湖湘杯2019两个密码题wp

原创

b0ring 于 2019-11-10 12:54:59 发布

分类专栏： crypto CTF 文章标签： 湖湘杯2019writeup

本文链接：https://blog.csdn.net/s1054436218/article/details/102996499

湖湘杯2019两个密码题wp

还是自己太菜的原因，这次湖湘杯只做出来4道题，然后5点的时候就放弃了去跟同学出去玩了，当时感觉进前50无望（这次湖湘杯py情况也很严重啊，可惜烽火台只报不封，挺恶心的）。不过无论如何，这次比赛还是有收获的，总结沉淀一下这两道密码学题目吧：

Oracle padding attack

忘了这个题目是啥了，但是攻击原理就是针对CBC模式的特殊攻击方式，Oracle padding attack。详细原理可以从下面获取（介绍的很详细，强烈推荐）：

https://www.freebuf.com/articles/database/151167.html

下面简单介绍一下利用原理：

Oracle padding attack的攻击条件还是比较苛刻的：

> 1. 在给服务器返回的数据中，iv可控。（iv是对称加密中的偏移向量，不清楚的同学可以详细了解一下CBC等加密模式的原理。）
> 2. 在响应数据中，攻击者可以通过报错等方式得知padding是否出错。
> 3. 服务器采用对称加密算法，使用CBC模式和PCK#5填充法。（这个填充法就是在最后一组长度不足分组长度的时候，填充剩余多少字节个数，例如16字节分组中最终还剩6个，那就填充6个ascii的6）
> 4. 初始的iv已知。

在服务器解密后，被解密的明文会和iv进行异或（这个是对称加密算法中CBC加密模式的基本原理，这里不懂建议回去复习一下，方便理解），之后会进行去填充处理，一般算法库对这种去填充的实现是：

1. 检查最后一个字节的ascii值，这里为了方便解释我们假设是6。
2. 检查后6个字节是不是6。
3. 如果是6，那么去掉这6个6，否则就报错或提示信息。

这里注意：上述过程中检查的是被我们解密后，再和初始向量异或后的明文串。如果向服务器返回的iv可控，那么我们就可以通过不断改变iv的方法来通过报错达到被解密的数据串可控的目的，同时也可以任意恢复原文！我们假设分组长度为16，简述这个过程如下：

1. 控制初始向量，前15个字符串任意，我们控制第16个字符串，从1爆破到255，不报错的那个结果肯定是使密钥解密出的明文串异或后为1。这是我们如果异或真正的初始向量，再异或1就恢复出了明文。这个明文再异或初始向量和我们期望解密的结果，那么这个串解密的结果就可控了。举个例子：假设我们爆破出的最后一个字符是0x20，原本初始向量的值是0x41，那么原本的明文串的最后一个字符就是：0x20 ^ 0x41 ^ 0x01=0x60。假设我们希望服务器解密这个串时最后一个字符是0x02，那么我们就可以调整初始向量为:0x60 ^ 0x41 ^ 0x02=0x23。
2. 根据第一步，我们已经爆破出了分组中的最后一个明文，先控制最后一个初始向量，使其经解密后结果为0x02，然后我们再控制倒数第二个初始向量，从1爆破到255，那么不报错的那个结果肯定是使密码解密处的明文串异或后为2。这时我们以此类推，爆破倒数第三个。

原理就是这些，这道题使用脚本攻击如下：

```
#enconding:utf-8
'''
    @Author: b0ring
    @MySite: https://unnamebao.github.io/
    @Date:  2019-11-10 12:02:53
    @Version: 1.0.0
'''

import socket
import base64
import codecs

def num2str(i):
    return ("%02x"%i).encode()

def postfix_str(postfix,iv,index):
    res_ = b""
    for i in range(int(len(postfix)/2)):
        res_ = num2str(int(postfix[len(postfix) - (i+1)*2:len(postfix)-i*2],16)^int(iv[32-(i+1)*2:32-i*2],16)^index) + res_
    return res_

def brute_one(IV,encryp_text,client,index,postfix):
    for i in range(256):
        if index == 0:
            if i == IV[-1]:
                continue
        if i == 0:
            print("[*] append:",postfix_str(postfix,IV,index))
        iv = b"41"*(16-index) + num2str(i) + postfix_str(postfix,IV,index)
        data_to_send = iv + encryp_text + b'\n'
        client.send(data_to_send)
        data_respond = client.recv(1024)
        # print(i,int(iv[32-index*2:32-index*2+2],16)^int(IV[32-index*2:32-index*2+2],16),data_respond)
        if b"padding error" in data_respond:
            continue
```

```
        else:
            return num2str(i^int(IV[32-index*2:32-index*2+2],16)^index)


if __name__ == "__main__":
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('183.129.189.62', 13706))
    data = client.recv(1024)
    print(data)
    c = data.decode().split("\n")[0].replace("Hey, new! Your passport is ","")[:64]
    IV= c[:32].encode()
    encryp_text = c[32:].encode()
    print("[*] IV:",IV)
    print(type(IV))
    postfix = b""
    for index in range(0,16):
        one = brute_one(IV,encryp_text,client,index+1,postfix)
        postfix = one + postfix
        print("[*] postfix:",postfix)
    print('------------------------------')
    payload = b""
    admin = b"Admin"
    for i in range(5):
        payload += num2str(admin[i] ^ int(postfix[i*2:i*2+2],16) ^ int(IV[i*2:i*2+2],16))
    for i in range(5,16):
        payload += num2str(11 ^ int(postfix[i*2:i*2+2],16) ^ int(IV[i*2:i*2+2],16))
    print(type(payload))
    print(payload)
    print(len(payload))
    payload += encryp_text + b'\n'
    client.send(payload)
    print(client.recv(1024))
```

Easy RSA

首先看一下题目吧：

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

flag = open("flag.txt","rb").read()
m=libnum.s2n(flag)
p=getPrime(1024)
q=getPrime(1024)
n=p*q
e=65537
c=pow(m,e,n)
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
dp=d%(p-1)
print dp,n,e,c

#84373069210173690047629226878686144017052129353931011112880892379361035492516066159394115482289291025932915787077633999791002846189004408043685986856359812230222233165493645074459765748901898518115384084258143483508823079115319711227124403284267559950883054402576935436305927705016459382628196407373896831725 22000596569856085362623019573995240143720890380678581299411213688857584612953014122879995808816872221032805734151343458921719334360194024890377075521680399678533655114261000716106870610083356478621445541840124447459943322577402684072179500812171300550579268160650682759996205027668663794655210422983706860538234480997785728787657827112606731857038891687027461957792503736425053757259252137968484955188784907860353630940865202570200215478270737685986001519287874341530036750962547922450142170446074408906941909891623188461043853116461233437951494899462512217740304844245818468411418196018745621092280167073642208406116553 14874271064669918581178066047207495551570421575260298116038863877424499500626920855863261194264169850678206604144314318171829367575688726593323863145664241189167820996601561389159819873734368810449011761054668595565217970516125181240869998009561140277444653698278073509852288720276008438965069627886972839146199102497874818473454932012374251932864118784065064885987416408142362577322906063320726241313252172382519793691513360909796645028353257317044086708114163313328952830378067342164675055195428728335222242094290731292113709866489975077052604333805889421889967835433026770417624703011718120347415460385182429795735
```

我们从中提取一些信息：

```
m = flag # m中的值是flag的明文串
n = p * q # 这个很清晰 学过RSA应该理解
e = 65537 # 提醒下，65537是个质数，虽然好像是不是质数无所谓
c = pow(m,e,n) # 这里就是典型的RSA加密了，如果已知d，可以通过 m = pow(c,d,n)解出明文，其中，(e,n)是公钥，d是私钥。
phi = (p-1)*(q-1) # 这里计算了n的欧拉函数，解释一下，欧拉函数是小于一个数有多少与其互质的数，质数的欧拉函数就是自身减一，而两个质数相乘的数，它的欧拉函数就是两个质数相加减一相乘
d = gmpy2.invert(e,phi) # 这里就是求同余phi下与e互质的数，就是私钥d
dp = d % (p-1) # 就是d对(p-1)求余
```

梳理一下：

```
e,c,n已知，dp = d % (p-1)已知，求m
```

这个地方涉及到一个原理：

```
对于任意因子a、b（不需要是质数），

m一        如果
```

而我们知道，d和e在同余phi下是互逆的，所以：

$d*$

因此，

$(dp*$ $-1)$ 一定是(p-1)的倍数，且p是素数。而且，我们又知道，$dp <$ （ 因此遍历1-65537 一

使用脚本如下：

```
#enconding:utf-8
'''
    @Author: b0ring
    @MySite: https://unnamebao.github.io/
    @Date:  2019-11-10 12:36:18
    @Version: 1.0.0
'''

import Crypto.Util.number as number
def fastpow(Co, CoCo, CoCoCo):
    CoCoCoCo = 1
    while CoCo != 0:
        if (CoCo & 1) == 1:
            CoCoCoCo = (CoCoCoCo * Co) % CoCoCo
        CoCo >>= 1
        Co = (Co * Co) % CoCoCo
    return CoCoCoCo


def gcd(num_1,num_2):
    p,q=max(num_1,num_2),min(num_1,num_2)
    if q == 0:
        return p
    r = p%q
    return gcd(q,r)

def EX_GCD(a,b,arr): #扩展欧几里得
    if b == 0:
        arr[0] = 1
        arr[1] = 0
        return a
    g = EX_GCD(b, a % b, arr)
    t = arr[0]
    arr[0] = arr[1]
    arr[1] = t - int(a // b) * arr[1]
    return g


def ModReverse(a,n):
    arr = [0,1,]
    gcd = EX_GCD(a,n,arr)
    if gcd == 1:
        return (arr[0] % n + n) % n
    else:
        return -1

dp = 84373069210173690047629226878686144017052129353931011112880892379361035492516066159394115482289291025932915787077633999791002846189004408043685986856359812230222233165493645074459765748901898518115384084258143483508823079115319711227124403284267559950883054402576935436305927705016459382628196407373896831725
n = 22000596569856085362623019573995240143720890380678581299411213688857584612953014122879995808816872221032805730
```

```
41513434589217193343601940248903770755216803996785336551142610007161068706100833564786214455418401244474599433225777402684072179500812171300550579268160650682759996205027668663794655210422983706860538234480997785728787657827112606731857038891687027461957792503736425053757259252137968484955188784907860353630940865202570200215478270737685986001519287874341530036750962547922450142170446074408906941909891623188461043853116461233437951494899462512217740304844245818468411418196018745621092280167073642208406110
e = 65537
c = 14874271064669918581178066047207495551570421575260298116038863877424499500626920855863261194264169850678206604144314318171829367575688726593323863145664241189167820996601561389159819873734368810449011761054668595565217970516125181240869998009561140277444653698278073509852288720276008438965069627886972839146199102497874818473454932012374251932864118784065064885987416408142362577322906063320726241313252172382519793691513360909796645028353257317044086708114163313328952830378067342164675055195428728335222242094290731292113709866489975077052604333805889421889967835433026770417624703011718120347415460385182429795735

# tmp1 = (dp * e) -1

for i in range(65537)[::-1]:
    if (dp*e-1)%i == 0 and number.isPrime((dp*e-1)//i + 1):
        p = (dp*e-1)//i + 1
        break
q = n // p
phi = (p-1)*(q-1)
d = ModReverse(e,phi)
m = fastpow(c,d,n)
print(number.long_to_bytes(m))
```

当然了，这里可以这样用是因为e比较小，那么如果e很大怎么办呢？其实我们已知e和dp在同余(p-1)下是互逆的，那么：

对于任意一个数r，

$r \equiv \mod p$

也就是说，对于任意的r，我们可以利用

$r - r$ 来找到p的一个倍数，再使用扩展欧几里得算法去求其与n的最大公因数，就可以算出p了，脚本具体

```
#enconding:utf-8
'''
    @Author: b0ring
    @MySite: https://unnamebao.github.io/
    @Date:  2019-11-10 12:36:18
    @Version: 1.0.0
'''

import Crypto.Util.number as number
def fastpow(Co, CoCo, CoCoCo):
    CoCoCoCo = 1
    while CoCo != 0:
        if (CoCo & 1) == 1:
            CoCoCoCo = (CoCoCoCo * Co) % CoCoCo
        CoCo >>= 1
        Co = (Co * Co) % CoCoCo
    return CoCoCoCo


def gcd(num_1,num_2):
    p,q=max(num_1,num_2),min(num_1,num_2)
    if q == 0:
        return p
```

```
    r = p%q
    return gcd(q,r)

def EX_GCD(a,b,arr): #扩展欧几里得
    if b == 0:
        arr[0] = 1
        arr[1] = 0
        return a
    g = EX_GCD(b, a % b, arr)
    t = arr[0]
    arr[0] = arr[1]
    arr[1] = t - int(a // b) * arr[1]
    return g


def ModReverse(a,n):
    arr = [0,1,]
    gcd = EX_GCD(a,n,arr)
    if gcd == 1:
        return (arr[0] % n + n) % n
    else:
        return -1

dp = 84373069210173690047629226878686144017052129353931011112880892379361035492516066159394115482289291025932915787077633999791002846189004408043685986856359812230222233165493645074459765748901898518115384084258143483508823079115319711227124403284267559950883054402576935436305927705016459382628196407373896831725
n = 22000596569856085362623019573995240143720890380678581299411213688857584612953014122879995808816872221032805734151343458921719334360194024890377075521680399678533655114261000716106870610083356478621445541840124447459943322577740268407217950081217130055057926816065068275999620502766866379465521042298370686053823448099778572878765782711260673185703889168702746195779250373642505375725925213796848495518878490786035363094086520257020021547827073768598600151928787434153003675096254792245014217044607440890694190989162318846104385311646123343795149489946251221774030484424581846841141819601874562109228016707364220840611
e = 65537
c = 14874271064669918581178066047207495551570421575260298116038863877424499500626920855863261194264169850678206604144314318171829367575688726593323863145664241189167820996601561389159819873734368810449011761054668595565217970516125181240869998009561140277444653698278073509852288720276008438965069627886972839146199102497874818473454932012374251932864118784065064885987416408142362577322906063320726241313252172382519793691513360909796645028353257317044086708114163313328952830378067342164675055195428728335222242094290731292113709866489975077052604333805889421889967835433026770417624703011718120347415460385182429795735

mp = fastpow(e,e*dp,n)-e
p = gcd(mp,n)
q = n // p
phi = (p-1)*(q-1)
d = ModReverse(e,phi)
m = fastpow(c,d,n)
print(number.long_to_bytes(m))
```