Chapter 3

WEP Vulnerabilities—Wired Equivalent Privacy?

Introduction

WEP has received an enormous amount of attention in the media as being flawed and broken. As its name implies, WEP was only intended to give wireless users the level of security implied on a wired network (which isn't much). Except in a fully switched environment, all wired traffic is exposed to the risk of eavesdropping (a.k.a., packet sniffing). WEP was not designed to be the end-all, be-all security solution for wireless networks and, as we shall see, WEP has a number of shortcomings, which make it vulnerable to several classes of attacks. The point of this chapter is to do more than just tell you that WEP is bad. Our goal is to paint a picture of what WEP was intended to do, how it works, and why it fails to live up to its design goals.

WEP 101

To truly understand the problems with WEP, we must first develop an understanding of how WEP works in its currently implemented form (Figure 3.1).

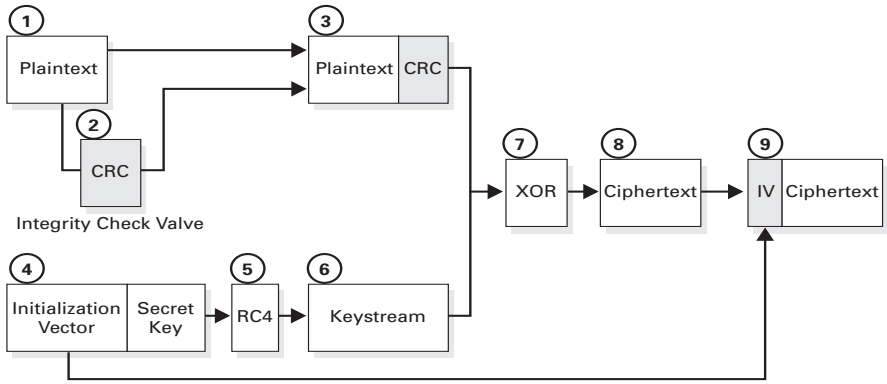

Figure 3.1 WEP encipherment.

The encryption process always begins with a plaintext message that we want to protect. First WEP performs a 32-bit cyclic redundancy check (CRC) checksum operation on the message. WEP calls this the integrity check value and concatenates it to the end of the plaintext message. Next, we take the secret key and concatenate it to the end of the initialization vector (IV). Plug this IV + secret key combination into the RC4 Pseudo-Random Number Generator (PRNG) and it will output the key stream sequence. The key stream is merely a series of 0s and 1s, equal in length to the plain text message plus CRC combination. Finally, we perform an exclusive OR operation (XOR) between the plain text message plus CRC combination and the key stream. The result is the cipher text. The IV (unencrypted) is prepended to the cipher text and included as part of the transmitted data.

Figure 3.2 is another way to look at the same operation. Again, we first take the integrity check value (the CRC) and append it to the end of the message. Then, we take this entire plaintext and XOR it with the key stream. The key stream is created by taking the secret key and appending it to the initialization vector and plugging it into the RC4 cipher.

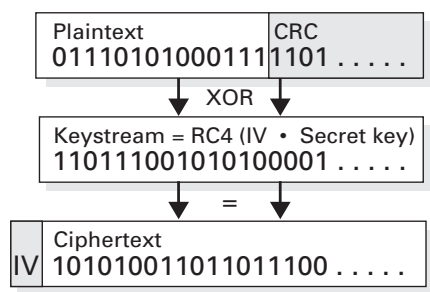

Figure 3.2 Encrypted WEP frame.

Note that after XORing the two values, we add the initialization vector to the beginning of the cipher text. The IV is prepended and included in clear text (unencrypted) because it is needed in the decryption process.

Decrypting the WEP Message

Decryption is the same process as encryption, but in reverse. We take the IV (which is sent in clear text) and prepend it to the secret key and plug that into the RC4 cipher to regenerate the key stream. Next, we XOR the key stream with the cipher text, which will give us the plain text value. Finally, we reperform the CRC-32 checksum on the message and ensure that it matches the integrity check value in our decrypted plain text. If the checksums do not match, the packet is assumed to have been tampered with and discarded.

Where Do IVs Come From?

One of the flaws in the implementation of the RC4 cipher in WEP is the fact that the 802.11 protocol does not specify how to generate IVs. Remember that IVs are the 24-bit values that are prepended to the secret key and used in the RC4 cipher. The reason we have IVs is to ensure that the value used as a seed for the RC4 PRNG is always different. RC4 is quite clear in its requirement that you should never, ever reuse a secret key. The problem with WEP is that there is no guidance on how to implement IVs. Do we choose IV values randomly? Do we start at 0 and increment by 1? Do we start at 16,777,215 and count backwards? Since each packet requires a unique seed for RC4, you can see that at high speeds, the entire 24-bit IV space can be used up in a matter of hours. Therefore, we are forced to repeat IVs, and violate RC4's cardinal rule of never repeating keys.

XOR Explained

Do you remember in school when you first learned addition and subtraction? Did your kindergarten teacher cover XOR too? WEP relies heavily upon the XOR operation, so if this is the first time you've seen this calculation performed, take a moment to orient yourself with its use. XOR is a binary logic operation that works like Figure 3.3.

Figure 3.3
XOR, a binary logic operation.

0 XORed with 0 = 0
0 XORed with 1 = 1
1 XORed with 0 = 1
1 XORed with 1 = 0

$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

The XOR operation is similar to saying "True if one value is different from the other value (i.e., one value is zero and the other value is one) and False if both values are the same (i.e., both values are zero or both values are one)."

Note that if you know two of the values in an XOR operation, you can derive the third. In other words, if you know that a number XORed with 0 equals zero, you can determine that the unknown number must be 0 (because 0 XORed with 0 equals 0). Similarly, if you know that 0 XORed with a number is equal to one, you can determine that the unknown number must be 1, because 0 XORed with 1 equals 1.

Key Management Problems

WEP uses a symmetric key encryption mechanism, meaning that the same shared secret (key) is used for both encryption and decryption. The key must be shared between the sender and receiver. One of the problems with the 802.11 protocol is that it does not address the issue of key management: How is the key distributed among users? This may not seem like a problem if you are using WEP in an environment with three laptops, but what happens if you try to deploy WEP across a campus of 5,000?

Each user must know the key and keep it a secret. What happens if one person leaves the company or has a laptop stolen? A new key must be given to every single user and re-entered in her client configuration. Also, if an attacker compromises the key from one session, the same key can be used to decrypt any other session, because everybody is using the same key.

RC4 Stream Cipher

WEP utilizes the RC4 Stream Cipher from RSA. This is the same cipher that is used in other crypto systems such as Secure Sockets Layer (SSL) (HTTPS).

The problem with WEP is that, again, the 802.11 protocol did not define how to implement IVs. As mentioned earlier, the key used in the RC4 cipher is a combination of a shared secret and an IV. The IV is a 24-bit binary number. Many manufacturers will claim to have 64-bit or 128-bit WEP, which is somewhat misleading, since 24 bits of each of these keys are the IV that is sent in clear text. Technically, the shared secret portions of the keys are really only 40 or 104 bits long.

Again, the issue with WEP is not the RC4 Cipher—it's how RC4 is implemented.

IV Collisions

When an IV is reused, we call this a collision. When a collision occurs, the combination of the shared secret and the repeated IV results in a key stream that has been used before. Since the IV is sent in clear text, an attacker who keeps track of all the traffic can identify when collisions occur. A number of attacks become possible upon the discovery of IV collisions.

A key stream attack is a method of deriving the key stream by analyzing two packets derived from the same IV. Simply stated, XORing the two cipher texts together will equal XORing the two plain texts together. Figure 3.4 shows this in detail.

In the upper left, we have taken 8 bits (plain text 1) and XORed them with our key stream. This results in cipher text 1. In the upper right, we have taken a different set of 8 bits (plain text 2), but XORed them with the same key stream, which results in a second cipher text. You will notice that XORing the two cipher texts together gives us a result equal to XORing the two plain texts together. Therefore, if both cipher texts are known (presumably

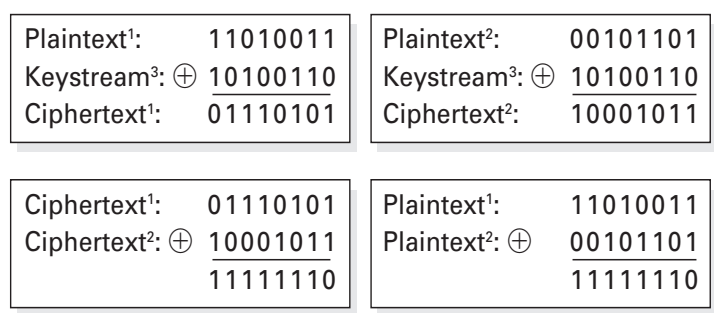

Figure 3.4 A key stream attack.

captured from a sniffer) and one plain text is known, the second plain text can be derived.

You may be wondering, "That's cool, but how do I know the first plain text, so I can derive the second plain text?" There are two ways. First, if you are able to see the target machine from a computer on the Internet (or a compromised host on the target LAN), you could send a packet to the target machine. Since it comes from you, the plain text payload of the packet would be known. You could identify the packet by forcing an unusual packet size and searching for that size in your sniffer log data.

An alternative way to learn the plain text of a packet is to guess. A number of TCP/IP protocols utilize known handshaking procedures. For example, DHCP, ARP, and other broadcast packets use well-documented signatures.

Keep in mind that key stream attacks only work when IVs are repeated. This is a major flaw in the implementation of WEP. Because the standard does not define how to implement IVs, they are often repeated and it is perfectly acceptable to reuse them. In fact, in order for an AP to remain compliant with the standard, it *must* accept IVs that have been reused. This violates a major tenet of RC4: It is unsafe to reuse the same key, ever! Keys should never be reused or repeated.

Message Injection

Once a key stream is known, a new message can be constructed by taking the new plain text and XORing it with the known key stream to create a new, forged cipher text. Again, since the 802.11 standard does not require the IV to change with every packet, each device must accept reused IVs.

For example, let's say we know the plain text and cipher text for a particular message. We could use this information to derive the key stream (Figure 3.5)

Using the key stream, we could take our own plain text and use the key stream to forge a new cipher text. This packet could then be injected into the network and decrypted by the target machine as a valid WEP packet (Figure 3.6).

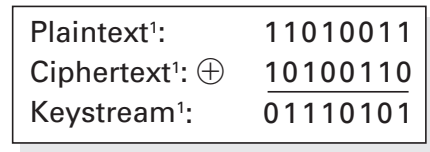

Figure 3.5
Deriving a key stream.

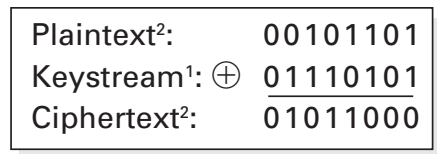

Figure 3.6
Forging a new cipher text.

Authentication Spoofing

A variation of the packet injection attack is authentication spoofing. In order to understand how this attack works, let's take another look at the shared key authentication process.

- Step 1—The client sends an authentication request to the AP.
- Step 2—The AP sends the client 128 bytes of challenge text.
- Step 3—The client encrypts the challenge text with its WEP key and sends the challenge response back to the AP.
- Step 4—The AP uses its knowledge of the WEP key to validate the challenge response and determine if the client does, in fact, know the shared secret key.
- Step 5—The AP responds to the client with a success or failure message.

The problem here is that if an attacker can observe this negotiation process, she will know the plain text (challenge text) and its associated cipher text (challenge response). Using the message injection attack methodology, the attacker could then derive the key stream, request authentication from the AP, and use the same key stream on the challenge text to create a valid challenge response. The attacker would then be authenticated to the AP even though she has no knowledge of the WEP key. This attack works because the challenge text is always 128 bytes and, again, because IVs can be repeated and reused.

Brute Force Attacks

Another approach to determining the WEP key is to use brute force. The shared secret portion of the WEP key is either 40 bits or 104 bits, depending on which key strength you are using. Security researcher Tim Newsham discovered that the key generators from some vendors are flawed. A brute force attack on a 40-bit key using a weak key generator could take less than a minute to crack.

Key generators enable a user to enter a simple pass phrase to generate the key, instead of entering the key manually with hexadecimal numbers. A 40-bit WEP key shared secret would require 10 hexadecimal numbers; a 104-bit WEP key shared secret would require 26 hexadecimal numbers. As a convenience, some vendors allow you to enter a pass phrase in ASCII that will generate the 10 or 26 hexadecimal numbers for you. The use of a key generator is completely proprietary and not part of any standard. However, note that several different vendors all use the same key generation algorithm.

Tim Newsham discovered that there are a number of problems with the key generators for several vendors. In one example, he noticed that for 40-bit keys, part of the key generation process included a 32-bit seed used in a PRNG. Because the highest bit of each ASCII character is always 0 and the key generator relied on XORing ASCII values, Tim discovered that instead of 00:00:00:00 – ff:ff:ff:ff (32 bits) of possible seeds, only values 00:00:00:00 – 00:7f:7f:7f needed to be considered. This reduced the actual entropy of the PRNG seed to 21 bits. Using a PIII/500 MHz laptop performing 60,000 guesses per second, Newsham was able to crack a 40-bit WEP key from a key generator in 35 seconds.

The moral of the story: Don't use key generators! Enter your WEP key using manual hexadecimal numbers. When done in this manner, a 40-bit WEP key would have taken 210 days to crack (not a terribly difficult task, when attacked by a Linux cluster).

Alternatively, you can implement 104-bit WEP. Tim noted that the key generator used for 104-bit WEP was not flawed. It was based on an MD-5 hash of the pass phrase. He estimated that a brute force of this key would take 10^{19} years. Clearly, brute forcing a 104-bit key is a much more difficult task then brute forcing a 40-bit key. When using WEP, always deploy the largest key size available.

Cracking WEP Keys

Programs such as AirSnort, WEPCrack, and dweputils crack WEP keys based on an attack described in a paper titled "Weaknesses in the Key Scheduling Algorithm of RC4" written by Scott Fluhrer, Itsik Mantin, and Adi Shamir. This paper identified certain IVs that leak information about the secret key. In fact, there are large classes of these weak keys. If you can collect enough cipher text that is derived from them, you can determine the secret key with relatively little work. This assumes, however, that the attacker has knowledge of the first few bytes of plain text. Interestingly enough, because of RFC 1042 (SNAP headers), all IP and ARP packets always start with 0xAA. Therefore, the first few bytes of plain text are (almost) always known. (IPX/SPX traffic uses a different SNAP header.)

Brute Force Attacks vs. FMS Attacks

Traditional brute force and FMS attacks represent two very different styles of attack. With a brute force attack, you only need to capture a single encrypted packet and then apply an enormous amount of computing power. (You probably want two packets: one to crack the key and one to double check that the cracked key works.) FMS attacks, on the other hand, rely on capturing an enormous amount of encrypted traffic, then using very little CPU power for a probabilistic algorithm to crack the key. In fact, the FMS crack scales linearly, which means that cracking a 128-bit key takes only slightly longer to crack then a 64-bit key, once you have captured enough weak keys.

Effective FMS Attacks

The problem for FMS attacks is capturing enough encrypted data to crack the key. In a high traffic network, this can be accomplished in a matter of hours. However, in a low traffic environment, this process can take days or weeks. To crack the WEP key using FMS, some attackers are simply patient and resort to doing sneaky things like putting AirSnort (or other tools) on a PDA and placing it in the bushes near the AP for days at a time. Other attackers have developed more clever techniques to artificially generate network traffic in order to capture cipher text to crack the key.

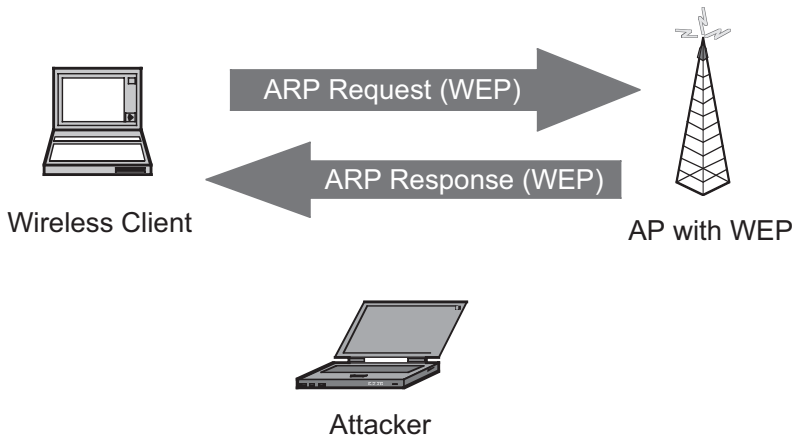

Figure 3.7 The attacker captures a legitimate, encrypted packet and guesses that it is an ARP request based on a known size (28 bytes).

One possible packet injection attack works like this: The attacker will capture the encrypted traffic and look for a known protocol negotiation based on the size of the captured packet; for example, an ARP request has a predictable size (28 bytes). Once captured, the attacker can simply re-inject the encrypted packet (ARP request) over and over again. The ARP response will generate new traffic, which the attacker can then capture. If the attacker repeats this process over and over again, it is possible to generate enough traffic for a successful FMS attack in about an hour. (See http://www.dachb0den.com for more information on this packet injection technique.)

Figures 3.7 and 3.8 show how this attack might be carried out.

Keep in mind that FMS attacks rely upon the attacker's ability to capture weak keys. Many hardware vendors have implemented firmware updates for their wireless NICs and APs that simply skip the specific IVs that cause these weak keys. This weak key avoidance technique renders the FMS attack useless. This is another reason why upgrading the firmware in all the devices in your wireless network is particularly important.

Orinoco Release Notes

Orinoco began implementing weak key avoidance in their firmware in the winter of 2002. The release notes for the Orinoco 8.10 firmware upgrade includes the following:

WEP Weak Key Avoidance

The key that is input to the WEP64 or 128 RC4 encryption algorithm consists of the secret key configured by the user (or via 802.1x) concatenated with

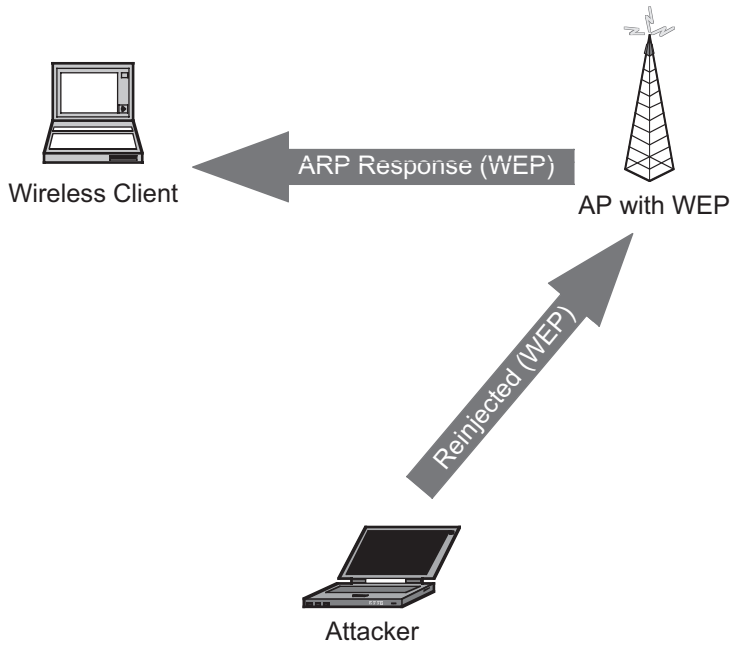

Figure 3.8
The attacker floods the network with the reinjected ARP reject. This results in a flood of ARP responses, which the attacker captures as part of an FMS attack.

the IV (Initialization Vector). The IV is determined by the transmitting station. By excluding certain IV values that would create so-called "weak keys," the weakness of WEP as described in "Weaknesses in the Key Scheduling Algorithm of RC4" by Scott Fluhrer, Itsik Mantin and Adi Shamir, and demonstrated through the AirSnort program, are avoided. Note that, as the IV is always determined by the transmitting station, there is no impact on interoperability. Stations/APs with weak key avoidance implemented can interoperate with stations/APs that do not have this. Of course, protection against this attack is provided only if all stations and APs implement this new scheme.

Now What?

Okay, now that you understand how WEP works and you've seen some of its shortcomings, it's time to take a deep breath and acknowledge that WEP isn't perfect, but it's not the end of the world. First of all, cracking WEP is not a trivial task. It requires a certain degree of skill and tenacity to pull off the attack. In a low traffic environment, you'll also need lots of patience in order to collect the large volume of packets needed to successfully crack the key (or

some additional skills to effectively mount a packet injection attack to trick the network into flooding traffic).

The obvious answer to the WEP problem is to extend the IV space and don't reuse IVs. These issues (and more) are addressed in the WPA protocol (see Chapter 5). If your environment doesn't support WPA, use WEP, but don't rely on it exclusively to keep your enterprise secure. At a minimum, change your WEP keys as often as practical and possible.

Summary

WEP has a number of well-documented vulnerabilities that significantly limit its ability to safeguard data. In this chapter, we reviewed how WEP and XORing work to help you understand the problems and go beyond the "WEP is Bad" headlines. The underlying encryption engine used by WEP is RC4, which is widely used in various Internet protocols including secure Web pages (HTTPS). When it comes to WEP flaws, the problem isn't RC4. The problem is the way that RC4 is implemented. In particular, the implementation of IVs is flawed because it allows IVs to be repeated and hence, violate the No. 1 rule of RC4: Never, ever reuse a key.

Newsham exposed another vulnerability of WEP by demonstrating that the key generator used by many vendors is flawed for 40-bit key generation. Using a typical laptop, he was able to crack a 40-bit key is less than a minute.

Another flaw of WEP, in the key scheduling algorithm, was discovered by Fluhrer, Mantin, and Shamir. This weakness, exploited by commonly available tools such as AirSnort, WEPCrack and dweputils, has the ability to crack WEP keys by analyzing traffic from totally passive data captures. If your network is consistently generating traffic at peak speeds, the WEP key (64 or 128 bit) can be cracked after capturing just a few hours of encrypted data. On a network with minimal activity, this attack could take days or even weeks to capture the requisite traffic. Some packet injection techniques, however, have the ability to artificially flood the network with activity to reduce the amount of time it takes to collect enough packets for an FMS attack. On the other hand, keep in mind that vendors who include weak key avoidance techniques in their firmware (which most do) are not vulnerable to FMS attacks. So, be sure to update your firmware on a periodic basis!

These issues don't make WEP useless, it just means that you have to be careful about how and when you use it. If you aren't able to implement anything else (such as WPA), and the only thing you have is WEP, then go ahead

and use it. If you're in a network with minimal security requirements, WEP may be appropriate.

I recommend using WEP and changing keys on a regular basis, if for no other reason, then because it identifies your network as private. Since the 802.11 protocol has no other way to tell the world that they shouldn't be attempting to associate with your AP, using WEP is a first line of defense to keep intruders out, or at least put them on notice that a No Trespassing sign has been posted.