RCTF 2019 baby_crypto & baby_aes Writeup

rmb122 / 2019-05-23 01:09:00 / 浏览数 3813 安全技术 CTF 顶(0) 踩(0)

密码学只做出来两题 baby, 暗示我还是学密码学的 baby (逃

baby_crypto

这题还算比较常规, 主要逻辑如下

```
while True:
   try:
       print("Input your cookie:")
       data_hex = sys.stdin.readline().strip()
       data = binascii.unhexlify(data_hex)
       assert(len(data) > iv_len + hash_len)
       iv, cookie_padded_encrypted, hv = data[:iv_len], data[iv_len: -hash_len], data[-hash_len:]
       cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=backend)
       decryptor = cipher.decryptor()
       cookie_padded = decryptor.update(cookie_padded_encrypted) + decryptor.finalize()
       try:
           cookie = unpad(cookie_padded)
       except Exception as e:
           print("Invalid padding")
           continue
       if not is_valid_hash(cookie, hv):
           print("Invalid hash")
           continue
       info = {}
       for _ in cookie.split(b";"):
           k, v = _.split(b":")
           info[k] = v
       if info[b"admin"] == b"1":
           with open("flag") as f:
               flag = f.read()
               print("Your flag: %s" %flag)
```

很明显的看到可以 `padding oracle`, 只要满足 `info[b"admin"] == b"1"` 就可以拿到 `flag`,
但在 `cookie` 后面设置了 hash 来效验 `cookie` 的有效性, 但是没有检测重复的键值
所以这里可以结合长度扩展攻击, 我们假设 `cookie` 为 `admin:0;username:abcde;password:abcde`
我们可以在原 `cookie` 后面添加一个 `;admin:1`, 得到
`admin:0;username:abcde;password:abcde\x80\x00\x00\x00\x00\x00\x00\x00\x00\x01\xa8;admin:1`,
因为顺序的关系, 这将覆盖之前的值, 从而满足条件. 脚本如下
一开始没有国内的服务器, 写完下午睡了一觉起来才跑完 233

```
import remotecli # https://github.com/rmb122/remoteCLI
import hashpumpy
from binascii import hexlify, unhexlify
import copy
from tqdm import tqdm


def padding(byte):
   padlen = 16 - len(byte) % 16
   byte += bytearray([padlen] * padlen)
   return byte


def addIvLastByte(iv, currIndex, midval):
   target = 16 + 1 - currIndex
   for i in range(currIndex, 16):
       iv[i] = midval[i] ^ target
   return iv


def xor(a, b):
   result = []
```

```
    for i in range(len(a)):
        result.append(a[i] ^ b[i])
    result = bytearray(result)
    return result

cli = remotecli.CLI()
cli.connect('207.148.68.109', 20000)
cli.sendLine('abcde')
cli.sendLine('abcde')

hv_hex_len = 40
iv_len = 16
orgCookie = 'admin:0;username:abcde;password:abcde'

cookie = cli.recvLinesUntilHave('Input your cookie:')[-2]
print(cookie)
hv_hex = cookie[-hv_hex_len:]
iv = cookie[:iv_len]
cookieEnc = cookie[iv_len: - hv_hex_len]

fakeHash, fakeCookie = hashpumpy.hashpump(hv_hex, orgCookie, ';admin:1', iv_len)
print(fakeCookie)
print(fakeHash)

fakeHash = bytearray(unhexlify(fakeHash))
fakeCookie = padding(fakeCookie)

assert len(fakeCookie) == 64
dummy = bytearray([0 for i in range(len(fakeCookie) + 16)]) # iv + cookie

for pos in range(64 + 16, 16, -iv_len):
    curr = dummy[pos - iv_len:pos]
    iv = bytearray([0 for i in range(iv_len)])
    midval = bytearray([0 for i in range(iv_len)])
    for currIndex in range(0, iv_len)[::-1]:
        for i in tqdm(range(0, 256)):
            iv[currIndex] = i
            cli.sendLine(hexlify(iv + curr + fakeHash))
            res = cli.recvline()
            #print(res)
            cli.recvline()
            if "Invalid padding" not in res:
                midval[currIndex] = (16 - currIndex) ^ iv[currIndex]
                if currIndex == 0:
                    tmp = xor(midval, fakeCookie[pos-iv_len*2:pos-iv_len])
                    for tmpPos in range(0, 16):
                        dummy[pos-iv_len*2 + tmpPos] = tmp[tmpPos]
                iv = addIvLastByte(iv, currIndex, midval)
                break

cli.sendLine(hexlify(dummy + fakeHash))
cli.console()
```

baby_aes

这题比较有意思, 操作还是比较硬核的, 主要逻辑

```
K = b"\x01\x23\x45\x67\x89\xab\xcd\xef\xfe\xdc\xba\x98\x76\x54\x32\x10"
    Ke = init(K)

    backend = default_backend()
    key = os.urandom(16)
    iv = encrypt(key, Ke)
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=backend)
    decryptor = cipher.decryptor()
    try:
        print("Input a hexstr to decrypt:")
        data = sys.stdin.readline().strip()
        ciphertext = binascii.unhexlify(data)
```

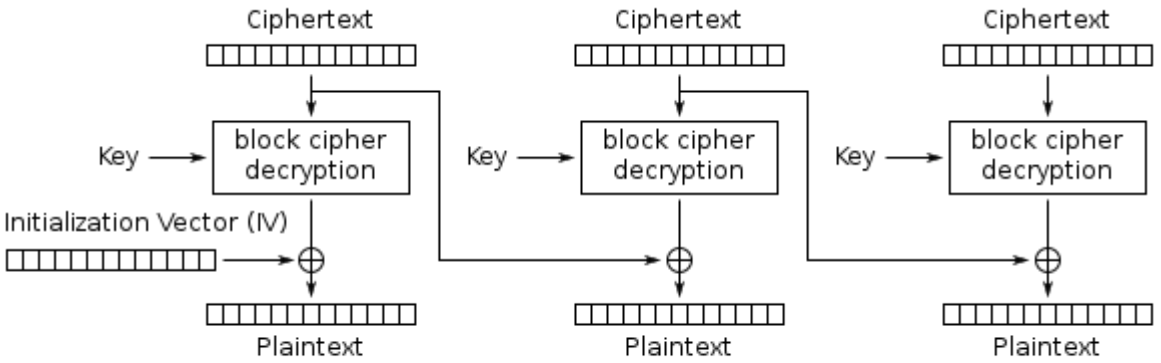
```
        plaintext = decryptor.update(ciphertext) + decryptor.finalize()
        print("Decrypted result:")
        print(binascii.hexlify(plaintext).decode())
    except Exception as e:
        pass

    with open("flag", 'rb') as f:
        flag = f.read()
        padder = padding.PKCS7(128).padder()
        flag_padded = padder.update(flag) + padder.finalize()
        encryptor = cipher.encryptor()
        flag_encrypted = encryptor.update(flag_padded) + encryptor.finalize()
        print("Your encrypted flag is:")
        print(binascii.hexlify(flag_encrypted).decode())
```

其中 `init` 函数是 AES 秘钥扩展, `encrypt` 是 AES 轮函数, 但是改变了 AES 原来的常数, 这两个函数也是本题的核心, 我们留到后面讲.
这里假设我们已经写出对应的解密函数,

看到 `iv = encrypt(key, Ke)`, 可以看到 iv 就是 key 的加密, 只要我们能获得 `iv`, 就能解密出 `key`, 从而解密得到 `flag`.

注意到这里是用 AES 解密输入的数据, 结合 CBC 模式



Cipher Block Chaining (CBC) mode decryption

我们可以输入两个相同分块(b1 + b1')长度的数据, 其中解密结果的第二块(o2)是这样算出来的
`xor(AESdec(b1'), b1) = o2`
而 `o2`, `b1` 都是已知的, 我们就可以解出 `AESdec(b1)`, 因为我们输入的两个分块相同,
我们将 `AESdec(b1)` 与 `o1` xor 一下, 就能得到 `iv`, 这时只要用 K 解密 `iv` 就能得到 `key`
从而解密 `flag`.

但问题就是这里出题人魔改了 AES, 不能直接解密, 这里最好自己写过一遍 AES 的实现, 否则接下来有些部分可能不太方便,
首先可以搜到作者魔改的原代码
可以看到 Sbox, T1-4 都被修改, 并且没有给出对应的逆变换

```
S = [0x63, 0x7c, 0x77, 0x7.... <-原来的
S = [0x93 ,0x43 ,0x5D ,0x6.... <-魔改之后的
```

但是从 `rcon = [....0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91 ]` 这最后几个
数据可以看出, 仍然还是在 `GF(2^8) mod x^8 + x^4 + x^3 + x + 1` 上的, 否则 0xc5 * 2 不会等于 0x91

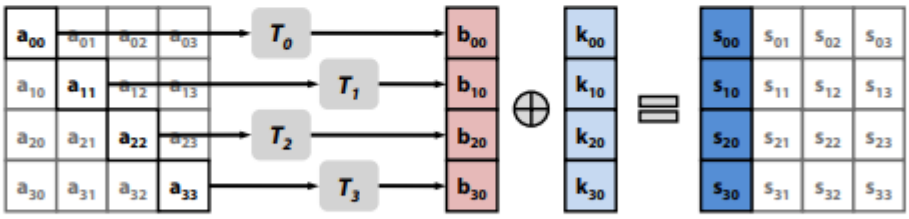
所以这里应该只是单纯的改了 Sbox, 不是改其他常数带来的副作用.

接下来可以看到 T 被完全修改了, 首先了解一下 T 变换是干嘛的,
这里借一张图说明

- Combine SubBytes, ShiftRows, MixColumns using the standard "T-table" approach. Update each column ($0 \leq j \leq 3$):

$$[s_{j0}, s_{j1}, s_{j2}, s_{j3}]^{\mathrm{T}} = T_0[a_{c_0 0}] \oplus T_1[a_{c_1 1}] \oplus T_2[a_{c_2 2}] \oplus T_3[a_{c_3 3}] \oplus k_j,$$

  where each T_i is $1KB$ and k_j is the jth column of the round key.
- Example ($j = 0$):



- Optimization approach: launch thread blocks containing multiple independent groups of 16 (1/2-warp) streams.

注: ShiftRow 与 ByteSub 之间的顺序不敏感, 可以在进入变换之前就 ShiftRow 好, 就跟上图一样, 取的不是原矩阵的一行, 而是 ShiftRow 之后矩阵的一行

T 变换是结合了 ByteSub MixColumn, 将 AES 轮函数中的两步并到一起, 加速效率的一种方法, 如果按照原来的矩阵乘法

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

```
d0 = 2 * ByteSub(b0) + 3 * ByteSub(b1) + 1 * ByteSub(b2) + 1 * ByteSub(b3)
d1 = 1 * ByteSub(b0) + 2 * ByteSub(b1) + 3 * ByteSub(b2) + 1 * ByteSub(b3)
d2 = 1 * ByteSub(b0) + 1 * ByteSub(b1) + 2 * ByteSub(b2) + 3 * ByteSub(b3)
d3 = 3 * ByteSub(b0) + 1 * ByteSub(b1) + 1 * ByteSub(b2) + 2 * ByteSub(b3)
```

算起来非常的麻烦, 但是看到

```
2 * ByteSub(b0)
1 * ByteSub(b0)
1 * ByteSub(b0)
3 * ByteSub(b0)
```

可以想到这结构完全是固定的, 因为加密的是字节, 定义域是 0-255, 完全可以将 0-255 的值带入 b0, 将所有值提前算出, 并成 4 个字节, 在使用时查表就行, 大大提高效率. 因为 GF(2^8) 上的加法实际上就是 xor, 所以

```
MixColumn(ByteSub(b0 b1 b2 b3)) = T1[b0] xor T2[b1] xor T3[b2] xor T4[b3]
```

在本题中, 假设进入轮函数之前 state 全是 0, 那么这里查表可以直接一步到位 `T1[0] ^ T2[0] ^ T3[0] ^ T4[0] = 0xaeaeaeae`,
而按照原列混合的矩阵算等于 `0x93939393` 说明列混合的矩阵也被修改, 这就比较麻烦了, 需要用一下 sage.

因为假设输入轮函数的 state 全是 0 ， 那么 subByte 的得到的是 0x93, 而 `T[0] = 0xF467D4E9`, 原 AES 的因数是 `(2, 3, 1, 1)`, 这里我们假设魔改之后的是 `(cofe[0], cofe[1], cofe[2], cofe[3])`

按照上面 T 变换的定义, `T[0]` 是这么来的

```
cofe[0] * b0
cofe[3] * b0
cofe[2] * b0
cofe[1] * b0
```

而本题 T[0] = 0xF467D4E9 可以写出
0x93 * cofe[0] = 0xF4, 0x93 * cofe[3] = 0x67, 0x93 * cofe[2] = 0xD4, 0x93 * cofe[1] = 0xE9,
在 0-255 的范围内爆破下,

```
sage: F.<x> = GF(2^8, modulus=[1,1,0,1,1,0,0,0,1])
sage: F.modulus()
x^8 + x^4 + x^3 + x + 1
sage: def f(num):
....:     global F
....:     return F.fetch_int(num)
....:
sage: for i in range(0,256):
....:     if f(0x93)*f(i)==f(0xf4):
....:         print(i)
....:
8
sage: for i in range(0,256):
....:     if f(0x93)*f(i)==f(0x67):
....:         print(i)
....:
9
sage: for i in range(0,256):
....:     if f(0x93)*f(i)==f(0xd4):
....:         print(i)
....:
7
sage: for i in range(0,256):
....:     if f(0x93)*f(i)==f(0xe9):
....:         print(i)
....:
5
```

那么四个因数是 (8, 5, 7, 9), 还原成矩阵求 GF(2^8) 上的逆矩阵, 再用一下 sage, 当然如果是大佬可以手算 _(:з」∠)_

```
sage: c = matrix(F, [[f(8), f(5), f(7), f(9)], [f(9), f(8), f(5), f(7)], [f(7), f(9), f(8), f(5)], [f(5), f(7), f(9), f(8)]])
sage: c.inverse()
[      x^7 + x^4 + x^2 + x           x^7 + x^6 + x^3       x^7 + x^4 + x^2 + 1 x^5 + x^4 + x^3 + x^2 + 1]
[x^5 + x^4 + x^3 + x^2 + 1       x^7 + x^4 + x^2 + x           x^7 + x^6 + x^3       x^7 + x^4 + x^2 + 1]
[      x^7 + x^4 + x^2 + 1 x^5 + x^4 + x^3 + x^2 + 1       x^7 + x^4 + x^2 + x           x^7 + x^6 + x^3]
[          x^7 + x^6 + x^3       x^7 + x^4 + x^2 + 1 x^5 + x^4 + x^3 + x^2 + 1       x^7 + x^4 + x^2 + x]
```

转换回数字表示就是
cofes = (150, 200, 149, 61)
带进 invMixColumn 就可以正确解密啦, 如果没有自己写过的话, 可以参考我写的辣鸡实现
修改 mixColumn 和 invMixColumn 里面的 polynomialMutil 函数乘的数为矩阵对应位置的数就行了

既然现在可以解密, 按着上面的思路就能拿 flag 了~
接下来 nc 一下, 输入 16 个 1,

```
Input a hexstr to decrypt:
31313131313131313131313131313131313131313131313131313131313131313131
Decrypted result:
3205fe135b595e72c90d2613ada3087812f10dee01e66c4d1e47089a0ff0f18c
Your encrypted flag is:
c2c06ee0e21dae7e5b64fcb84397b4ed920c28bb81a676d817a4b920564bd04dd2a570900ff2e9d5fee9cb74c37c4812
```

```
from Crypto.Cipher import AES as stdAES
from Crypto.Util.strxor import strxor

K = b"\x01\x23\x45\x67\x89\xab\xcd\xef\xfe\xdc\xba\x98\x76\x54\x32\x10"
hexstr = '11111111111111111111111111111111'.encode()
dec = unhexlify('3205fe135b595e72c90d2613ada3087812f10dee01e66c4d1e47089a0ff0f18c')
midVal = strxor(dec[16:32], hexstr[0:16])
iv = strxor(dec[0:16], midVal)
aes = AES()
key = aes.decryptBlock(iv, K)
print(key)
print(iv)
flag = unhexlify('c2c06ee0e21dae7e5b64fcb84397b4ed920c28bb81a676d817a4b920564bd04dd2a570900ff2e9d5fee9cb74c37c4812')
stdaes = stdAES.new(key, stdAES.MODE_CBC, iv)
```

```
print(stdaes.decrypt(flag))

'''
b'N\t\x9c\xce*\xfa\xc1\x02\x94\xd1\x02\xf2\xb8d*E'
b'\x11\xc5\xc2\xcck\x8e\x03\x0e\xe6{\x1f\xb8\x93b\xc8\xc5'
b'RCTF{88358abe-e571-4bdf-95a3-93e9d8ddf558}\x06\x06\x06\x06\x06\x06'
'''
```

这样子求解, 比直接爆破四个因数优雅很多, 而且之后遇到类似题目, 修改列混合的因数, 可以直接按照上面的方法通杀