陇原战疫2021网络安全大赛_Crypto_复现

原创

M3ng@L 已于 2022-02-28 11:49:10 修改 6394 收藏

分类专栏： CTF比赛复现 文章标签： Crypto python

于 2022-02-28 11:47:30 首次发布

陇原战疫2021网络安全大赛_Crypto_复现

mostlycommon

keywords: 共模攻击

Decryption code

```
from Crypto.Util.number import *
import gmpy2

e1 = 65536
e2 = 270270
n = 122031686138696619599914690767764286094562842112088225311503826014006886039069083192974599712685027825111684
852235230039182216245029714786480541087105081895339251403738703369399551593882931896392500832061070414483233029 0
671174109524996554821601040277304627404973472127522695895262675041002627073670202446135 03
c1 = 394490164037354058923435072007400984775810396059796034847743477143816352119255859248127279914002780318923 91
996192354880233130336052873275920425836986816735715003772614138146640312241166362203750473990403841789871473337 0
674507276004863307234611006029527362323066024815653488348112927495472406194000847121496 73
c2 = 439414048358202739641420987820610435221253502807293661163119431711086891081144444472955119690901071295301 87
119024651382804933594308335681000311125969011096172605146903018110328309963467134604392943061014968838406604211 9
963224682767447140637357865052494167083943941693243159451454778834380035693724601722682 77

_,s,t = gmpy2.gcdext(e1,e2)
print(s,t)
m = pow(c1,s,n) * pow(c2,t,n) % n
print(gmpy2.gcd(e1,e2))
print(bytes.decode(long_to_bytes(gmpy2.iroot(m,2)[0])))
```

easytask

keywords: `GGH`， `embedded technique`

解题细节思路请见paper，(20条消息) GGH非对称密码体制破解方法_M3ng@L的博客-CSDN博客

简单来说，使用`embedded technique`方法将CVP问题转换为SVP问题，进而求得干扰向量$_e$的大小，减去$_e$再乘以公钥向量B的逆即可

Decryption code

```
from sage.modules.free_module_integer import IntegerLattice
c = e = [151991736758354,115130361237591,58905390613532,130965235357066,74614897867998,48099459442369,4589448578
2943,7933340009592,25794185638]
B = W = [[-10150241248,-11679953514,-8802490385,-12260198788,-10290571893,-334269043,-11669932300,-2158827458,-7
021995],
[52255960212,48054224859,28230779201,43264260760,20836572799,8191198018,14000400181,4370731005,14251110],
[2274129180,-1678741826,-1009050115,1858488045,978763435,4717368685,-561197285,-1999440633,-6540190],
[45454841384,34351838833,19058600591,39744104894,21481706222,14785555279,13193105539,2306952916,7501297],
[-16804706629,-13041485360,-8292982763,-16801260566,-9211427035,-4808377155,-6530124040,-2572433293,-8393737],
[28223439540,19293284310,5217202426,27179839904,23182044384,10788207024,18495479452,4007452688,13046387],
[968256091,-1507028552,1677187853,8685590653,9696793863,2942265602,10534454095,2668834317,8694828],
[33556338459,26577210571,16558795385,28327066095,10684900266,9113388576,2446282316,-173705548,-577070],
[35404775180,32321129676,15071970630,24947264815,14402999486,5857384379,10620159241,2408185012,7841686]]

c = matrix(c)
B = matrix(B)
temp = B.stack(c).augment(vector([0]*B.ncols()+[1]))
e = IntegerLattice(temp).shortest_vector()[:-1]
m = B.solve_left(c - matrix(e))
print(m)
m = [ 877  619  919  977  541  941  947 1031  821]

from Crypto.Util.number import *
from Crypto.Cipher import AES
import hashlib
c = 0x1070260d8986d5e3c4b7e672a6f1ef2c185c7fff682f99cc4a8e49cfce168aa0
m = [877,619,919,977,541,941,947,1031,821]
key = hashlib.sha256(str(m).encode()).digest()
aes = AES.new(key,AES.MODE_ECB)
flag = aes.decrypt(long_to_bytes(c))
print(flag)
```

Perference

hxp |伏尔加CTF 2016资格：crypto300"XXY"写

Mathematics of Public Key Cryptography (auckland.ac.nz)

GGH | 4XWi11的博客

Nguyen's Attack：Intended Solution to GGH in GYCTF 2020 | Soreat_u's Blog (soreatu.com)

Civet cat for Prince

keywords: `AES_CBC`， `xor`， `狸猫换太子`

Description

```
from Crypto.Cipher import AES
import os
from hashlib import sha256
import socketserver
import signal
import string
import random

table = string.ascii_letters + string.digits
BANNER = br'''
 .d8888b.  d8b                   888                            888
```

```
d88P  Y88b Y8P                    888                             888
888    888                        888                             888
888        888 888  888  .d88b.  888888        .d8888b  8888b.  888888
888        888 888  888 d8P  Y8b 888          d88P"        "88b 888
888    888 888 Y88  88P 88888888 888          888      .d888888 888
Y88b  d88P 888  Y8bd8P  Y8b.     Y88b.        Y88b.    888  888 Y88b.
 "Y8888P"  888   Y88P    "Y8888   "Y888        "Y8888P "Y888888  "Y888



 .d888                        8888888b.          d8b
d88P"                         888   Y88b         Y8P
888                           888    888
888888  .d88b.  888d888       888   d88P 888d888 888 88888b.   .d8888b  .d88b.
888    d88""88b 888P"         8888888P"  888P"   888 888 "88b d88P"    d8P  Y8b
888    888  888 888           888        888     888 888  888 888      88888888
888    Y88..88P 888           888        888     888 888  888 Y88b.    Y8b.
888     "Y88P"  888           888        888     888 888  888  "Y8888P  "Y8888
'''

guard_menu = br'''
1.Tell the guard my name
2.Go away
'''

cat_menu = br'''1.getpermission
2.getmessage
3.say Goodbye
'''


def Pad(msg):
    return msg + os.urandom((16 - len(msg) % 16) % 16)


class Task(socketserver.BaseRequestHandler):
    def _recvall(self):
        BUFF_SIZE = 2048
        data = b''
        while True:
            part = self.request.recv(BUFF_SIZE)
            data += part
            if len(part) < BUFF_SIZE:
                break
        return data.strip()

    def send(self, msg, newline=True):
        try:
            if newline:
                msg += b'\n'
            self.request.sendall(msg)
        except:
            pass

    def recv(self, prompt=b'[-] '):
        self.send(prompt, newline=False)
        return self._recvall()

    def proof_of_work(self):
```

```
        proof = (''.join([random.choice(table) for _ in range(12)])).encode()
        sha = sha256(proof).hexdigest().encode()
        self.send(b"[+] sha256(XXXX+" + proof[4:] + b") == " + sha)
        XXXX = self.recv(prompt=b'[+] Give Me XXXX :')
        if len(XXXX) != 4 or sha256(XXXX + proof[4:]).hexdigest().encode() != sha:
            return False
        return True

    def register(self):
        self.send(b'')
        username = self.recv()
        return username

    def getpermission(self, name, iv, key):
        aes = AES.new(key, AES.MODE_CBC, iv)
        plain = Pad(name)+b"a_cat_permission"
        return aes.encrypt(plain)

    def getmessage(self, iv, key, permission):
        aes = AES.new(key, AES.MODE_CBC, iv)
        return aes.decrypt(permission)

    def handle(self):
        signal.alarm(50)
        if not self.proof_of_work():
            return
        self.send(BANNER, newline=False)
        self.key = os.urandom(16)
        self.iv = os.urandom(16)
        self.send(b"I'm the guard, responsible for protecting the prince's safety.")
        self.send(b"You shall not pass, unless you have the permission of the prince.")
        self.send(b"You have two choices now. Tell me who you are or leave now!")
        self.send(guard_menu, newline=False)
        option = self.recv()
        if option == b'1':
            try:
                self.name = self.register()
                self.send(b"Hello " + self.name)
                self.send(b"Nice to meet you. But I can't let you pass. I can give you a cat. She will play with
you")
                self.send(b'Miao~ ' + self.iv)
                for i in range(3):
                    self.send(b"I'm a magic cat. What can I help you")
                    self.send(cat_menu, newline=False)
                    op = self.recv()
                    if op == b'1':
                        self.send(b"Looks like you want permission. Here you are~")
                        permission = self.getpermission(self.name, self.iv, self.key)
                        self.send(b"Permission:" + permission)
                    elif op == b'2':
                        self.send(b"Looks like you want to know something. Give me your permission:")
                        permission = self.recv()
                        self.send(b"Miao~ ")
                        iv = self.recv()
                        plain = self.getmessage(iv, self.key, permission)
                        self.send(b"The message is " + plain)
                    elif op == b'3':
                        self.send(b"I'm leaving. Bye~")
                        break
                self.send(b"Oh, you're here again. Let me check your permission.")
```

```
                self.send(b"Give me your permission:")
                cipher = self.recv()
                self.send(b"What's the cat tell you?")
                iv = self.recv()
                plain = self.getmessage(iv, self.key, cipher)
                prs, uid = plain[16:],plain[:16]
                if prs != b'Princepermission' or uid != self.name:
                    self.send(b"You don't have the Prince Permission. Go away!")
                    return
                else:
                    self.send(b"Unbelievable! How did you get it!")
                    self.send(b"The prince asked me to tell you this:")
                    f = open('flag.txt', 'rb')
                    flag = f.read()
                    f.close()
                    self.send(flag)
            except:
                self.request.close()
        if option == b'2':
            self.send(b"Stay away from here!")
        self.request.close()


class ThreadedServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    pass


class ForkedServer(socketserver.ForkingMixIn, socketserver.TCPServer):
    pass


if __name__ == "__main__":
    HOST, PORT = '0.0.0.0', 10005
    print("HOST:POST " + HOST + ":" + str(PORT))
    server = ForkedServer((HOST, PORT), Task)
    server.allow_reuse_address = True
    server.serve_forever()
```

Analysis

首先是执行`proof of work`函数，简单的爆破四个字符使得其连接上剩余已知字符的`sha256`等于函数给出的`sha256`值

然后进入正题，第一段守卫处没有有用信息，需要输入你的`name`（之后称为`m1`），然后在`magic cat`处可以获得

- `getpermission()`

得到对`m1+b'a_cat_permission'`的AES_CBC加密之后得到的密文，其中`iv`是系统生成的已知量，`key`未知

```
    def getpermission(self, name, iv, key):
        aes = AES.new(key, AES.MODE_CBC, iv)
        plain = Pad(name)+b"a_cat_permission"
        return aes.encrypt(plain)
```

- `getmessage()`

输入一段密文进行AES_CBC解密，其中`iv`由我们给出，`key`是之前使用的，得到解密之后给出的明文

```
    def getmessage(self, iv, key, permission):
        aes = AES.new(key, AES.MODE_CBC, iv)
        return aes.decrypt(permission)
```

总共有三次询问的机会（循环次数为3），循环完成之后由守卫进行验证

提交给守卫某个`密文`和自己给出的偏移量`iv`，使得进行相同的`key`的AES_CBC解密之后得到的明文是`m1+b'Princeperimission'`

```
self.send(b"Give me your permission:")
cipher = self.recv()
self.send(b"What's the cat tell you?")
iv = self.recv()
plain = self.getmessage(iv, self.key, cipher)
prs, uid = plain[16:],plain[:16]
if prs != b'Princepermission' or uid != self.name:
    ...
```

以上就是对题目代码的分析

简单来说，我们需要想办法把从`magic cat`处得到的`perimission = m1 + b'a_cat_perimission'`的密文换成`m1 + b'Princepermission'`的密文

其中可以用到的是`magic cat`会提供一次或两次`解密`的机会（用自己给出的`iv`）

显然没有办法知道`key`进而直接求解，但是我们不需要老老实实地直接找 $need\_m + {}_{need\_m}$ 的密文，因为我们可以构造合适的`iv`代入最后的AES_CBC解密

先令

$m = input\_name$

$m = a\_cat\_permission$

$need\_m = {}_{m}$

$need\_m = Princepermission$

由于AES_CBC加密是分块加密，而每一块的大小是`128bits`也就是`16字节`，而由于题目要求，${}_{m}$ 和 ${}_{m}$ 以及 $need\_m$ 都是16字节大小的

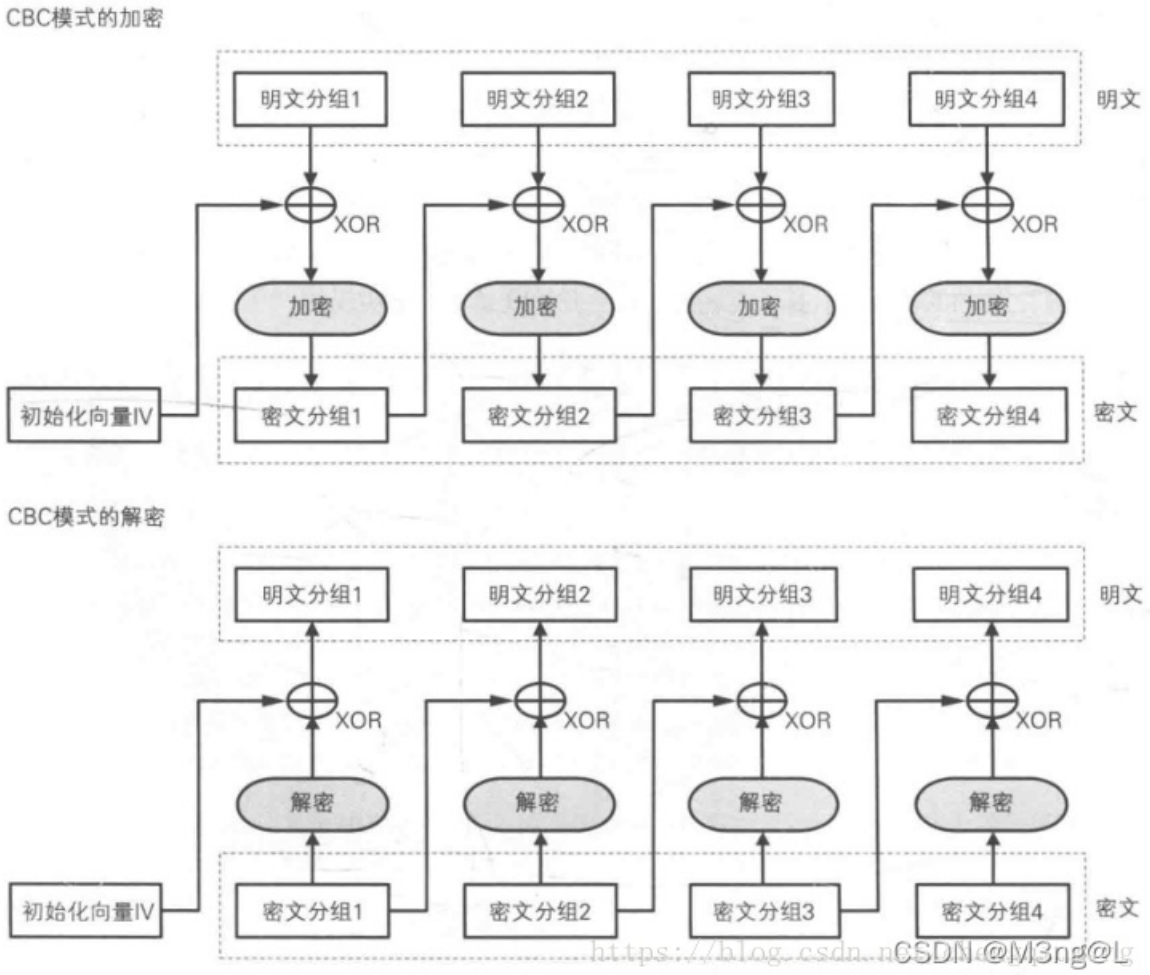
这就意味着加密得到的密文块会有特殊的性质

令

通过图片明确一下AES_CBC加解密过程

加密过程是，先将明文按`16字节`一分组拆开，然后进行如图加密（先进行异或，非明文分组1的异或对象是前一个密文分组或者是明文分组1的异或对象初始化向量`iv`），再分别投入加密器，得到各个密文分组后拼接起来成为最后的密文

解密过程是，将密文按`16字节`一分组拆开，然后分别投入解密器，得到的结果与前一个密文分组或者初始化向量进行异或，最后得到各个明文分组再直接拼接在一起组成最后的明文



其中加密器和解密器在本题算是黑盒，只需要知道进去的数据和原来是不一样的即可

我们构造的密文要先经过一次解密器，这就意味着非经过AES_CBC加密产生的密文（比如 c ）都会产生一个未知的数据：那么这时就需要用到 `magic cat` 提供的 `解密` 的机会

经过解密器之后会进行异或，那么这就是本题的关键之一了，异或的性质就是两个一样的数异或均为0，也就是说对其他一起的数不会有影响，利用这个性质我们构造

$fake\_c = c \oplus$

使用系统给出的 `iv` 代入 `magic cat` 提供的一次解密的机会

先经过解密器得到$m \oplus iv \oplus$

在与初始化向量 `iv` 异或得到$m =$

接着构造

$fake\_iv =$

那么最后传入的密文是 $fake\_c +$ ，传入的 `iv` 也就是 $fake\_iv$

推导一下最后的解密过程，（AES_CBC是分块解密的）

$fake\_c$ 单独进行解密得到$m \oplus iv \oplus$

然后再与$fake_iv$异或得到

$m \oplus iv \oplus$ decrypted $m \oplus$

=

得到明文分组1，符合最后的判断条件

然后c单独进行解密，得到$c \oplus$

再与$fake_c$进行异或得到$c \oplus m \oplus$

这就是明文分组2，符合最后的判断条件

Decryption code

```
from re import L
from pwn import *
import hashlib,string,random
from Crypto.Cipher import AES

io = remote("node4.buuoj.cn","27370")
temp = io.recvline()
# print(temp)
temp1 = temp.split(b"==")
# print(temp1)
part_proof = bytes.decode(temp1[0].split(b"XXXX")[1])[1:-2]
sha = bytes.decode(temp1[1]).strip()
table = string.ascii_letters + string.digits
while True:
    XXXX = "".join([random.choice(table)for _ in range(4)])
    temp_proof = XXXX + part_proof
    temp_sha = hashlib.sha256(temp_proof.encode()).hexdigest()
    if sha == temp_sha:
        io.recvuntil(b"[+] Give Me XXXX :")
        io.sendline(XXXX.encode())
        break
io.recvuntil(b"[-] ")
io.sendline(b"1")
io.sendline(b"1" * 16)
io.recvuntil(b'Miao~ ')
iv = io.recvuntil(b"\n")[:-1]
# print(iv)
io.recvuntil(b'[-]')
io.sendline(b'1')
io.recvuntil(b'Permission:')
cat_permission = io.recvline()[:-1]
# print(cat_permission)
io.recvuntil(b"[-]")
io.sendline(b'2')
io.recvuntil(b"Looks like you want to know something. Give me your permission:")

m2 = b"a_cat_permission"
m1 = b'1' * 16 # your name,直接16字节，不会进行pad()函数，如果那样的话，m1也会成为一个未知量
need_m2 = b"Princepermission"
need_m1 = m1
c1 = cat_permission[:16]
c2 = cat_permission[16:]
fake_c1 = xor(xor(c1,m2),need_m2)

io.sendline(fake_c1)
io.recvuntil(b"Miao~ ")
```

```
io.sendline(iv)
io.recvuntil(b"The message is ")
m = io.recvuntil(b"\n")[:-1]
# print(plain)
io.recvuntil(b"[-]")
io.sendline(b'3')
io.recvuntil(b"Give me your permission:")

fake_permission = fake_c1 + c2
fake_iv = xor(xor(m,m1),iv)

io.sendline(fake_permission)
io.recvuntil(b"What's the cat tell you?")
io.sendline(fake_iv)
io.interactive()
```

Perference

完美起航-20211107陇原战疫Crypto方向WP (okgoes.cn)

(11条消息) CBC模式解读_实践求真知-CSDN博客_cbc模式