Padding Oracle Attack实验报告

一、漏洞简介

一些系统在解密给定的密文 时，会验证填充(padding)是否有效，如果填充无效则抛出错误。这种看似无害的行为导致了一种 称为padding oracle的攻击。该攻击最初由 Serge Vaudenay 于 2002 年发布，许多知名系统都被发 现容易受到这种攻击，包括 Ruby on Rails、ASP.NET 和 OpenSSL。

二、实验过程

2.1 任务一

任务一需要验证PKCS#5 填充方案。

我们首先创建长度分别为5、10、16字节的文件P1、P2、P3：

```
echo -n "12345" > P1
echo -n "0123456789" > P2
echo -n "0123456789abcdef" > P3
```

之后使用openssl对文件进行加密、解密并不删除padding，指令如下所示：

```
openssl enc -aes-128-cbc -e -in P1 -out C1
openssl enc -aes-128-cbc -d -nopad -in C1 -out P11
```

查看解密后的文件：

```
[12/25/21]seed@VM:~/.../Labsetup$ xxd P11
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
[12/25/21]seed@VM:~/.../Labsetup$ xxd P22
00000000: 3031 3233 3435 3637 3839 0606 0606 0606  0123456789......
[12/25/21]seed@VM:~/.../Labsetup$ xxd P33
00000000: 3031 3233 3435 3637 3839 6162 6364 6566  0123456789abcdef
00000010: 1010 1010 1010 1010 1010 1010 1010 1010  ................
```

可以看出，当文件大小为5字节时，填充了十一个0x0b，当文件大小为10字节时，填充了六个0x06，当文件大小为16字节时，填充了十六个0x10，这符合我们的预期。

2.2 任务二

在这个任务中，我们提供了一个托管在 5000 端口的padding oracle 服务器。服务器里面有一个 秘密消息，它打印出这个秘密消息的密文。使用的加密算法和方式为AES-CBC，加密密钥为K，我们需要使用padding oracle攻击来推导这条消息。

我们首先查看初始向量和密文，如下所示：

```
[12/25/21]seed@VM:~/.../Labsetup$ nc 10.9.0.80 5000
010203040506070801020304050607 08a9b2554b0944118061212098f2f238cd779ea0aae3d9d020
f3677bfcb3cda9ce
```

可以看到密文共有32字节，故加密时，分为了两块。我们首先把传入的密文第一块全置为0，再让第一块的最后一字节遍历0-255，即可找到中间值的最后一字节，如下所示：

```
[12/25/21]seed@VM:~/.../Labsetup$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0xcf
CC1: 000000000000000000000000000000cf
P2:  00000000000000000000000000000000
```

可以看到当取0xcf的时候，padding验证通过，说明：$D2[15] = 0xcf \oplus 0x01$

说明D2最后一字节为0xce。

接下来我们要让$CC[15] = 0xce \oplus 0x02$，并遍历CC[14]，结果如下所示：

```
[12/25/21]seed@VM:~/.../Labsetup$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0x39
CC1: 0000000000000000000000000039cc
P2:  00000000000000000000000000000000
```

同理有D2[14] = 0x3b。

接下来，让$CC[14] = 0x3b \oplus 0x03$，遍历CC[13]，结果如下所示：

```
[12/25/21]seed@VM:~/.../Labsetup$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0xf2
CC1: 00000000000000000000000000f238cd
P2:  00000000000000000000000000000000
```

得知，D2[13] = 0xf1。

同样地，我们可以知道D2[12] = 0x1c，D2[11] = 0x45，D2[10] = 0xec。

综上D2[10:15] 为 0xec, 0x45, 0x1c, 0xf1, 0x3c, 0xce。

2.3 任务三

任务三与任务二类似，但是每次与oracle建立新连接时，oracle都会生成一个新的密钥和IV来加密秘密消息，所以我们必须在脚本中实现自动化攻击。

首先查看初始向量和密文，如下图所示：、

```
[12/25/21]seed@VM:~/.../Labsetup$ nc 10.9.0.80 6000
6eef926732ac199bf1da992d173538fa8a07c79175c296a6156c9a6bcc390cebd1db987487a23ed133
45ead0bc8d750ce29e8275e37de3412cafd6a30ab16b3c
```

可以看出密文共有48字节，说明加密时分为了三块。

我们把任务一中的工作封装程循环，共有三个循环，每个循环寻找一个中间块。在每一个循环中，首先修改CC1，使其后K位，使其解密后的明文后K位为K+1。之后让CC1[15-K]从0到255遍历，如果某个值使得padding检查成功，修改字节数组D的值，最终让D和密文（或者初始向量）异或，最终得到明文。

修改代码如下所示：

```
    for K in range(16):
        for index in range(K):
            CC1[15 - index] = D1[15 - index] ^ (K + 1)
        for i in range(256):
            CC1[15 - K] = i
            status = oracle.decrypt(CC1 + C1 )
```

```
            if status == "Valid":
                x = i ^ (K + 1)
                D1[15 - K] = (x).to_bytes(1,byteorder='little')[0]

    CC1 = bytearray(0x00 for i in range(16))

    for K in range(16):
        for index in range(K):
            CC1[15 - index] = D2[15 - index] ^ (K + 1)
        for i in range(256):
            CC1[15 - K] = i
            status = oracle.decrypt(IV + CC1 + C2)
            if status == "Valid":
                x = i ^ (K + 1)
                D2[15 - K] = (x).to_bytes(1,byteorder='little')[0]

    CC1 = bytearray(0x00 for i in range(16))

    for K in range(16):
        for index in range(K):
            CC1[15 - index] = D3[15 - index] ^ (K + 1)
        for i in range(256):
            CC1[15 - K] = i
            status = oracle.decrypt(IV + C1 + CC1 + C3)
            if status == "Valid":
                x = i ^ (K + 1)
                D3[15 - K] = (x).to_bytes(1,byteorder='little')[0]
    ################################################################

    # Once you get all the 16 bytes of D2, you can easily get P2
    P1 = xor(IV, D1)
    P2 = xor(C1, D2)
    P3 = xor(C2, D3)
    print("D1:  " + D1.hex())
    print("D2:  " + D2.hex())
    print("D3:  " + D3.hex())
    print("P1:  " + P1.hex())
    print("P2:  " + P2.hex())
    print("P3:  " + P3.hex())
    print("P1:",P1)
    print("P2:",P2)
    print("P3:",P3)
```

结果如下所示:

```
[12/25/21]seed@VM:~/.../Labsetup$ python3 manual_attack.py
C1:  77970a5acff52b65e6064b8de63f052f
C2:  ced9f9dcc07ecea6692c15479f2ae785
D1:  9cdfc0c1a0b39b374c85c5842825380c
D2:  32d24e7a83944916c66739e8c658774a
D3:  afadd8fce82091f840044b18c103e587
P1:  285e5f5e29285e5f5e29205468652053
P2:  454544204c6162732061726520677265
P3:  61742120285e5f5e29285e5f5e290202
P1: bytearray(b'(^_^)(^_^) The S')
P2: bytearray(b'EED Labs are gre')
P3: bytearray(b'at! (^_^)(^_^)\x02\x02')
```

得到了秘密值，符合我们的预期。

三、总结

通过这次实验，复习了对称加密的分组加密的工作流程，学习了实际中经常用到的PKCS#5 填充方案。并爆破出了存储在服务器端的秘密值。这让我对分组对称加密、padding oracle有了更加深刻的理解。

最后身份感谢老师和助教的指导！