SCOUT
AN IMPROVED TOOL FOR CRACKING WEP KEYS

K. Vikram

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur 208016
Uttar Pradesh, India
Email: vikram@iitk.ac.in

ABSTRACT

With Network Security becoming more and more important, most network protocols provide some level of security against attacks. The IEEE 802.11 protocol, designed for wireless local area networks is no exception. In this protocol, the layer which implements this security is part of the data link layer and is called WEP, which stands for *Wired Equivalent Privacy*. As the name suggests, it was intended to provide as much privacy in a wireless system as in a wired one. Unfortunately, as is well known, it fell short of the claims of its designers. Although a lot has been said about the potential insecurity of WEP, few have actually implemented a direct attack on an actual system. Here we shall also describe a tool, Scout[1], which implements a particular attack and which cracks both the 5 byte and the 13 byte key of WEP with a greater probability than that of existing tools.

1. INTRODUCTION

With the advent of a large number of wireless technologies, wireless networking is becoming increasingly popular. The most popular protocol until a few years ago was the Bluetooth[8], which was primarily used for short range digital communications, for instance to connect a mobile phone and a digital camera to a laptop. Towards the end of the last decade another protocol emerged, which could be used for communication over longer distances. The IEEE 802.11[11] protocol soon became a very popular standard for building wireless networks, and has also been used in applications where Bluetooth was used earlier. Even though most of the devices that implement these standards have been used indoors, there have also been attempts at using them outdoors and increasing the range at which these off-the-shelf wireless devices communicate.

As wireless devices percolate into the market and as more and more people start using them, there is an increasing demand for secure communication links. People, using such devices for their businesses or other sensitive operations, would sooner or later have to entrust their wireless systems with confidential data. If their networks are susceptible to malicious attacks, it would prove detrimental to them and their business. Even for home use, a user would find it highly desirable to prevent their neighbours from snooping into their networks. Security, therefore, is a prime concern for most users, more so for wireless networks, where one cannot impose a strict physical boundary to the data channel.

Communication protocol suites can be made secure at any of its layers. Introducing security features at the lowermost layers of the protocol stack would make the system more efficient, as much of the computation would then be done in the hardware. This would also reduce the load on the upper application layers which can assume the link to be secure and not bother about security at all. The IEEE 802.11 committee probably had something of this nature in mind when they proposed the link layer security protocol, namely WEP[11]. WEP is not really what it claims to be, though. It is fraught with security loopholes which makes it susceptible to attacks.

Highly secure protocols, though desirable in certain situations, can pose a challenge for Internet crime investigators and law enforcers. Using these protocols, criminal organizations can exchange messages that would be difficult to intercept. There have been raging philosophical debates on whether such security techniques should be made public or not, but we shall not get into it. Our primary purpose is to investigate the WEP protocol specifications and exploit its loopholes to make it possible to monitor traffic that is being sent over WEP. Of course, this is not as easy as it sounds and if WEP is used in a very careful manner, cracking it might not be possible except with high computing power.

In what follows we shall learn in more detail about the IEEE 802.11 protocol in Section 2. In Section 3 we shall look at the security features of WEP and their shortcomings. In the section after that, we shall also see how these shortcomings could be exploited to attack a system and obtain the *secret* key. Section 5 has more details of the actual implementation of the attack on a real 802.11 system.

2. IEEE 802.11

The IEEE 802.11b[11] standard contains the specification for the physical and media access layer of a wireless network. Since its release in 1997, it has been widely used for the formation of wireless networks and has become a de facto standard for the building of wireless LANs. The increased mobility that end users can enjoy has made the technology so attractive that it has replaced wired networks in many places. Also, with the proliferation of 802.11 compliant devices, wireless communication has become cheaper than wired communication in certain setups. There have also been successful attempts in using these devices, which were originally designed for indoor communication, for long range outdoor links. Apart from increase mobility and low cost, wireless networks are desirable also because of their flexibility. Setting up these networks takes up much less effort than wired ones. On the fly ad-hoc networks have also become popular and there

[1] A watcher who explores carefully in order to obtain information

has been a lot of research lately in the area of mobile ad-hoc networks or MANETS. The IEEE 802.11 protocol provides almost all of the above functionalities, and therefore has two modes of operation, an infrastructure mode and an ad-hoc mode. The protocol has been designed in such a manner that to the upper layers it appears as a standard IEEE 802 LAN. Therefore the standard TCP/IP suite of protocols can be used over the MAC layer of the 802.11 standard.

The protocol, as mentioned above, can operate in one of the two possible modes. In the *ad-hoc* mode, mobile stations form a network on the fly without any extra infrastructure. An instance of such a network could be in a meeting or a conference where people arrive with their mobile devices and share information among themselves without the need for any devices to be installed at the conference site. Such a network is known as an *Independent Basic Service Set* or IBSS in the terminology of the 802.11 standard.

The other mode of operation, the *infrastructure* mode, is more commonly used. A network in this mode consists of one or more *Access Points*(AP) which can optionally be connected to a backbone network. A mobile station will have to *associate* itself with one of these access points to be a part of this network. If more than one AP is visible to the mobile station, then it can choose to associate with any AP. The mobile node can also dissociate from an AP and re-associate with another AP when it changes its location, a handoff occurring in such a case. At any instant a station could be associated to only one AP. If the security layer is enabled then the station also has to authenticate to the AP. The security layer known as WEP is discussed in more detail in Section 3.

Once a station is part of the network it can send packets to any other node on the same network or even to the backbone network. All packets to and from a station is routed through the AP that it is associated to. In this mode, stations are not allowed to directly communicate with each other. Access Points have functionalities similar to that of the stations, plus some additional ones like bridging, etc. which enable it to forward packets either to other APs or to the backbone network. Further details about the protocol and the architecture of the system can be found in the IEEE 802.11 standard specifications manual.

3. WEP AND ITS INSECURITY[5, 12]

WEP which stands for Wired Equivalent Privacy, is the security mechanism in the link layer of the 802.11 protocol. It was intended to provide as much privacy and security on a wireless medium as there is in the physical security inherent in a wired medium. The idea was, again, to make this protocol look like a standard Ethernet based LAN to the upper layers. The entire privacy hinges on a secret key which is known only to the users of the network. The knowledge of the key in the wireless domain, therefore, is analogous to physical access to the network wire on a wired local area network. A user who does not have physical access to the wire in a wired network would not be able to connect to it. Likewise, a user who does not possess the key would be unable to connect to the wireless network.

3.1. Properties of the WEP algorithm

The IEEE 802.11 standard specifications mentions the following properties of WEP:

- *It is reasonably strong.* The security afforded by the algorithm relies on the difficulty of discovering the secret key through a brute force attack. This is in turn related to the length of the secret key and the frequency of changing keys.
- *It is self-synchronizing.* WEP is self-synchronizing for each message. In other words, nodes do not have to share state information in order to understand the encrypted ciphertexts sent by the other nodes and if a couple of packets in between are missed, it does not affect the operation of the algorithm.
- *It is efficient.* The WEP algorithm can efficiently be implemented either in hardware or in software.
- *It is optional.* The WEP security can be turned off or on, as and when desired.

Most of these properties are accurate enough, except that the security provided is not as much as it should have been, primarily due to the combination of certain sound cryptographic primitives in insecure ways. We will visit the exact details of the weaknesses in the WEP algorithm in Section 3.2

3.2. The theory of its operation[11]

Crucial to the operation of WEP is a stream cipher[10]. Stream ciphers possess certain features that sometimes make them the preferred mechanism of encryption. The stream ciphers are synchronous, since their key bit stream is completely determined by the random seed, which is the key. These ciphers are more amenable to formal and mathematical analysis and their encryption speed is much higher than that of block ciphers. In the WEP algorithm, a particular stream cipher known as the RC4 is used for generating the key stream.

For discussing the cryptographic algorithm[5, 14], we start off with assuming that both the nodes which would like to communicate know the secret key, say k. Now, depending on the authentication mechanism, the station authenticates to the Access Point (AP). The standard specifies two kinds of authentication schemes, open system authentication and shared key authentication. The former is a trivial authentication algorithm that effectively does not provide any security. The latter uses the WEP algorithm and requires that WEP also be enabled when it is used. It is essentially a challenge response protocol in which the AP throws a plaintext as a challenge and the station encrypts it according to the WEP algorithm, using the secret key, k and sends it back to the AP which checks if the challenge had been encrypted correctly. Once the station has authenticated to the AP, or if this is an instance of a communication between two APs, packet exchange can now begin.

Suppose the plaintext message that the sender wants to send is M. The sequence of steps that occur in the process are as follows Sender:

- First, compute the checksum $C(M)$ of this message, M and append it to the message. The checksum algorithm is independent of k.

- Choose a random number, v of a fixed length and prepend it to k. Call this random number the Initialization Vector or IV.
- Compute the RC4 keystream using this combined key, $< v, k >$.
- Compute $< M, C(M) > \oplus RC4(< v, k >)$. This is the ciphertext, K. Transmit this and the IV, v over the wireless link.

Receiver:

- Knowing v(from message) and k(shared), compute $RC4(< v, k >)$
- Compute $K \oplus RC4(< v, k >)$ to obtain $< M', C' >$
- Check if $C' = C(M')$. If it is then accept the message M', otherwise drop the packet.

The particular checksum algorithm used in the standard is the CRC-32 algorithm and is of 4 bytes. The size of v is usually 24 bits and the size of k is either 40 bits or 104 bits, depending on the mode that the system is running in.

3.3. Problems with WEP and related attacks[7, 5]

The security issues that WEP addresses can broadly be divided into three categories, namely privacy, data integrity and access control. Unfortunately, it seems, none of these issues are successfully addressed by WEP. Its loopholes are as follows.

With respect to Privacy:

- *We can obtain more than one packet with the same key stream used for encryption and mount a dictionary based attack.*
  Key Idea: $(A \oplus X) \oplus (B \oplus X) = A \oplus B$
  Why is an IV used at all? Suppose if we didn't then all ciphertexts would have been decrypted through the same key. Instead of decrypting the packets, if one simply obtains the XOR of two encrypted packets, he would obtain the XOR of the plaintexts, as shown in the equation above. Then by some statistical analyses, he could infer what the plaintexts were. Quite often the IP packets contain redundant data, and predicting its headers is possible.

  So the designers of WEP included this IV, so that the stream keys are not generally identical for two packets. Unfortunately the IV space is typically only of 24 bits, and so after 2^{24} or 16 million packets we would have to repeat the IV. Thus it is practical to obtain many ciphertexts with the same IV and hence the same keystream cipher. Moreover even if the same machine is not reusing the IV, other machines might be using it at the same time. Also the IV values are not that random. In the worst case each access point might reuse its IV after around 5 hours. So obtaining packets with same IVs and therefore same key streams is still easier. All this would be passive. We could even mount an active attack where we send some text that we are aware of. We then sniff the ciphertext off the air.

  After statistical analysis, when we get the plaintexts, we automatically get the key stream corresponding to a particular IV as well. We can create a dictionary of IVs and key streams. This dictionary is enough for our purposes, although we do not know the key. The size of the dictionary would approximately be $2^{24} \times 1500 \approx 25GB$, which is quite a manageable size.

  Not only this. Armed with this information, we can according to [6] also figure out the secret key, k used on the network. For this we actually do not need the entire key stream (practically around 1500 bytes - the maximum size of a packet), but just the first word of the key stream. The idea behind this is, as explained in [6], that there are classes of weak IVs which leak some information about the bytes of the key. This idea has been implemented in Airsnort[1] and even in Scout. The theoretical details of these weaknesses of the RC4 algorithm will be discussed in Section 4.1
- *Fool the Access Point (active)*:
  Once we authenticate ourselves to the network (authentication spoofing - refer to problems with access control mechanisms), we can change the destination IP field of an encrypted packet read off the air (by using the technique mentioned in the next section) to send it to one of our machines out on the Internet through the base station (or AP). We can also change the port number to 80, in case the firewall in between causes a problem. The access point will happily decrypt it and send it to our machine on the Internet.
- *Double Encryption (active)*:
  We can do the reverse of the above. After sniffing a packet off the air, we send that packet from an Internet connection to our mobile host (which is assumed to be authenticated). The AP will encrypt the packet again, effectively decrypting it. The problem with this is the usage of the same IV. Getting the timing right is quite difficult.
- *Reaction Attacks (active)*:
  We flip just two particular bits in the encrypted packet, such that the checksum is maintained, and send it to some station. By going into the details of the CRC-32 algorithm, we can convince ourselves that if the XOR of the corresponding plaintext bits was 0, then the station will drop the packet. Otherwise it will return an ACK. By observing this reaction for a whole lot of transformed packets, we can figure out which bits were flipped and therefore learn the key stream and the plaintext.
- *Bad Key Management & Another Dictionary Attack*:
  The WEP key, k, is usually required to be entered manually. So we can expect easily memorable keys to be used. Moreover, these common strings are often not even hashed to obtain a key, instead their ASCII equivalent is directly used. A dictionary of possible keys can also prove to be another method of intrusion into the network.

With respect to Integrity:

- *We can change parts of the message and fix the checksum to reflect those changes.*
  Key Idea: $C(A \oplus B) = C(A) \oplus C(B)$
  The problem arises because the CRC-32 algorithm is linear as shown above. So if I decide to change some bits of a message, say A, then I will first construct a bit sequence B,

which has 1s in the positions which I want to flip. Then I generate $C(B)$ and XOR it with $C(A)$. This again means flipping some bits in $C(A)$. The effect flipping of bits would be the same, whether it is done before the encryption of after. Therefore even if we XOR C(B) with X XOR C(A), the effect will be the same.

This attack can be used maliciously if additional knowledge of the packets are available. For example, we can change just the command bits of a telnet session packet to make the server run whatever we want.

With respect to Access Control:

- *Authentication mechanism is flawed.*
  If I sniff the packets in an authentication session, then I have both the plaintext and ciphertext for a particular IV. From this I can figure out the key stream and use this key stream to authenticate myself as well.
- *By building the dictionary as mentioned above, we can even construct packets.* In fact for this only a single IV would be enough. Once we can construct packets and also read them, even if we have no knowledge of the key, we can use this network for our purposes. The authentication protocol itself provides us with one such plaintext/ciphertext pair.

4. THEORY BEHIND THE IMPLEMENTED ATTACK[9]

Most of the attacks given in Section 3.3 were statistical in nature. Although so much has been said about the insecurity of the WEP algorithm, it is not trivial to actually crack a WEP key in practice. There are many problems involved, some of them even theoretical in nature, due to the probabilistic nature of the attack.

One has to decide what one wants from an attack. There are two possibilities. Either you would be satisfied by obtaining the capability of reading off the plaintext data from the encrypted data and possibly inject encrypted traffic into the network, without ever getting to know the WEP secret key. Or you wouldn't rest in peace till you get to know the secret key, after which doing anything would be possible.

As regards the former, a lot of techniques are discussed in Section 3.3, which try to get hold of ciphertext/plaintext message pairs. From this we obtain the RC4 stream's first couple of thousand bytes which suffice in decrypting and encrypting packets with the IV used in the pair. Most of the attacks in this category were active in nature. Active attacks are more difficult than passive ones for two reasons. One is that such attacks are more traceable, and secondly they are sometimes difficult to implement requiring fiddling with the low level driver software or even firmware on the card. For these reasons, mostly passive attacks are spoken of.

The only public domain software tools[3] for attacking WEP all use a particular passive method of procuring the secret key. All of them make use of the theoretical results given in [6]. The attack described here relies heavily on the weaknesses in the RC4 algorithm itself and without its understanding, one cannot really appreciate the details of the attack. This attack is popular for the simple reason that it is the most powerful and gets us straight to the secret key. Airsnort[1] is one such open source cracking tool which implements this attack. Our tool is an improvement over Airsnort and makes use of the same theoretical results stated in [6].

4.1. The RC4 algorithm and its weakness[11, 6]

RC4, designed by Ron Rivest in 1987, is a particular stream cipher. In other words, the RC4 algorithm takes an input a *random seed* of any length and outputs a *pseudo random number sequence*. The random number sequence acts like a key stream for the encryption algorithm of WEP as explained in Section 3.2. RC4 consists of two parts, a key scheduling algorithm(KSA) which turns a random key (whose typical size is $40-256$ bits) into an initial permutation of S of $\{0, ..., N-1\}$, where N is a power of 2 and is typically 256. The other part is a pseudo random number generator(PRGA), which uses this permutation to generate a pseudo-random number sequence. The KSA and the PRGA algorithms are shown in Figure 1.

KSA(K)

Initialization:
For $i = 0..N-1$
  $S[i] = i$
$j = 0$

Scrambling:
For $i = 0..N-1$
  $j = j + S[i] + K[i \ mod \ l]$
  $Swap(S[i], S[j])$

PRGA(K)

Initialization:
$i = 0$
$j = 0$

Generation loop:
$i = i + 1$
$j = j + S[i]$
$Swap(S[i], S[j])$
$Output \ \ x = S[S[i] + S[j]]$

Figure 1: The KSA and the PRGA Algorithms

The PRGA initializes two indices i and j to 0 and then loops over four simple operations which increment i as a counter, increment j pseudo randomly, exchange the two values of S pointed to by i and j, and output the value of S pointed to by $S[i] + S[j]$. The KSA consists of N loops that are similar to the PRGA round operation. It initializes S to be the identity permutation and i and j to 0, and applies the PRGA round operation N times, stepping i across S, and updating j by adding $S[i]$ and the next word of the key (in cyclic order). Each round of the KSA is called a *step*

We shall now look at a particular key attack that is mounted by most tools available for this purpose, including ours. We will assume that the same secret key is used with numerous different initialization vectors, as is done in WEP, and the attacker knows the first word RC4 output corresponding to each initialization vector. Armed with this the attacker can reconstruct the secret part of the key.

The attack is as follows. Suppose we know the first A words of the secret key ($K[3], ..., K[A+2]$; with $A = 0$ initially) and we would want to know the next word, $K[A+3]$. We run the KSA for IVs of the form $(A+3, N-1, X)$ for various different values of X. At the first step j is advanced by $A+3$, and then $S[i]$ and $S[j]$ are swapped, resulting in the key setup state shown in Figure 2. The array on the top is is the secret key prepended with the IV and the array below is a window into the permutation where the pointers i and j are shown.

Then, on the next step, i is advanced, and then the advance on j is computed, which comes out to be 0 (modulo N). Subsequently, $S[i]$ and $S[j]$ are swapped, resulting in the permutation as given in Figure 3.

Following this, at the next step, j is advanced by $X + 2$, which means that for each X, j is assigned a different value and for all further steps in the KSA, each IV acts differently. Since the attacker knows the value of X and $K[3], ...K[A+2]$, he can compute the exact behaviour of the key setup until he reaches step $A+3$. At this point, he knows the value of j_{A+2} and the exact values of the permutation S_{A+2}. If the value at $S_{A+2}[0]$ or $S_{A+2}[1]$ has been disturbed, the attacker discards this particular IV. Otherwise, j is advanced by $S_{A+2}[i]$ + $K[A+3]$, and then the swap is done, resulting in the permutation given in Figure 4.

The attacker now knows the permutation S_{A+2} and the value of j_{A+2}. In addition, if he knows the value of $S_{A+3}[A+3]$, he knows its location in S_{A+2}, which is the value of j_{A+3}, and hence he would be able to compute $K[A+3]$. We also mention here, without proof, that in such a condition the attacker can obtain the correct value of $K[A+3]$ with a confidence level of 5%. By observing enough IVs with the above configuration, the attacker can get to the value of $K[A]$ with a fair amount of confidence. What is given here is a brief idea of the attack and for the interested reader, further details about the weakness of the RC4 algorithm and related attacks can be found in [6].

It is worthy to note, that the IVs that reveal key bytes are not just the kind that have been mentioned here. In fact, any IV of n words that, after n steps, leaves $S_n[1] < n$ and $S_n[1] + S_n[S_n[1]] = n + B$ will be good enough for the above attack. Here B is that byte of the secret key that we are seeking. Such IVs are more popularly known as *weak* IVs. IVs of the form $(A+3, N-1, X)$ form a subset of the set of *weak* IVs. Thus if one has less number of IVs to go by and has enough computing power, then he can extract information from all such IVs.

5. IMPLEMENTATION DETAILS OF THE ATTACK

The actual implementation of the attack, starting from the streamlining of the operation of collecting the packets to getting to the actual secret WEP key involved a lot of effort. Though theoretically the attack seems quite feasible, there are quite a few practical difficulties. The major difficulty lies in the fact that the cracking operation is highly probabilistic and one can never be sure if he has enough packets to arrive at the key. In the following sections we shall look at the specific implementation issues that came up during the development and testing of the tool.

5.1. Sniffers

For the purpose of collecting enough packets, a packet sniffer for wireless interfaces was required. There are a host of such sniffers available, quite a lot of which are open source as well. Sniffers are nothing but programs which put the network device, on which they are sniffing, into what is known as the *promiscuous* mode. In the promiscuous mode, a network device will read all packets on the medium whether they are addressed to it or not. For wireless network cards, this is typically known as the RF monitoring mode. Although, any of these sniffers could have been used for our purposes, a sniffer customized for our needs would be more efficient and sufficiently flexible. Moreover, with the availability of a packet capture library such as `pcap` writing a sniffer of our own is quite easy. An additional issue for wireless cards is that putting them into the RF monitoring mode requires support from the driver. We used the Linux `wlan-ng` driver[15] along with a wireless card with a Prism2 chipset as this combination was the most convenient among others.

5.2. Description of Scout

Scout, the tool we have developed for cracking WEP keys, has two components. One is the sniffer which can be deployed on a separate machine to incessantly listen for packets on the wireless medium. The sniffer logs three pieces of information corresponding to each encrypted data packet. They are, the Initialization Vector used to encrypt that packet, the first byte of the encrypted part of the packet and the BSSID of the AP to which this packet belongs. All packets which do not possess these pieces of information, possibly because they are not encrypted at all, are simply ignored. These three information fields add up to 10 bytes per packet. Therefore only 10 bytes of data is kept for each packet and the rest is discarded. This ensures that a considerable amount of relevant data can be collected without using too much disk space. On a 10Mbps connection used under full load, assuming an average packet size of 1KB, a maximum of 1GB of disk space would be used up if the sniffer is left running for a day. Other than this information, some number of full encrypted data packets corresponding to each BSSID are also stored, so that they can be later used to verify the secret key. In our tool, we store upto 10 encrypted packets for each AP in the network.

The mechanism for verifying a secret key, once you have the packets, is quite straightforward. All that one has to do is to *decrypt* a packet assuming that the given key was the one that was actually used for encrypting it and check whether the packet checksum is correct. If it is, then it is very likely that the given key was used for encrypting this packet. The same operation could be performed on a few more packets and if the results concur, then we can be pretty sure that the key is correct. If indeed that was the key used for encryption, then for all the packets the checksum should match correctly. In our case, where we check against 10 different packets, the probability that an incorrect key is passed off as correct is too small to be of any significance.

The second component reads the information that the first part has dumped into a file, for its processing. It first separates out the information for each BSSID and also checks for the *weak* IVs. The

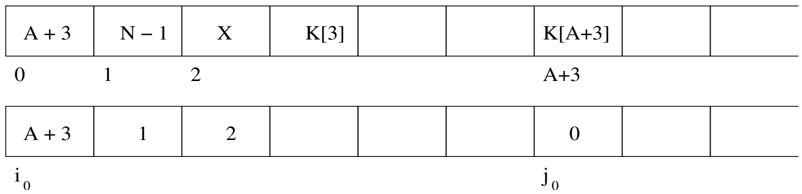

Figure 2: State of the permutation after the first step of KSA

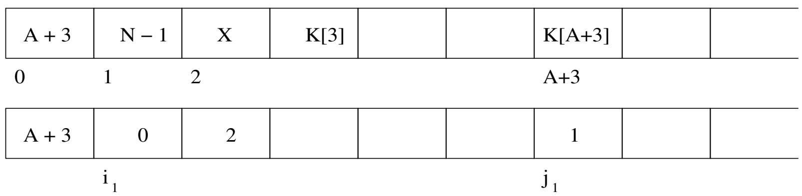

Figure 3: State of the permutation after the second step of KSA

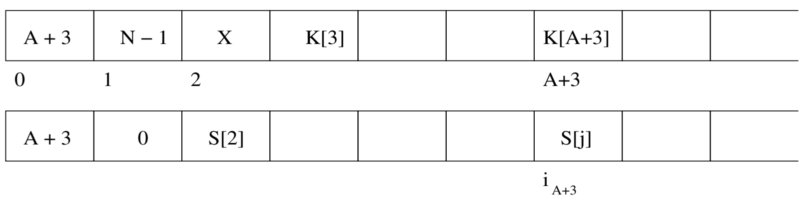

Figure 4: State of the permutation after the $A + 3^{th}$ step of KSA

criteria for the weakness of IVs is described in detail in Section 4.1. This tool actually operates in one of the two possible modes. In the faster mode, the set of IVs recognized to be weak is a subset of all the weak IVs. The testing of such IVs can be done with very little computation and is used by Airsnort [1] to check for IV weakness while sniffing packets itself. Such IVs can also be classified as to which key byte they reveal. In the other *enhanced* mode, the exact criterion for weakness as given in end of Section 4.1 is used. As the program reads the file, it builds the data structures which systematically keep track of such information as which IV would reveal information about which byte and for which AP. No information is gleaned from the packets for which the IVs are not weak. After the file has been read, the structures are all ready to be used for the cracking procedure. The advantage of having two different components is that they can be run on different machines. Existing tools have both these functionalities combined into one program. This would not be appropriate when we would like to run the sniffer on a small Linux box with high speed network cards but low computational power. With our design, it would be possible to run the sniffer on a separate machine that could be placed anywhere which would dump that data into a central location. From that location then, the cracker running on a high performance machine can read off this data and crack the key. In many cases, especially while cracking the 13 byte key of WEP, the time required for cracking might be very high and might run into weeks. For such instances, we can also run the cracker on a computational grid that might be geographically distant from the site where the packet data is collected. This would be possible since the rate of output of information from the sniffer is low enough that the data can be sent over the Internet. Scout, therefore, is a potentially powerful tool that can be used to crack virtually any 802.11 network secured using WEP.

The cracking procedure is the same as discussed in Section 4.1. To obtain the first word (a byte in this case) output of the RC4 key stream, we need to know the first byte of the data part of the unencrypted packet. If we XOR that with the first byte of the encrypted packet, we get the first byte of the key stream. Incidentally, all 802.11 data packets begin with the SNAP header the first byte of which is $0xAA$. So knowing this we can always infer the first byte of the RC4 output.

The cracking algorithm which makes use of results from Section 4.1 is shown in Figure 5:

The function `checkKey` checks if the key is correct using the mechanism for checking keys given earlier and returns SUCCESS if the key is correct and FAILURE otherwise. The `suggestForIV` function takes in an IV, checks for its weakness and returns the value of the key byte it suggests, using also the first word output of the corresponding RC4 key stream output. This value is correct around 5% of the time. In the *enhanced* mode, each IV collected is checked for weakness on the fly whereas in the faster mode the IVs would be statically checked for weakness and classified depending on which key byte they reveal. The function `sort` sorts the input array using the frequency value for defining an ordering among the elements. The function `breadth` returns the search breadth corresponding to a particular key byte and can in general depend on how much information

```
TryByte(whichByte, guess, keyLength):

if(keyLength==whichByte)
  return checkKey(guess, keyLength);
for i in 0..255
begin
  bytes[i].index=i;
  bytes[i].frequency=0;
end
for each weak IV, v
begin
  suggestedByte = suggestForIV(v);
  bytes[suggestedByte].frequency++;
end
/* sort using the bytes[i].frequency */
sort(bytes);
for i in 0..breadth(whichByte)
begin
  guess[3+which] = bytes[i].index;
  if(TryByte(whichByte+1, guess, keyLength)
      == SUCCESS)
    return SUCCESS;
end
return FAILURE;
```

Figure 5: The Cracking Algorithm

is there to reveal that byte. TryByte is a recursive function and is initially called as `TryByte(0, guess, keyLength)`, where keyLength would either be 5 or 13 depending on the mode that WEP is being used in. When the function returns a SUCCESS, the parameter `guess` would contain the secret part of the key.

The above procedure is almost the same as the one used by Airsnort[1], except for the breadth parameter for each recursive call being a variable in our case, instead of a constant value as in Airsnort. This breadth parameter can be chosen intelligently for each key byte, depending on how much information has been revealed for that byte. We have used a heuristic for the breadth parameter which works well for us but which can be further tuned for other setups. If the number of IVs that reveal information about this byte is greater than 25, we let the breadth remain as it is. If it is between 8 and 25, we set the breadth equal to the number of IVs. This means that all the bytes suggested by some or the other IV is checked for correctness. If the number is below 8, then we conclude that enough information is not available for this byte and we perform a blind search *only* for this byte. Such an arrangement would especially be useful in cases when the packets do not seem to be enough to crack the key and a somewhat blind search has to be performed on particular key bytes only. The design of Scout is also more suited for such cases, in which the cracker which requires high computational power can run on a different machine than the one which is sniffing.

The tool was tested by running it on a Compaq laptop with the Linux Operating System. Two wireless cards were plugged into this machine, with one of them sniffing packets from the air and the other injecting packets into the wireless network. Netperf[13], a network throughput analyzer was used for injecting packets into the network. For the 5 byte key, packets collected over a period of 4 hours was enough to crack the key in a few seconds. For the 13 byte key, packets collected for around 15 hours was enough to crack the key in about 50 hours.

6. CONCLUSION AND FUTURE WORK

Considering the performance of our tool, it is clear that the security offered by WEP can be bypassed with little effort. WEP, therefore, is not a complete solution to protect wireless networks from attacks. Moreover, it has been noticed that most wireless networks do not even have WEP enabled simply because it is not enabled by default. The manual key exchange and key management involved brings in a human element and this increases the scope for a compromise in security. Nevertheless, even a proper key management does not ensure complete safety against attackers. With reasonable amount of computational power most wireless network could be broken even if they are careful in configuring its security using WEP. For instance, the cracker part of the tool can be comfortably parallelized and run on a cluster or a grid. Such a mode of operation would lead to a dramatic decrease in the time required to obtain the secret key of a network. Parallelizing the cracking algorithm is conceptually simple. All that has to be done is to break up the search space of keys into disjoint parts and search for the key in each of them parallely. Since the searches in each of them are independent of each other, the communication between the parallel threads is practically zero. Parallelizing the algorithm is something that has still not been attempted actually and is a promising lead that might conclusively demonstrate the complete insecurity of the WEP algorithm.

Keeping the insecure nature of WEP in mind, a new 802.11X[4] standard has been proposed that is aimed at providing enhanced security[2]. This standard makes use of user authentication and dynamic key exchange. This is currently supported using the Extensible Authentication Protocol (EAP)[2] available in various Remote Authentication Dial-In User Service (RADIUS)[2] implementations. In this authentication method, with the help of a centralized EAP/RADIUS server a different set of encryption keys is negotiated for each session. Therefore, if the key is at all leaked, only the data of that session is vulnerable. Other standards are also under review but none of them have been actually approved as yet. Meanwhile, the best security solution is to encrypt at higher layers. End to End Security by using Public Key Infrastructures (PKI) or the use of Virtual Private Networks (VPN) are currently the safest approaches to protect a wireless network against attacks and they would be so for at least a few more years.

7. REFERENCES

1. Airsnort. http://airsnort.shmoo.com/.
2. The Unofficial 802.11 Security Web Page.
3. Wireless Technology Comparison.
4. 3COM. 802.1X Security - Designing a Secure Network.
5. Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html, 2001.
6. Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. *Lecture Notes in Computer Science*, 2259, 2001.
7. Ian Goldberg. The Insecurity of 802.11.

8. J. Haartsen, M. Naghshineh, J. Inouye, O. Joeressen, and W. Allen. Bluetooth: Vision, Goals, and Architecture. *Mobile Computing and Communications Review*, October 1998.

9. David Hulton. Practical Exploitation of RC4 Weaknesses in WEP Environments, February 2002.

10. Michael R A Huth. *Secure Communicating Systems: Design, Analysis and Implementation*. Cambridge University Press, 2001.

11. IEEE Computer Society LAN MAN Standards Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Std. 802.11, 1997.

12. Intel Corp. Overview of ieee 802.11b security.

13. Rick Jones. Netperf. http://www.netperf.org/.

14. Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP.

15. Absolute Value Systems. Linux wlan-ng. http://www.linux-wlan.com/linux-wlan/.