Pro1 重安装键攻击

本次project的所有代码在压缩包之中，具体的运行方法请见Readme.txt文件。

协议复现

协议复现代码在Ubuntu17.4 下使用g++ 编译生成elf文件。

客户端

客户端主要有两个线程，主线程主要负责数据传输部分，子线程负责握手部分，监听来自接入点的数据包。

子线程时刻开着一个recv，能够在任意时刻收到来自接入点的数据包，实现在进行发送密文数据的同时一旦收到来自接入点的请求，能够重新进行握手包的发送、接收和重新初始化参数操作。

主线程调用了openssl库对于数据进行加密。

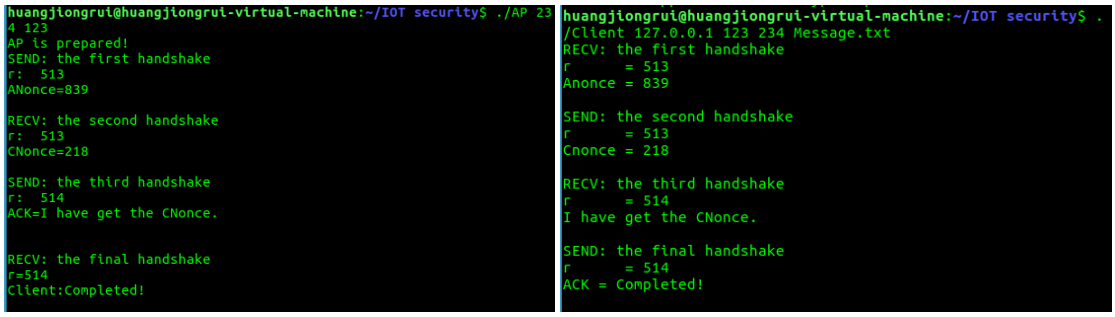
接入点

接入点因为一开始写成了单线程的，然后又懒得改，就直接在上面调，最后还真给我调出来了。单线程实现重新发包和接收加密数据。首先需要给接入点中的所有recv设置接收的最长时间以防止程序出现阻塞。其次对于部分函数进行循环，其跳出循环的条件是循环进行的时间。

运行结果截图

握手部分

正常运行



出现丢失

```
huangjiongrui@huangjiongrui-virtual-machine:~/IOT security$ ./AP 23
4 123
AP is prepared!
SEND: the first handshake
r:  693
ANonce=66

RECV: the second handshake
r:  693
CNonce=279

SEND: the third handshake
r:  694
ACK=I have get the CNonce.


Ciphertext = D76A796CC3D3296423BA318C96100BA8
Ciphertext = 06475E97DCEE7B89418CD9CF022A9439
Ciphertext = 1D9EB6E4DB58402CD4A7E3C84359DF02
SEND: the third handshake
r:  695
ACK=I have get the CNonce.


RECV: the final handshake
r=695
Client:Completed!
```

```
huangjiongrui@huangjiongrui-virtual-machine:~/IOT security$ ./Clien
t 127.0.0.1 123 234 Message.txt
RECV: the first handshake
r       = 693
Anonce = 66

SEND: the second handshake
r       = 693
Cnonce = 279

RECV: the third handshake
r       = 694
I have get the CNonce.

-------Nonce = 0 ----------
Plaintext  = POST, GET, HTTP,
Ciphertext = D76A796CC3D3296423BA318C96100BA8
Key = 87252A38EFF36E21779611C4C2445B84

-------Nonce = 1 ----------
Plaintext  =  INPUT, OUTPUT,
Ciphertext = 06475E97DCEE7B89418CD9CF022A9439
Key = 260E10C789BA57A90ED98D9F577EB819

-------Nonce = 2 ----------
Plaintext  = POST, GET, HTTP,
Ciphertext = 1D9EB6E4DB58402CD4A7E3C84359DF02
Key = 4DD1E5B0F7780769808BC380170D8F2E

RECV: the third handshake
r       = 695
I have get the CNonce.

SEND: the final handshake
r       = 695
ACK = Completed!
```

数据传输部分

正常运行

因为正常运行的时候AP和Client的输出是一模一样的，所以只截取了一张图

```
-------Nonce = 0 ----------
Plaintext  = POST, GET, HTTP,
Ciphertext = ECEB6A4E50628DBB56B40B5B8A5B6947
Key = BCA4391A7C42CAFE02982B13DE0F396B

-------Nonce = 1 ----------
Plaintext  =  INPUT, OUTPUT,
Ciphertext = 84E4589448A7E8E1FEFBF914D7AF1E59
Key = A4AD16C41DF3C4C1B1AEAD4482FB3279

-------Nonce = 2 ----------
Plaintext  = POST, GET, HTTP,
Ciphertext = 7F0A2EACB98B7957D87A791A887E5DEB
Key = 2F457DF895AB3E128C565952DC2A0DC7

-------Nonce = 3 ----------
Plaintext  =  INPUT, OUTPUT,
Ciphertext = 0D774180ED01714600A3B37B87FCA46A
Key = 2D3E0FD0B8555D664FF6E72BD2A8884A
```

出现丢失

```
-------Nonce = 0 ----------
Plaintext  =  INPUT, OUTPUT,
Ciphertext = A76C6468BAA7420138C34594971077A4
Key = 87252A38EFF36E21779611C4C2445B84

-------Nonce = 1 ----------
Plaintext  = POST, GET, HTTP,
Ciphertext = 76414393A59A10EC5AF5ADD7032AE835
Key = 260E10C789BA57A90ED98D9F577EB819

-------Nonce = 2 ----------
Plaintext  =  INPUT, OUTPUT,
Ciphertext = 6D98ABE0A22C2B49CFDE97D04259A30E
Key = 4DD1E5B0F7780769808BC380170D8F2E

-------Nonce = 3 ----------
Plaintext  = POST, GET, HTTP,
Ciphertext = EC7FBB14B0E1AE9F8B2DE705DD733E78
Key = BC30E8409CC1E9DADF01C74D89276E54
```

可以发现重新连接成功后的明文是继续发送的，只是将Nonce又重新初始化为0.

重安装键攻击

攻击设计

中间人攻击

中间人攻击的代码在Ubuntu17.4 下使用g++ 编译生成elf文件。

作为一个中间人转发client与AP之间的所有消息。

对于client发给AP的消息，检查其中的ACK包并且实施拦截，即不转发该包。此时由于client已经初始化完成了参数，可以按client发送的顺序截获一些加密的数据。对于AP传给client的所有包，中间人全部如实转发。

因此，由于中间人不断地截获ACK包，使AP的参数无法初始化，AP不断地要求client进行重发，导致了client的参数不断地重新初始化。因此，我们得到了一些用相同密钥加密的密文块。密文块的数量取决于中间人在捕获了多少次ACK包后选择对于ACK包进行转发。密文块越多，在之后的字典攻击中破解密钥就越方便。中间人将所有截获到的加密数据块用写文件的方式储存下来，为接下来的字典攻击做准备。

字典攻击

字典攻击的代码在Ubuntu17.4下使用python2.7进行解释

由于已知的明文只有“H T P O S G E I N U , SPACE” 这几个字符，所以可以使用字典攻击的方法对密文进行破解。本次project我只破解了Nonce=0时的密钥。

首先将所有的可能的异或值生成一个字典，把异或值作为键，对应的两个字符作为值。由于会出现'E'⊕'I' ==','⊕' '等类似情况，所以此时该键可以对应超过两个字符。同时，将键值0对应于'', 放入字典中。 将所有密文块两两异或，分别放入对应的列表中，所以对于同一个明文会有密文块数减一个异或的密文。

新建一个plaintext的列表，通过之前生成的字典对于每一个表内的数字进行查询，将所有可能的字符全部储存到plaintext列表的对应位置。选出对应位置字符串上出现频率最高的字符（可能会有多个，因为键值对应不止一对字符），打印出列表。

注意此时的列表可能会有大量的可能字符，所以需要做人工的填词游戏。相同密钥加密的密文块数量越多，需要人工进行校验的部分就越少。所以在进行中间人攻击的时候，最好有足够多的密文块能够被抓到。

在人工校验完成后，直接将该明文和已加密的密文进行异或就可以得到对应的密钥。

运行截图

Adversary:

捕获到的所有密文

captured_cipher

```
1
35A3AB0603323085E81DD3C848196CD2
C0418A9694546976FD60435F0F4ADD0F
2
22A9AC7E0F5A2394EC1DD3C848196CD2
C0418A9694546976FD60435F0F4ADD0F
A0B101E326EC652C5C0570248DA24BB7
21234C1C0DA9C08A8A2367F0DEF74929
42E924545AAF782F713A1E02A897A16B
3
49CCB7077B4222949011A3CE551D15A6
AB2491EFE04C6876816C3359124EA47B
CBD41A9A52F4642C2009002290A632C3
4A46576579B1C18AF62F17F6C3F3305D
298C3F2D2EB7792F0D366E04B593D81F
4
22A9AC7E0F5A2394EC1DD3C848196CD2
C0418A9694546976FD60435F0F4ADD0F
A0B101E326EC652C5C0570248DA24BB7
21234C1C0DA9C08A8A2367F0DEF74929
42E924545AAF782F713A1E02A897A16B
5
49CCB7077B4222949011A3CE551D15A6
A4359193EC24756C8115473C7C51A47B
B7A101E326F47E2B5805702A86A632C3
4557571975D9DC90F6566393ADEC305D
55F924545AB76328753A1E0CA393D81F
6
```

decrypt.py

```
huangjiongrui@huangjiongrui-virtual-machine:~/IOT security$ python decrypt.py
The possible plaintext:
(one character means this byte is definite
while more than one characters means these are possible characters of the byte.)

['P', 'O', 'HSTO', 'T', ',', ' ', 'G', 'E', 'T', ',', ' ', 'I', 'N', 'P', 'U', 'T']
['G', 'E', 'HSTO', ',', ' ', 'H', 'T', 'T', 'P', ',', ' ', 'I', 'N', 'P', 'U', 'T']
[',', ' ', 'HSTO', 'U', 'T', 'P', 'U', 'T', ',', ' ', 'P', 'O', 'S', 'T', ',', ' ']
['G', 'E', 'HSTO', ',', ' ', 'H', 'T', 'T', 'P', ',', ' ', 'I', 'N', 'P', 'U', 'T']
[',', ' ', 'HSTO', 'U', 'T', 'P', 'U', 'T', ',', ' ', 'P', 'O', 'S', 'T', ',', ' ']
['H', 'T', 'HSTO', 'P', ',', ' ', 'I', 'N', 'P', 'U', 'T', ',', ' ', 'P', 'O', 'S']
[',', ' ', 'HSTO', 'T', 'T', 'P', ',', ' ', 'I', 'N', 'P', 'U', 'T', ',', ' ', 'O']
[' ', 'P', 'HSTO', 'S', 'T', ',', ' ', 'G', 'E', 'T', ',', ' ', 'H', 'T', 'T', 'P']
The output may need manually check, Please run the GetKey.py now.
```

GetKey.py

```
huangjiongrui@huangjiongrui-virtual-machine:~/IOT security$ python GetKey.py
Please see the decrypt.py's output, and manually check the word.
Please input one of the plaintext and its No.
input the plaintext after check (eg. 'POST, GET, INPUT')
'POST, GET, INPUT'
the No. of the input plaintext.(begin from 1)
1
the key is :[101, 236, 248, 82, 47, 18, 119, 192, 188, 49, 243, 129, 6, 73, 57, 134]
```

可以在client端或者AP端看到Nonce=0时真实的key

两者确实是同一个key，破解正确。

效果分析

本次project成功模拟了WPA2协议以及复现了简单的重安装攻击。有些缺陷的部分在于当捕获的同样密钥加密的密文块数量较少时，对于明文的破译大量依赖人工。所以，在进行中间人攻击的时候，在进行转发ACK包之前，需要截获足够的ACK包，得到足够的用同样密钥加密的密文块。

当然在最后进行密钥的验证时，我因为在AP和client都打印了key值，所以直接进行了比对。在真正的攻击场景中，攻击者应该使用该密钥对于其他的密文进行破译，观察明文是否与预期一致。