Łamanie haseł WPA2-PSK

Rafał Kamiński

1. Cel projektu

Celem projektu jest stworzenie programu umożliwiającego wylistowanie pobliskich sieci WiFi WPA2-PSK, a następnie umożliwia złamanie hasła do jednego z nich.

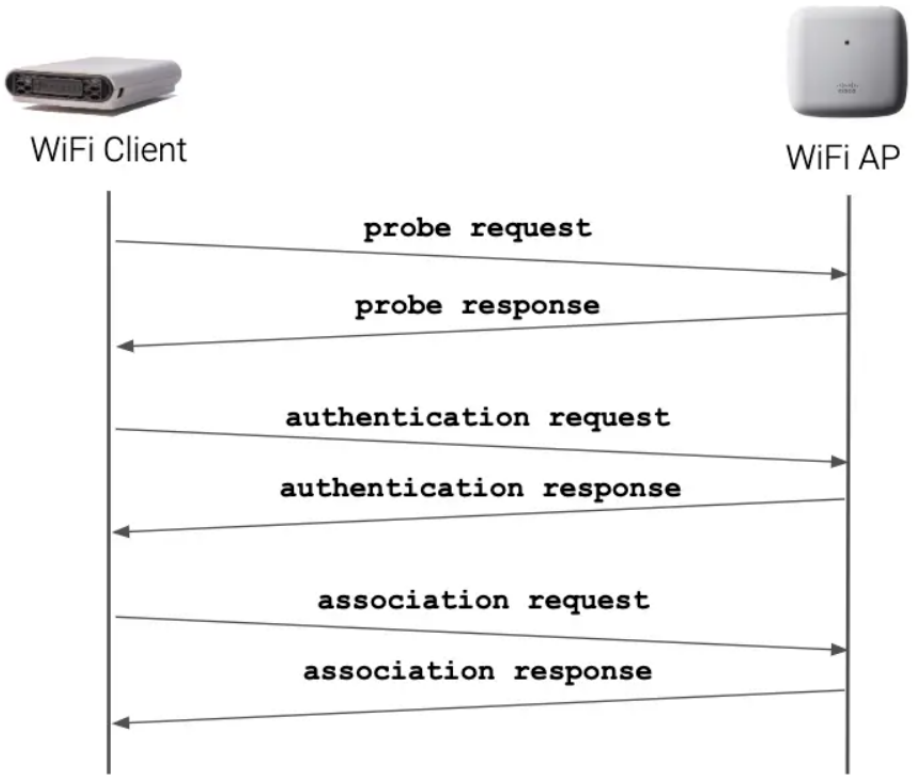
2. Wstęp teoretyczny

Podłączenie do sieci wifi przebiega w trzech etapach. Pierwszym z nich jest Authentication and Association, mający na celu skanowanie i znalezienie sieci WiFi do których chcemy dołączyć. Drugim jest 4-way handshake, mający zapewnić połączenie szyfrowane połączenie do sieci. Ostatnim etapem jest uzgodnienie adresu IP podłączonego urządzenia, co nie będzie ważne w tym projekcie.

a. Authentication and Association 802.11, 802.11n

Proces skanowania dzieli się na aktywne i pasywne. Aktywne skanowanie przeskakuję po wszystkich kanałach po kolei i wysyła pakiet Probe Request, nasłuchując na zwrotny Probe Response. Pasywne przeskakuje po kanałach i nasłuchuję na Beacon Probe rozgłaszane przez punkt dostępu ogłaszające sieć

Proces łączenia:

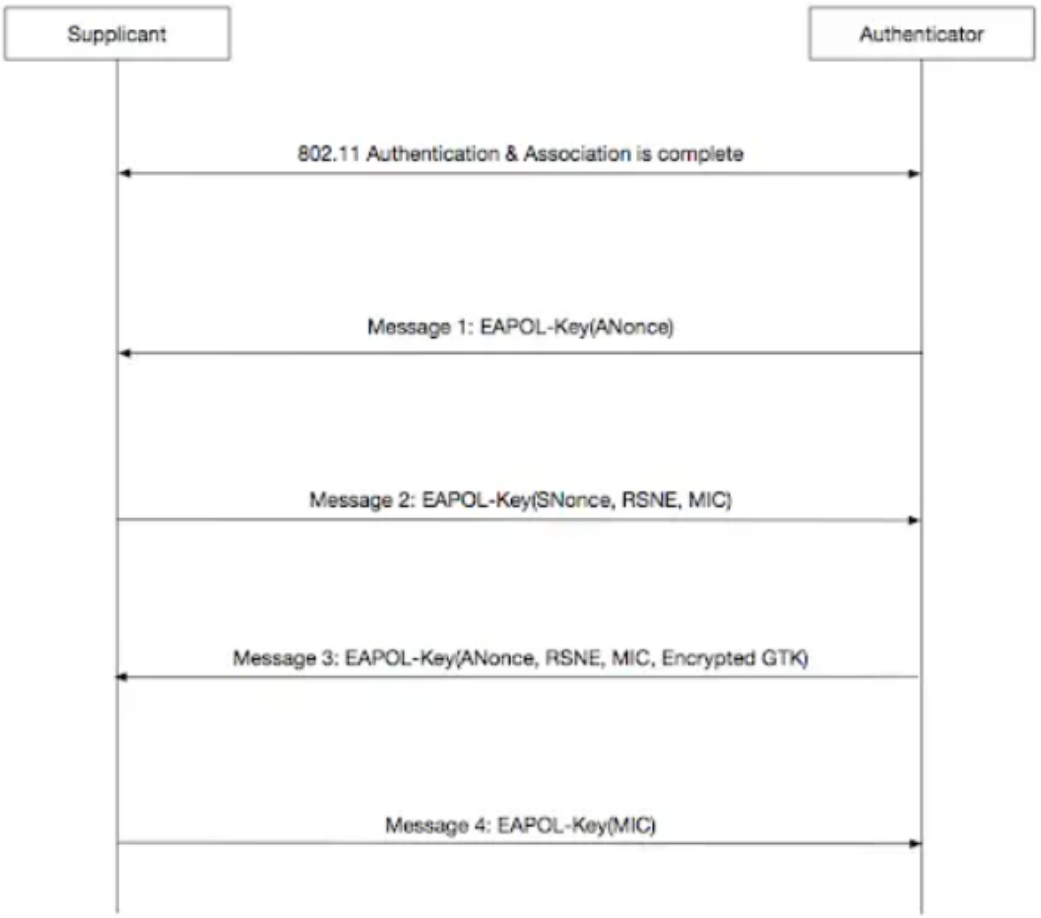


(możlwe dodatkowe pakiety ACK)

Autentykacja i asocjacja w tej części nie ma na celu zapewniania bezpiecznego łącza, a jedynie wykrycie sieci i uzgodnienie parametrów połączenia.

b. 4-way handshake 802.1x

Proces połączenia:



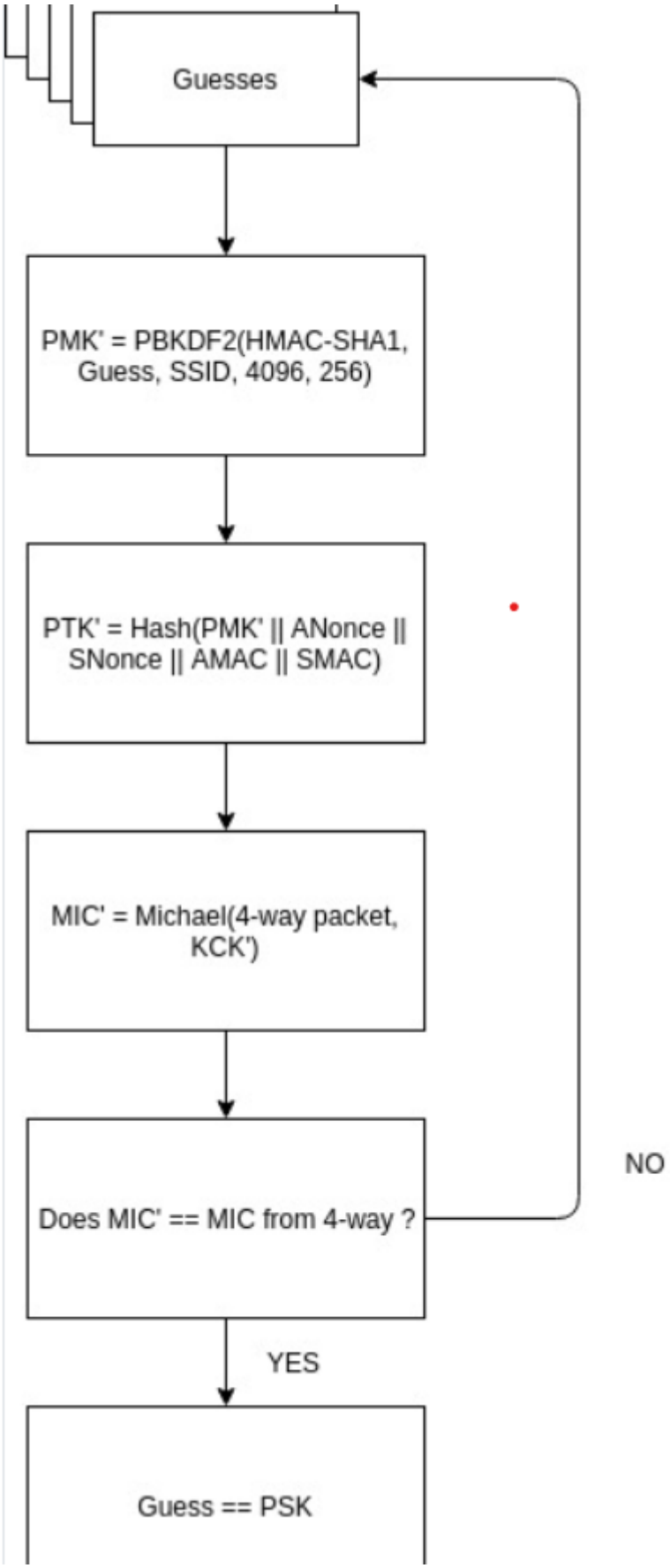
Po wykonaniu Autentykacji i Asocjacji, punkt dostępowy rozpoczyna 4-way handshake. Następuję wymiana wartości Nonce, adresów MAC i pozostałych potrzebnych parametrów do szyfrowania.

Klient i punkt dostępowy uzyskają własny klucz przejściowy par (PTK). Pakiet PTK jest uzyskiwany na podstawie par kluczy głównych (PMK) każdego urządzenia. Pakiet PTK składa się z ANonce punktu dostępu, SNonce klienta oraz adresów MAC punktu dostępu i klienta.

c. Łamanie hasła

Podczas nasłuchiwania ruchu sieci, możliwe będzie wyłapanie 4-way handshake'u. Po jego złapaniu, będziemy mieć możliwość przeprowadzenia lokalnego ataku próbującego odgadnąć hasło do sieci.

Proces łamania hasła:



Czekanie na połączenie się użytkownika może być zbyt czasochłonne. Można jednak wykorzystać w tym miejscu dodatkowy atak wylogowywujący użytkownika, aby ten połączył się ponownie moment później. Jest to możliwe, ponieważ pakiety kończące połączenie nie są szyfrowane.

3. Odbieranie i wysyłanie pakietów przy wykorzystaniu socketów w Pythonie

    a. Tryb monitors interfejsu sieciowego

Domyślnie interfejs sieciowy odbierał będzie jedynie ramki przeznaczone do nas. W celu odbierania wszystkich pakietów z poziomu standardu 802.11 w Pythonie musimy ustawić nasz interfejs sieciowy w tryb monitora. Wykorzystana do tego została komenda:

```
sudo airmon-ng check kill && sudo airmon-ng start wlp4s0
```

Aby wrócić nasz interfejs do oryginalnego stanu możemy wykorzystać komendę:

```
sudo airmon-ng stop wlp4s0mon && sudo ifconfig wlp4s0 down && sudo
iwconfig wlp4s0 mode managed && sudo ifconfig wlp4s0 up && sudo service
NetworkManager restart
```

    b. Odbieranie pakietów

Aby Pythonowy socket mógł odbierać pakiety jako czyste bajty, musimy zastosować następujące ustawienia:

```
ETH_P_ALL = 0x0003
sock = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
socket.ntohs(self.ETH_P_ALL))
sock.bind((“wlp4s0mon”, 0))
```

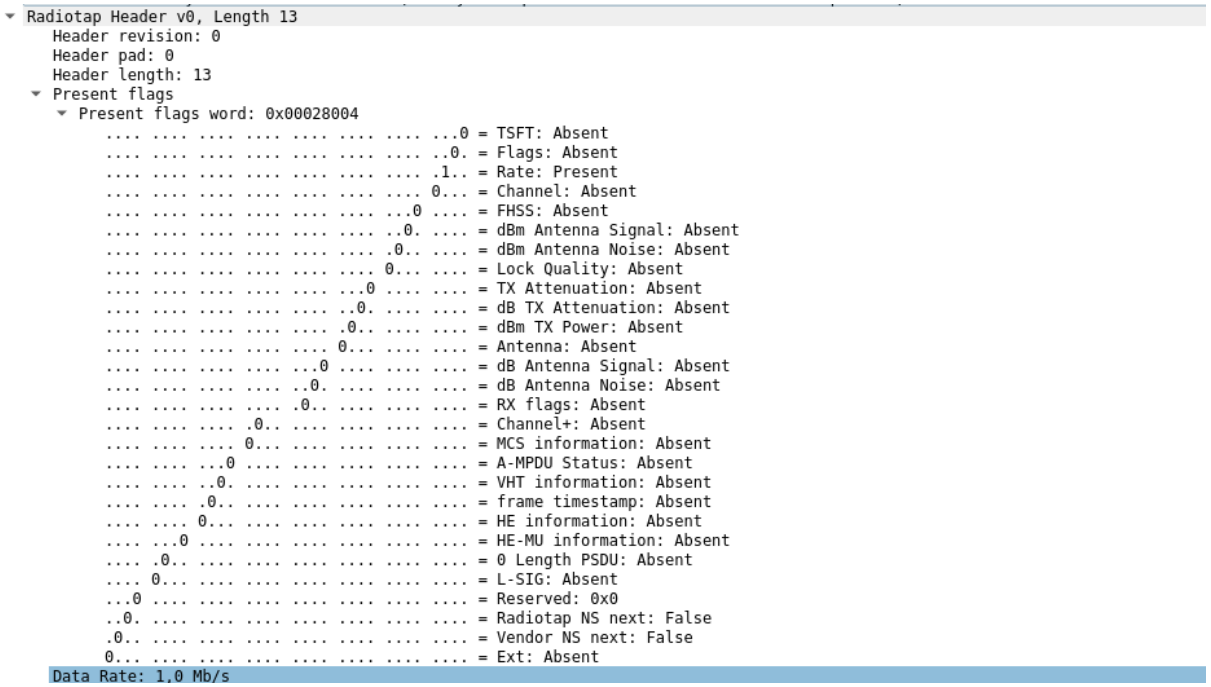
W tym trybie będą odbierane wszystkie pakiety 802.11 z interfejsu w trybie monitora.

    c. Wysyłanie pakietów

W celu wysyłania pakietów wymagane są delikatnie inne ustawienia:

```
sock = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
socket.PACKET_OTHERHOST)
sock.bind((“wlp4s0mon”, 0))
```

Jeżeli chcemy wysyłać pakiety przez interfejs w trybie monitora, przy wysyłaniu będziemy musieli dodatkowo doklejać na początku wiadomości tzw “RadioTap Header” wyjaśniony dokładniej w przyszłym rozdziale. W przypadku pominięcia go lub przesłania go z wartościami niezgodnymi ze stanem faktycznym interfejs odrzuci pakiet i nie prześle go dalej. Doświadczalnie zostało przeze mnie ustalone, że najłatwiej jest dokleić do wiadomości pusty header, nie niosący ze sobą żadnych dodatkowych informacji, wtedy mamy pewność, że interfejs prześle go dalej. Niniejszy header:



4. Framework projektu

    a. Interakcje z socketami

W celu ułatwienia interakcji z socketami, zaprojektowałem dwie klasy pośredniczące w tym procesie: PacketSniffer oraz PacketSender. Obie wymagają podania interfejsu na jakim mają pracować, ale dodatkowo klasa PacketSniffer wymaga podania obiektu filtrującego i parsującego ramki w celu lepszego ich zwracania.

```
class PacketSniffer:

    ETH_P_ALL = 0x0003

    def __init__(self, iface: str, parser: Parser, _filter: Filter =
AllMatchFilter()):
        self._iface = iface
        self._parser = parser
        self._filter = _filter

    def listen(self) -> Iterator:
        with socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
socket.ntohs(self.ETH_P_ALL)) as sock:
            sock.bind((self._iface, 0))
            while True:
                frame = sock.recv(2000)
                if self._filter.match(frame):
                    yield self._parser.parse(frame)
```

```
class PacketSender:

    def __init__(self, iface: str):
        self._iface = iface
        self._socket = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
socket.PACKET_OTHERHOST)
        self._socket.bind((self._iface, 0))

    def send(self, raw_data: bytes,
radiotap_header=b"\x00\x00\x08\x00\x00\x00\x00\x00"):
        self._socket.sendall(radiotap_header + raw_data)
```

b. Filtry

Filtry są klasami odpowiedzialnymi za odrzucanie błędnych oraz nie interesujących nas pakietów. Przyjmują wiadomość w postaci bajtów i odpowiednio sprawdzają czy konkretne pola ramki spełniają dane warunki. Przykładowy filtr:

```
class CRC32Filter(Filter):

    def match(self, frame: bytes) -> bool:
        _, frame = RadioTapHeaderParser.parse(frame)

        actual_crc = calculate_crc32(frame[:-4])
        received_crc = frame[-4:]

        return actual_crc == received_crc
```

Aby umożliwić logiczne łączenie warunków wprowadzone zostały klasy filtrów FilterAlternative oraz FilterAggregate, przyjmujące listę filtrów jako parametry i sprawdzają kolejno czy przynajmniej jeden filtr przeszedł test, lub czy wszystkie filtry przeszły test.

c. Parsery

Parsery mają za zadanie przefiltrowana już ramkę przekształcić w zadany obiekt, w celu łatwiejszej interakcji z nim w przyszłości. Przykładowa implementacja:

```
class ManagementFrameParser(Parser):

   def parse(self, frame: bytes) -> ManagementFrame:
       radio_tap_header, frame = RadioTapHeaderParser.parse(frame)
       frame_control = FrameControlParser.parse(frame[0:2])
       duration: str = frame[2:4].hex()
       dest_addr: str = ":".join(map(lambda byte:
hex(byte).lstrip("0x").zfill(2), frame[4:10]))
       src_addr: str = ":".join(map(lambda byte:
hex(byte).lstrip("0x").zfill(2), frame[10:16]))
       bssid: str = ":".join(map(lambda byte:
hex(byte).lstrip("0x").zfill(2), frame[16:22]))
       sequence_control: str = frame[22:24].hex()
       body: ManagementFrameBody = ManagementFrameBodyParser.parse(frame[24:
-4])
       fcs: str = frame[-4:].hex()

       return ManagementFrame(
           radio_tap_header,
           frame_control,
           duration,
           dest_addr,
           src_addr,
           bssid,
           sequence_control,
           body,
           fcs
       )
```

d. Przykład użycia

Przykładowa implementacja odbierania ramek Beacon Frame lub Probe Response Frame i wyświetlanie ich jako ramkę Management Frame (znaczenia ramek omówione w późniejszych rozdziałach):

```
iface = "wlp4s0mon"

packet_sniffer = PacketSniffer(
    iface,
    ManagementFrameParser(),
    FilterAggregate(
        RadioTapHeaderFilter(),
        CRC32Filter(),
        ManagementFrameFilter(),
        FilterAlternative(
            BeaconFrameFilter(),
            ProbeResponseFrameFilter()
        )
    )
)

for packet in packet_sniffer.listen():
    packet: ManagementFrame
    print(packet)
```

Przykładowa implementacja wysyłania pakietów:

```
iface = "wlp4s0mon"

# random mac from android hot spot
frame = DeauthenticationFrame("a2:cf:2d:38:59:0c")

sender = PacketSender(iface)

while True:
    for _ in range(64):
        sender.send(frame.to_bytes())
    sleep(1)
```

5. Listowanie pobliskich sieci WiFi

Ramki 802.11 posiadają dzielą się na kilka podtypów, m.in. Control i Management. Informacje rozgłaszające sieć i zarządzające procesem łączenia są w typem Management i na nim się na razie skupimy.

a. Budowa Management Frames 802.11

i. RadioTap Header

Ramki odbierane przez interfejs w trybie monitora mają dołączony tzw RadioTap Header, który jest dodawany przez kartę sieciową w momencie odbioru ramki. Zawierą on dodatkowe informacje o przychodzącym pakiecie m.in. siłę sygnału.

```
⊞ Frame 308: 94 bytes on wire (752 bits), 94 bytes captured (752 bits)
⊟ Radiotap Header v0, Length 26
    Header revision: 0
    Header pad: 0
    Header length: 26
  ⊞ Present flags
    MAC timestamp: 110317703
  ⊞ Flags: 0x10
    Data Rate: 2.0 Mb/s
    Channel frequency: 2437 [BG 6]
  ⊞ Channel type: 802.11b (0x00a0)
    SSI Signal: -44 dBm
    SSI Noise: -84 dBm
    Antenna: 0
    SSI Signal: 40 dB
⊞ IEEE 802.11 Beacon frame, Flags: ........C
⊞ IEEE 802.11 wireless LAN management frame
```

ii. Management Frame

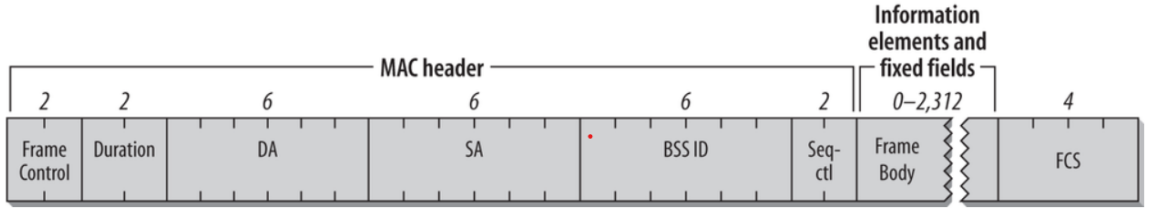
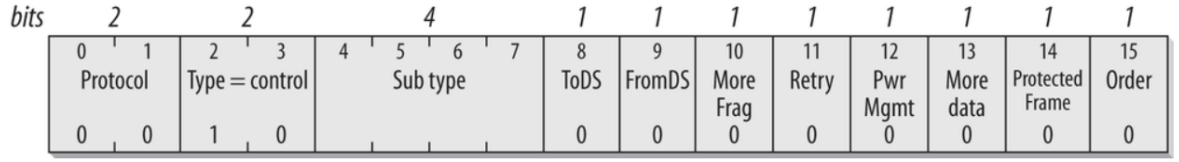


Figure 4-20. Generic management frame

Ważnymi dla nas polami są: Frame Control, z którego możemy wyciągnąć informację o rodzaju ramki; BSSID, będącym adresem MAC punktu dostępowego oraz Frame Body, z którego wyciągnąć można wiele informacji, jak na przykład SSID, czyli nazwę sieci.

Frame Control zawiera dane o wersji protokołu oraz rodzaju ramki. Stąd jesteśmy w stanie określić że ramka 802.11 jest faktycznie typu Management i jakich danych możemy się w niej spodziewać:



Protokół na ten moment zawsze ma wartość 0. Dla pakietów Management typ będzie wynosić również 0. Podtyp pomoże nam określić, czy mamy do czynienia z Beacon Probe, Probe Response czy jeszcze czymś innym.

Various 802.11 Frame Types and Subtypes

Type Value B3..B2	Type Description	Subtype Value B7 .. B4	Subtype Description
00	Management	0000	Association Request
00	Management	0001	Association Response
00	Management	0010	Reassociation Request
00	Management	0011	Reassociation Response
00	Management	0100	Probe Request
00	Management	0101	Probe Response
00	Management	0110	Timing Advertisement
00	Management	0111	Reserved
00	Management	1000	Beacon
00	Management	1001	ATIM
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1101	Action
00	Management	1110	Action No Ack (NACK)
00	Management	1111	Reserved

Frame Body dzieli się na dwie sekcje. Pierwsza część jest stałej długości i niesie konkretne informację. Druga część posiada zmienną długość i jest listą parametrów nadawanej w postaci: ID Elementu (1B), Długość w bajtach (1B), Wartość (zmienna długość). SSID sieci jeśli zostało nadane, znajdzie się pod ID Elementu równym 0, a kanał na jakim nasłuchuje punkt dostępu pod ID równym 3.

b. Filtrowanie ramek

W celu wylistowania pobliskich sieci będzie zależało nam na ramkach 802.11 będących w typie Management z podtypem Probe Response lub Beacon Probe. Będziemy więc przyjmować jedynie ramki z wersją protokołu i typem równymi 0, oraz podtypem równym 0101 lub 1000. Z nich wyciągnąć można SSID i zapisać je w celu dalszego działania programu.

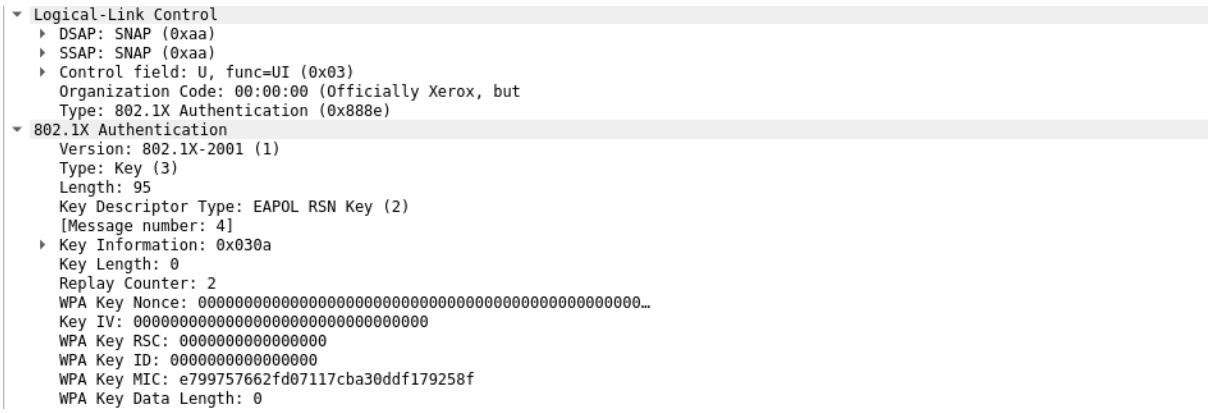
W trakcie odbierania ramek zmieniany będzie kanał karty sieciowej, aby wyłapać sieci nadające na innych częstotliwościach.

W celu sprawdzania poprawności ramek liczone jest crc32 i porównywane z ramka.

6. Przechwytywanie 4-way handshake

a. Budowa ramek

Ramki EAPOL 802.1x przesyłane są poprzez ramki 802.11 QoS Data, których budowa poza innym Frame Control (typ=x, podtyp=y), nie różni się od opisanych wcześniej Management Framów. Zawiera ona jednak dodatkowo w ciele wartości Logical-Link-Control oraz interesujące nas wartości z ramek 802.1x:



Kolejne ramki zawierają inne potrzebne na danym etapie wartości przesyłane w polach kluczy:

1) ANonce
2) SNonce, RSNE, MIC
3) ANonce, RSNE, MIC, encrypted GTK
4) MIC

W procesie łamania haseł nie potrzebna jest nam część wartości, więc nie wgłębiam się w ich znaczenie.

b. Filtrowanie ramek

W celu sprawdzania poprawności ramek liczone jest crc32 i porównywane z ramka.

Zależeć nam będzie na ramkach 802.11 QoS Data, które są typu 10 oraz podtypu 1000 zawierającym w ciele wartości Logical-Link-Control z typem podanym jako 802.1x Authentication o wartości 888e.

7. De-autentykacja użytkownika

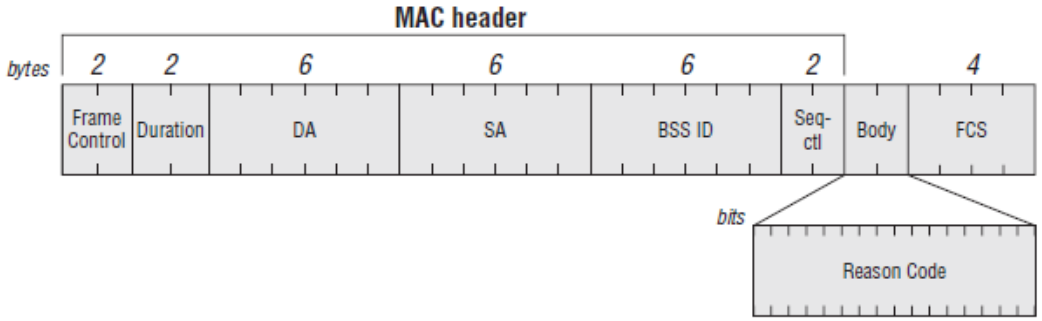
Problemem w powyższym procesie jest oczekiwanie na połączenie się użytkownika do sieci. Można jednak zaradzić temu, poprzez wyrzucenie użytkownika z sieci samodzielnie i poczekanie na jego (często automatyczne) ponowne połączenie.

Ramki Deauth są częścią standardu 802.11, więc nie są szyfrowane i powiązane w jakikolwiek sposób z punktem dostępowym. Może więc je nadać dowolna osoba w pobliżu w celu wyrzucenia użytkowników z sieci.

a. Budowa ramki Deauth

Ramki Deauth są typu Management z podtypem 1100. Nie są zabezpieczane przez 802.1x, co pozwala każdemu wysłać taką ramkę podszywając się pod punkt dostępu.

Docelowym adresem może być konkretna maszyna lub adres broadcast, co umożliwia wyrzucenie z sieci każdego podłączonego użytkownika. Ramka:



W ciele zawarty jest powód rozłączenia. W projekcie zastosowany został status 7:

7	Class 3 frame received from nonassociated station	Client attempted to transfer data before it was associated.

Po otrzymaniu go urządzenie rozłączy się z sieci i będzie próbowało połączyć się ponownie, co umożliwia przechwycenie jego handshake.

8. Łamanie hasła do sieci WiFi WPA2-PSK

Ten rozdział będzie bardziej ogólny, niekompletny i potencjalnie błędny ponieważ nie udało mi się zaimplementować poniższej części w celu sprawdzenia jej poprawności.

a. Potrzebne informacje z 4-way handshake

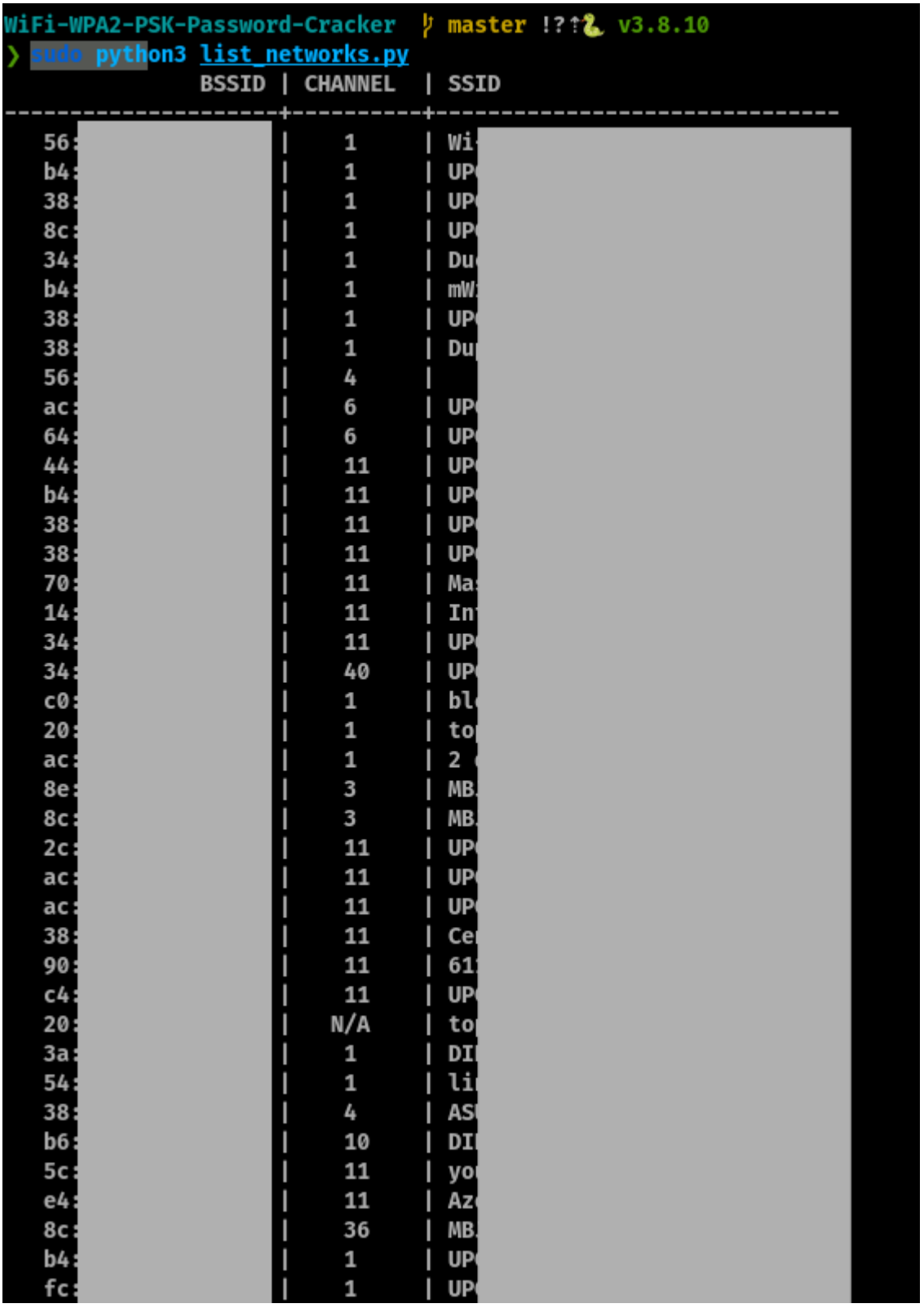
- ANonce - z ramki 1
- SNonce - z ramki 2
- MAC punktu dostępowego
- MAC logującego się użytkownika
- Payload i MIC ramki 2 lub 4

b. Proces zgadywania haseł

- Ustal testowane hasło
- Oblick PMK na podstawie hasła i SSID sieci
- Oblicz PTK na podstawie PMK, ANonce, SNonce i adresów MAC
- Oblicz MIC jako keyed-hash drugiej lub czwartek ramki handshaku, przy użyciu PTK jako klucza i porównaj z MIC ramki, jeśli są identyczne, to hasło zostało znalezione

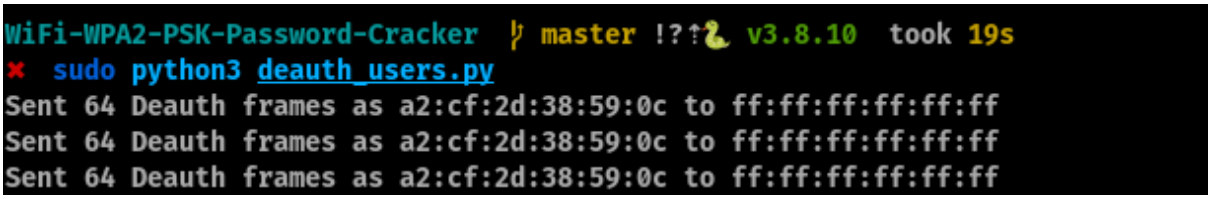
9. Działanie programu

    a. Znajdowanie pobliskich sieci



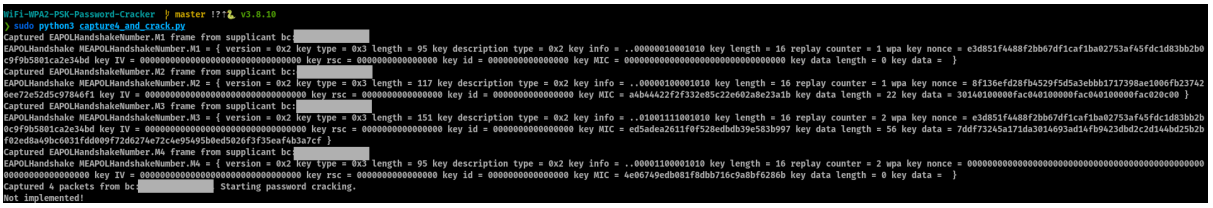
b. De-Autentykacja użytkownika

Cześć wartości zhardcodowanych w kod zamiast podawanych jako parametry programu.



c. Nasłuchiwanie 4-way handshake i łamanie hasła

Cześć wartości zhardcodowanych w kod zamiast podawanych jako parametry programu.



10. Podsumowanie

Bezpieczeństwo sieci WiFi WPA2-PSK zależne głównie jest od siły ustalonego hasła, ponieważ schemat logowania pozwala na lokalne łamanie haseł bez wykrycia o ile będzie możliwe podsłuchanie 4-way handshake. Warto pomyśleć nad dodatkowymi mechanizmami zabezpieczeń takimi jak na przykład certyfikaty jeżeli zależy nam na zwiększeniu bezpieczeństwa.

Dzięki temu projektowi nauczyłem się wiele na temat działania sieci WiFi i jej skomplikowaniu oraz jak testować ich bezpieczeństwo przy użyciu ogólnodostępnych narzędzi i języków programowania.

Bibliografia:

1. https://netbeez.net/blog/how-wifi-connection-works/
2. https://netbeez.net/blog/station-authentication-association/
3. https://netbeez.net/blog/secure-network-4-way-handshake/
4. https://en.wikipedia.org/wiki/IEEE_802.11
5. https://en.wikipedia.org/wiki/IEEE_802.1X
6. https://security.stackexchange.com/questions/66008/how-exactly-does-4-way-handshake-cracking-work
7. https://cylab.be/blog/32/how-does-wpawpa2-wifi-security-work-and-how-to-crack-it
8. https://community.nxp.com/t5/Wireless-Connectivity-Knowledge/802-11-Wi-Fi-Connection-Disconnection-process/ta-p/1121148
9. https://www.ii.pwr.edu.pl/~kano/course/module8/8.2.1.4/8.2.1.4.html
10. https://howiwifi.com/2020/07/13/802-11-frame-types-and-formats/
11. https://en.wikipedia.org/wiki/802.11_Frame_Types
12. https://www.oreilly.com/library/view/80211-wireless-networks/0596100523/ch04.html
13. https://www.radiotap.org
14. https://wifinigel.blogspot.com/2013/11/what-are-radiotap-headers.html
15. https://erg.abdn.ac.uk/users/gorry/eg3576/crc.html
16. http://zlib.net/crc_v3.txt
17. https://mrncciew.com/2014/10/11/802-11-mgmt-deauth-disassociation-frames/
18. https://support.zyxel.eu/hc/en-us/articles/360009469759-What-is-the-meaning-of-802-11-Deauthentication-Reason-Codes-
19.