Airscanner

Cracking WEP

Since mid-summer 2001, wireless network security has been a hot topic. The fact that most wireless networks are not secured has been reported, discussed, lectured on, and more. About the only way you could have avoided this is if you were completely removed from every form of paper and electronic media.

However, all this media hype has not had the desired effect! Believe it or not, an estimated 60[nd]70% of all wireless networks in existence are still NOT using even the most basic of protections. As a result, any hacker can drive around the block and probably find an open and hackable wireless network. This is problem number one.

Problem number two is a bit more technical, and will be the main subject for this article. This problem concerns the weaknesses surrounding WEP. The *Wireless Equivalent Privacy* protocol defines how a wireless network is secured. In short, it determines what encryption and authentication method is used to secure wireless data. The problem with WEP is in the way the data is encrypted. As discovered by three researchers, WEP can be cracked by anyone with a *sniffer*, which is the name given to the hardware device or software that can capture data as it flies through the air. This basically means that all those companies that think they are securely using their wireless network are doing so under false pretenses. This article deals with this issue, and explains just what cracking WEP means. I will show you just how hackers will attack you and take advantage of this weakness to capture your secret key right out of the air, which they can then use to connect to your secure wireless network.

Home Wireless Local Area Network Users

Before delving into the details of WEP, it must be understood that there are two main categories of WEP users. Many home users don't bother setting up WEP because they considered it a waste of time since WEP is crackable, or because they find it too complicated. If you are the latter, find someone who knows how to enable WEP, and offer them dinner. Regardless of the issues surrounding WEP, it should be understood that cracking WEP is not as easy as everyone makes it sound. Although cracking WEP is possible on the typical home-owned WLAN, it would take two to four weeks to capture enough data to successfully extract the key. In other words, by simply enabling WEP and changing the secret key periodically, you can be fairly certain that your WLAN will not be hijacked by a hacker. That said, let's take a look at how cracking WEP appears from a hackers point of view.

The Secret Key

As was previously mentioned, WEP incorporates two main types of protection: a secret key and encryption. The secret key is a simple 5- or 13-character password that is shared between the access point and all wireless network users. This key is all-important to WEP in that it is also used in the encryption process to uniquely scramble each packet of information with a unique password. This ensures that if a hacker cracks one packets key, he won't be able to view every packet's information.

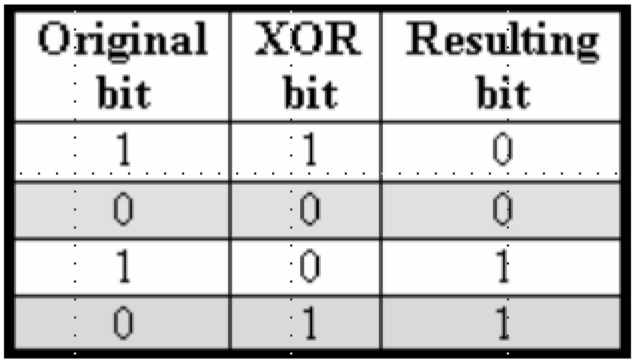
To do this, WEP defines a method to create a unique secret key for each packet using the 5- or 13-characters of the pre-shared key and three more psuedo-randomly selected characters picked by the wireless hardware.

For example, let's assume that our pre-shared key was "games". This word would then be merged with "abc" to create a secret key of "abcgames", which would be used to encrypt the packet. The next packet would still use "games", but concatenate it this time with "xyz" to create a new secret key of "xyzgames". This process would randomly continue during the transmission of data. This changing part of the secret key is called the Initialization Vector because it initializes the encryption process for each packet of data sent.

XOR

It is important to understand the basics of XOR when discussing RC4 and WEP because it is used in the encryption process to create the encrypted data.

XOR is just a simple binary comparison between two bytes that produces another byte as a result of a simple process. In short, it takes each corresponding bit in a byte and compares them by asking "Is this bit different from that bit?" If the answer is yes, the result is 1; otherwise it is a 0. Figure 1 illustrates.



Original bit	XOR bit	Resulting bit
1	1	0
0	0	0
1	0	1
0	1	1

Figure 1

XOR Byte Comparison Table.

From this illustration, you should also note one other thing. Just as the resulting bit can be deduced by comparing the first two columns, the same can be said about the original bit if the XOR bit and resulting bit are compared. (for example, 1 XOR 1 = 0 □ 0 XOR 1 = 1). This is an important part of how and why WEP is crackable.

RC4

RC4 is the encryption algorithm used to cipher the data sent over the airwaves. It is important that data is scrambled; otherwise, anyone could "see" everything using a sniffer. This includes all e-mails, Web pages, documents, and more. RC4 is a very simple and fast method of encryption that scrambles each and every byte of data sent in a packet. It does this through a series of equations using the previously discussed secret key.

RC4 actually consists of two parts: the Key Scheduling Algorithm and the Psuedo Random Generation Algorithm. Each part is responsible for a different part of the encryption process. However, before discussing the algorithm in detail, we need to understand what an array is.

Array Swapping

An array is a programming term used to hold multiple values. For example, consider the alphabetArray(26). This array would hold 26 values, with each value represented by the number in the array. For example:

```
alphabetArray(1)=a
alphabetArray(2)=b
alphabetArray(3)=c
```

However, in the case of a secure application of an array, we want to scramble the values held in each position. If this wasn't done, a hacker could easily predict what was in alphabetArray(26). To do this, a swapping function can be performed on the array. For example, consider the following illustration:

```
alphabetArray(1)=A
alphabetArray(2)=B

Swap (alphabetArray(1), alphabetArray(2))
Swap(A, B) □ (B,A)

alphabetArray(1)=B
alphabetArray(2)=A
```

As you can see, the values held in each array position were switched with each other. If this same process is done over and over again using a psuedo random routine, you won't be able to tell what value is held in any of the array positions.

KSA

The Key Scheduling Algorithm is the first part of the encryption process. The following is the algorithm actually used in RC4 by line with an explanation for each line.

Algorithm

```
1. Assume N = 256
2. K[] = Secrete Key array
3. Initialization:
4. For i = 0  to N – 1
5.      S[i] = i
6. j = 0
7. Scrambling:
8. For i = 0 ... N – 1
9.      j = j + S[i] + K[i]
10.     Swap(S[i], S[j])
```

Explanation

1. N is an index value. It determines how strong the scrambling process is. WEP uses a value of 256.

2. K is the letter used to symbolize the secret key array. In the case of a five-character, pre-shared key, this value would be the three-character IV + five-character pre-shared key eight-character secret key. Each character is held in the corresponding K position. This value does not get scrambled.

3. This starts the initialization of the KSA. It basically is used to seed the empty State (S[]) array with values 0[nd]255.

4. This is the start of the loop process that increases the value of i each time the algorithm loops.

5. Once it is done, the S array will hold values 0[nd]255 in corresponding array position 0[nd]255.

6. j is used to hold a value during the scrambling process, but it must first be initialized to ensure that it always starts at 0.

7. This starts the scrambling process that creates the psuedo random S array from the previously seeded S array.

8. Another loop that ensures the scrambling process occurs 256 times.

9. This is the equation used to merge the properties of the secret key with the state array (S[]) to create a psuedo random number, which is assigned to j.

10. Finally, a swap function is performed to swap the value held in S[i] with the value held in S[j].

As you can see, this is not a terribly complex process. Some simple math based on the secret key, and you have a psuedo random state array. The next part takes this array and creates a stream of data that is used to encrypt the data to be sent over the airwaves.

PRGA

The PRGA (Psuedo Random Generation Algorithm) is the part of the RC4 process that outputs a streaming key based on the KSA's psuedo random state array. This streaming key is then merged with the plaintext data to create a stream of data that is encrypted. Following is the algorithm and its explanation:

Algorithm

```
1. Initialization:
2. i = 0
3. j = 0
4. Generation Loop:
5.     i = i + 1
6.     j = j + S[i]
7.     Swap(S[i], S[j])
8.     Output z = S[S[i] + S[j]]
9.     Output XORed with data
```

Explanation

1. Again, before using the PRGA, the i and j values must be initialized.

2. i initialized to 0.

3. j initialized to 0.

4. This starts the stream-generation processes. It will continue until there is no more data, which in WEP's case is the end of the packet of data[md]or about 1,500 bytes.

5. i is added to itself to keep a running value used in the swap process.

   Note: This value will ALWAYS equal 1 the first time through the PRGA loop (i = i + 1 i = 0 + 1 = 1).

6. j is used to hold the psuedo random number in the S[] position, with the previous S[] added to it.

   Note: This value will ALWAYS hold the value held in S[1] for the first iteration of the PRGA (j = j + S[i] j = 0 + S[1]).

7. Another swap function is performed that switches the values held in the i position and j position of the state array.

8. z is calculated based on an addition of the value held in the state array, as represented by the addition of the values held in S[i] added to S[j]. (This will be better understood after seeing the example later in the article.)

9. Finally, the z value is XORed with the plaintext to create a new and encrypted value. This can be represented by the equation encrypted data = z XOR plaintext.

   Note: XOR only requires that you know ANY two of the values to deduce the third. In other words, if the plaintext is known and the encrypted data is captured by a sniffer, a hacker can deduce the z value outputted by the PRGA.

CRC

There is one final part of the data-transmission process that needs to be mentioned due to the fact that it adds additional data to the packet. This is the CRC, or Cyclic Redundancy Checksum value.

When a packet is sent across a network, there has to be a way for the receiving party to know that the packet was not altered or corrupted in transmission. This is accomplished via the CRC. Before the data is packaged and sent, a value is calculated by the CRC algorithm that is based on the bytes of the data. This value is then appended to the actual data and sent to the receiving party. Once the packet is received, the CRC value is removed, and a NEW CRC value is calculated on the received data. If the NEW CRC value matches the ORIGINAL CRC value, the packet is assumed to be complete; otherwise, the packet is considered corrupted and is dumped. As you will see next, this does affect the whole encryption process.

Putting It All Together

Now that we have briefly covered the basics, let's take a look at how it all works together. Figures 2 and 3 provide a graphical representation of the whole encryption process, and decryption process.

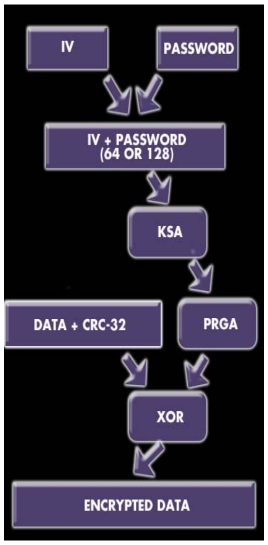


Figure 2

Graphical representation of RC4 encryption process.

As illustrated in figure 2, the IV is first created by the access point, and is merged with the pre-shared key to create a secret key. This key is then used by the KSA to create a psuedo random state array, which is then used by the PRGA to create a streaming key that is XORed with the plaintext data and its CRC value. As a result, the encrypted data is created and sent to the receiving party of the WLAN, where it is then unencrypted.

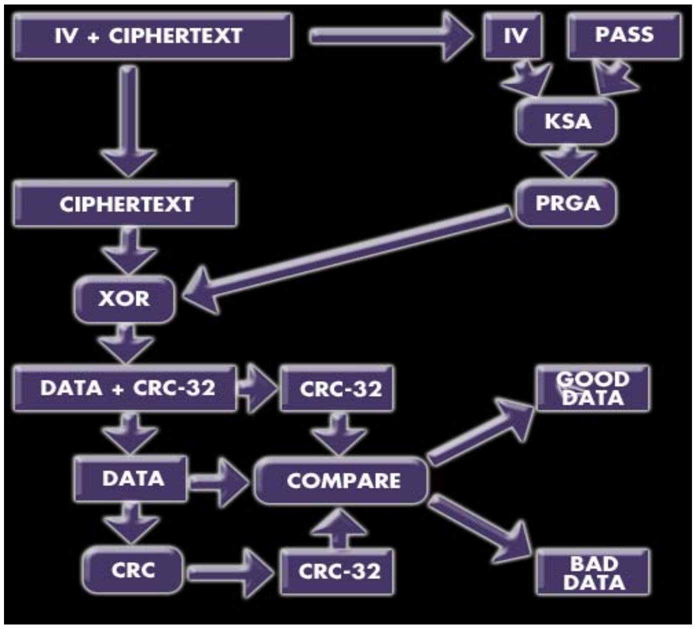


Figure 3

Graphical representation of RC4 decryption process.

From figure 3 you can see that once the data is received, the IV, which is sent as plaintext appended to the encrypted text, is removed and merged with the pre-shared password to create the same secret key used in the encryption process. This key is then used by the KSA to create a duplicate psuedo random state array value that is used by the PRGA to create the same streaming key used to encrypt the plaintext. This streaming key is XORed with the encrypted text, which results in the creation of the plaintext and CRC value. The CRC value is removed, and a new CRC value is deduced, which is compared to the original CRC value. The data is then either accepted or rejected.

Cracking WEP

Now that we understand the basics of how WEP works, let's review a few points.

- The IV is sent as plaintext with the encrypted packet. Therefore, ANYONE can easily sniff this information out of the airwave and thus learn the first three characters or the secret key.

- Both the KSA and PRGA leak information during the first few iterations of their algorithm. The i will always be 1, and j will always equal S[1] for the first iteration of the PRGA, and the KSA is easily duplicable for the first three iterations due to the fact that the first three characters of the secret key is passed as plaintext.

- XOR is a simple process that can be easily used to deduce any unknown value if the other two values are known.

In addition to these previously explained points, there are several more that make WEP dangerous.

- There is a 5% probability that the values held in S[0][nd]S[3] will NOT change after the first three iterations of the KSA. In other words, any hacker can guess what will happen during the KSA process with a 5% likelihood of being correct.

- The first value of the encrypted data is always the SNAP header, which equals "AA" in hex or "170" in decimal form. This essentially means that by sniffing the first byte of encrypted text and XORing it with 170, any hacker can deduce the first output byte of the PRGA.

- In the WEP encryption process, it has been determined that a certain format of an IV indicates that it is a weak IV and subject to cracking. The format is (B + 3, 255, x) where B is the byte of the secret key being cracked. However, we know the first three characters due to the IV, so we want to crack the pre-shared password that starts after the IV. The 255 value indicates that the KSA is at a vulnerable point in the algorithm, and the value "x" can be any value.

Now that these points have been provided, let's take a look at how a hacker would use this knowledge to crack WEP.

Walking Through the KSA

As previously mentioned, the IV is sent as plaintext. This can be easily sniffed out of the air and used to re-create the first three iterations of the KSA. To illustrate, we will do just that. Follow closely and pay attention to the math[el]this will get a bit technical!

Before getting into the details, we need to define a few values.

- The captured weak IV is 3, 255, 7. This value was chosen because it was tested and is known to be a real weak IV.

- The pre-shared password is 22222. Although a hacker would not know this before cracking WEP, we need to define it so you can see the cracking process in action.

- N is 256.
- If a value exists that is greater than N (256), a modulus operation must be performed on it. This basically divides the number by 256, which results in a leftover number called the modulus. This is the value that is passed on through the calculation.
- The initialization process of the state array has already occurred and seeded the state array with the 256 values.

First, we need to clarify our key array as a hacker would see it after capturing the IV.

K[0]=3	K[1]=255	K[2]=7	K[3]=?	K[4]=?	K[5]=?	K[6]=?	K[7]=?

Next, we need to define and track the state array values, i value, and j value. This will be done before each loop is processed, so you can see how the values change. We will not show all 256 state array values because they are useless to the cracking WEP process. Only the first four state array values and any value that has changed will be shown.

KSA loop 1

i=0	j=0	S[0]=0	S[1]=1	S[2]=2	S[3]=3		

j=j + S[i] + K[i mod l] = 0 + S[0] + K[0] = 0 + 0 + 3 = 3 à j = 3

In this equation, you can see that the j and i value were 0, which is used by the S[] array (S[0] = 0) and the K[] array (K[0] = 3). This resulted in the values of 0, 0, and 3 being added together to assign the value of 3 to j. This value is then passed on to the swap function below.

i=0, j=3

Swap (S[i], S[j]) à Swap (S[0] , S[3]) à S[0] = 0 , S[3] = 3 à S[0] = 3 , S[3] = 0

In this process, you can see that values held in S[0] and S[3] are swapped. This is an important process to watch, but remember there is a 5% chance that the values held in S[0] S[3] will not change after the first 4 KSA/PRGA loops.

KSA loop 2

i=1	j=3	S[0]=3	S[1]=1	S[2]=2	S[3]=0		

j=j + S[i] + K[i mod l] = 3 + S[1] + K[1 mod 8] = 3 + 1 + 255 = 259 mod 256 = 3 à j = 3

i=1, j=3

Swap(S[i], S[j]) à Swap (S[1] , S[3]) à S[1]=1 , S[3]=0 à S[1]=0 , S[3]=1

Note that in this loop the value of i increases by one and that a modulus operation was performed to determine the value of j. It is only coincidental that j = 3 again.

KSA loop 3

i=2	j=3	S[0]=3	S[1]=0	S[2]=2	S[3]=1		

j=j + S[i] + K[i mod l] = 3 + S[2] + K[2] = 3 + 2 + 7 = 12 à j = 12

i=2, j=12

Swap(S[i], S[j]) à Swap (S[2] , S[12]) à S[2]=2 , S[12]=12 à S[2]=12 , S[12]=2

Note that up to this point, only KNOWN values are used. Any hacker can reproduce this process up to this point. However, in the next step, the secret key is unknown, so a hacker has to stop.

KSA loop 4

i=3	j=12	S[0]=3	S[1]=0	S[2]=12	S[3]=1	S[12]=2	

j=j + S[i] + K[i mod l] = 12 + S[3] + K[3] = 12 + 1 + ? = ?

i=3, j=?

Swap(S[i], S[j]) à Swap (S[3] , S[?]) à S[3]=1 , S[?]=? à S[3]=?? , S[??]=1

So, now a hacker is up against a wall. However, what if there were a way to determine the j value at this point? Fortunately, for a hacker, there is a way. A simple XOR calculation, and he can determine this value from the first iteration of the PRGA process.

Knowing this, let's reflect on the XOR process that creates the encrypted data. The final step of the RC4 process is to XOR a PRGA byte with a byte of the plaintext data. Since XOR works in both directions, we also know that we can get deduce the first byte of the PRGA if we XOR the first byte of the encrypted data with the first byte of plaintext. Fortunately, for a hacker this is easy thanks to the SNAP header (170 in decimal) and the use of a sniffer to capture the encrypted byte. In our example, we will provide the captured encrypted byte value (165 in decimal), which changes from packet to packet. The following equation illustrates the XOR process:

z = 0xAA(SNAP) XOR Ciphertext byte1 = 170 (Dec) XOR 165 (Dec) = 15 è z = 15

As a result of this XOR calculation, a hacker can deduce that the PRGA value is 15 (decimal). Now, he can reverse-engineer the PRGA process, and use this to determine the missing j value. First, let's remind ourselves of the known loop values as they would occur entering loop 4 of the KSA. Remember, these values can be easily reproduced by the use of the IV values.

KSA loop 4

i=3	j=12	S[0]=3	S[1]=0	S[2]=12	S[3]=1	S[12]=2	

1. Initialization:
2. i=0
3. j=0
4. Generation:
5. i = i + 1 = 0 + 1 = 1
6. j = j + S[i] = 0 + S[1] = 0 + 0 = 0
7. Swap (S[i], S[j]) à Swap (S[1] , S[0]) à S[1]=0 , S[0]=3 à S[1]=3 , S[0]=0
8. z = S[S[i] + S[j]] = S[S[1] + S[0]] = S[3 + 0] = S[3] = ?
9. ?=15 è S[3] =15 at KSA4

From the previous discussion, you know that i will always equal 1 for the first iteration of the PRGA (line 5). This then means that j will always equal S[0] (line 6). As we can see from the KSA loop 4 input values, S[1] = 0. This then results in j being assigned the value of 0 (line 6). The values held in S[i] and S[j] are then swapped, which means that S[1] is swapped with S[0] resulting in S[1] = 3 and S[0] = 0 (line 7). These values are then added together, and used to pull a value from the state array. In this case, the combined S[i] and S[j] values = 3 (line 8). However, the S[3] value referenced to here is from the completion of KSA loop 4, which is unknown to us. Fortunately, due to the XOR process, we know that the resulting value is 15, which means that S[3] will equal 15 at the output of KSA loop 4. Knowing this, a hacker only needs to reverse the KSA loop 4 process to deduce the secret key value.

Let's now walk though this as a hacker would.

KSA loop 4

i=3	j=12	S[0]=3	S[1]=0	S[2]=12	S[3]=1	S[12]=2	

S[3]=15 , S[15]=S[3]t-1 à S[3]=15 , S[15]=1

First, we know that the final step in the KSA loop is to swap values. Knowing the values of the state array after loop 4 completes and before it starts is important. Thanks to the XOR weakness, we know S[3] will equal 15, and we can make an educated guess that S[15] will hold the value held by S[3] before loop 4, which is 1 in this case. As a result, a hacker can deduce that S[3]=15 and S[15]=1 after the swap.

Swap (S[3] , S[15]) à S[3]=1 , S[15]=15

Next, a hacker swaps the values held in these positions, which leaves S[3] equaling 1.

j=j + S[i] + K[i mod 256] = 12 + S[3] + K[3] = 12 + 1 + K[3] = 15

A hacker then plugs the values into the equation that would produce the j value. This fills in all the fields except the value of the secret key array.

à K[3] = 15 – 12 – 1 = 2

After a simple reverse calculation, the value 2 is produced, which is the first byte of out secret key!

Simplifying the Process

This was a down-and-dirty look at how a hacker could deduce a secret key, byte by byte. If they had to do this by hand, the threat of this weakness would be seriously lessened. However, this process has been written into a program that can perform this process in seconds if enough data is captured. For example, WEPCrack (which was written as an educational tool) and AirSnort are both programs that can crack the secret key in a matter of seconds if enough data is present. The catch is with the data.

Due to the requirements, roughly 7GB of data must be captured, on average, to crack the password. This is A LOT of information. In fact, most home users and small businesses will have a tough time meeting this mark in two weeks. However, on the other hand, if a WLAN is fully maxed out, it can send this much data in two to four hours. So, the threat of WEP is a real and dangerous risk.

Patching the Hole

For those people who are concerned about this threat, there are several things that can be done to secure the WEP hole. First, use WEP. Although this may seem ridiculous, the simple fact that you use WEP will cause most hackers to skip your WLAN and move on to an unprotected target.

Second, use a RADIUS server for authentication. This will ensure that each user is permitted access to the internal network only with a user name and password. Although this is some protection, the RADIUS server should also use a time limit on the keys. This is due to further weaknesses and dangers, known as ARP poisoning, in which a hacker can take over an existing session and bypass any RADIUS requirements. By setting the time to 30 minutes, you can be sure that no hacker can successfully crack WEP.

Fourth, set up a VPN on top of the WLAN connection. This will provide further protection and require yet another password to connect. The downside of this is that it will slow the connection speed due to VPN encryption overhead.

Fifth, control access to the internal network using the user name and passwords. This will protect your resources if a laptop is stolen or if the account information is pilfered.

In addition to these measures, there are additional things one can do to increase security. Tokens, DMZs, radiation zones, and more can be used to control who has what access, where, and for how long. In short, just as with a regular network, you have to weigh the need with the costs of having users jump through more security gates to access their data. Too much security, and no one will use the service or will find ways around it. Too little, and you may have the wrong people accessing your data.

Summary

If nothing else, this article should have enlightened you about how a hacker cracks WEP. If it can be illustrated in a few pages of text, imagine how easy it is to automate using a computer. Although the danger is real for everyone who uses WEP, the threat really only applies to those who have highly trafficked WLANs. In these cases, WEP should be only one part of the security defenses built to protect your wireless network from intrusion.