

Chapter 8

BREAKING THROUGH

*“To advance irresistibly, push through their gaps.”
—Sun Tzu*

If you have already read the wireless penetration testing section of the template in Appendix G, you will find that this chapter is a more detailed walk-through. If you understand how WLANs work, comprehend the general wireless security principles, and have researched both tools of the trade and test and attack planning chapters, you might skip this one. Otherwise, stay with us and read the answers to your questions.

The Easiest Way to Get in

The first thing any attacker looks for is “low-hanging fruit.” An inexperienced attacker will search for it because he or she can’t get into anything else, whereas an experienced Black Hat will look for it to save time and to be sure that (unless it’s a honeypot) no IDS and egress filtering is present and hosts on the network are easy to break into for further backdoor planting. Despite the opinion of a few “security experts,” the amount of wide-open wireless networks is incredible. By “wide open” we mean no WEP, no MAC filtering, no closed ESSID, no protocol filtering, and most likely AP management interface accessible from the WLAN. There are a variety of reasons why this situation exists, the major one being the users’ (or even





Chapter 8 • Breaking Through

system administrators') laziness and ignorance. When attacking such networks, a cracker has only three main concerns: physical network reachability, connectivity to the Internet, and the (rare) possibility of a honeypot trap. Let's explore each in further detail.

- Physical network reachability: Even if a network is wide open, it is no good (for a cracker) if the only way to connect to it is to sit with a laptop right under the office window.
- Connectivity to the Internet: Is it present and how "fat" is the "pipe"?
- Honeypot trap: Is trouble on the way?

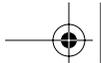
The first issue, reachability, is addressed by a high-gain antenna. A high-gain omnidirectional might look like a walking stick or a pool cue and will not raise any suspicions. The majority of Yagis can pass for poster holders and even the directional dishes would not surprise anyone as long as the cracker passes himself or herself off as telecom engineer troubleshooting a link or even an amateur radio enthusiast. It is truly amazing when you sit in the park with a huge antenna in the middle of nowhere and present yourself as a university student doing research. The second issue, connectivity, can be sorted via multiple means; for example, by looking at the DHCP traffic present, a gateway IP would be shown. We have to admit, we like Ettercap. Press "p/P" for the Ettercap plug-ins available. The plug-in that discovers LAN gateways is called triton. The last issue, the honeypot trap, is difficult to solve. Use your intuition and skill to determine whether this low-hanging fruit is poisoned. Looking for sniffers helps; check out the hunter plug-in in Ettercap (Figure 8-1).

Of course, as a corporate penetration tester you can simply ask if there are honeypots, but that would spoil both fun and the challenge, would it not?

A Short Fence to Climb: Bypassing Closed ESSIDs, MAC, and Protocols Filtering

Let us explore slightly more protected WLANs. How about so-called closed networks? ESSID makes a bad shared secret. The reason is that it is not removed from all management frames. For example, reauthenticate and reassociate frames will contain the ESSID value. Thus, a net-





A Short Fence to Climb: Bypassing Closed ESSIDs, MAC, and Protocols Filtering

```

ettercap 0.6.b
-----
          9 hosts in this LAN (192.168.77.5 : 255.255.255.0)
          1) 192.168.77.5          1) 192.168.77.5

Starting hunter plugin...
Scanning: ANY

Most probably sniffing NICs:
- NONE

Less probably sniffing NICs:
- 192.168.77.6
- 192.168.77.250

hunter plugin ended. (press 'q' to quit...)

-----
Your IP: 192.168.77.5 MAC: 00:04:75:E7:26:51 Iface: eth0 Link: SWITCH
Host: kabanos.core.arhont.com (192.168.77.5) : 00:04:75:E7:26:51

```

Figure 8.1 Ettercap hunter plug-in.

work with roaming hosts will not benefit from the closed ESSIDs at all and sending a deauthenticate frame to one or more hosts on the closed WLAN is easy:

```

arhontus:~# ./ssid_jack -h
Ssid Jack: Proof of concept so people will stop calling an ssid
a password.
Usage: ./ssid_jack -b <bssid> [ -d <destination mac> ] [ -c
<channel number> ] [ -i <interface name> ]

```

```

        -b: bssid, the mac address of the access point (e.g.
00:de:ad:be:ef:00)
        -d: destination mac address, defaults to broadcast
address.
        -c: channel number (1-14) that the access point is on,
defaults to current.
        -i: the name of the AirJack interface to use (defaults to
aj0).

```

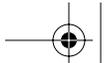
```

arhontus:~# ssid_jack -b 00:02:2d:ab:cd: -c 11
Got it, the ssid is (escape characters are c style):
"ArhOnt-X"

```

On a BSD platform, use the `dinject-death` utility from Wnet and sniff the passing traffic while using it.





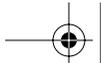
Chapter 8 • Breaking Through

Of course, such methodology will only work against a network with several reachable associated hosts present. In the rare case of a lonely access point, your best bet would be to guess the closed ESSID. It is surprising, but many users enable closed ESSID but do not change the actual ESSID value from the default (perhaps counting on the fact that it is not broadcasted anyway). Use the OUI, which is the first 3 bytes of the MAC address, to find out the access point manufacturer (see RFC 1700) and check the default ESSID values for the access points produced by this particular vendor and supporting closed ESSIDs. You can find these values and many other interesting facts in Appendix H.

MAC filtering is also trivial to bypass, even though we have seen some wi-fi inexperienced security consultants claiming it to be a good protection – shame on you guys. Sniff the network traffic to determine which MAC addresses are present. When the host quits the network, assume it's MAC and associate. You can also change your MAC and IP address to the same values as those on the victim's host and coexist peacefully on the same (shared) network (piggybacking). Surely you would need to disable ARPs on your interface and go to Defcon 1 with your firewall. You would also have to be careful about what traffic you send out to the network to prevent the victim host from sending too many TCP resets and ICMP port unreachables, so their rare and megaexpensive knowledge-based IDS does not get triggered. You should try to restrict your communications to ICMP when communicating with the outside world. You can use any Loki-style ICMP-based backdoor (e.g., encapsulate data in echo replies or any other ICMP types that do not illicit responses). If you want to enjoy full network interoperability, you don't have to wait for the host to leave and can simply kick it out. Such action might lead to user complaints and an IDS alarm, in particular if WIDS is in place, but who cares, especially since you urgently need to check the latest posts at <http://www.wi-foo.com>. Therefore, try to use your common sense and pick a host that does not seem to generate any current traffic and send it a deassociate frame spoofing your MAC address as an access point. At the same time, have a second client card plugged in and configured with the MAC of a target host and other WLAN parameters to associate. It is a race condition that you are going to win, because no one can stop you from flooding the spoofed host with deassociate frames continuously. To flood the host with deassociate frames from Linux you can use wlan_jack:

```
arhontus:~# ./wlan_jack -h
Wlan Jack: 802.11b DOS attack.
```





A Short Fence to Climb: Bypassing Closed ESSIDs, MAC, and Protocols Filtering

```
Usage: ./wlan_jack -b <bssid> [ -v <victim address> ] [ -c
<channel number> ] [ -i <interface name> ]
  -b: bssid, the mac address of the access point (e.g.
      00:de:ad:be:ef:00)
  -v: victim mac address, defaults to broadcast address.
  -c: channel number (1-14) that the access point is on,
      defaults to current.
  -i: the name of the AirJack interface to use (defaults to
      aj0).
```

```
arhontus:~# ./wlan_jack -b 00:02:2d:ab:cd: -v 00:05:5D:F9:ab:cd -
c 11
Wlan Jack: 802.11 DOS utility.
```

```
Jacking Wlan...
```

Alternatively, you can employ File2air. If running HostAP drivers, you can launch Void11 or craft your own frames with Libwlan. Another way of flooding the host with deassociate frames is using Mike Schiffman's *omerta* utility under HostAP and employing the Libradiate library. In this book we do not describe Libradiate, because it ceased to be supported more than a year ago and at the moment *omerta* is probably the only tool worth mentioning here that employs Libradiate. On the OpenBSD platform you can employ the *dinject-disas* utility, perhaps run from a simple looping shell script. Finally, a different way of launching very efficient DoS attacks with AirJack is using *fata_jack*. Please consult the wireless DoS attacks section at the end of this chapter to learn more about it.

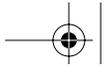
Just to remind you how to change a MAC address when you need it:

```
# ifconfig wlan0 hw ether DE:AD:BE:EF:CO:DE      (Linux ifconfig)
# ip link set dev wlan0 address DE:AD:BE:EF:CO:DE (Linux iproute)
# ifconfig wi0 ether DEADBEEFCODE                (FreeBSD)
# sea -v wi0 DE:AD:BE:EF:CO:DE                    (OpenBSD)
```

Sea is a separate utility that does not come with OpenBSD but can be found at <http://www.openbsd.org>.

Protocol filtering is harder to bypass. Unfortunately for system administrators and fortunately for attackers, very few access points on the market implement proper protocol filtering and they tend to be high-end, expensive devices. Also, protocol filtering applies only to a few specific situations in which user activity is limited to a narrow set of actions, for example, browsing a corporate site through HTTPS or sending e-mails via Secure Multipurpose Internet Mail Extensions (S/MIME) from PDAs given to employees for these aims specifically. SSH port forwarding might help, but you have to be sure that both sides support SSHv2.





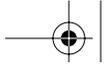
Chapter 8 • Breaking Through

The main attacks against networks protected by protocol filtering are attacks against the allowed secure protocol (which might not be as secure as it seems). Good examples of such insecurity are well-known attacks against SSHv1 implemented in Dug Song's *Dsniff* by the `sshshow` and `sshmitm` utilities. Whereas `sshshow` can help an attacker disclose some useful information about the bypassing SSH traffic (e.g., the authentication attempts or length of transmitted passwords or commands with both SSHv1 and SSHv2 traffic), `sshmitm` is a powerful man-in-the-middle for SSHv1 utility that allows SSHv1 password login capture and connection hijacking attacks. Unfortunately, although the majority of complete networked operational systems currently support SSHv2, SSHv1 is often the only choice available to log in to routers, some firewalls, and other networking devices and this is still preferable to `telnet` or `rlogin`. On wired networks, traffic redirection via DNS spoofing is necessary for `sshmitm` to work. However, Layer 2 `monkey_jack`-style man-in-the-middle attacks can successfully replace DNS spoofing on 802.11 links, leaving fewer traces in the network IDS logs unless a proper wireless IDS is implemented (which is rarely the case).

The creator of *Dsniff* did not leave HTTPS without attention as well. `webmitm` can transparently proxy and sniff HTTPS traffic to capture most of the "secure" SSL-encrypted Web mail logins and Web site form submissions. Again, `dnsspoof` traffic redirection for `webmitm` can be substituted by a wireless-specific man-in-the-middle attack, raising fewer system administrators' eyebrows. Another remarkable man-in-the-middle tool specifically designed for attacking various SSL connections (HTTPS, IMAPS, etc.) is *Omen*. Just like `webmitm`, more information on using *Omen* follows in the next chapter.

If network designers and management decided to rely on SSH, HTTPS, and so on as their main line of defense and did not implement lower-layer encryption and proper mutual authentication (e.g., 802.1x/EAP-TLS or better), you might not even have to attack Layer 6 security protocols. Nothing would stop a cracker from associating with the target network, running a quick `nmap` scan, and launching an attack against the discovered `sshd` (e.g., using `sshnuke` to exploit the CRC32 vulnerability, if you want to be as 1337 as Trinity). Of course, the real-life CRC32 bug was patched eons ago, but new `sshd` vulnerabilities tend to appear on a regular basis. As for HTTPS security, the latest CGI vulnerability scanners support HTTPS (e.g., Nikto with the `-ssl` option) and in the majority of cases the difference in exploitation of the discovered CGI holes over the HTTPS protocol is limited to changing the target port to 443 from 80 or piping data through `stunnel`.





Picking a Trivial Lock: Various Means of Cracking WEP

Finally, a desperate cracker can always resort to brute force. There are a variety of utilities and scripts for SSH brute forcing: `guess-who`, `ssh-crack`, `ssh-brute.sh`, `55hb_v1.sh`, and so on. With SSL-protected Web logins you can try the `php-ssl-brute` script. Although brute forcing leaves telltale multiple login signs in the logs, wireless attackers might be unconcerned, as it is more difficult to locate and prosecute a cracker on a WLAN anyway. Although brute force is both time and battery power consuming for a mobile wireless attacker, if it is the only choice available, someone will eventually give it a try and perhaps succeed.

Picking a Trivial Lock: Various Means of Cracking WEP

The next step on your way to complete WLAN control is cracking WEP. As mentioned, wireless attacks do not start and end with cracking WEP, as many security experts might tell you. However, if the attacker cannot break WEP (if present), all he or she can do is disrupt the network operations by DoS attacks on layers below the protocol WEP implementation.

From the section dealing with WEP cracking tools, you have probably gathered that there are three major ways of attacking WEP:

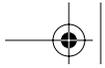
- Brute-forcing and improved brute-forcing
- FMS attack
- Improved FMS attack

Because this book is a down-to-earth guide to wireless security and hundreds of pages have already been written on WEP weaknesses and cracking mathematics, we do not aim to provide a comprehensive guide to the mathematical internals of WEP cracking attacks. Nevertheless, we believe it is important to present some cryptological data on WEP as an act of homage to all researchers who contributed to the WEP analysis and flaw enumeration.

WEP Brute-Forcing

Pure WEP keyspace brute-forcing with tools such as `wep_tools` or `dwepcrack` brute-forcing options is realistic only against 40-bit WEP keys.





Chapter 8 • Breaking Through

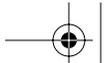
Even with this limited key size, it might take about 50 days on a single average Pentium III host. Nevertheless, an efficient distributed attack against 40-bit WEP is possible and one should never underestimate the potential of dictionary attacks, which are also applicable to 128-bit and higher WEP key size. In particular, it applies to the use of the newer Wepattack tool that can run dictionary attacks against a single captured data packet encrypted using WEP.

Tim Newsham has pointed out that the algorithm accepted as the de facto standard for 40-bit WEP key generation by many wireless equipment vendors is extremely flawed. It starts from folding a password string into a 32-bit number that reduces the keyspace from 2^{40} to 2^{32} bits. This number is employed to seed a pseudorandom number generator (PRNG; see Chapter 11), which is used to derive all four 40-bit WEP keys used on the network. Although the PRNG-generated keyspace has a cycle length of 2^{32} bits, because of the way the values are derived from the PRNG, the actual cycle length of drawn values is only 2^{24} bits. To be more specific, a seed x produces the same keys as a seed $x + 2^{24}$. To make the situation even worse, the method chosen to fold a password string into a 32-bit seed ensures that the high bit of each of the four bytes always equals zero. The effect of these weaknesses combined is that the algorithm can only generate 2^{21} unique sets of WEP keys, corresponding to seeds between 0 and 0x1000000, which do not have bits 0x80, 0x8000, or 0x800000 set. Thus, it takes 2^{21} operations or less to crack any set of WEP keys generated from a password processed with such an algorithm. In Newsham's observations, this corresponds roughly to 90 seconds of cracking time on a 233-MHz PII or 35 seconds on a 500-MHz PIII; this is quite a difference if compared to 50 days of brute-forcing without this flaw.

However, not all vendors used the vulnerable key generation algorithm (to our knowledge, 3Com never did), 40-bit keys aren't used much anymore, and there are tools that ensure proper 40-bit key generation. An example of such a tool is `dwepkeygen`, included as part of BSD-airtools. In addition, to crack WEP using `wep_tools`, a large (about 24 Gb) pcap-format dump file is required. Thus, although Newsham's comments are interesting and have their place in the history of wireless cryptanalysis, we do not recommend trying the attack he developed or using brute-forcing in general against 128/104-bit WEP keys used by modern wireless networks.

However if you have truly massive traffic dump files, trying a dictionary attack using `wep_tools` or `dwepcrack` could bring success. Even better, you can try your luck with a dictionary attack against a single captured data packet or limited-size traffic dumps using Wepattack.





Picking a Trivial Lock: Various Means of Cracking WEP

The FMS Attack

The most common attack against WEP is Scott Fluhrer, Itsik Mantin, and Adi Shamir's (FMS) key recovery methodology discovered in 2001 (the original paper entitled "Weaknesses in the Key Scheduling Algorithm of RC4" is available from http://www.cs.umd.edu/~waa/class-pubs/rc4_ksaproc.ps). As you already know, this attack was implemented first by the Wep_crack and then by AirSnort. For those interested in how the attack algorithms work, we present a brief explanation here. If you are already familiar with the FMS attack or aren't interested in the "theoretical" cryptanalysis, feel free to skip this section and move forward.

The FMS attack is based on three main principles:

1. Some IVs set up RC4 cipher (see Chapter 11) the way it can reveal key information in its output bytes.
2. Invariance weakness allows use of the output bytes to determine the most probable key bytes.
3. The first output bytes are always predictable because they contain the SNAP header defined by the IEEE specification.

A WEP key can be defined as $K=IV.SK$ where SK is the secret key. The RC4 operation in a nutshell is $K=IV.SK \rightarrow KSA(K) \rightarrow PRNG(K) \text{ XOR data stream}$. The scheduling algorithm KSA(K) works in the following way:

```

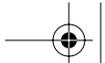
Initialization:
  For i = 0 \x{2026} N - 1
    S[i] = i
  j = 0
Scrambling:
  For i = 0 \x{2026} N - 1
    j = j + S[i] + K[i mod l]
    Swap(S[i], S[j])
  
```

The PRNG works as:

```

Initialization:
  i = 0
  j = 0
Generation Loop:
  i = i + 1
  j = j + S[i]
  Swap(S[i], S[j])
  Output Z = S[S[i] + S[j]]
  
```





Chapter 8 • Breaking Through

Some IVs initialize the PRNG the way the first byte in the stream is generated using a byte from the secret key. Because the first data byte that the PRNG output is XORed with is predictable (SNAP header), it is easy to derive the first PRNG byte. The values we can get from weak IVs are only true about 5 percent of the time; some are true about 13 percent of the time. Taking into account the key size, it takes six to eight million packets of analysis to determine the correct WEP key. The theoretical packets throughput maximum (“wire speed”) on the throughput-comparable to 802.11b LAN 10Base-T shared Ethernet is 812 frames per second (frame size of 1,518 bits). If we divide 6,000,000 by 812 we will get about 7,389 seconds or just above 2 hours necessary to accumulate enough packets for efficient WEP cracking. However, as we will see, the reality is different.

The basic FMS attack comes down to searching for IVs that conform to the $(A + 3, N - 1, X)$ rule, where A is the byte in the secret key you are cracking, N is the size of the S-box (256) and X is a random number. It is advised that the following equations are applied right after the KSA:

$$\begin{aligned} X &= SB+3[1] < B+3 \\ X + SB+3[X] &= B+3 \end{aligned}$$

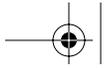
The main problem is that such an equation is dependent on the previous key bytes, so it must be applied to the entire packet dump for every key byte that is tested. In its classical form, the FMS attack tests only the first byte of the output because it is very reliable; we know that the first byte of the SNAP header is nearly always `0xAA`.

An Improved FMS Attack

To bypass this problem and optimize the FMS attack, H1kari of Dasb0den Labs has analyzed the patterns of weak IVs appearance and how they relate to the key bytes they rely on. As he pointed out in the “Practical Exploitation of RC4 Weaknesses in WEP Environments” article (a must-read for any serious wireless security professional; available at <http://www.dachb0den.com/projects/bsd-airtools/wepexp.txt>), a basic pattern present can be defined as follows:

```
Definitions:
let x = iv[0]
let y = iv[1]
let z = iv[2]
let a = x + y
let b = (x + y) - z
```





Picking a Trivial Lock: Various Means of Cracking WEP

```

Byte 0:
  x = 3 and y = 255
  a = 0 or 1 and b = 2
Byte 1:
  x = 4 and y = 255
  a = 0 or 2 and b = SK[0] + 5
Byte 2:
  x = 5 and y = 255
  a = 0 or 3 and b = SK[0] + SK[1] + 9
  a = 1 and b = 1 or 6 + SK[0] or 5 + SK[0]
  a = 2 and b = 6
Byte 3:
  x = 6 and y = 255
  a = 0 or 4 and b = SK[0] + SK[1] + SK[2] + 14
  a = 1 and b = 0 or SK[0] + SK[1] + 10 or SK[0] + SK[1] + 9
  a = 3 and b = 8
Byte 4:
  x = 7 and y = 255
  a = 0 or 5 and b = SK[0] + SK[1] + SK[2] + SK[3] + 20
  a = 1 and b = 255 or SK[0] + SK[1] + SK[2] + 15 or
    SK[0] + SK[1] + SK[2] + 14
  a = 2 and b = SK[0] + SK[1] + 11 or SK[0] + SK[1] + 9
  a = 3 and b = SK[0] + 11
  a = 4 and b = 10
  
```

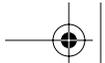
The resulting distribution pattern would be similar to this:

Secret	Key	Byte											
	0	1	2	3	4	5	6	7	8	9	a	b	c
			+		+		+		+		+		+
0	8	16	16	16	16	16	16	16	16	16	16	16	16
1	8		16	16	16	16	16	16	16	16	16	16	16
2		16	8		16	16	16	16	16	16	16	16	16
a	3			16	8		16	16	16	16	16	16	16
4					16	8		16	16		16	16	16
V	5					16	8		16	16	16		16
a	6						16	8		16	16	16	16
l	7							16	8		16	16	16
u	8								16	8		16	16
e	9									16	8		16
s	a										16	8	16
	b											16	8
	c												16
	d												16

8 - 8-bit set of weak ivs
 16 - 16-bit set of weak ivs
 + - 2 additional x and y dependent 8-bit weak ivs

From this distribution a rough estimate of weak IVs per key byte can be derived. There are other means of deriving this value as outlined in the referenced article. However, the real catch is to find an algorithm that





Chapter 8 • Breaking Through

will allow filtering out weak IVs based on the secret key byte that they can attack. This can be done with an algorithm similar to this:

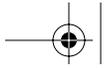
```
let l = the amount of elements in SK

i = 0
For B = 0 ... l - 1
  If (((0 <= a and a < B) or
      (a = B and b = (B + 1) * 2)) and
      (B % 2 ? a != (B + 1) / 2 : 1)) or
      (a = B + 1 and (B = 0 ? b = (B + 1) * 2 : 1)) or
      (x = B + 3 and y = N - 1) or
      (B != 0 and !(B % 2) ? (x = 1 and y = (B / 2) + 1) or
      (x = (B / 2) + 2 and y = (N - 1) - x) : 0)
  Then ReportWeakIV
```

Such methodology effectively reduces the search time for each key by at least 1/20, thus giving us the time necessary to crack WEP. Now you don't need to collect 6,000,000 packets or more; half a million packets could be sufficient! This is the improved FMS attack as implemented by BSD-airtools `dwepcrack`; read its source code to discover and learn more.

The practicality of WEP cracking attacks is still denied by many. There are statements that, for example, a home or SOHO WLAN will not generate enough traffic to collect a sufficient amount of weak or interesting IVs for the key compromise in a reasonable time period. You just saw a methodology that can significantly cut the necessary data collected and this methodology has been implemented in a security auditing tool since the year 2001! However, even if the most commonly used WEP cracking tool, AirSnort, is employed, the results can be less than encouraging for the few remaining WEP enthusiasts. In our experience it takes only 3,000 to 3,500 interesting IVs frames to break the WEP key for either 64-bit or 128-bit WEP keys using AirSnort. The only difference mentioned between cracking the keys of both sizes is the amount of time necessary to collect these frames. It took 10 to 20 percent more time to collect the necessary amount of interesting IVs frames to obtain a 128-bit key on a testing wireless network. Our record of breaking a 64-bit WEP with AirSnort is 1 hour 47 minutes on a point-to-point 802.11b link with one of the hosts flood pinging the other (approximately 300 packets per second). Such an attack required 107 minutes * 300 packets/second = 1,926,000 packets, much less than the 6,000,000 packets estimated theoretically. It could've been sheer luck, but would you base your network security on guesswork considering how lucky or unlucky an attacker might be?





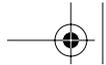
Picking a Trivial Lock: Various Means of Cracking WEP

On a large, corporate wireless network, 300 packets per second is neither unusual nor unexpected, especially with 802.11a and 802.11g standards offering higher bandwidth and network throughput. The presence of “chatty” network protocols (RIP, link-state routing protocols “hello” packets, spanning tree, HSRP, VRRP, NetBIOS, IPX RIP and SAP, AppleTalk, etc.) might dramatically decrease the time needed to crack WEP. It also generates wireless traffic even when no user activity is present. Imagine a large wireless Novell-based network running NetBIOS over IPX and using three Cisco routers with turned-on hot standby for failover resilience and enabled CDP (we have seen networks like this in the United Kingdom on several occasions). Such a network does not have to be the WLAN itself; leaking wired traffic on the wireless side is sufficient and we have frequently seen access points plugged directly into the switch or hub. Let’s say there are 100 hosts on the network and no user activity present. In one hour, every host will generate approximately 1,200 NetBIOS keep-alives, 40 IPX RIPs, and 40 SAPs, and each router will send 1,200 HSRP Hello packets and 60 CDP frames if the defaults aren’t changed (they rarely are), as well as the obvious 40 RIPs. Thus, the number of generated packets will be $100 \times (1,200 + 40 + 40) + 3 \times (1,200 + 60 + 40) = 131,900$ packets per hour. Thus, accumulating the 2,000,000 packets necessary to crack WEP with AirSnort in our example will take approximately 15 hours. With `dwepcrack` as few as 500,000 packets might be needed, which translates into approximately 3 hours, 47 minutes, without a single user logged in! Remember that this network is both perfect and hypothetical. In reality, a Novell server might send more than one SAP in 90 seconds because a single SAP packet can advertise up to seven services and the server might run more. NLSIP might be running and STP traffic could be present. We frequently find networks with system administrators completely unaware of the unnecessary and unused STP traffic on the network and some higher end switches and even wireless access points have STP enabled by default. Mind the traffic!

Finally, in some cases, old 802.11b cards use the same IV value or start counting IV numbers from 0 each time the card is initialized and increments these numbers by one. This also significantly cuts the time necessary to crack WEP.

How about cracking WEP on 802.11a networks? It is essentially the same. The only difference is that we aren’t aware of decent 802.11a support on BSD and AirSnort will not work with `ark_5k`. However, you can save a `pcap-format` 802.11a traffic dump file obtained using an Atheros chipset card in the RFMON mode and `tcpdump` (or `Kismet`) and feed it to AirSnort or even `dwepcrack` (after booting into BSD). If you want real-time WEP cracking on an 802.11a network, use `wepcrack` and the power





Chapter 8 • Breaking Through

of `at/crond` as we have described. For example, you can pipe `tcpdump` output into `prism-getIV.pl` and then process the `IVFile.log` file with `WEPCrack.pl`.

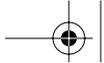
Picking the Trivial Lock in a Less Trivial Way: Injecting Traffic to Accelerate WEP Cracking

The attacks against WEP we have reviewed so far are purely passive and rely on traffic being present on the wireless network. But can we generate the additional WLAN traffic without even being associated to the network? The answer is positive and we have reviewed the tools such as `reinj` or `Wepwedgie` in Chapter 5. There are claims that `reinj` can reliably cut WEP cracking time to less than one hour and there is no reason not to believe these claims (shouldn't a security professional be paranoid anyway?). Thus, the arguments like "this SOHO network generates too little wireless traffic to be a suitable target for WEP cracking" fail; nothing stops the cracker from introducing additional network traffic using the tools we have described. Even more, the attacks on WLANs could include host discovery and even port scanning via the wireless traffic injection without even knowing WEP. TCP SYNs can be predictable and thus injected; the same applies to TCP ACKs, TCP RSTs, TCP SYN-ACKs, and ICMP unreachables such as ICMP port unreachable. At the moment, one Linux tool to launch attacks of this class, the `Wepwedgie`, is under active development and the working beta version should be available as this book hits the shelves—watch out! You don't have to wait until the WEP key is cracked to proceed with further network analysis; use `Wepwedgie` while cracking the key and save your time.

Field Observations in WEP Cracking

To end the WEP cracking story, here are some observations from our practical work. There are specific conditions in which RF noise, an unreliable link, or host deassociation or deauthentication can increase rather than decrease the amount of WEP-encrypted traffic flowing through the wireless net.





Cracking TKIP: The New Menace

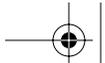
One such condition is the presence of connection-oriented protocol links. Imagine two hosts communicating over the wireless link using TCP or SPX. If the link is unreliable or fails, the data segments will be retransmitted many times until the whole datagram is eventually passed. The amount of packets necessary to transmit the same amount of data will increase and so will the amount of interesting IV frames to catch. Even more, to alleviate the awful link problem, the system administrator might decrease the frame size as all wireless networking manuals and how-tos advise. This will surely help, but it will also increase the amount of fragments sent, with each fragment having its own very special IV. Please note that the casual RF problems of multipath, active interference, and hidden nodes are common reasons to decrease the wireless frame size; truly, “the network stability and network security are two sides of the same coin” (Dan Kaminskiy). It is interesting that no research has been done to establish the mathematical relation between the preset 802.11 frame size and the time efficiency of WEP cracking. Surely it is a useful topic that many wireless hackers might like to investigate.

Another case of link disruption generating excessive amounts of traffic is triggering routing updates. Imagine a link-state routing protocol (let’s say OSPF) running over the wireless network. Should the link to one of the routers go down, an LSA flood will follow, giving a new data to the Dijkstra algorithm to work on. Now imagine that the link goes down periodically, thus creating a “flapping route.” In a situation in which both designated and backup routers’ links go down, router elections will take place: more packets, more IVs. Distance vector protocols like RIP and IGRP aren’t any better; not only do they constantly generate volumes of wireless network traffic, but should the link go down, a flood of triggered updates will begin. These examples demonstrate that wireless DoS attacks (both first and second OSI layer) are not just a mere annoyance or possible man-in-the-middle attack sidekicks, but can constitute part of a greater network intrusion plan involving accelerating the shared WEP key disclosure.

Cracking TKIP: The New Menace

As you will see in the following Defense chapters, 802.11i TKIP eliminates the vulnerabilities of WEP we have described and is considered to be practically uncrackable, or is it? When the TKIP keys are generated,





Chapter 8 • Breaking Through

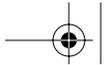
distributed, and rotated using 802.1x and RADIUS, a cracker won't get far trying to crack the keys. Instead, he or she will probably choose a more lateral approach, trying to attack the 801.1x itself. However, if 802.1x cannot be used, a preshared key (PSK) will substitute it as a key establishment method. Although each client host can have its own PSK, at the moment the only real-world implementation of the PSK available is a single PSK per ESSID, just like WEP was. However, the PSK is not used to encrypt data like WEP. Instead, it is employed to generate pairwise transient keys (PTK) for each TKIP-protected connection. These keys are distributed by a four-way handshake and, apart from the PSK, use two nonces from the two first packets of the handshake and two MAC addresses of the involved hosts. Because the handshake packets and the MAC addresses are easy to sniff out, once you know the PSK, you can easily produce all the PTKs you need and the network is yours to take. As usual, the handshake can be initiated by a DoS attack deassociating a client host from the AP. This already eliminates the advantage of TKIP preventing the "nosey employee attack" (users on the same WLAN sniffing each other's traffic). Such an attack can be mitigated by users not knowing the PSK, which creates additional load on the system administrator, who is now also responsible for entering the key on every user's box.

But can an outside attacker obtain the PSK and take over the WLAN? With some luck he or she can. In a four-way handshake, the PTK is used to hash the frames. Because we know both nonces and both MACs, all we need to derive the PSK from the PTK is to crack the hash. Offline hash cracking is neither new nor hard to perform. We deal with it in this chapter, too, in a section devoted to attacks against EAP-LEAP. A PSK is 256 bits long; this is a significantly large number. Although this is great from the cryptographic point of view, no user would ever remember or easily enter a password string that long. Thus, the PSK is generated from an ASCII passphrase in accordance with the following formula:

$$\text{PMK} = \text{PBKDF2}(\text{passphrase}, \text{ssid}, \text{ssidLength}, 4096, 256)$$

where PBKDF2 is a cryptographic method from the PKCS #5 v2.0 Password-based Cryptography Standard. In a nutshell, the string of the passphrase, the ESSID, and its length are hashed 4,096 times to generate a 256-bit key value. Interestingly, neither the length of the passphrase nor the length of the ESSID has a significant impact on the speed of hashing. As stated in the 802.11i standard, a typical passphrase has approximately 2.5 security bits per single character. The n bits passphrase should produce a key with $2.5 * n + 12$ security bits. In accordance with





COMP NOTE: Please note that RH is too long

The Frame of Deception: Wireless Man-in-the-Middle Attacks and Rogue Access

this formula (and the 802.11i standard), a key generated from a passphrase less than 20 characters in length is not sufficiently secure and can be cracked. Just how many users (or even system administrators) usually choose and remember passwords of 20 characters or more?

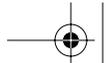
The practical attack against PSK-using TKIP would resemble an offline WEP cracking with WEPattack. The handshake frames capture can be done after deassociating a wireless host by one of the DoS attacks described in this chapter. Robert Moskowitz, who proposed this attack, considers it to be easier to execute than, for example, brute-forcing or running dictionary attacks against WEP. Although no ready tool to perform the offline TKIP cracking exists at the moment of writing, the bounty is too high and most likely by the time you buy this book, the cracking underground will come up with one. After all, we are talking about a hash-cracking tool similar to `md5crack` and a shell script to send deassociate frames and capture the handshake afterward to provide the feed for a hash cracker. Similar functionality is already implemented in a wireless attack tool, namely the `Asleep-imp`.

What would be the impact of such an attack? The wireless networks that do not use 802.1x for TKIP keys distribution and rotation are primarily the networks lacking a RADIUS server due to installation difficulties, price, or other reasons. The networks using legacy wireless hardware and firmware incapable of handling 802.1x also fall into this category. This means that SOHO networks and public hotspots (mind the users bringing “ancient” unupdated client cards) are the networks expected to be susceptible to offline TKIP cracking attacks. These are precisely the kind of networks on which users and administrators are likely to set simple, easy-to-crack passwords that can be found in a modest dictionary. This is clearly a case of Murphy’s Law at work.

The Frame of Deception: Wireless Man-in-the-Middle Attacks and Rogue Access Points Deployment

Our next stop is wireless man-in-the-middle attacks. The first question you might have is why we need man-in-the-middle attacks on 802.11 LANs at all. On the switched wired networks, man-in-the-middle attacks are frequently used to allow the possibility of traffic sniffing. 802.11 LANs are shared medium networks by definition, and once





Chapter 8 • Breaking Through

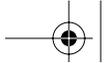
you've dealt with the encryption (if present) you can sniff all the packets on the LAN even without being connected to it. We have already answered this question when describing Dsniff utilities: The answer is connection hijacking and traffic injection. Positioning yourself between two wireless hosts gives an unmatched opportunity to inject commands and even malware into the traffic streams between both hosts. Becoming a rogue access point or wireless bridge means there are far more than two hosts to target with the connection hijacking or traffic injection and modification tools we review in the next chapter.

A specific implication of man-in-the-middle attacks is providing a rogue access point to attack one-way 802.1x authentication systems that use EAP-MD5. To perform such an attack, your rogue AP will also have to be a rogue RADIUS server providing fake credentials in the form of always positive authentication reply to the deceived client hosts. As you will see later, setting both a rogue access point and a RADIUS server on a laptop is not as difficult as you might think. However, such an attack would have a limited use, because the current 802.1x solutions support mutual (client-to-server and server-to-client) authentication and will use EAP-MD5 as a fallback solution only.

Wired man-in-the-middle attacks can be performed using DNS spoofing, ARP cache poisoning, or sneaking into the switch room and changing some cable plug-in positions (a la Kevin Style). Wireless man-in-the-middle attacks are akin to the latter case, but you can be miles away from the switch room. Man-in-the-middle attacks on WLANs can occur on both the first and second OSI layers. Layer 1 man-in-the-middle attacks refer to jamming an existing wireless AP while providing your own clear signal AP at least five channels away from the attacked AP channel. The jamming can be performed using a specific jamming device or by flooding the AP channel with junk traffic (e.g., using FakeAP, Void11 or File2air). If a jamming device is used, the defending side will need a decent frequency analyzer to detect the jamming attack; traditional wireless IDS won't help.

Of course, the parameters of your rogue AP (ESSID, WEP, MAC) should reflect the parameters of the legitimate access point. Layer 2 attacks differ by using a spoofed deassociation or deauthentication frames flood to kick the target host from its link with a legitimate AP. This is generally more efficient than the channel jamming. A determined attacker can easily combine both Layer 1 and Layer 2 attacks to reach the maximum effect. The majority of modern client cards will detect the new rogue AP on a channel different from the one they currently use and





The Frame of Deception: Wireless Man-in-the-Middle Attacks and Rogue Access

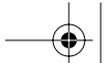
automatically associate with it if the association with the legitimate AP has been made hard or impossible. However, if the clients are preset to work at the specific frequency only, the chances of a successful man-in-the-middle attack are dramatically decreased because the attack will depend on outspoofing or overpowering the legitimate AP on the channel it runs. Such an attempt is likely to end up as a DoS attack due to the RF interference.

When launching man-in-the-middle attacks, you don't have to pose as an access point in all cases; sometimes an attacker might want to knock off a selected client host and substitute his or her machine as that host to the access point and the rest of the network. This task is significantly easier: A client host is likely to have lower EIRP, so you don't have to set your host as an access point (emulating the attacked host's IP and MAC is enough) and a quick man-in-the-middle attack against a single host is less likely to cause user complaints and disturbance in the logs. Besides, you can be closer to the victim machine than you are to the access point.

DIY: Rogue Access Points and Wireless Bridges for Penetration Testing

Many wireless security literature sources depict wireless man-in-the-middle attackers as people carrying hardware access points and accumulator batteries around. Frankly, this is ridiculous and makes it sound more like a van-in-the-middle attack. How long would you be able to wander around with a heavy battery, an access point, a laptop, cables, and antennas? Also, it is much easier to hijack connections and inject data if you do it on one of the hijacking machine network interfaces rather than force a hardware access point in a repeater mode to route all traffic through the Ethernet-connected attacking host (how would you do it in reality?). Thus, the optimal solution is to set a software-based access point on a client card plugged into the attacker's laptop (or even PDA). A second plugged-in card can be used as a jamming/frame-generating device to bring down a legitimate AP. Both cards might have to run using different drivers or at least be produced by different vendors to provide proper functionality separation. Several variations of the attack exist, such as using two bridged access point-enabled client cards or using two laptops instead of one, with the obvious functionality of one being used as an access point and another as a DoS-launching platform.





Chapter 8 • Breaking Through

The access point functionality can be set using the following:

- HostAP and Prism54g on Linux (Prism chipset cards)
- HermesAP drivers on Linux (Hermes chipset cards)
- Patched Orinoco driver + `monkey_jack` on Linux (Hermes chipset cards)
- `Ifconfig mediaopts hostap paramater` or WiFi BSD drivers on FreeBSD (Prism chipset cards)
- `wicontrol mediaopt hostap paramater` on Open and NetBSD (Prism chipset cards)
- ZoomAir Access Point software on Windows 95/98/NT/2000 (ZoomAir cards only, these cards have a Prism chipset)

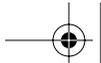
Our discussion will be mainly devoted to Linux-based access points, because we had more play time with them. There is nothing wrong with using BSD-based APs in wireless security auditing. A Windows-based ZoomAir access point is easy to set up, but offers limited functionality, and there are hardly any decent hijacking or traffic injection tools for the Microsoft platform.

The easiest way to launch a man-in-the-middle attack is by using the `monkey_jack` utility provided with AirJack, assuming your AirJack compilation and configuration went well as we described in Chapter 5:

```
arhontus:~# ./monkey_jack
Monkey Jack: Wireless 802.11(b) MITM proof of concept.
Usage: ./monkey_jack -b <bssid> -v <victim mac> -C <channel
number> [ -c <channel number> ] [ -i <interface name> ] [ -I
<interface name> ] [ -e <ssid> ]
-a: number of disassociation frames to send (defaults to 7)
-t: number of deauthentication frames to send (defaults
to 0)
-b: bssid, the mac address of the access point (e.g.
00:de:ad:be:ef:00)
-v: victim mac address.
-c: channel number (1-14) that the access point is on,
defaults to current.
-C: channel number (1-14) that we're going to move them to.
-i: the name of the AirJack interface to use (defaults to
aj0).
-I: the name of the interface to use (defaults to eth1).
-e: the ssid of the AP.
```

Supply all the necessary parameters, press Enter, and see your host's Hermes/Orinoco chipset card being inserted between the target host on





The Frame of Deception: Wireless Man-in-the-Middle Attacks and Rogue Access

the WLAN and the access point. To amplify the attack on the first layer, use the highest EIRP you can reach with your cards and available antennas on both flooding and the AP cards. Try `-v FF:FF:FF:FF:FF:FF` for a weapon of mass deception.

Alternatively you can set an access point employing two Prism chipset cards and `hostap` drivers and use `FakeAP` as a channel flooding tool on one of the cards, while the second card runs in a Master mode (AP). Flooding a channel with beacons is not as efficient as sending deauthentication frames, so you might opt for combining one card running under `HostAP` and one using `airjack_cs`. To do the latter, edit the `/etc/pcmcia/config` file and bind one card to the `"hostap_cs"` and another to `"airjack_cs"` modules. Restart the PCMCIA services, insert both cards, and go. Use `wlan_jack` or `fata_jack` to deassociate hosts from the network AP. Alternatively, you can stick to `HostAP` drivers only, install `Libradiate`, and use `omerta` to generate deassociation frames sent by one of the cards. Even better, you can strike with `Void11` using an opportunity to deauthenticate multiple hosts, run concurrent floods, or even try to take down the legitimate access point with authentication or association frames bombardment. The choice is yours.

Installing and setting `HostAP` drivers is very easy. Grab the latest version of `HostAP` from the CVS at <http://hostap.epitest.fi/>, do `make` && `make_pccard` as root (we assume you use a PCMCIA client card), restart the PCMCIA services, and insert your card. You should see something like this:

```

arhontus:~# lsmod

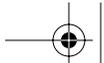
Module                Size  Used by    Tainted: P
hostap_cs             42408  0 (unused)
hostap                61028  0 [hostap_cs]
hostap_crypt          1392   0 [hostap]

arhontus:~# iwconfig
wlan0      IEEE 802.11b  ESSID:"test"
Mode:Master Frequency:2.422GHz Access Point: 00:02:6F:01:ab:cd
Bit Rate:11Mb/s Tx-Power:-12 dBm Sensitivity=1/3
Retry min limit:8 RTS thr:off Fragment thr:off
Encryption key:off
Power Management:off
Link Quality:0 Signal level:0 Noise level:0
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:0 Invalid misc:425 Missed beacon:0

```

The card automatically runs in the access point (Master) mode with the default ESSID `"test."` Note that if you insert a Hermes chipset card, it





Chapter 8 • Breaking Through

will work with `hostap_cs`, but you cannot place it into the Master or Repeater modes, the interface is `eth1`, and the default ESSID is blank. To change the card modes use `iwconfig <interface> mode ad-hoc || managed || master || repeater || secondary || monitor`. Read the fine manpages to learn more about the modes supported. Try the Repeater mode with HostAP and Prism chipset card to insert a rogue repeater into the testing wireless network as another man-in-the-middle attack possibility:

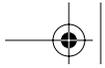
```
arhontus:~# iwconfig wlan0 channel 1 txpower 100mW mode repeater
essid Sly
arhontus:~# iwconfig wlan0
wlan0 IEEE 802.11b ESSID:"Sly"
Mode:Repeater Frequency:2.412GHz Access Point:
00:00:00:00:00:00
      Bit Rate:2Mb/s Tx-Power=20 dBm Sensitivity=1/3
      Retry min limit:8 RTS thr:off Fragment thr:off
```

Another similar and rather fanciful thing to try is inserting a double card wireless bridge into a point-to-point link (a true man-in-the-middle attack, because the best position for the attacker would be right between the endpoints, in the middle of the Fresnel zone). For this attack you'll need to have bridging and 802.11d (if you want to use the Spanning Tree Protocol, or STP) support enabled in the Linux kernel and bridging tools (<http://bridge.sourceforge.net/>) installed. Setting a wireless bridge is similar to setting a wireless distribution system (WDS), but you'll have to use another wireless interface on a second card instead of the usual wired interface:

```
iwpriv wlan0 wds_add 00:22:22:22:22:22
brctl addbr br0
brctl addif br0 wlan1
brctl addif br0 wlan0
brctl addif br0 wlan0wds0
ifconfig wlan1 0.0.0.0
ifconfig wlan0 0.0.0.0
ifconfig wlan0wds0 0.0.0.0
ifconfig br0 <insert IP here> up
```

Then the bridge can be set to participate in the STP process and add new distribution links automatically. To accomplish the latter, the command `prism2_param wlan0 autom_ap_wds 1` is used. As the `README.prism2` file outlines, you can use several commands to check the operation of your bridge:





The Frame of Deception: Wireless Man-in-the-Middle Attacks and Rogue Access

'brctl show' should show br0 bridge with the added interfaces and STP protocol enabled.

'brctl showstp br0' should show more statistics about each bridge port. The state' parameter should show 'learning' for a few seconds and change to 'forwarding' afterward.

'brctl showmacs br0' can be used to check behind which bridge port each known MAC address is currently allocated.

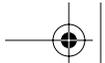
Now you probably want to become a root bridge on the STP network. Run Ettercap on one of the wireless interfaces, go to the plug-ins selection ("p/P") and select the plug-in lamia. The priority value for the root bridge should be as low as possible—select zero. You might also need to set your MAC address to a lower value in case there is another bridge with a zero priority. When a tie based on a priority value takes place, the lower MAC wins.

Imagine the amount of traffic you will get through on a busy wireless network using such a bridge!

If you only have a Hermes/Orinoco chipset card (we strongly recommend that you have three different chipset cards [Cisco Aironet, Prism, and Hermes] for proper wireless security testing), you can use HermesAP (<http://www.hunz.org/hermesap.html>) to set a software-based access point. HermesAP is much younger than HostAP and lacks many of the features of HostAP, but it is catching up. Installing HermesAP is more complicated than setting up HostAP because both the Hermes card firmware update and orinoco driver/pcmcia-cs patching are required; see the README file (<http://www.hunz.org/README>). Once set, HermesAP is configurable via Linux Wireless Extensions, and supports WDS, RFMON, and closed ESSIDs. Because we don't know how to generate traffic (other than beacons) with HermesAP, we do not review it any further in the man-in-the-middle attacks discussion. Nevertheless, HermesAP is a very interesting project and we hope that this paragraph will spark more interest in its development and attract more hackers on its side.

Finally, on the BSD side you can set an access point functionality with a command like `wicontrol -n foobared -p 6 -f 6 -e 0` (this is an OpenBSD example, as we are going to use Wnet later; `-p 6` stands for hostap mode, `-f` sets channel, `-e 0` means WEP is not required to associate). The interface set to act as an access point can then be employed to bombard the network with deassociation and deauthentication frames (Wnet `dinject`) telling the defenseless hosts to disconnect from the current access point. Yes, this means that under OpenBSD you might not





Chapter 8 • Breaking Through

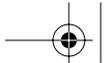
need a second card to perform an efficient man-in-the-middle attack, thus saving some configuration time and a lot of battery power. You will probably need to write a small shell script to make `dinject` tools send multiple deauthenticate or deassociate frames for a successful DoS attack. Also, don't forget that you are limited to Prism chipset cards only.

Hit or Miss: Physical Layer Man-in-the-Middle Attacks

To conclude the man-in-the-middle attack section, we would like to share some thoughts on Layer 1 attack attempts. On a physical layer there are two possible avenues reinforcing a chance of a successful man-in-the-middle assault:

1. Network management is restricted by the legal FCC, ETSI, or equivalent EIRP output regulations. At the same time, the attackers do not care about these restrictions (when an attack is launched the law is broken anyway) and can easily surpass all legal power output limits imposed. For instance, a cracker can use a powerful 23 dBm (200 mW) PCMCIA client card with a decent gain antenna (e.g., 24 dBm dish or grid directional). The EIRP would reach about 45 dBm (subtract 2–3 dBm for the obvious connectors and pigtail loss), which equals about 31.62 W of output. Such output is much higher than the legally permitted 1 W point-to-multipoint wireless LAN EIRP and should be significantly higher than the allowed EIRP on the majority of point-to-point wireless links deployed.
2. 802.11 hosts are supposed to associate with a wireless access point on the basis of basic error ratio (BER). In practical terms, it comes down to the signal strength and SNR ratio, assuming all other parameters such as ESSID and WEP key are correct. Theoretically, introducing the rogue access point with a very high EIRP as described earlier should be able to force the hosts on a WLAN to associate with the rogue and not the legitimate AP. The reality is not that simple, as many wireless clients tend to reassociate with the AP they were associated with before and will only change the frequency to a different one in case of a very powerful RF noise flood hitting the used channel. These association choice features are usually built into the card's firmware. In several cases, such as the AirPort client card configuration under Mac OS X, it is possible to configure manually whether





The Frame of Deception: Wireless Man-in-the-Middle Attacks and Rogue Access

the host will join the AP with the highest SNR or stick with the most recently associated access point. Of course, roaming WLANs are at greater danger from physical layer man-in-the-middle attacks, because roaming hosts should associate on the basis of AP signal strength. Nevertheless, for the reasons outlined earlier, Layer 1 man-in-the-middle wireless attacks are rather unreliable and should be supplementary to the data link attacks employing targeted deassociation and deauthentication frame floods.

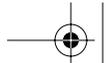
Phishing in the Air: Man-in-the-Middle Attacks Combined

A man-in-the-middle attack does not have to be limited to a single layer. Just like the defense-in-depth would cover all seven layers of the OSI model, so can the attack-in-depth, efficiently sneaking under and over the safeguards deployed. Consider the possible disadvantages of the Layer 1 man-in-the-middle attack we have discussed. Nevertheless, if both Layer 1 and Layer 2 attacks are combined, the outcome is almost certain. Not only do you deassociate the hosts from the network AP to lure them to yours, you also overpower the AP, making sure that your rogue AP is preferred. At the same time, you can flood the legitimate AP channel with noise.

This is not hard to accomplish. For example, you can combine the HostAP Master mode (the rogue AP ≥ 5 channels away) with FakeAP (generating noise on the network AP channel) and Void11 (single or mass host deassociation). If EAP-MD5 is used on the network, you can add the hostapd authenticator and authentication server functionality to trick the connecting hosts into an association with your rogue AP and obtain the password. In a few pages, we review this attack in more detail. Finally, if higher layer security protocols such as SSH or SSL are involved, you can add man-in-the-middle attacks against these protocols to the combined Layers 1 and 2 man-in-the-middle attack for the full efficiency.

An interesting and rather specific case is when the wireless access point or authentication server uses Web-based user authentication, as commonly done by wireless hotspots. This can be performed using NoCat (see Chapter 13) or by employing various proprietary hotspot user authentication solutions. In such a case, the appearance of the user login Web page defines the trust. Once you can fake the page, the



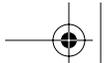


Chapter 8 • Breaking Through

unsuspecting users would happily log in and enter their credentials, only to be told later that “a network error has occurred and the connection was lost.” Even better, a sequence of other Web pages can be faked to present the target with common login pages (e.g., eBay, Paypal, Hotmail) for more credentials to grab. A suite to abuse users’ trust in such a sneaky way is called Airsnarf. It doesn’t matter if the connection uses SSL or PGP keys a la NoCat, the end users won’t know it and some of them will inevitably associate with the rogue AP and enter their credentials. The question is how many of them. Airsnarf, as presented first at Defcon 11, uses Layer 1 overpowering to overcome the legitimate network AP. This, of course, brings in all the previously discussed problems of Layer 1 man-in-the-middle attacks. What if the clients are set to use a specific channel? What if the interference is too strong? What if the rogue AP is PDA-based and uses a casual built-in antenna in a CF client card, whereas the AP under attack has a high IR value and is connected to a high gain antenna via an amplifier?

This is exactly the case when combining a Layer 1 and Layer 2 attack is necessary for success. The Airsnarf + HostAP + Void11 + FakeAP combination immediately comes to mind. In fact, a determined attacker can also try to shut the legitimate access point down at the same time. This can be done using other instances of Void11, hammering the AP with authentication and association frame floods. If the attacker can associate with the hotspot or is an already associated rogue user, he or she can launch higher layer DoS attacks to disable the network AP first. Such attacks can be SNMP-based (how many users or “administrators” don’t change the default community names?) or employ more traditional DoS attacks, such as SYN flooding. We found out that many commonly deployed access points have problems dealing with intensive traffic using large packets and can be knocked out by `ping -s 65507 -f` or similar actions. At the same time the rogue AP, perhaps a Zaurus PDA in the attacker’s pocket using Airsnarf from an `ipkg` package, will entrap unsuspecting users and snatch their user names and passwords. This underlines the necessity of profound AP testing for resistance to various common higher layer attacks as well as known Layer 2 wireless threats before the production cycle starts. If, in the process of a security audit, a penetration tester can crash or freeze the AP, too bad. This isn’t just a DoS attack; it signifies an additional vulnerability of every host on the tested WLAN to the man-in-the-middle menace. To reduce this particular threat, make sure that any kind of AP management from the wireless side is turned off completely and no open AP ports are presented to the users on the WLAN.





Breaking the Secure Safe

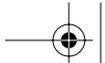
The final barriers you might need to bypass to associate with the wireless network are 802.1x-based authentication and higher layer VPNs. Attacking 802.1x and VPNs requires prior knowledge of the involved protocol's structure and operation. We strongly suggest reading Chapters 10 and 13 to learn more about 802.1x/EAP and Chapter 14 to review common VPN protocols before trying to understand the attacks we describe here.

Crashing the Doors: Authentication Systems Attacks

If the 802.1x implementation protecting the attacked network is using EAP-TLS, EAP-TTLS, or EAP-PEAP (reviewed in the Defense part of the book), the cracker might be out of luck and have to resort to DoS, social engineering, or wired side attacks against the certificate server or authority. There are theoretical investigations into possible man-in-the-middle attacks against tunneled authentication protocols—see “The Compound Authentication Binding Problem” IETF draft at <http://www.ietf.org/internet-drafts/draft-puthenkulam-eap-binding-02.txt>. Only time will tell if practical implementations of such attacks will come into existence. In a few cases, EAP-TTLS might be set to use older authentication methods such as MS-CHAP. These methods are vulnerable to an attack should the attacker manage to insert himself or herself into the tunnel.

An improved dictionary attack or plain old brute-force approach can be taken against Cisco EAP-LEAP because it employs user passwords, not host certificates. The EAP-LEAP dictionary attack improvement, first presented by Joshua Wright at Defcon 11, represents a formidable threat to WLANs that depend on LEAP security features. The main principle behind the attack is EAP-LEAP using MS-CHAPv2 in the clear to authenticate users. Thus, it inherits several MS-CHAPv2 flaws including plaintext user names transmission, weak challenge/response DES key selection, and an absence of salt in the stored NT hashes. Let us take a closer look at how the LEAP challenge/response operates. First, the authenticator (access point) sends a random 8-bit challenge to the supplicant (client host). The supplicant uses an MD4 hash of the authentication password to generate three different DES keys. Each of these keys is used to encrypt the challenge received and the ciphertext ($3 \times 64 = 192$





Chapter 8 • Breaking Through

bits in total) is sent back to the authenticator as a response. The authenticator checks the response and issues an authentication success or failure frame back to the supplicant, depending on the result.

Unfortunately, five nulls are consistent in every LEAP challenge/response exchange, making the third DES key weak. Because the challenge is known, calculating the remaining two DES keys takes less than a second. The trouble is that the third flawed DES key allows calculating the last two bits of the NT hash, leaving only 6 bytes to brute-force or run against a dictionary. That should not be difficult, because MD4 is fast, resource-economical, and insecure.

The attack against EAP-LEAP implemented by Joshua Wright in his *Asleap-imp* tool is as follows:

- Calculate a large list of MD4-hashed passwords.
- Capture EAP-LEAP challenge/response frames.
- Extract challenge, response, and username.
- Use the response to calculate the last two bits of the MD4 hash.
- Run the dictionary attack against the hash taking the two known last bits into account.

Another tool that uses the same attack against EAP-LEAP and was posted to the public domain is *Leap*. Check out the detailed description of *leapcrack*, *leap*, and *Asleap-imp* use in Chapter 6.

EAP-MD5, the original (and fallback) implementation of EAP, is vulnerable to man-in-the-middle attacks against the AP because there is no AP/server-to-host authentication. A rogue access point placed between the EAP-MD5 supplicant and the RADIUS server can easily snatch the user credentials sent to the authentication server and even authenticate users employing false credentials. To perform such an attack, the cracker might install RADIUS on the rogue AP host and direct user traffic to this illicit RADIUS server. An alternative path is to employ the *HostAP* *hostapd* daemon-supported minimal coallocated authentication server. This server requests the identity of the wireless client and will authorize any host capable of sending a valid EAP Response frame. No keys are required and any client can authenticate. This is not the functionality you would like to employ in a real-world access point, but for a man-in-the-middle attack in the process of penetration testing it is really what the doctor ordered. To start *hostapd* with the authentication server capability, use the `hostapd -xm wlan0` command. When the *hostapd* authentication server is enabled, clients not supporting 802.1x will not be able to send data frames through the rogue AP.





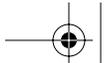
Breaking the Secure Safe

Finally, there is a whole spectrum of DoS attacks against various implementations of EAP:

- *DoS attacks based on flooding with EAPOL-Start frames.* A cracker can try to crash the access point by flooding it with EAPOL-Start frames. The way to avoid this attack is to allocate limited resources on receipt of an EAPOL-Start frame.
- *DoS attacks based on cycling through the EAP Identifier space.* A cracker can bring down the access point by consuming all EAP Identifier space (0–255). As the EAP Identifier is required to be unique within a single 802.1x port only, there is no reason for an AP to lock out further connections once the Identifier space has been exhausted. Nevertheless, some access points do just that.
- *DoS attacks against clients based on sending premature EAP Success frames.* The IEEE 802.1x standard enables a client to avoid bringing up its interface if the required mutual authentication is not completed. This allows a well-implemented supplicant to avoid being tricked by a rogue authenticator AP flooding with premature EAP Success frames.
- *DoS attacks against clients based on spoofing EAP Failure frames.* The EAP specification requires supplicant clients to be able to use alternative indications of successful or failed 802.1x authentication. Thus, a well-implemented supplicant should not be fooled by a cracker flooding the network with EAP Failure frames. A supplicant that receives EAP-Failure frames from a rogue authenticator outside of the legal 802.1x exchange should ignore the frames. Not all supplicant clients possess such capability. If the proper authenticator AP wishes to remove the supplicant client, it would follow the EAP failure by the deassociation frame. There is nothing to stop attackers from imitating such a situation. File2air is the current tool of choice to launch such attacks.
- *DoS attacks using malformed EAP frames.* An example of such an attack is a FreeRADIUS 0.8.1 crash caused by an EAP TLS packet with flags 'c0' and with no TLS message length or TLS message data. This attack was reported at <http://www.mail-archive.com/freeradius-users@lists.cistron.nl/msg15451.html>.

How about practical implementations of these attacks? Unfortunately, there is no Nemesis or Wnet-style custom frame-generation toolkit for 802.1x/EAP at the time of writing. As mentioned earlier, you can always try to create your EAP frames in binary and send them using File2air.





Chapter 8 • Breaking Through

Besides, QA Cafe has released a commercial EAP-testing Linux suite they call EAPOL (<http://www.qacafe.com/eapol/>). You can only run EAPOL using Cisco Aironet 350 cards. A demo version of the suite, which includes binaries for Red Hat and Debian distribution, is available for download from the QA Cafe Web site. Here is the description of all tests supplied by the demo version of EAPOL as stated at <http://www.qacafe.com/eapol/test-summary-demo.htm#4>:

Authenticator sends EAPOL packets to supplicant's unicast MAC address:

Description:

- step 1. Send EAPOL-Logoff to place controlled port in unauthorized state
- step 2. Send EAP-Start to initiate authentication
- step 2. Wait for EAPOL packet from Authenticator (up to txWhen seconds)
- step 3. Verify destination MAC address is supplicant's MAC address

Reference: IEEE Std 802.1X-2001
Section 7.8 EAPOL Addressing

NOTE: The authenticator should be in the CONNECTING state after the EAPOL-Logoff/EAPOL-Start packets are sent by the supplicant.

Basic case of authenticator-initiated authentication:

Description:

- step 1. Send EAPOL-Logoff to place controlled port in unauthorized state
- step 2. Initiate ICMP Ping on LAN port to Trusted host
- step 3. Continue ping attempts for 120 seconds
- step 4. Verify authentication occurs for the configured type
- step 5. Verify ICMP ping to Trusted host

Reference: IEEE Std 802.1X-2001
Section 8.4.2.1 Authenticator initiation

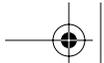
Basic case of supplicant-initiated authentication:

Description:

- step 1. Send EAPOL-Logoff to place controlled port in unauthorized state
- step 2. Send EAPOL-Start to initiate authentication process
- step 3. Verify authentication occurs for the configured type
- step 4. Verify ICMP ping to trusted host

Reference: IEEE Std 802.1X-2001
Section 8.4.2.2 Supplicant initiation





Breaking the Secure Safe

Authenticator sends EAP Failure after supplicant sends EAP-Logoff:

Description:

- step 1. Send EAPOL-Logoff to place controlled port in unauthorized state
- step 2. Wait up to 15 seconds for EAP Failure packet from Authenticator

Reference: IEEE Std 802.1X-2001
Section 8.5.4.4 Disconnected

Authenticator sends EAP Failure if identity is unknown:

Description:

- step 1. Configure the supplicant to use unknown identity
- step 2. Send EAP-Start
- step 3. Wait for EAP Identity request
- step 4. Respond with unknown identity
- step 5. Verify an EAP Failure is received

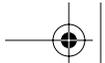
Reference: IEEE Std 802.1X-2001
Section 8.5.8.6 FAIL

NOTE: This test uses the Identity 'badUserName' which must not be a valid user name on your Backend authentication server.

The test summary of the full EAPOL suite gives an idea of how many possible DoS attacks against the EAP do exist. The summary can be viewed at <http://www.qacafe.com/eapol/test-summary.htm>. The EAPOL setup for wireless 802.1x authentication testing needs a Linux machine with one Ethernet and wireless interface. One interface of the EAPOL-running host is the 802.1x supplicant interface connected to the authenticator device (access point). The second interface must be connected to the trusted part of the device (access point Ethernet port) or network that does not require 802.1x authentication (wired LAN into which the tested AP is plugged). EAPOL is a lab testing suite for wireless security software and protocol developers, beta testers, and security consultants, not a canned "script kiddie" DoS tool. However, because the information about attacks exists "in the wild," we expect that hacked-up Xsupplicant clients and HostAP-based authenticators implementing the attacks described are under development in the hacker community and will surface soon.

To summarize, the main problem of EAP frames is the same with the 802.11 management and control frames: lack of proper authentication and integrity protection (secure checksums).





Chapter 8 • Breaking Through

Tapping the Tunnels: Attacks Against VPNs

Attacks on higher layer VPNs is hardly a wireless-specific topic that surely deserves a book of its own. Here we can only provide some directions for a security professional or enthusiast to follow in his or her future research into it. Point-to-Point Tunneling Protocol (PPTP) and various IPsec implementations are the most common VPN solutions encountered. PPTP took a heavy battering from the security community and multiple tools have built-in options to attack PPTP tunnels. Anger is one such tool:

```
arhontus:~# ./anger -- h
```

```
usage: anger [ -v ] [ -d device ] output1 [ output2 ]
```

```
Write sniffed challenge/responses to output1.
```

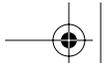
```
If output2 is given it will perform an active attack on  
PPTP connections and write the password hashes to output2.
```

```
-d Device to open for sniffing.  
-v Some diagnostics.
```

As the documentation packaged with the tool states, Anger is a PPTP sniffer and attack utility. It sniffs PPTP MS-CHAP challenge/response packets and outputs them in a format suitable for feeding to the infamous L0phtcrack password cracking program. Anger implements an active attack against the MS-CHAPv1 password change protocol. When the sniffer detects a PPTP client attempting to log in using MS-CHAPv1, it fakes a password change command from the server. If the deceived user follows the dialog to change his or her password, Anger logs the hashes of the current password as well as the hashes of the new password chosen. These hashes can be given to L0phtcrack to crack the password or be used with a hacked-up PPP client for use with the Linux PPTP client to log onto the network. There are other utilities implementing the PPTP password change attack besides Anger, such as deceit by Aleph One (<http://packetstormsecurity.nl/new-exploits/deceit.c>).

After the publication and exploitation of flaws in the MS-CHAP protocol, Microsoft released a new version of MS-CHAP. This new version is not vulnerable to the password change attack. It does not perform a challenge/response authentication based on the weak LM hashes, and possesses the capability of server authentication. Microsoft has added a number of new steps to the response-to-challenge generation and implemented SHA1 hashing. However, the sniffer can still precompute





Breaking the Secure Safe

hashes, and L0phtcrack does not require any changes to handle MS-CHAPv2 cracking.

The latest versions of Anger support sniffing MS-CHAPv2 challenge/response packets. The outlines for MS-CHAPv2 have the LM hash set to all zeros, as it is not available. Unfortunately, it is not possible to use the command-line version of L0phtcrack to crack MS-CHAPv2 entries because it does not attempt to get the NT response via a dictionary attack, unless there is an LM response present. However, you can use the Windows GUI version of L0phtcrack to crack the MS-CHAPv2 entries. In such a case, you must disable the cracking of the LM hash and enable cracking of the NT hash in the L0phtcrack options panel because L0phtcrack will not recognize the all-zeros LM response field as invalid and will still try to crack it. Replacing this field with something else leads to a parsing error.

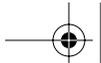
Ettercap possesses a whole collection of plug-ins written to sniff PPTP tunnels, decapsulate traffic, and get user log-in passwords:

```
H03_hydra1 1.1 -- PPTP: Gets the passwords
H04_hydra2 1.0 -- PPTP: Decapsulates connections
H05_hydra3 1.0 -- PPTP: Forces renegotiation
H06_hydra4 1.0 -- PPTP: Forces PAP authentication
H07_hydra5 1.0 -- PPTP: Tries to force cleartext
H08_hydra6 1.0 -- PPTP: Forces chapms from chapmsv2
```

If you use PPTP on your WLAN, you should know how disruptive these plug-ins can be if PPTP is the only or best defensive measure standing between the cracker and WLAN traffic. If your interest in PPTP security lies beyond trying a few of the underground attack tools available, recommended reading includes “Cryptanalysis of Microsoft’s PPTP Authentication Extensions (MS-CHAPv2)” by Bruce Schneier and Dr. Mudge (<http://www.counterpane.com/pptpv2-paper.html>) and a follow-up to this paper published by Team Teso (<http://www.team-teso.net/releases/chap.pdf>).

The main prerequisite to attacking IPsec VPNs is understanding how IPsec works. Without such an understanding, the discussion here makes little sense. Skip to Chapter 14 to learn more about the workings of the IPsec protocol and you will see that actually there is no such thing as an attack against IPsec; there are only attacks against specific IPsec modes or implementations. IPsec implementations that have known security





Chapter 8 • Breaking Through

problems such as buffer overflows or man-in-the-middle attack susceptibilities include the following:

- Cisco VPN Client 3.5
- Cisco VPN Client 1.1
- SafeNet/IRE SoftPK and SoftRemote
- PGPFreeware 7.03 - PGPNet
- WAVEsec

To poke around the IPSec-protected LAN use `IKEProber.pl` by Anton T. Rager or `Ike-scan` by Roy Hills. `IKEProber` and `Ike-scan` are Internet Key Exchange (IKE) packet manglers written to discover and fingerprint IKE-running hosts. The command syntaxes of both tools is as follows:

```
arhontus:~# ./ike-scan -h
Usage: ike-scan [options] [hosts...]
Hosts are specified on the command line unless the --file option
is specified.
```

Options:

```
--help or -h    Display this usage message and exit.
```

```
--file=<fn> or -f <fn>
```

Read hostnames or addresses from the specified file instead of from the command line. One name or IP address per line. Use "-" for standard input.

```
--sport=<p> or -s p    Set UDP source port to <p>, default=500,
0=random.
```

Some IKE implementations require the client to use UDP source port 500 and will not talk to other ports. Note that superuser privileges are normally required to use nonzero source ports below 1024. Also only one process on a system may bind to a given source port at any one time.

```
--dport=<p> or -d p
```

Set UDP destination port to <p>, default=500. UDP port 500 is the assigned port number for ISAKMP and this is the port used by most if not all IKE implementations.

```
--retry=<n> or -r n    Set total number of attempts per host to
<n>, default=3.
```

```
--timeout=<n> or -t n
```

Set initial per-host timeout to <n> ms, default=500. This timeout is for the first packet sent to each host. Subsequent



Breaking the Secure Safe

timeouts are multiplied by the backoff factor which is set with backoff.

`--interval=<n> or -i <n>`

Set minimum packet interval to `<n>` ms, default=75. This controls the outgoing bandwidth usage by limiting the rate at which packets can be sent. The packet interval will be greater than or equal to this number and will be a multiple of the select wait specified with `--selectwait`. Thus `--interval=75 --selectwait=10` will result in a packet interval of 80 ms. The outgoing packets have a total size of 364 bytes (20 bytes IP hdr + 8 bytes UDP hdr + 336 bytes data) when the default transform set is used, or bytes if a custom transform is specified. Therefore for default transform set: 50 = 58240bps, 80 = 36400bps and for custom transform: 15 = 59733bps, 30 = 35840bps.

`--backoff= or -b ` Set timeout backoff factor to ``, default=1.50.

The per-host timeout is multiplied by this factor after each timeout. So, if the number of retries is 3, the initial per-host timeout is 500 ms and the backoff factor is 1.5, then the first timeout will be 500 ms, the second 750 ms and the third 1125 ms.

`--selectwait=<n> or -w <n>`

Set select wait to `<n>` ms, default=10. This controls the timeout used in the `select(2)` call. It defines the lower bound and granularity of the packet interval set with `--interval`. Smaller values allow more accurate and lower packet intervals; larger values reduce CPU usage. You don't need to change this unless you want to reduce the packet interval close to or below the default select wait time.

`--verbose or -v`

Display verbose progress messages. Use more than once for greater effect:

- 1 - Show when hosts are removed from the list and when packets with invalid cookies are received.
- 2 - Show each packet sent and received.
- 3 - Display the host and backoff lists before scanning starts.

`--lifetime=<s> or -l <s>`

Set IKE lifetime to `<s>` seconds, default=28800. RFC 2407 specifies 28800 as the default, but some implementations may require different values.

`--auth=<n> or -m <n>`

Set auth. method to `<n>`, default=1 (preshared key). RFC defined values are 1 to 5. See RFC 2409 Appendix A.

`--version or -V`

Display program version and exit.

Chapter 8 • Breaking Through

`--vendor=<v>` or `-e <v>`

Set vendor id string to MD5 hash of <v>. Note: this is currently experimental.

`--trans=<t>` or `-a <t>`

Use custom transform <t> instead of default set. <t> is specified as enc,hash,auth,group.

e.g., 2,3,1,5. See RFC 2409 Appendix A for details of which values to use. For example, `--trans=2,3,1,5` specifies Enc=IDEA-CBC, Hash=Tiger, Auth=shared key, DH Group=5

If this option is specified, then only the single custom transform is used rather than the default set of 8 transforms. As a result, the IP packet size is 112 bytes rather than the default of 364.

`--showbackoff[=<n>]` or `-o[<n>]`

Display the backoff fingerprint table. Display the backoff table to fingerprint the IKE implementation on the remote hosts. The optional argument specifies time to wait in seconds after receiving the last packet, default=60. If you are using the short form of the option (-o) then the value must immediately follow the option letter with no spaces, e.g. `-o25` not `-o 25`.

`--fuzz=<n>` or `-u <n>` Set pattern matching fuzz to <n> ms, default=100.

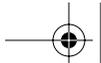
This sets the maximum acceptable difference between the observed backoff times and the reference times in the backoff patterns file. Larger values allow for higher variance but also increase the risk of false positive identifications.

Report bugs or send suggestions to ike-scan@nta-monitor.com
See the ike-scan homepage at <http://www.nta-monitor.com/ike-scan/>

arhontus:~# perl IKEProber.pl
ikeprober.pl V1.13 -- 02/14/2002, updated 9/25/2002
By: Anton T. Rager - arager.com

Usage:

```
-s SA [encr:hash:auth:group]
-k x|auser value|user value [KE repeatedX
  times|ascii_supplied|hex_supplied]
-n x|auser value|user value [Nonce repeatedX
  times|ascii_supplied|hex_supplied]
-v x|auser value|user value [VendorID
  repeatedX|ascii_supplied|hex_supplied]
-i x|auser value|user|rawip value [ID
  repeatedX|ascii_supplied|hex_supplied|Hex_IPV4]
-h x|auser value|user value [Hash
  repeatedX|ascii_supplied|hex_supplied]
-spi xx [SPI in lbyte hex]
-r x [repeat previous payload x times]
```



Breaking the Secure Safe

```
-d ip_address [Create Init packet to dest host]
-eac [Nortel EAC transform - responder only]
-main [main mode packet instead of aggressive mode - logic
      will be added later for correct init/respond]
-sa_test 1|2|3|4 [1=86400sec life, 2=0xffffffff life, 3=192
                group attribs, 4=128 byte TLV attrib]
-rand randomize cookie
-transforms x [repeat SA transform x times]
```

Use these tools to discover vulnerable IPSec implementations on LAN, download appropriate exploit code, compile it, and give it a try.

WAVEsec mobile IPSec implementation is exploitable with `kracker_jack` from the AirJack suite:

```
arhontus:~# ./kracker_jack
Kracker Jack: Wireless 802.11(b) MITM proof of concept (with a
bite).
```

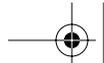
```
Usage: ./kracker_jack -b <bssid> -v <victim mac> -C <channel
number> [ -c <channel number> ]
V <victims ip address> -s <server mac> -S <server ip address>
[ -i <interface name> ] [ -I <interface name> ] [ -e <ssid> ]
n <netmask> -B <broadcast address>
```

```
-a: number of disassociation frames to send (defaults to 7)
-t: number of deauthentication frames to send (defaults to 0)
-b:..bssid, the mac address of the access point (e.g.,
    00:de:ad:be:ef:00)
-v: victim mac address
-V: victim's ip address
-s: wavesec server mac address
-S: wavesec server ip address
-B: network broadcast address
-n: netmask address
-c: channel number (1-14) that the access point is on, defaults
to current
-C: channel number (1-14) that we're going to move them to
-i: the name of the AirJack interface to use (defaults to aj0)
-I: the name of the interface to use (defaults to eth1)
-e: the ssid of the AP
```

If you want to find more on how `kracker_jack` performs a man-in-the-middle attack against WAVEsec, check out Abaddon's Black Hat briefings presentation at <http://802.11ninja.net/bh2002.ppt>.

As a less specific attack against IKE, you can try `IKECrack`, which works against IKE phase 1 aggressive mode and MD5_HMACs only. `IKECrack` (`ikecrack-snarf-1.00.pl` on the site) is a Perl script that takes a pcap-format file as an input and attempts a real-time brute-force of the PSK.





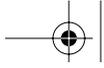
Chapter 8 • Breaking Through

Finally, a desperate attacker can resort to DoS attacks against IPSec, perhaps to force the system administrator to bring down the IPSec tunnel for a while to determine what went wrong. If there is mission-critical traffic on the wireless link, the attacker's hope is that it will be allowed to pass unprotected while the network administration is searching for the source of the IPSec tunnel failure. A cracker can try to stop ISAKMP for IPSec traffic with a `h09_ropcr` Ettercap plug-in (likely to work only against the aggressive IKE mode). Less specific attacks such as flooding UDP port 500 on IKE-running hosts can also be launched. There is a report (<http://www.securiteam.com/windowsntfocus/6N00G0A3FO.html>) that continuous flooding of UDP 500 port on a Windows 2000 machine with large (more than 800 bytes) UDP packets can use all available CPU cycles and lock up the targeted machine.

The Last Resort: Wireless DoS Attacks

Multiple DoS attacks against various wireless (and even wired) protocols, security protocols included, are mentioned elsewhere in the chapter. In many cases these attacks can be part of a sophisticated penetration plan and assist in social engineering, man-in-the-middle attempts, stealing, or cracking secret keys. However, a desperate attacker might launch a DoS attack to "compensate" for the effort spent on failed access attempts. Besides, wireless DoS attacks per se can be launched by the competitors, for political reasons, out of curiosity, and so forth; the situation is no different from DoS attacks on public networks such as the Internet. Unfortunately, due to the nature of the RF medium and design of the core 802.11 protocols, wireless networks cannot be protected against Layer 1 and certain Layer 2 DoS attacks. This is why, in our opinion, 802.11 links should not be used for mission-critical applications in theory. In the real world, there are cases when 802.11 is the only choice, and cases of system administrators or network designers being unaware or dismissive of the problem and going forward with the WLAN installation anyway. This is why you, as a security professional, should be able to demonstrate various wireless DoS dangers to your clients. If you are a system administrator or a wireless enthusiast, you can always check out how wireless DoS attacks work on your network, perhaps to know what to expect when your WLAN is attacked and to generate IDS signatures. For your convenience, we have categorized known wireless DoS attacks:





The Last Resort: Wireless DoS Attacks

1. Physical Layer Attacks or Jamming

There is nothing you can do about RF jamming short of triangulating the jamming device and tracking its owner. Even then the jammer owner is likely to claim that he or she did nothing illegal, because anyone is allowed to transmit anything in the ISM band. You will have to prove that the attacker's transmission is intentional and that he or she has exceeded the FCC EIRP limit (most likely this is the case) in a court of law. The jamming device can be a custom-built transmitter or a high-output wireless client card or even access point (e.g., Demarctech offers an AP with 500-mW output!) flooding the selected channel(s) with junk traffic. FakeAP, Void11, File2air, or any other 802.11 frame-generating tool can be used to run the flood. A completely custom-built jammer can employ harmonics and transmit at about 1.2 GHz or even about 600 MHz. Such a device would be easier to build than the 2.4 to 2.5 GHz jammer, and you'll need a decent, expensive frequency counter to discover the attack and its source. If one wants to build a very powerful 2.4 to 2.5 GHz jamming device, the core for such a device is elsewhere; it's called a microwave oven's magnetron. Check out Vjacheslav (Slava) Persion's Web page (<http://www.voltagelabs.com/pages/projects/herf005/>) for examples of microwave magnetron-based transmitters in action. The main disadvantages of Layer 1 attacks from the attacker's perspective are time, effort, and expenses to build a jammer, and the fact that such a device would have to be positioned quite close to the attacked network for an efficient attack. It is very likely that once the attack is discovered, the jammer is lost and can serve as hard evidence in court.

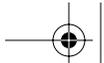


2. Spoofed Deassociation and Deauthentication Frames Floods

These attacks are probably the most well-known and used DoS attacks on 802.11 LANs. In the beginning of this chapter we discussed deauthentication frames floods when applied to bypassing MAC address filtering and closed ESSIDs.

Just as in the case of jamming, there is little you can do to eliminate the threat. The 802.11i developers have discussed the possibility of authenticated deauthentication (pardon the tautology) and deassociation. However, as far as we know, the idea did not get any further in practical terms. A variety of tools can be used to launch deauthentication





Chapter 8 • Breaking Through

and deassociation floods, including `dinject`, `wlan_jack`, `File2air`, `void11`, and `omerta`. `Void11` is probably the most devastating tool mentioned because it provides “canned” mass flood and match list flood capabilities:

```
arhontus# void11_hopper >/dev/null &  
arhontus# void11_penetration -D wlan0 -S ihatethisnetwork -m 30
```

or

```
arhontus# void11_hopper >/dev/null &  
arhontus# echo DE:AD:BE:EF:13:37 > matchlist  
arhontus# void11_penetration -l matchlist -D wlan0
```

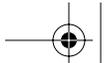
The capability to attack hosts from a matchlist can be very useful when implementing active defenses on your WLAN.

An extension of the deauthentication or deassociation frames flood attack is sequential multiframe attacks, such as sending deauthentication or deassociation frames followed by a forged probe responses and beacon frames flood providing incorrect information (ESSID, channel) about an access point to associate with. If 802.1x is used on the network, an EAP-Failure frame can preclude the deauthenticate or deassociate + fake probe responses frames train. Such an attack guarantees that the targeted host is dropped from the WLAN like a lead weight and will have difficulties reassociating. A forged probe responses flood might or might not have a significant detrimental effect on reassociation, depending on the passive versus active scanning priority implemented by the attacked host wireless card firmware. An example of deauthenticate + fake probe response frame attack is given in the `file2air` README file; this or other (`void11` + FakeAP?) tools can be used to launch this type of attack.

3. Spoofed Malformed Authentication Frame Attack

This attack is implemented in practice by the `fata_jack` utility written for AirJack by “loud-fat-bloke” (Mark Osborne; <http://www.loud-fat-bloke.co.uk>). It is based on the `wlan_jack` code, but sends altered spoofed authentication request frames instead. As the author of the tool states, the sent frame has a destination address of the AP and a source address of the attacked client and is an authentication frame with an unknown algorithm (type 2) and a sequence number and status code both set to 0xffff.





The Last Resort: Wireless DoS Attacks

As a result of an attack, the AP sends the impersonated client a reply frame. This frame says “Received an authentication frame with authentication sequence transaction sequence number out of expected sequence” (i.e., code 0x000e). This causes the client to become unauthenticated from the AP. In our experience, the client becomes deassociated and starts behaving erratically, exhibiting difficulties reassociating and sudden channel hops.

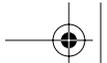
4. Filling Up the Access Point Association and Authentication Buffers

Many access points do not implement any protection against these buffers being overflowed and will crash after an excessive amount of connections are established or authentication requests sent. This applies to software access points as well; for example, an OpenBSD 3.1-based AP. Void11 implements both association and authentication frames floods with random flooding host interface MAC addresses. A small progtest utility that comes as an example code with libwlan for Linux HostAP also associates a great number of fake stations with an access point to see if it will crash or freeze. Alternatively, you can associate to the AP and then start fast MAC address changes at the associated interface. This variation of the association buffer overflow attack is implemented by a `macfld.pl` script by Joshua Wright:

```
arhontus# perl macfld.pl
macfld: Need to specify number of MAC's to generate with -c|--count
Usage:
  macfld [options]
    -c, --count
    -u, --usleep (microseconds)
    -f, --dataflush
    -p, --pingtest
    -i, --interface WLANINT
    -a, --apaddr
    -s, --srcaddr
    -d, --debug
    -h, --help
```

We strongly believe that the access point and wireless bridge manufacturers should implement these and similar tools to test their equipment before the production cycle begins.





Chapter 8 • Breaking Through

5. Frame Deletion Attack

The idea behind this attack is to corrupt the bypassing frame's CRC-32 so that the receiving host will drop it. At the same time, the attacker sends a spoofed ACK frame to the sender telling it that the frame was successfully received. As a result, the corrupt frame is efficiently deleted without being resent. Because authenticating all CSMA/CA frames is not resource-feasible, there is nothing that can be done to stop frame deletion attacks. To corrupt the CRC, the attacker might try to send the same frame with the corrupt CRC at the same time with the legitimate sender or emit a lot of noise when the sender transmits the last 4 bytes of the frame. Providing a reliable frame CRC corruption is probably the trickiest part of the attack. Of course, if implemented successfully, such an attack is not easy to detect or defend against. However, at the time of writing, it is purely theoretical and we have yet to see someone making the theoretical practical.

6. DoS Attacks Based on Specific Wireless Network Settings

There are somewhat obscure attack possibilities based on exploiting specific Layer 2 settings of 802.11 LANs, such as the power-saving mode or virtual carrier sense (RTS/CTS)-enabled networks.

In power-saving mode attacks, a cracker can pretend to be the sleeping client and poll the frames accumulated for its target from the access point. After the frames are retrieved, the access point discards the buffer contents. Thus, the legitimate client never receives them. Alternatively, our cracker can spoof traffic indication map (TIM) frames from the access point. These frames tell the sleeping clients whether the data has arrived for them to wake up and poll it. If a cracker can deceive the clients to believe that no pending data was received by the AP, they remain asleep. In the meanwhile, the access point accumulates the unpolled packets and is forced to discard them at some point or suffer a buffer overflow. This attack is more difficult to accomplish, because the cracker has to find the way to stop the valid TIM frames from reaching the intended hosts. Finally, a cracker can spoof beacons with TIM field set or ATIM frames on ad-hoc WLANs to keep the hosts awake even if there is no data to poll. This would efficiently cancel the power-saving mode operation and increase the client host's battery drain.





The Last Resort: Wireless DoS Attacks

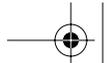
The DoS attacks against the virtual carrier sense-implementing networks are prioritization attacks by nature. A cracker can constantly flood the network with request to send (RTS) frames with a large transmission duration field set, thus reserving the medium for his or her traffic and denying other hosts from accessing the communication channel. The network is going to be overwhelmed by the clear to send (CTS) responses to every RTS frame received. The hosts on the WLAN will have to obey these CTS frames and cease transmitting.

Although there are no specific tools available to launch these attacks, in practice, File2air, a hex editor, and some additional shell scripting come to mind.

7. Attacks Against 802.11i Implementations

Nothing is without a flaw, and new security standards can introduce new potential security flaws even as they fix the old ones. The risk/benefit ratio is what matters in the end, and in the case of the 802.11i security standard the balance is positive: It is better to have it than not. Nevertheless, there are a few problems with 802.11i implementations that can be exploited to launch rather sneaky DoS attacks. In this chapter we have already reviewed DoS attacks against 802.1x/EAP authentication protocols that might force an unsuspecting network administrator to switch to other, less secure means of user authentication, if persistent. Another avenue for possible DoS attacks against 802.11i-protected networks is corrupting the TKIP Michael message integrity checksum. In accordance with the standard, if more than one corrupt MIC frame is detected in a second, the receiver shuts the connection down for a minute and generates a new session key. Thus, a cracker corrupting the frame MICs a few times every 59 seconds should be able to keep the link down. However, launching this attack is not as easy as it seems. Because understanding all the “whys” and “why nots” of the MIC corruption attack requires an understanding of MIC (and TKIP in general) operations, a detailed discussion of this attack belongs in Chapter 12, where you can find it. Here we state that running this attack by sending different MIC frames with the same IV does not appear to be easy to implement or even possible. An attacker would have to resort to means similar to the CRC-32 corruption in the frame deletion attack described earlier; for example, emit a jamming signal when the part of the frame containing the MIC is transmitted. For now, like the frame deletion attack, the corrupt MIC attack remains purely theoretical.





Chapter 8 • Breaking Through

To conclude this chapter, even the latest wireless safeguards aren't 100 percent safe. In the following discussion, you are invited to observe (or participate in) the security horrors that can follow a successful attack on a WLAN.

Summary

There are several levels of possible wireless protection ranging from the limited RF signal spread to RADIUS-based authentication and VPN deployment. However, there is a counter-countermeasure for practically every countermeasure available to WLAN defenders. This is similar to developing missiles, antimissiles, and fake targets and jammers to deflect the antimissiles in military practice. A skilled penetration tester has to be familiar with the means of getting through various wireless defense mechanisms and must be able to implement these methods when needed. Wireless penetration testing is not limited to finding networks and cracking WEP, and as the sophistication of wireless defenses grows, so does the complexity of attacks aimed at bypassing them.

