YourFlagIsWhat?'
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1
1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 1
1 1 0 0 0 1 0 0 1 0 0 1 0 0 1 1
0 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1
0 1 0 1 1 0 0 1 0 1 1 0 0 0 1 1
1 1 1 0 0 0 1 1 1 1 1 0 1 0 1 1
```

3.2 **(■4(■■88888)+16)**

```
...
{
 ...
 (■4(■■88888)+16)
 ■
 # ■■■■■■
 (1+(|¯8)■1)■■UCS(■)
}
'YourFlagIsWhat?'
```

这是一个独立的表达式，其值是固定的。

```
(■4(■■88888)+16)

20 16
```

(20 16) ■ a表示将a填充为20*16的矩阵

```
{(■4(■■88888)+16)■(1+(|¯8)■1)■■UCS(■)}'YourFlagIsWhat?'

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1
1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1
1 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0
0 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1
1 0 1 0 0 1 0 0 1 1 0 1 0 1 1 0
0 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1
1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 0 1 1 1 0 1 1 1 0 1
0 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1
0 0 1 1 1 1 0 0 0 1 0 0 1 0 0 1
0 0 1 0 1 1 0 1 1 0 1 0 0 1 0 0
1 1 0 1 0 1 1 0 0 1 0 1 1 0 0 0
1 1 1 1 0 0 0 1 1 1 1 1 0 1 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1
1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1
1 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0
0 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1
```

3.3 **10■■■**

```
...
{
 ...
 10■■■
 # ■■■■■■(8*40)■■(20*16)■■
 (■4(■■88888)+16)■(1+(|¯8)■1)■■UCS(■)
}
'YourFlagIsWhat?'
```

■ ■分别表示将矩阵的行、列倒序。

10■将矩阵的列向上位移10

3.4 **2⊥8(+/■■(7*2)-■9.1■■'FlagIsWhat')**

我们看一个稍微复杂点的

```
2
⊥
8(+/■■(7*2)-■9.1■■'FlagIsWhat')
■
# ■■■■■■
10■■■(■4(■■88888)+16)■(1+(|¯8)■1)■■UCS(■)
}
'YourFlagIsWhat?'
```

8(+/■■(7*2)-■9.1■■'FlagIsWhat')是一个独立的表达式，我们分别运行以下子表达式

```
9.1■2
2

9.1■10.3
9.1

■10
1 2 3 4 5 6 7 8 9 10

■9.1■■'FlagIsWhat'
9

8(+/■■(7*2)-■9.1■■'FlagIsWhat')
8 40
```

可以推断

- 7*2表示7的2次方，
- ■'FlsWhat'计算字符的长度
- a■b计算a,b最小值，■a对a取整
- ■a返回一个1-a的数组
- +/a表示对a求和

所以8(+/■■(7*2)-■9.1■■'FlagIsWhat')■10■■■(■4(■■88888)+16)■(1+(|¯8)■1)■■UCS(■)就是将矩阵转回8*40

下面就是再转回ascii

```
{2⊥8(+/■■(7*2)-■9.1■■'FlagIsWhat')■10■■■(■4(■■88888)+16)■(1+(|¯8)■1)■■UCS(■)}'YourFlagIsWhat?'

255 0 255 189 214 107 214 214 255 0 0 189 214 214 255 214 148 148 189 255 255 189 214 148 189 107 66 0 66 0 66 107 41 0 107 41
```

3.5 移位异或**a≠8↑(1,a←(8■2)■■)**

```
...
{
 ...
 {
   a≠8↑(1,a←(8■2)■■)
 }
 ¨
 2⊥8(+/■■(7*2)-■9.1■■'FlagIsWhat')■10■■■(■4(■■88888)+16)■(1+(|¯8)■1)■■UCS(■)
}
'YourFlagIsWhat?'
```

这里我们遇到一个子函数，通过以上的分析，我们已经能阅读一些代码。

(8■2)就是(2 2 2 2 2 2 2 2)，(8■2)■■表示转换成二进制。

结合实验和文档：

- a←(8■2)■■表示赋值给a
- ≠指异或
- 8↑a是保留前8位

因此，这个子函数就是f(x)=x^shift(x), shift函数为x右移一位，在前面补1。

3.6 最后一段

```
(■(41(41)0+140)(■UCS('µ■»Õ■$#■$èáË■■■`âÞ■■#"!■"KE(©$#■$Q<k'))146)
{
```

```
 +/
 ■
 ≠
 33+
 2⊥(1(5)×8)
 ■
 ■
 {
   a≠8↑(1,a←(8■2)■■)
 }
 ¨
 2⊥8(+/■■(7*2)-■9.1■■'FlagIsWhat')■10■■■(■4(■■88888)+16)■(1+(|¯8)■1)■■UCS(■)
}
'YourFlagIsWhat?'
```

¨按列读取矩阵，将矩阵转换成一个一维矩阵。我们省略一些前面已经分析过的语法，直接得到以下结果

```
{33+   2⊥(1(5)×8)   ■   ■   {     a≠8↑(1,a←(8■2)■■)   }   ¨   2⊥8(+/■■(7*2)-■9.1■■'FlagIsWhat')■10■■■(■4(■■88888)+16)■(1
```

```
136 103 52 118 118 118 103 52 168 95 142 116 116 116 95 142 40 50 139 156 156 156 50 139 40 189 214 74 74 74 189 214 104 44 17
```

■就是函数左边的参数，即密文。与我们加密的结果异或并求和后，如果结果是0，则答案正确。

4. 逆向

整个加密函数其实不难，主要过程有

1. 将明文转码为二进制码，并填充至16*20，得到一个矩阵
2. 做三次矩阵变换， 10■■■
3. 将矩阵转换为8*40矩阵
4. 按列将矩阵转换为一维矩阵，并做一个移位异或计算，得到一个一维矩阵。
5. 将矩阵恢转成8*40，转为ascii码, 并+33

因此我们逆向过程也很简单

4.1 将密文-33, 转成二进制的8*40矩阵

```
(8■2)■(■(41(41)0+140)(■UCS('µ■»Õ■$#■$èáË■■■`âÞ■■#"!■"KE(©$#■$Q<k'))146-33)
```

```
1 1 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0
0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1
0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 1
1 1 0 1 1 1 1 1 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1
0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 0
1 1 0 1 0 0 1 1 0 0 1 0 1 0 0 1 1 1 1 1 0 1 1 1 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0
0 0 1 0 1 1 0 1 1 1 0 1 1 0 1 0 0 0 1 0 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 1 0
0 0 1 0 0 0 0 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1
```

4.2 按行读矩阵，每8位作为一个数，传给 移位异或 的逆运算

```
a = "11011111001011111101110000100001001000000010100100101101110101100010000000100011001010100010001111101110001011000010100111
for i in range(len(a)/8):
   print int(a[i*8:i*8+8],2),
```

```
# (223 47 220 33 32 41 45 214 32 35 42 35 238 44 41 223 33 238 32 45 45 35 238 41 38 211 41 238 38 32 45 218 37 42 214 33 185
```

4.3 移位异或 的逆运算

因为移位运算前面补1，所以可以利用shift(x)的第一位和f(x)的第一位异或，求x的第一位，即shift(x)的第二位，依次求得x

```
a = (223,47,220,33,32,41,45,214,32,35,42,35,238,44,41,223,33,238,32,45,45,35,238,41,38,211,41,238,38,32,45,218,37,42,214,33,18
)

def dexhr(r):
   r = bin(r)[2:]
   r = (8-len(r))*'0'+r
   x = ""
   shift_x = "1"
   for index in range(8):
       n = int(shift_x[-1])^int(r[index])
```

```
        x += str(n)
        shift_x += str(n)
    return x


for i in a:
    x = dexhr(i)
    print int(x, 2),

# (106 202 104 193 192 206 201 100 192 194 204 194 75 200 206 106 193 75 192 201 201 194 75 206 196 98 206 75 196 192 201 108
```

4.4 矩阵变换 的逆运算

```
■■10■(20 16)■(8■2)■(106 202 104 193 192 206 201 100 192 194 204 194 75 200 206 106 193 75 192 201 201 194 75 206 196 98 206

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0
1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1
0 1 1 1 0 1 0 1 1 0 1 1 1 1 0 1
0 1 1 0 1 1 1 1 0 1 1 1 1 0 1 0
0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0
0 1 0 1 1 0 1 1 0 0 0 1 0 0 0 0
0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1
0 0 0 0 1 1 1 0 1 1 1 0 0 0 1 0
1 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1
1 0 0 1 0 1 1 1 1 0 0 1 0 1 0 1
1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0
1 0 1 0 0 0 0 0 0 0 1 1 0 0 1 0
1 1 0 0 1 1 0 0 1 1 0 1 1 0 1 0
1 1 1 1 0 1 0 0 0 1 1 0 0 1 1 1
```

4.5 转成8*40得到flag

```
2⊥(8 40)■■■10■(20 16)■(8■2)■(106 202 104 193 192 206 201 100 192 194 204 194 75 200 206 106 193 75 192 201 201 194 75 206 19

70 76 65 71 56 98 51 54 99 57 48 50 45 55 100 50 55 45 52 57 57 48 45 56 101 55 49 45 52 51 52 48 98 57 55 48 56 97 53 101
```

将以上ascii转成字符串即为flag`FLAG8b36c902-7d27-4990-8e71-4340b9708a5e`

打完收工

上一篇：【2018年 网鼎杯CTF 第四场... 下一篇：【2018年 网鼎杯CTF 第四场...

1. 2 条回复

就不告诉你 2018-08-29 22:12:20

不错，今天做的时候发现APL语言了，研究了好久，谢大佬解惑啦

0 回复Ta


饭饭之交 2018-08-29 23:18:25

厉害了，今天就只base64出来了，然后就看不懂了

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板