Cyber Security Coursework - Protocol Analysis

James Donohue - james.donohue@bbc.co.uk

Introduction

This is my report for the 'protocol analysis' part of the Cyber Security coursework assignment.

The third-party tools used in this report were:

- `file(1)` - command-line tool common on UNIX-based systems for identifying file types
- Wireshark v2.2.4 - graphical network protocol analyser

`capture1.pcap`

The `file` tool identifies this capture as follows:

```
tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length 65535)
```

From this we can learn that the capture was created on a little-endian processor (such as the Intel x86 family). The 'capture length' indicates that captured data was limited to 65535 bytes per packet (Lamping, Sharpe, and Warnicke 2014).

Opening the file in Wireshark shows it contains 44 packets. Using the *Statistics* > *Endpoints* report we can see the packets relate to 6 different Ethernet addresses and 5 IPv4 addresses.

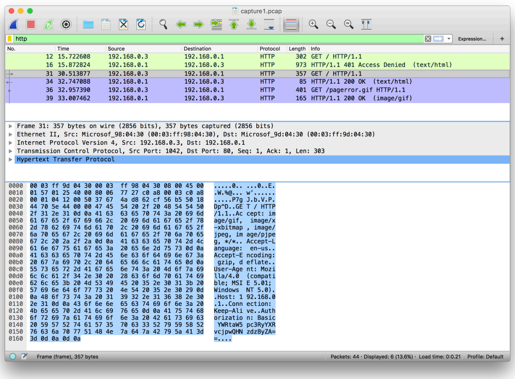
Find the username & password from the HTTP Basic type of authentication

By using the `http` view filter in Wireshark to show only HTTP packets (see Figure 1) we can see two separate HTTP connections are made from host `192.168.0.3` to port 80 on host `192.168.0.1`. The first of these connections contains a single request (`GET /`) which results in a `401 Access Denied` response.

The second connection (coloured in purple on Figure 1) contains two requests made using HTTP Basic authentication. The content of these requests can be seen most easily using the *Follow > TCP Stream' feature (see Figure 2).

Basic HTTP authentication, originally specified in HTTP/1.0, allows for simple challenge-response authentication of web clients using the `Authorization` header. The value after the scheme name `Basic` is a Base64 encoding of the username and password, separated by a colon. As no encryption is performed on the password, the Basic scheme is explicitly non-secure (Berners-Lee, Fielding, and Frystyk 1996). In this case the client sends the following header:

```
Authorization: Basic YWRtaW5pc3RyYXRvcjpwQHNzdzByZA==
```



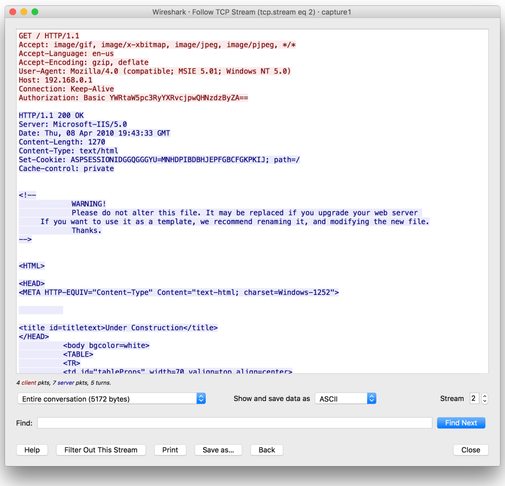
```
http
```

```
No.  Time       Source       Destination  Protocol Length Info
  12 15.722608  192.168.0.3  192.168.0.1  HTTP     302 GET / HTTP/1.1
  16 15.872824  192.168.0.1  192.168.0.3  HTTP     973 HTTP/1.1 401 Access Denied  (text/html)
  31 30.513877  192.168.0.3  192.168.0.1  HTTP     357 GET / HTTP/1.1
  34 32.747088  192.168.0.1  192.168.0.3  HTTP      85 HTTP/1.1 200 OK  (text/html)
  36 32.957390  192.168.0.3  192.168.0.1  HTTP     401 GET /pagerror.gif HTTP/1.1
  39 33.007462  192.168.0.1  192.168.0.3  HTTP     165 HTTP/1.1 200 OK  (image/gif)
```

```
▶ Frame 31: 357 bytes on wire (2856 bits), 357 bytes captured (2856 bits)
▶ Ethernet II, Src: Microsof_98:04:30 (00:03:ff:98:04:30), Dst: Microsof_9d:04:30 (00:03:ff:9d:04:30)
▶ Internet Protocol Version 4, Src: 192.168.0.3, Dst: 192.168.0.1
▶ Transmission Control Protocol, Src Port: 1042, Dst Port: 80, Seq: 1, Ack: 1, Len: 303
▶ Hypertext Transfer Protocol
```

```
0000  00 03 ff 9d 04 30 00 03  ff 98 04 30 08 00 45 00   .....0.. ...0..E.
0010  01 57 01 25 40 00 80 06  77 27 c0 a8 00 03 c0 a8   .W.%@... w'......
0020  00 01 04 12 00 50 37 67  4a d8 62 cf 56 b5 50 18   .....P7g J.b.V.P.
0030  44 70 5e 44 00 00 47 45  54 20 2f 20 48 54 54 50   Dp^D..GE T / HTTP
0040  2f 31 2e 31 0d 0a 41 63  63 65 70 74 3a 20 69 6d   /1.1..Ac cept: im
0050  61 67 65 2f 67 69 66 2c  20 69 6d 61 67 65 2f 78   age/gif,  image/x
0060  2d 78 62 69 74 6d 61 70  2c 20 69 6d 61 67 65 2f   -xbitmap , image/
0070  6a 70 65 67 2c 20 69 6d  61 67 65 2f 70 6a 70 65   jpeg, im age/pjpe
0080  67 2c 20 2a 2f 2a 0d 0a  41 63 63 65 70 74 2d 4c   g, */*.. Accept-L
0090  61 6e 67 75 61 67 65 3a  20 65 6e 2d 75 73 0d 0a   anguage:  en-us..
00a0  41 63 63 65 70 74 2d 45  6e 63 6f 64 69 6e 67 3a   Accept-E ncoding:
00b0  20 67 7a 69 70 2c 20 64  65 66 6c 61 74 65 0d 0a    gzip, d eflate..
00c0  55 73 65 72 2d 41 67 65  6e 74 3a 20 4d 6f 7a 69   User-Age nt: Mozi
00d0  6c 6c 61 2f 34 2e 30 20  28 63 6f 6d 70 61 74 69   lla/4.0  (compati
00e0  62 6c 65 3b 20 4d 53 49  45 20 35 2e 30 31 3b 20   ble; MSI E 5.01; 
00f0  57 69 6e 64 6f 77 73 20  4e 54 20 35 2e 30 29 0d   Windows  NT 5.0).
0100  0a 48 6f 73 74 3a 20 31  39 32 2e 31 36 38 2e 30   .Host: 1 92.168.0
0110  2e 31 0d 0a 43 6f 6e 6e  65 63 74 69 6f 6e 3a 20   .1..Conn ection: 
0120  4b 65 65 70 2d 41 6c 69  76 65 0d 0a 41 75 74 68   Keep-Ali ve..Auth
0130  6f 72 69 7a 61 74 69 6f  6e 3a 20 42 61 73 69 63   orizatio n: Basic
0140  20 59 57 52 74 61 57 35  70 63 33 52 79 59 58 52    YWRtaW5 pc3RyYXR
0150  76 63 6a 70 77 51 48 4e  7a 64 7a 42 79 5a 41 3d   vcjpwQHN zdzByZA=
0160  3d 0d 0a 0d 0a                                     =....
```

Figure 1: Wireshark filtering for only HTTP packets in capture1.pcap



```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 192.168.0.1
Connection: Keep-Alive
Authorization: Basic YWRtaW5pc3RyYXRvcjpwQHNzdzByZA==

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 08 Apr 2010 19:43:33 GMT
Content-Length: 1270
Content-Type: text/html
Set-Cookie: ASPSESSIONIDGGQGGGYU=MNHDPIBDBHJEPFGBCFGKPKIJ; path=/
Cache-control: private


<!--
            WARNING!
            Please do not alter this file. It may be replaced if you upgrade your web server
     If you want to use it as a template, we recommend renaming it, and modifying the new file.
            Thanks.
-->


<HTML>

<HEAD>
<META HTTP-EQUIV="Content-Type" Content="text-html; charset=Windows-1252">



<title id=titletext>Under Construction</title>
</HEAD>
          <body bgcolor=white>
          <TABLE>
          <TR>
          <td id="tableProps" width=70 valign=top align=center>
```

```
4 client pkts, 7 server pkts, 5 turns.
Entire conversation (5172 bytes)    Show and save data as ASCII    Stream 2
```

Figure 2: Wireshark Follow TCP Stream showing HTTP Basic authentication in capture1.pcap

The server responds with `HTTP/1.1 200 OK`, indicating that the authentication attempt was successful. The plaintext of the Base64 value can easily be determined using a trivial fragment of Python:

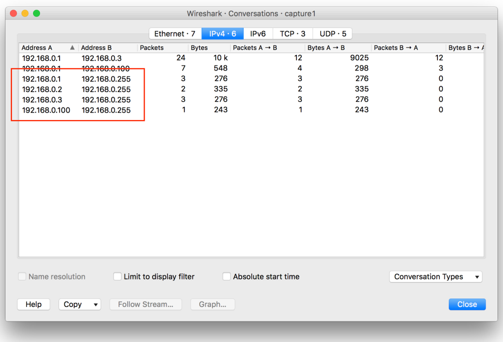
```
>>> import base64
>>> encoded = 'YWRtaW5pc3RyYXRvcjpwQHNzdzByZA=='
>>> base64.b64decode(encoded)
'administrator:p@ssw0rd'
```

Therefore the username sent was `administrator` and the password was `p@ssw0rd`.

Which hosts appear to be sending broadcast IP packets?

In IPv4, broadcast packets are those sent to a special address which causes them to be received by all hosts on a given subnet. The special address has all its unmasked bits set to 1 (Mogul 1984), which means ending in a sequence of one or more 255s when printed in typical 'dotted quad' form.

The Wireshark *Statistics* > *Conversations* view (shown in Figure 4) shows that four hosts are sending packets to `192.168.0.255`, which is the broadcast address for the `192.168.0.0/24` network.



Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B →
192.168.0.1	192.168.0.3	24	10 k	12	9025	12	
192.168.0.1	192.168.0.100	7	548	4	298	3	
192.168.0.1	192.168.0.255	3	276	3	276	0	
192.168.0.2	192.168.0.255	2	335	2	335	0	
192.168.0.3	192.168.0.255	3	276	3	276	0	
192.168.0.100	192.168.0.255	1	243	1	243	0	

Figure 3: Wireshark 'Conversations' view for capture1.pcap with broadcast traffic highlighted

The hosts that appear to be sending broadcast IP packets (via UDP) are therefore:

```
192.168.0.1
192.168.0.2
192.168.0.3
192.168.0.100
```

The packets are UDP datagrams on ports 137 and 138 (NetBIOS), meaning we can infer that these hosts are likely to be running Windows.

`capture2.pcap`

The Address Resolution Protocol (ARP) (Plummer 1982) is used to determine the hardware (MAC) address of a target host based based on its IP address. In normal operation, the host attempting to resolve a MAC address sends a broadcast 'request' message containing the IP address that is being queried. This will be received by all network interface cards (NICs) on the same subnet. The target host then sends a ‘reply’ message containing its own MAC address to the host that made the request, which caches it for later reuse.

Because ARP has no built-in support for authenticating such messages, it is possible for a malicious user to use it to disrupt network activity. In ‘ARP poisoning’, an attacker sends fake ARP requests or replies with the intention of either intercepting network traffic between two hosts (man-in-the-middle attack) or producing a denial-of-service (DoS) (Mishra 2016).

Method of detection

ARP poisoning can be suspected when a host begins sending ARP packets containing a ‘sender’ hardware (MAC) address that is different to a MAC address already associated with the given protocol address (IP) in earlier traffic.

Therefore, a simple implementation of an ARP poisoning sensor is to record the sender MAC address associated with the IP address in each ARP packet. When a new ARP packet is received, if a different sender MAC is given for the IP in address in question than one previously recorded, an alert can be issued and any other required action taken.

In order to demonstrate this approach using the supplied packet capture file, a Scapy script (`poisonarp.py`) was written. This script implements the above approach by iterating over each packet in the capture and storing a mapping of IP address to MAC address in a dictionary object.

The script can be executed as follows:

```
$ python poisonarp.py cwk_pcaps/capture2.pcap
```

Example output

```
Read 328 packets from cwk_pcaps/capture2.pcap
WARNING: suspected ARP poisoning attack at packet 298
  Attacker: 00:03:ff:98:98:02 (last seen using 192.168.0.2)
  Victim: 00:00:00:00:00:00 (192.168.0.1)
  Hijacked resource: 00:03:ff:98:98:03 (192.168.0.3)
```

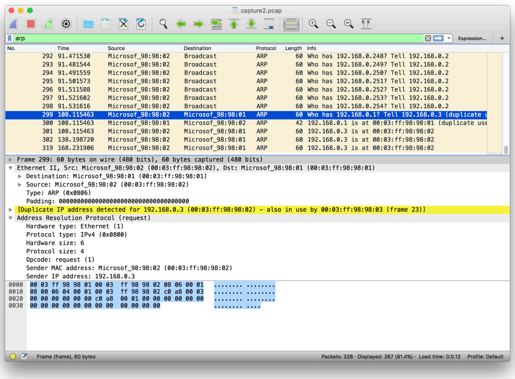
Limitations

To identify the victim, this script uses the ‘target’ hardware and protocol addresses given in the ARP packet, which is sufficient for the packet capture given. However, ARP poisioning can also be performed by broadcasting malicious ARP replies (so-called Gratuitous ARP or GARP). In this scenario the intended victim may not be immediately apparent.

Also, there are some scenarios in which the MAC address for an IP address may change legitimately (for example due to replacement of a part). These events will produced ‘false positives’ in a sensor that uses this approach.

Alternatives: Wireshark

Wireshark also identifies this type of attack using the message 'Duplicate IP address detected' (see yellow text in Figure **??**).



```
No.   Time        Source             Destination        Protocol  Length  Info
  292 91.471530   Microsof_98:98:02  Broadcast          ARP         60 Who has 192.168.0.248? Tell 192.168.0.2
  293 91.481544   Microsof_98:98:02  Broadcast          ARP         60 Who has 192.168.0.249? Tell 192.168.0.2
  294 91.491559   Microsof_98:98:02  Broadcast          ARP         60 Who has 192.168.0.250? Tell 192.168.0.2
  295 91.501573   Microsof_98:98:02  Broadcast          ARP         60 Who has 192.168.0.251? Tell 192.168.0.2
  296 91.511588   Microsof_98:98:02  Broadcast          ARP         60 Who has 192.168.0.252? Tell 192.168.0.2
  297 91.521602   Microsof_98:98:02  Broadcast          ARP         60 Who has 192.168.0.253? Tell 192.168.0.2
  298 91.531616   Microsof_98:98:02  Broadcast          ARP         60 Who has 192.168.0.254? Tell 192.168.0.2
  299 108.115463  Microsof_98:98:02  Microsof_98:98:01  ARP         60 Who has 192.168.0.1? Tell 192.168.0.3 (duplicate u
  300 108.115463  Microsof_98:98:01  Microsof_98:98:02  ARP         42 192.168.0.1 is at 00:03:ff:98:98:01 (duplicate use
  301 108.115463  Microsof_98:98:02  Microsof_98:98:01  ARP         60 192.168.0.3 is at 00:03:ff:98:98:02
  302 138.198720  Microsof_98:98:02  Microsof_98:98:01  ARP         60 192.168.0.3 is at 00:03:ff:98:98:02
  319 168.231906  Microsof_98:98:02  Microsof_98:98:01  ARP         60 192.168.0.3 is at 00:03:ff:98:98:02

▶ Frame 299: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
▼ Ethernet II, Src: Microsof_98:98:02 (00:03:ff:98:98:02), Dst: Microsof_98:98:01 (00:03:ff:98:98:01)
  ▶ Destination: Microsof_98:98:01 (00:03:ff:98:98:01)
  ▶ Source: Microsof_98:98:02 (00:03:ff:98:98:02)
    Type: ARP (0x0806)
    Padding: 000000000000000000000000000000000000
▶ [Duplicate IP address detected for 192.168.0.3 (00:03:ff:98:98:02) - also in use by 00:03:ff:98:98:03 (frame 23)]
▼ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Microsof_98:98:02 (00:03:ff:98:98:02)
    Sender IP address: 192.168.0.3

0000  00 03 ff 98 98 01 00 03  ff 98 98 02 08 06 00 01   ........ ........
0010  08 00 06 04 00 01 00 03  ff 98 98 02 c0 a8 00 03   ........ ........
0020  00 00 00 00 00 00 c0 a8  00 01 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....

Frame (frame), 60 bytes          Packets: 328 · Displayed: 267 (81.4%) · Load time: 0:0.12   Profile: Default
```

Figure 4: Wireshark showing duplicate IP address usage in capture2.pcap

`capture3.pcap`

The `file` tool identifies this capture as follows:

```
capture3.pcap: tcpdump capture file (little-endian) - version 2.4 (802.11, capture length 65535)
```

This reveals that unlike the other capture files (which were captured on an Ethernet network) this capture was created using a wireless (802.11) network.

Wireshark reports that the capture file contains 25895 packets. Using the display filter `wlan.fc.protected == 1` reveals that 59.5% of the packets (100% of data frames) are flagged as `protected` (see Figure 5). This bit indicates that the packet data is encrypted using an 802.11 encryption mechanism such as WEP (Wired Equivalent Privacy) or TKIP (Orebaugh, Ramirez, and Beale 2006).

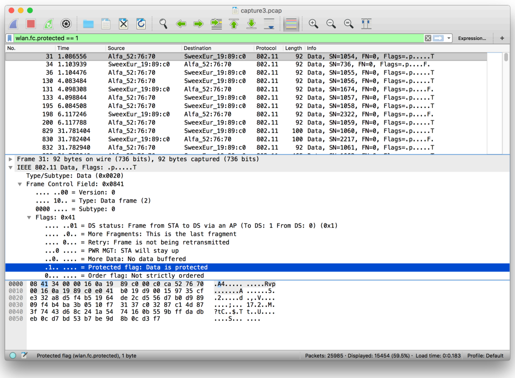
This means that it will not be possible to simply use Wireshark's protocol dissection functionality to extract passwords from the capture, as the relevant data is encrypted. However, the display filter `wlan.wep.iv`, which identifies frames that include the WEP Initialization Vector (IV), shows the same list of packets, meaning that all of these packets are encrypted using WEP.

The stream cipher used by WEP has been shown to be insecure and vulnerable to passive attack by an eavesdropper (Fluhrer, Mantin, and Shamir 2001), given access to sufficient IVs. The tool aircrack-ng can be used to crack WEP encryption using a packet capture file as follows:

```
$ aircrack-ng cwk_pcaps/capture3.pcap
```

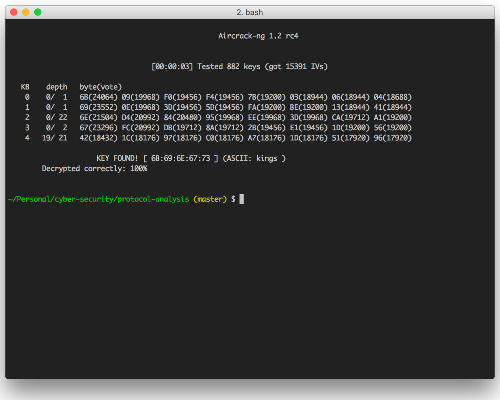
This successfully extracts the WEP key `6B:69:6E:67:73` ("kings") from the capture file as shown in figure 6.

If we input this key in hexadecimal form into Wireshark (using *Preferences > Protocols > IEEE 802.11 > Decryption Keys*) it will automatically decrypt the WEP-encrypted packets. Using the `http.request` display filter we can see that the HTTP requests being sent include the header (Figure 7):



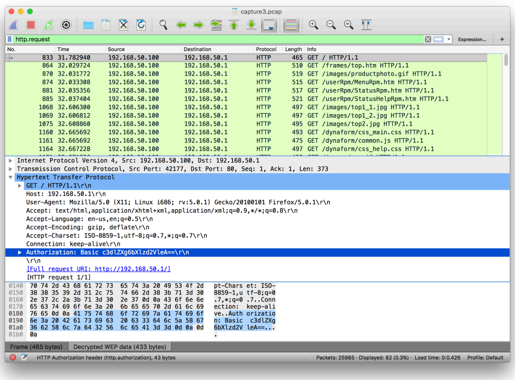
```
capture3.pcap
wlan.fc.protected == 1
No.   Time        Source              Destination         Protocol  Length Info
   31 1.086556    Alfa_52:76:70       SweexEur_19:89:c0   802.11     92 Data, SN=1054, FN=0, Flags=.p.....T
   34 1.103939    SweexEur_19:89:c0   Alfa_52:76:70       802.11     92 Data, SN=736, FN=0, Flags=.p....F.
   36 1.104476    Alfa_52:76:70       SweexEur_19:89:c0   802.11     92 Data, SN=1055, FN=0, Flags=.p.....T
  130 4.083484    Alfa_52:76:70       SweexEur_19:89:c0   802.11     92 Data, SN=1056, FN=0, Flags=.p.....T
  131 4.098308    SweexEur_19:89:c0   Alfa_52:76:70       802.11     92 Data, SN=1674, FN=0, Flags=.p....F.
  133 4.098844    Alfa_52:76:70       SweexEur_19:89:c0   802.11     92 Data, SN=1057, FN=0, Flags=.p.....T
  195 6.084508    Alfa_52:76:70       SweexEur_19:89:c0   802.11     92 Data, SN=1058, FN=0, Flags=.p.....T
  198 6.117246    SweexEur_19:89:c0   Alfa_52:76:70       802.11     92 Data, SN=2322, FN=0, Flags=.p....F.
  200 6.117788    Alfa_52:76:70       SweexEur_19:89:c0   802.11     92 Data, SN=1059, FN=0, Flags=.p.....T
  829 31.781404   Alfa_52:76:70       SweexEur_19:89:c0   802.11    100 Data, SN=1060, FN=0, Flags=.p.....T
  830 31.782404   SweexEur_19:89:c0   Alfa_52:76:70       802.11    100 Data, SN=2217, FN=0, Flags=.p....F.
  832 31.782940   Alfa_52:76:70       SweexEur_19:89:c0   802.11     92 Data, SN=1061, FN=0, Flags=.p.....T

▶ Frame 31: 92 bytes on wire (736 bits), 92 bytes captured (736 bits)
▼ IEEE 802.11 Data, Flags: .p.....T
    Type/Subtype: Data (0x0020)
  ▼ Frame Control Field: 0x0841
      .... ..00 = Version: 0
      .... 10.. = Type: Data frame (2)
      0000 .... = Subtype: 0
    ▼ Flags: 0x41
        .... ..01 = DS status: Frame from STA to DS via an AP (To DS: 1 From DS: 0) (0x1)
        .... .0.. = More Fragments: This is the last fragment
        .... 0... = Retry: Frame is not being retransmitted
        ...0 .... = PWR MGT: STA will stay up
        ..0. .... = More Data: No data buffered
        .1.. .... = Protected flag: Data is protected
        0... .... = Order flag: Not strictly ordered

0000  08 41 34 00 00 16 0a 19  89 c0 00 c0 ca 52 76 70   .A4..... .....Rvp
0010  00 16 0a 19 89 c0 e0 41  b0 19 d9 00 15 97 35 cf   .......A ......5.
0020  e3 32 a8 d5 f4 b5 19 64  de 2c d5 56 d7 b0 d9 89   .2.....d .,.V....
0030  09 f4 b4 ba 3b 05 10 f7  31 37 c0 32 87 c1 4d 87   ....;... 17.2..M.
0040  3f 74 43 d6 8c 24 1a 54  74 16 0b 55 9b ff da db   ?tC..$.T t..U....
0050  eb 0c d7 bd 53 b7 be 9d  8b 0c d3 f7               ....S... ....

Protected flag (wlan.fc.protected), 1 byte    Packets: 25985 · Displayed: 15454 (59.5%) · Load time: 0:0.183    Profile: Default
```

Figure 5: Wireshark showing encrypted Wifi packets in capture3.pcap



```
2. bash

                                 Aircrack-ng 1.2 rc4


                     [00:00:03] Tested 882 keys (got 15391 IVs)

   KB    depth   byte(vote)
    0    0/  1   6B(24064) 09(19968) F0(19456) F4(19456) 7B(19200) 03(18944) 06(18944) 04(18688)
    1    0/  1   69(23552) 0E(19968) 3D(19456) 5D(19456) FA(19200) BE(19200) 13(18944) 41(18944)
    2    0/ 22   6E(21504) D4(20992) 84(20480) 95(19968) EE(19968) 3D(19968) CA(19712) A1(19200)
    3    0/  2   67(23296) FC(20992) DB(19712) 8A(19712) 2B(19456) E1(19456) 1D(19200) 56(19200)
    4   19/ 21   42(18432) 1C(18176) 97(18176) C0(18176) A7(18176) 1D(18176) 51(17920) 96(17920)

                  KEY FOUND! [ 6B:69:6E:67:73 ] (ASCII: kings )
        Decrypted correctly: 100%


~/Personal/cyber-security/protocol-analysis (master) $ 
```

Figure 6: Output from aircrack-ng after cracking WEP on capture3.pcap



```
capture3.pcap
http.request
No.   Time        Source           Destination    Protocol  Length Info
  833 31.782940   192.168.50.100   192.168.50.1   HTTP      465 GET / HTTP/1.1
  864 32.029724   192.168.50.100   192.168.50.1   HTTP      510 GET /frames/top.htm HTTP/1.1
  870 32.031772   192.168.50.100   192.168.50.1   HTTP      519 GET /images/productphoto.gif HTTP/1.1
  874 32.033308   192.168.50.100   192.168.50.1   HTTP      515 GET /userRpm/MenuRpm.htm HTTP/1.1
  881 32.035356   192.168.50.100   192.168.50.1   HTTP      517 GET /userRpm/StatusRpm.htm HTTP/1.1
  885 32.037404   192.168.50.100   192.168.50.1   HTTP      521 GET /userRpm/StatusHelpRpm.htm HTTP/1.1
 1068 32.606300   192.168.50.100   192.168.50.1   HTTP      497 GET /images/top1_1.jpg HTTP/1.1
 1069 32.606812   192.168.50.100   192.168.50.1   HTTP      497 GET /images/top1_2.jpg HTTP/1.1
 1075 32.608860   192.168.50.100   192.168.50.1   HTTP      495 GET /images/top2.jpg HTTP/1.1
 1160 32.665692   192.168.50.100   192.168.50.1   HTTP      493 GET /dynaform/css_main.css HTTP/1.1
 1161 32.665692   192.168.50.100   192.168.50.1   HTTP      475 GET /dynaform/common.js HTTP/1.1
 1164 32.667228   192.168.50.100   192.168.50.1   HTTP      497 GET /dynaform/css_help.css HTTP/1.1

▶ Internet Protocol Version 4, Src: 192.168.50.100, Dst: 192.168.50.1
▶ Transmission Control Protocol, Src Port: 42177, Dst Port: 80, Seq: 1, Ack: 1, Len: 373
▼ Hypertext Transfer Protocol
  ▶ GET / HTTP/1.1\r\n
    Host: 192.168.50.1\r\n
    User-Agent: Mozilla/5.0 (X11; Linux i686; rv:5.0.1) Gecko/20100101 Firefox/5.0.1\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-us,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Connection: keep-alive\r\n
  ▶ Authorization: Basic c3dlZXg6bXlzd2VleA==\r\n
    \r\n
    [Full request URI: http://192.168.50.1/]
    [HTTP request 1/1]

0140  70 74 2d 43 68 61 72 73  65 74 3a 20 49 53 4f 2d   pt-Chars et: ISO-
0150  38 38 35 39 2d 31 2c 75  74 66 2d 38 3b 71 3d 30   8859-1,u tf-8;q=0
0160  2e 37 2c 2a 3b 71 3d 30  2e 37 0d 0a 43 6f 6e 6e   .7,*;q=0 .7..Conn
0170  65 63 74 69 6f 6e 3a 20  6b 65 65 70 2d 61 6c 69   ection:  keep-ali
0180  76 65 0d 0a 41 75 74 68  6f 72 69 7a 61 74 69 6f   ve..Auth orizatio
0190  6e 3a 20 42 61 73 69 63  20 63 33 64 6c 5a 58 67   n: Basic  c3dlZXg
01a0  36 62 58 6c 7a 64 32 56  6c 65 41 3d 3d 0d 0a 0d   6bXlzd2V leA==...
01b0  0a                                                 .

Frame (465 bytes)   Decrypted WEP data (433 bytes)
HTTP Authorization header (http.authorization), 43 bytes    Packets: 25985 · Displayed: 82 (0.3%) · Load time: 0:0.426    Profile: Default
```

Figure 7: Wireshark showing decrypted HTTP traffic from capture3.pcap

```
Authorization: Basic c3dlZXg6bXlzd2VleA==
```

Using the same Python fragment used for `capture1.pcap` above we can therefore discover that the HTTP Basic username was ‘sweex’ and the password was ‘mysweex’.

References

Berners-Lee, T., R. Fielding, and H. Frystyk. 1996. “RFC1945: Hypertext Transfer Protocol – HTTP/1.0.” https://tools.ietf.org/search/rfc1945.

Fluhrer, Scott, Itsik Mantin, and Adi Shamir. 2001. “Weaknesses in the Key Scheduling Algorithm of RC4.” *Selected Areas of Cryptography* 2259. Springer: 1–24.

Lamping, Ulf, Richard Sharpe, and Ed Warnicke. 2014. “Wireshark User’s Guide: Capturing with Tcpdump for Viewing with Wireshark.” https://www.wireshark.org/docs/wsug_html_chunked/AppToolstcpdump.html.

Mishra, Charit. 2016. *Mastering Wireshark*. Birmingham, UK: Packt Publishing.

Mogul, Jeffrey. 1984. “RFC919: Broadcasting Internet Datagrams.” https://tools.ietf.org/html/rfc919.

Orebaugh, Angela, Gilbert Ramirez, and Jay Beale. 2006. *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Rockland, MA: Syngress.

Plummer, David C. 1982. “RFC829: An Ethernet Address Resolution Protocol.” https://tools.ietf.org/html/rfc826.