CBC-Padding攻击

原创

Risker_GML 于 2018-10-20 13:23:15 发布 1059 收藏 3
文章标签： CTF Crypto

本文链接：https://blog.csdn.net/qq_38412357/article/details/83212668

欢迎关注我的新博客：http://mmmmmmlei.cn

对于 CBC 翻转字节攻击以及 Padding Oracle Attack 这块的知识一直不怎么会运用，所以今天复现了一道在Xman 夏令营打排位赛的一道密码题，算是 Padding Oracle 的一个简化版，思想大同小异，只是这个题没有和服务器交互。

这个题也是仿 hack.lu 2016 的一道题，文章末尾有链接地址。

看下题，给了 `AESCipher.py` , `lockfile.py` 和 `flag.encrypted` 三个文件。
代码：

AESCipher.py

```python
from Crypto import Random
from Crypto.Cipher import AES


class AESCipher(object):

    def __init__(self, key):
        self.bs = 32
        self.key = key

    @staticmethod
    def str_to_bytes(data):
        u_type = type(b''.decode('utf8'))
        if isinstance(data, u_type):
            return data.encode('utf8')
        return data

    def _pad(self, s):
        return s + (self.bs - len(s) % self.bs
                    ) * AESCipher.str_to_bytes(chr(self.bs - len(s) % self.bs))

    @staticmethod
    def _unpad(s):
        return s[:-ord(s[len(s) - 1:])]

    def encrypt(self, raw):
        raw = self._pad(AESCipher.str_to_bytes(raw))
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return iv + cipher.encrypt(raw)

    def decrypt(self, enc):
        iv = enc[:AES.block_size]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return cipher.decrypt(enc[AES.block_size:])
```

lockfile.py

```python
#!/usr/bin/env python3
import sys
import hashlib
from AESCipher import *


class FileLocker(object):

    def __init__(self, keys):
        assert len(keys) == 4
        self.keys = keys
        self.ciphers = []
        for i in range(4):
            self.ciphers.append(AESCipher(keys[i]))

    def enc(self, plaintext):
        stage1 = self.ciphers[0].encrypt(plaintext)
        stage2 = self.ciphers[1].encrypt(stage1)
        stage3 = self.ciphers[2].encrypt(stage2)
        ciphertext = self.ciphers[3].encrypt(stage3)
        return ciphertext

    def dec(self, ciphertext):
        stage3 = AESCipher._unpad(self.ciphers[3].decrypt(ciphertext))
        stage2 = AESCipher._unpad(self.ciphers[2].decrypt(stage3))
        stage1 = AESCipher._unpad(self.ciphers[1].decrypt(stage2))
        plaintext = AESCipher._unpad(self.ciphers[0].decrypt(stage1))
        return plaintext


if __name__ == "__main__":
    if len(sys.argv) != 3:
        # PASSWORD SHOULD BE Visible character
        print("Usage: ./lockfile.py plainfile password")
        exit()

    filename = sys.argv[1]
    plaintext = open(filename, "rb").read()

    password = sys.argv[2].encode('utf-8')
    assert len(password) == 8
    i = len(password) / 4
    keys = [
        hashlib.sha256(password[0:i]).digest(),
        hashlib.sha256(password[i:2 * i]).digest(),
        hashlib.sha256(password[2 * i:3 * i]).digest(),
        hashlib.sha256(password[3 * i:4 * i]).digest(),
    ]
    s = FileLocker(keys)

    ciphertext = s.enc(plaintext)

    open(filename + ".encrypted", "w").write(ciphertext)
```

可以看到加密程序对于 key 的处理很特别，八字符的 key 被分成了四组，每组两个字符哈希后作为 AES 密钥，把明文加密了四次。采用的是下一轮加密上一轮的密文这种形式。

注意到这里的填充方式：

```
def _pad(self, s):
        return s + (self.bs - len(s) % self.bs
                    ) * AESCipher.str_to_bytes(chr(self.bs - len(s) % self.bs))
```

采用的是类似 PKCS5 的填充方式，也就是说无论明文多少位都需要填充。

直接爆破八个字符显然不可能，数量级是`len(dict)^8`,我们考虑把每步分解，如果四轮加密中，每一轮都能知道当前的两个字符是否正确，那么数量级就变成了`4*(len(dict)^2)`,还是很容易的。

既然使用了这个填充规则，每一轮就可以通过判断解密后的最后填充来判断解密密钥是否正确，这是判断填充正确与否的代码：

```
def checkPadding(raw):
    s=raw[-1]
    if s==chr(0) or s==chr(1):
        return False
    if raw[len(raw)-ord(s):]==ord(s)*s:
        return True
    else:
        return False
```

最后一个字节是`chr(0)`肯定不正确，最后一个字节是`chr(1)`倒是有可能正确，正好缺了一个字节，但只会发生在填充最原始的明文的时候，因为第二轮开始加密的都是上一轮的密文，都是 AES 分组大小的倍数，不可能差一个字节，而且爆破的过程中如果解密出来最后一个字节正好是 chr(1),就会认为填充正确，判断错误的几率还是很大的，所以代码里直接返回 False 了。

exp 如下：

```python
# -*- coding:utf-8 -*-
import hashlib
import string
import libnum
from AESCipher import *

def checkPadding(raw):
    s=raw[-1]
    if s==chr(0) or s==chr(1):
        return False
    if raw[len(raw)-ord(s):]==ord(s)*s:
        return True
    else:
        return False
flag_enc=open("flag.encrypted").read()

dict=[]
for x in string.printable:
    for y in string.printable:
        dict.append(hashlib.sha256(x+y).digest())

cipher=flag_enc
for i in range(4):
    for key in dict:
        raw=AESCipher(key).decrypt(cipher)
        if checkPadding(raw):
            cipher=AESCipher._unpad(raw)
            break
flag=""
i=0
while i<len(cipher)-1:
    if cipher[i]=='1':
        flag+=chr(int(cipher[i:i+3]))
        i=i+3
    else:
        flag+=chr(int(cipher[i:i+2]))
        i=i+2
print flag
```

需要注意的是解密出来的明文是一串数字:

`120109971101236510811410197100121457610197114110101100456766674580971001001051101031 25`

用正常 `hex解码` 是乱码，观察发现首部 120,109,97 都像是 ascii 值，数字其实是 flag 字符串每个字符十进制 ascii 值连起来的。

> 运行结果： xman{Already-Learned-CBC-Padding}

参考：

hack.lu 2016 原题目

https://p-te.fr/2016/10/20/hack-lu-cryptolocker/

https://gophers-in-the-shell.herokuapp.com/hack-lu-2016-cryptolock-crypto-200-pts/