Lab 9: Wireless Security 2

Introduction

In this lab students will explore ways to perform wireless attacks and understand potential defenses. The attacks that will be covered are inspecting & modifying wireless card parameters, changing the wireless transmission channel, flooding attacks, and cracking keys of WEP protected networks.

Software Requirements

All required files are packed and configured in the provided virtual machine image.

- The Virtual Machine running Ubuntu 16.04
- Wireshark: Network protocol analyzer
- Aircrack- ng: a suite of tools to assess WiFi network security http://aircrack-ng.en.softonic.com/

Task 1: Cracking the WEP Password

Aircrack-ng

Aircrack-ng is a network software suite consisting of a detector, packet sniffer, WEP and WPA/WPA2-PSK cracker and analysis tool for 802.11 wireless LANs.

Step 1: Install Aircrack-ng. You can install the tool in the SeedVM through

```
sudo apt-get update
sudo apt-get install -y aircrack-ng
```

Step 2: Cracking the WEP.cap (trace provided in eDimension). It has contained large enough number of captured WEP packets for the aircrack-ng to crack the WEP protocol. Run the following code to get the correct password.

```
aircrack-ng <cap file>
```

Task 2: Cracking the WEP Packet

With the WEP password, an attacker is able to join a cracked WiFi network as well as cracking the captured WEP packets. In this task, we will see how we can use the cracked password to further crack the captured packets.

Step 1: Recall the WEP encryption process

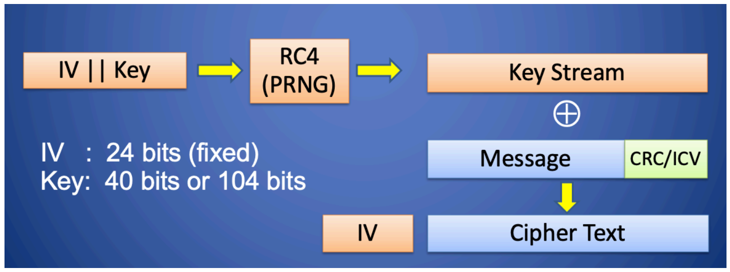

Fig. 1 WEP Encryption

In the WEP encryption process, the initialization vector (IV) concatenated with the key is fed into a RC4 pseudorandom number generator (PRNG). The PRNG generates one byte of pseudorandom number, i.e., the key, in each iteration. The PRNG can generate a stream of infinite keys as needed for the WEP encryption. The key stream encrypts the messages and the CRC through the XOR operation.

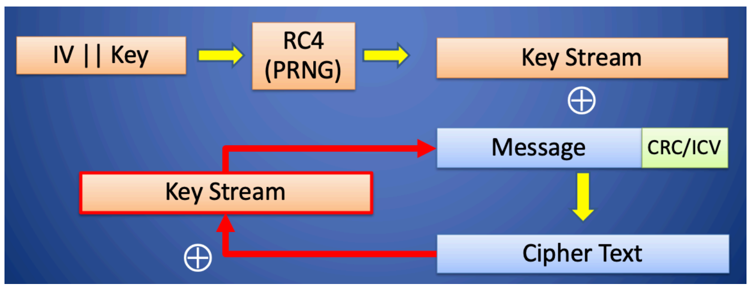

Fig. 1 WEP Decryption

The decryption process is straight forward by XORing the cipher text with the same key stream as in the encryption process. Note that the same key stream can be generated because we have cracked the keys of the WEP packets.

Step 2: Implement the RC4 Algorithm

The first part of RC4 is the key-scheduling algorithm (KSA). KSA initializes the permutation in the array S. The array S is mixed with the 'input key', which is *IV|| Key* in this case. IV is transmitted in plaintext while the Key is what we get in the Task 1. '||' means concatenation. For example, if IV is '*0x1A1B1C*' (24 bits) and Key is '*0x01234567*' (40 bits), then the concatenated input key is '*0x1A1B1C01234567*'.

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap(S[i],S[j])
endfor
```

Fig3. Key-scheduling algorithm (KSA):

The next part of RC4 is the pseudo-random generation algorithm (PRGA). PRGA modifies the state of "S" and outputs a byte of the keystream in each iteration. The key stream will do bitwise XOR with the cipher text (both message and CRC/ICV) to recover the original text messages.

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap(S[i],S[j])
    output S[(S[i] + S[j]) mod 256]
endwhile
```

Fig. 4 Pseudo-random generation algorithm (PRGA)

Note that PRGA outputs a random number at the end of **each iteration** instead of the end of the while loop. Actually, the while loop is an infinite loop to generate key streams. In Python, the keyword *yield* allows you to output in the middle of a loop. In the main function, you can use the *next(<PRNG>)* to visit the output random number. Please refer to the skeleton code for details.

Step 3: Verify Your Results
After you have gone through the decryption process, you will get a raw packet consists of hex numbers which is hardly be ready by human body. You must have a method to verify whether your crack is successful.

ASCII Code

A straightforward way is to translate the bytes into readable ASCII code. For example,

```
bytes.fromhex("7061756c").decode()
```

```
'paul'
```

However, if the WEP packet doesn't contain readable texts, e.g., only contains data, this method will fail.

Checksum

Each WEP packet has a checksum (also known as CRC or ICV in the context of WEP) attached after the message. ICV is calculated from the message through the CRC32 algorithm and encrypted together with the message. You can find the **encrypted** ICV from Wireshark.

```
▼ WEP parameters
    Initialization Vector: 0xcdd23a
    Key Index: 0
    WEP ICV: 0x5db2d69a (not verified)
```

ICV can be used to verify the integrity of the packet received from WiFi. In our case, the captured ICV will match CRC32(Message) only when you have correctly decrypted the packets. In Python, we can directly use the *crc32* method in *binascii* library to ease our life. For example,

```python
input = 'AAAA03000000080600010800060400010000EA66BFB69AC10000
crcle = binascii.crc32(bytes.fromhex(input)) & 0xffffffff
```

One special thing you need to take care of is that *binascii.crc32* gives the CRC in **little endian**, while the ICV in WEP protocol follows the **big endian**. You can convert the endian through the following code.

```python
# binascii.crc32 is in little endian, convert it to big endian
crc=struct.pack('<L', crcle)
```

Assignment:

1. Read the instructions above and use the skeleton python code to finish all the tasks. You may crack any broadcast WEP packet, e.g., SN=2000, for the lab.

2. Demonstrate the following in the report
   a. Justify the correctness of your implementation of the RC4 algorithm
   b. The cracked **payload** and **ICV** of one broadcast packet