攻防世界crypto高手题之streamgame1

原创

沐一 · 林 于 2021-09-21 12:05:50 发布 56 收藏 1

分类专栏： CTF 密码学 文章标签： ctf

本文链接：https://blog.csdn.net/xiao__1bai/article/details/120399710


CTF 同时被 2 个专栏收录

167 篇文章 6 订阅

订阅专栏


密码学

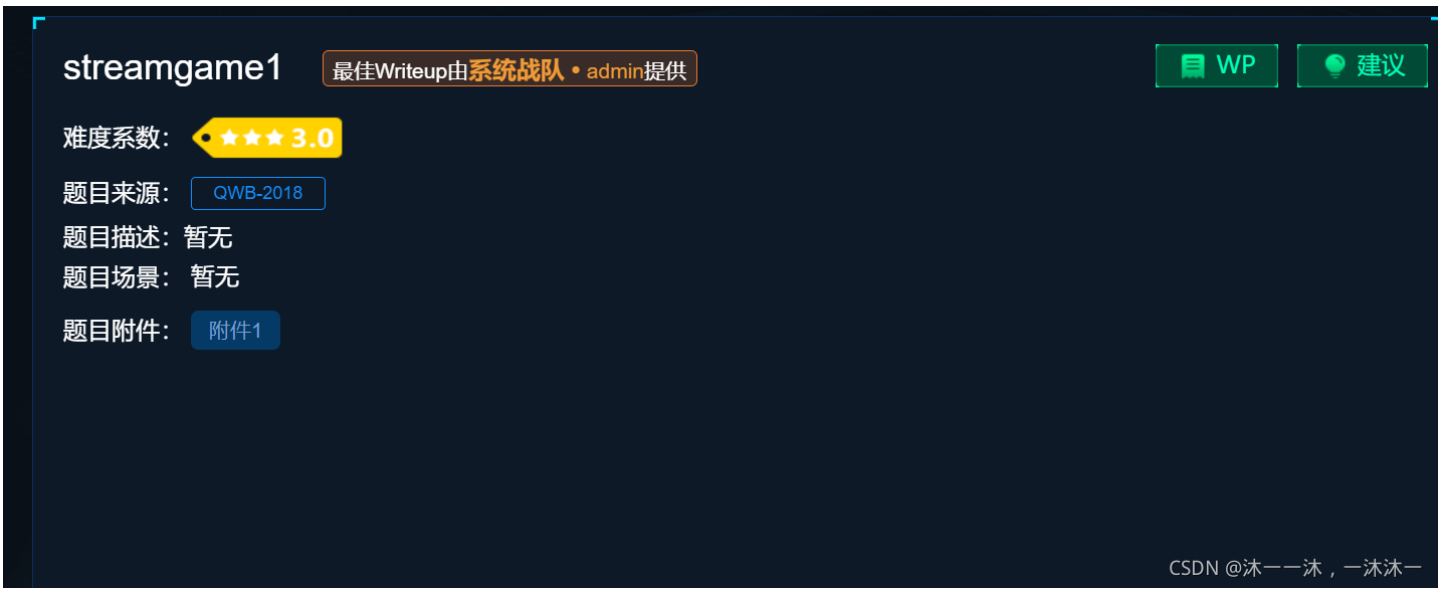
51 篇文章 1 订阅

订阅专栏

攻防世界crypto高手题之streamgame1

继续开启全栈梦想之逆向之旅~

这题是攻防世界crypto高手题的streamgame1



.

.

下载附件，是典型的 `LFSR` 类型：

```
from flag import flag
assert flag.startswith("flag{")
# 作用：判断字符串是否以指定字符或子字符串开头flag{
assert flag.endswith("}")
# 作用：判断字符串是否以指定字符或子字符串结尾}，flag{}，6个字节
assert len(flag)==25
# flag的长度为25字节，25-6=19个字节
#3<<2可以这么算，bin(3)=0b11向左移动2位变成1100，0b1100=12(十进制)
def lfsr(R,mask):
    output = (R << 1) & 0xffffff    #将R向左移动1位，bin(0xffffff)='0b111111111111111111111111'=0xffffff的二进制补码
    i=(R&mask)&0xffffff             #按位与运算符&：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0
    lastbit=0
    while i!=0:
        lastbit^=(i&1)    #按位异或运算符：当两对应的二进位相异时，结果为1
        i=i>>1
    output^=lastbit
    return (output,lastbit)



R=int(flag[5:-1],2)
mask    =   0b10100011000100011100

f=open("key","ab")   #以二进制追加模式打开
for i in range(12):
    tmp=0
    for j in range(8):
        (R,out)=lfsr(R,mask)
        tmp=(tmp << 1)^out   #按位异或运算符：当两对应的二进位相异时，结果为1
    f.write(chr(tmp))  #chr() 用一个范围在 range（256）内的（就是0～255）整数作参数，返回一个对应的字符。
f.close()
```

.

.

明文`key`如下，需要转成能推出`初始状态值`(flag)的对应`位数`的`flag`：

.

.

(这里积累第一个经验)

`LFSR`类型我在博客`https://blog.csdn.net/xiao__1bai/article/details/120392307`已经总结过了，所以这里直接应用前面的总结步骤仿照着做即可：

```
from flag import flag
assert flag.startswith("flag{")
# 作用：判断字符串是否以指定字符或子字符串开头flag{
assert flag.endswith("}")
# 作用：判断字符串是否以指定字符或子字符串结尾}，flag{}，6个字节
assert len(flag)==25
# flag的长度为25字节，25-6=19个字节
#3<<2可以这么算，bin(3)=0b11向左移动2位变成1100，0b1100=12(十进制)
def lfsr(R,mask):
    output = (R << 1) & 0xffffff    #将R向左移动1位，bin(0xffffff)='0b111111111111111111111111'=0xffffff的二进制补码
    i=(R&mask)&0xffffff             #按位与运算符&：参与运算的两个值，如果两个相应位都为1，则该位的结果为1，否则为0
    lastbit=0
    while i!=0:
        lastbit^=(i&1)    #按位异或运算符：当两对应的二进位相异时，结果为1
        i=i>>1
    output^=lastbit
    return (output,lastbit)


R=int(flag[5:-1],2)
mask    =   0b10100110001000111100

f=open("key","ab")    #以二进制追加模式打开
for i in range(12):
    tmp=0
    for j in range(8):
        (R,out)=lfsr(R,mask)
        tmp=(tmp << 1)^out    #按位异或运算符：当两对应的二进位相异时，结果为1
    f.write(chr(tmp))  #chr() 用一个范围在 range（256）内的（就是0～255）整数作参数，返回一个对应的字符。
f.close()
```

但是flag不可能有76位这么长，所以这19位应该是2进制数，这才能对应后面的mask的19位为一个循环。

flag长度19字节，4*19=76位

同样的逻辑，out左移动一位，右边补上lastbit值

可以推出反馈函数lastbit=R3⊕R4⊕R5⊕R9⊕R13⊕R14⊕R17⊕R19

也就是说19位为一个循环，那么key值只要取前19位即可反推flag

.

.

(这里积累第二个经验)

关键就是这个`key`值怎么取，一开始我直接复制粘贴`字符转16进制`，然后取`前19`位，可想而知当然是错的了。

然后查了好些资料发些他们从文件中读取二进制数的代码还是不够简便，于是我吸取了我前面`base64编码`的代码写了一串从文件中读取`对齐`的`二进制`的代码，以后也直接拿来用即可。
(在线转换都会省略最开头的0导致结果位数错误，进而导致结果错误，所以自己要注意。)

```
f = open('5key','rb') #以二进制格式打开文件
content = f.read()  #读取的是\xhh类型的十六进制
#print(content)
key=['{:0>8}'.format(str(bin(i)).replace('0b','')) for i in content] #从base64编码汲取的经验，二进制8位对齐。
print(''.join(key)[:19])
```

.

.

最终解题脚本，还是一样要注意小端顺序：

```
f = open('5key','rb') #以二进制格式打开文件
content = f.read()  #读取的是\xhh类型的十六进制
key=['{:0>8}'.format(str(bin(i)).replace('0b','')) for i in content] #从base64编码汲取的经验，二进制8位对齐。
key1=''.join(key)[:19] #lastbit全部值，就是反馈函数生成的值，32位的key1
key2=key1
flag=[]
for i in range(19):
 output='?'+key1[:18] #?0100000111111011110111011111000,因为后面有key1=str(lastbit)+key1[:31]，key1不断填补，output不断取前31位，所以这里output每次把?定在第i位上，注意output和key1是独立的分开的。
 flag.append(str(int(key2[-1-i])^int(output[-3])^int(output[-4])^int(output[-5])^int(output[-9])^int(output[-13])^int(output[-14])^int(output[-17])))
#这里之所以取负数是因为我们截取是从左往右取，而在计算机中是小端顺序，应该从右往左取才对，这里的key2[-1-i]就是小端左到右的第i位，也就是前面分析的反馈函数生成值的第i位。flag值的原第i位,现在在第19位，可以通过⊕的可逆性来求，就是R19=lastbit ⊕ R3 ⊕ R4 ⊕ R5 ⊕ R9 ⊕ R13 ⊕ R14 ⊕ R17
 key1=str(flag[i])+key1[:18]#不断填补key1,让key1向右推进,
print("flag{"+bin(int(''.join(flag[::-1]),2)).replace('0b','')+"}")#这里是经过一系列操作，首先把flag变成小端顺序的[::-1]，然后就是转十六进制。
```

.

.

结果：

```
└─$ python 2.py
flag{1110101100001101011}
```

.

.

.

总结：

1：（这里积累第一个经验）

LFSR 类型我在博客 https://blog.csdn.net/xiao__1bai/article/details/120392307 已经总结过了，所以这里直接应用前面的总结步骤仿照着做即可：

```
from flag import flag
assert flag.startswith("flag{")
# 作用：判断字符串是否以指定字符或子字符串开头flag{
assert flag.endswith("}")
# 作用：判断字符串是否以指定字符或子字符串结尾}，flag{}，6个字节
assert len(flag)==25
# flag的长度为25字节，25-6=19个字节
#3<<2可以这么算，bin(3)=0b11向左移动2位变成1100，0b1100=12(十进制)
def lfsr(R,mask):
    output = (R << 1) & 0xffffff    #将R向左移动1位，bin(0xffffff)='0b111111111111111111111111'=0xffffff的二进制补码
    i=(R&mask)&0xffffff             #按位与运算符&：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0
    lastbit=0
    while i!=0:
        lastbit^=(i&1)     #按位异或运算符：当两对应的二进位相异时，结果为1
        i=i>>1
    output^=lastbit
    return (output,lastbit)


R=int(flag[5:-1],2)
mask     =    0b10100011000100011100

f=open("key","ab")    #以二进制追加模式打开
for i in range(12):
    tmp=0
    for j in range(8):
        (R,out)=lfsr(R,mask)
        tmp=(tmp << 1)^out    #按位异或运算符：当两对应的二进位相异时，结果为1
    f.write(chr(tmp))  #chr() 用一个范围在 range（256）内的（就是0～255）整数作参数，返回一个对应的字符。
f.close()
```

但是flag不可能有76位这么长，所以这19位应该是2进制数，这才能对应后面的mask的19位为一个循环。

flag长度19字节，4*19=76位

同样的逻辑，out左移动一位，右边补上lastbit值

可以推出反馈函数lastbit=R3⊕R4⊕R5⊕R9⊕R13⊕R14⊕R17⊕R19

也就是说19位为一个循环，那么key值只要取前19位即可反推flag

2：

(这里积累第二个经验)

关键就是这个 key 值怎么取，一开始我直接复制粘贴 字符转16进制，然后取 前19 位，可想而知当然是错的了。

.

然后查了好些资料发些他们从文件中读取二进制数的代码还是不够简便，于是我吸取了我前面 base64编码 的代码写了一串从文件中读取 对齐 的 二进制 的代码，以后也直接拿来用即可。

(在线转换都会省略最开头的0导致结果位数错误，进而导致结果错误，所以自己要注意。)

```
f = open('5key','rb') #以二进制格式打开文件
content = f.read()  #读取的是\xhh类型的十六进制
#print(content)
key=['{:0>8}'.format(str(bin(i)).replace('0b','')) for i in content] #从base64编码汲取的经验，二进制8位对齐。
print(''.join(key)[:19])
```

解毕！敬礼！