Wireless Security: from WEP to 802.11i.

Lucas Hendrickson[1]
Department of Mathematics and Computer Science
University of Wisconsin-Superior
lhendric@students.uwsuper.edu

Victor Piotrowski
Department of Mathematics and Computer Science
University of Wisconsin-Superior
vpiotrow@uwsuper.edu

Abstract

Recent IT surveys indicate that the greatest barrier in deploying wireless LANs is security. While an adversary needs a physical access to a wired network, this is not necessary in a wireless LAN. Therefore, wireless communication standards include security features at the data link layer. In particular, the 802.11 specification defines Wired Equivalent Privacy (WEP). WEP was design at a time when the strong cryptographic algorithms were subject to very prohibitive export laws. This and other factors resulted in inherently flawed security scheme.

In this paper, we discuss many WEP vulnerabilities such as poor key management, message injection, and authentication spoofing. Although there are programs such as AirSnort or WEPCrack, we have decided to write our own Windows-based cracking application in C#. We use it to demonstrate brute force attacks and FMS attacks. FMS attacks are based on the concept of weak-keys as described in 2001 paper "Weaknesses in the Key Scheduling Algorithm of RC4" by Fluhrer, Mantin, and Shamir [6].

In the final part of the paper, we discuss recent efforts of hardware vendors to improve wireless LAN security. In particular, there are several firmware upgrades which eliminate FMS weak keys and improve key management so Initialization Vector collisions are less frequent. Originally called WEP2, the Wi-Fi Protected Access (WPA) scheme is supported in popular wireless products released in the second half of 2003. We discuss its improvement over WEP, relation to the future 802.11i standard, challenges in its implementation, and possible attacks.

---

[1] Student presenter.

Introduction

With the coming of 802.11 wireless networks, wires that were once everywhere have now disappeared, to be replaced by radio waves traveling in the air. Anyone who owns a laptop with a wireless card can attest to the convenience and "wow" factor of surfing the internet without a single wire hooked up to anything. Yet, as most things are in life, there is a downside. In a traditional wired network, an attacker needs physical access to the network to intercept traffic or to launch attacks. Consequently, we have a concept of physical security in the sense that we can lock up computers and the wires that connect them from unauthorized users. In wireless networks, we don't have physical security, but it is common to think that wireless signals are contained inside the walls of a building. You may even try to confirm this by walking outside with a laptop or PDA with a wireless card and seeing that you no longer get a signal. However, it is easy to construct a more powerful antenna at a low cost that will greatly increase the range of wireless devices. There are even reports of very capable antennas capturing traffic from miles away. This poses a very serious security risk and should be a major consideration when installing a wireless network. In reality, most wireless networks that are deployed are left unprotected. There are even databases [11] of these unprotected networks on the web. For those who want to secure they wireless networks, 802.11 provides a way to encrypt transmissions. Unfortunately, the encryption available has its own severe weaknesses. In this paper we discuss some of those known weaknesses and implementation of known attacks.

Weaknesses

The 802.11 standard method of encryption is called Wired Equivalent Privacy or WEP. WEP was designed to encrypt traffic, and uses the Rivest Cipher 4 (RC4) for its encryption engine. WEP was not designed to be a complete end-to-end security solution. In fact, it allows anyone who knows the shared key to have access to all of the traffic on that wireless network. This has some similarities with a wired network. Hence the term Wired Equivalent Privacy.

RC4 is a stream cipher. It works by producing a stream of pseudorandom bytes which is then XORed with a plaintext to produce a ciphertext. The WEP implementation of RC4 requires two parameters. First, it needs a key of a certain length, typically 40-bit or 104-bit. Secondly, it requires a 24-bit number called the initialization vector (IV) that is attached as a prefix to the key. The IV is used to produce variability in the encrypted output, so that different packets use different keystreams. Immediately we can notice problems. First, the size of the IV is too small. We are limited to only 16777216 possible values, which can be consumed in a busy network in a matter of hours. It is common to see packets with the same IV. This is called an IV collision. What it means is that that we have two packets that are encrypted with the same keystream. Knowing parts of the plaintext, we can reasonably guess what the keystream is. The more IV collisions we have, the better guesses we can make. If enough IV's and repeats are collected, it is possible to create a dictionary of keystreams for each IV allowing someone

to decrypt all traffic without ever knowing the key. Secondly, what makes matters worse is that the 802.11 standard has no implementation details on how IV's are generated. It is recommended, but not required, that they are changed for each packet. In reality, most wireless cards do in fact change the IV for each packet. With the equipment we had at our disposal, we noticed this behavior. Our older hardware would initialize the IV to 0 upon power up, but our newer equipment would start with different values.

As mentioned before, RC4 is a stream cipher so it is vulnerable to so called message injection. If you know the plaintext as well as the ciphertext for a particular message, then an XOR of the plaintext and ciphertext will produce the keystream. If we know a keystream for a particular IV, we can encrypt our own plaintext and send a forged ciphertext.

In WEP, authentication is optional and, if enabled, is based on a challenge and response protocol. An access point will send a 128-bit value to a client. The client then encrypts this value using the shared key and sends it back. This mechanism of challenge and response gives an attacker a plaintext and ciphertext which is sufficient for the message injection attack mentioned above. Therefore, we are able to generate the correct response for any challenge sent to us by an access point. There is also another method for authentication used by most access points. It is based on MAC addresses, but considering how easy it is to spoof a MAC address, this is not a very effective way to keep determined people out of your network. In WEP, authentication is done on the client only and there is no way to authenticate an access point. Therefore, we can connect to an attacker's access point that claims it is a member of the network. We call these rouge access points.

When transmitting wireless packets, we also want to make sure that they arrive without modification. This is called integrity checking. This functionality is provided by a Cyclic Redundancy Check (CRC). CRC is used mainly to protect against a transmission error. It does little to protect against malicious modification by an attacker. In fact, Borisov et al. [3] showed that if an attacker knows the complete ciphertext and the desired modification on the plaintext, the linearity of both the CRC and RC4 allows the attacker to modify the ciphertext and the CRC so that it will be accepted.

The above mentioned WEP vulnerabilities allow message forging and spoofing, but cannot be used for a successful attack on the key. The attacks on the key itself can be done by brute force for a 40-bit key in a matter of days but for a 128-bit key or longer it is impractical. However, there is a method of attack that works for any key length in a reasonable amount of time. That weakness is described in the seminal paper *Weaknesses in the Key Scheduling Algorithm of RC4* by Fluhrer, Mantin, and Shamir [6]. This weakness combined with WEP's use of the IV produces a very serious flaw. Attacks based on this flaw are called FMS attacks.

FMS Attack

The problem discovered in [6] was in the key scheduling phase of the RC4 algorithm. They identify several classes of weak keys which reveal information about the internal state of the algorithm. The most relevant to our discussion is when an IV is prepended to a key, which is how WEP uses RC4. We call an IV "resolved" or "weak" if it is of a certain form. An example of a resolved IV is any key of the following form [B+3, 0xFF, X] where B is a byte number of the key and X is any value.

This weakness is not a reflection on the RC4 algorithm and it is still widely used and considered very secure (the most popular application of RC4 is SSL). RSA recommends [10] that while using RC4 you discard the first 256 bytes of the keystream. If WEP did this, the FMS attack would not work.

The weakness stems from the fact that the IV and the key are used as seed values for the pseudorandom number generation. Because the IV is sent as plaintext, an attacker knows enough to perform the first several stages of the RC4 algorithm. There is a chance that once the algorithm has completed, these values will not change. Knowing this and the plaintext of the first byte of encrypted traffic, we are able to guess the bytes of the key 5% of the time [6]. To do this successfully, we need to gather several weak IVs and calculate the predicted key byte values. We take all of these guesses, and the most frequently appearing values should be the bytes of the key.

Fluhrer et al. described the attack in general but did not implement it. They made this clear in their paper: "Note that we have not attempted to attack an actual WEP connection, and hence do not claim that WEP is actually vulnerable to this attack." However, in a paper published a month later in August 2001, Stubbfield et al. [8] described their successful implementation of the FMS attack. They reported that the attack took a week in total to implement, from purchasing the equipment needed, coding the attack, generating the traffic, and eventually breaking the key.

Later that same month, perhaps the most popular cracking utility based on the FMS attack, AirSnort, was released to the general public as an open source project. Originally a console based program, it evolved to include a graphical interface and became easier to use. Currently, http://airsnort.shmoo.com/ is the home page for the AirSnort project.

Hardware preparation

The first step towards breaking a WEP key using the FMS attack requires capturing encrypted wireless packets. When a wireless card receives a signal from some source, it will ignore the packet if its destination is for another MAC address or if the packet is encrypted and the card does not know the correct key. However, we need to collect these normally discarded packets to successfully attack the key. To do this requires both a card that has the capability for raw packet capture as well as the drivers that will make this data available to the application layer. The companies that sell wireless cards do not include drivers that will allow this raw packet capture (often referred as putting a card into "monitor mode"). The most popular cards for this raw data capture are based on either the Orinoco or the Prism 2 chipset. The only free drivers available for these cards

are for Linux. There are a few companies that sell wireless monitoring programs that come with their own proprietary drivers that work under Windows, but the cost for this software is several thousand dollars and was therefore outside of our budget.

Finding cards that did have the required chipset proved more difficult than originally planned. Most manufactures do not necessarily display the chipset of their device in an easy to find location. Also, the manufacturers will often change the chipset while keeping the same model number and changing only the revision number. One site that we found helpful for determining the chipset was http://www.linux-wlan.com/ which had a list of 802.11 cards and displayed what chipset is used for each model. This is also the location to find monitor mode drivers for Prism based chipsets.

Luckily we had an older wireless card based on the Prism 2 chipset, specifically a D-Link DWL-650 revision 2I. We ran into difficulties trying to configure this card in low end Dell Optiplex GX1 running Fedora Core 1 with a PCI to PCMCIA bridge. As a result, we decided to try a different wireless card, Intel PRO/Wireless 2011B LAN PCI Adapter. While this card was also based on the Prism 2 chipset, we could not put the card into monitor mode. To our surprise when we tried using the Orinoco drivers from the AirSnort website, we were able to capture encrypted traffic. Later we discovered that it was not uncommon for other users to capture traffic by swapping the drivers.

Now that we had a capture working, we decided to test if we could crack a key using AirSnort. We set AirSnort to run and created traffic by having another computer that knew the encryption key ping flood the access point. AirSnort's interface keeps a count on how many packets have been captured, how many of them are encrypted, and how many are weak. In all the attempts we initially tried, AirSnort would report finding between 5 and 15 weak packets within the first few minutes, and regardless of how many packets it would capture after that, it would not report finding any more weak keys. This is surprising as we should see weak packets in a proportional relation to the capture time.

Discouraged, we attempted a dual capture of the same data using a demo version of the commercial product AiroPeek [12] in Windows using a D-Link DWL-G650 revision B2 and Ethereal in Linux using the Intel card. Upon comparing the two captures we found a discrepancy. The packets in Ethereal were all 12 bytes shorter than the ones captured by AiroPeek. Comparing corresponding packets from both captures we discovered the type of packet, the source and destination addresses, packet sequence and fragment numbers were identical. However, the IV and other parameters were different. Upon closer inspection where we expected to find the IV we instead saw the first bytes of encrypted data. The IV was missing. This lost information was essential for the breaking process.

Installing the Intel card on a Dell Dimension XPS T800r running SuSE 8.2 produced the same results. However, installing the D-Link DWL-650 with our PCI bridge on the same system provided us with better luck. The PCI bridge and the wireless card were detected and worked fine. Attempts to install the Prism drivers proved successful and we were able to capture encrypted traffic. To confirm that we were getting all of the bytes of the

packets, we again used AiroPeek on one computer and Ethereal on our Dimension. The packets were the same in both captures.

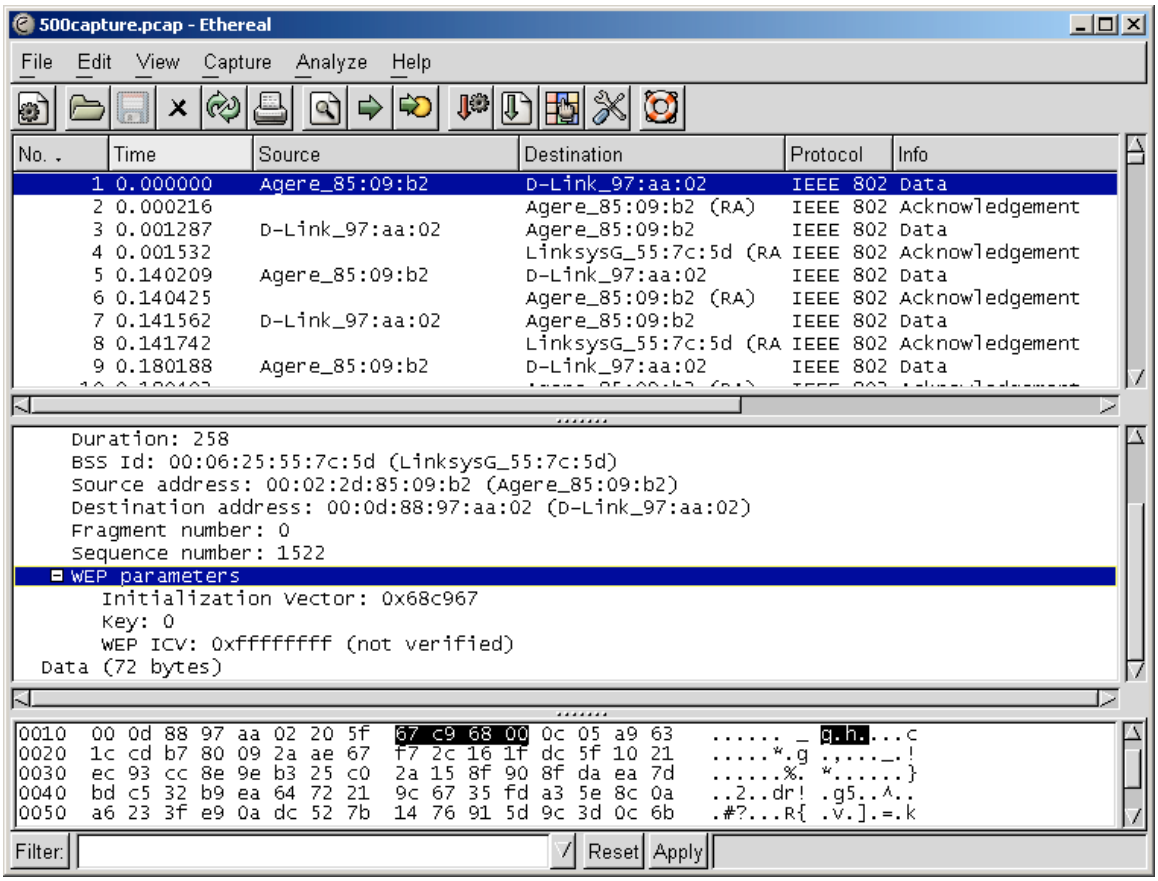

Figure 1: Ethereal Capture of Wireless Traffic

Generating the traffic was not too difficult. The problem was to generate many different IV's to capture. While our solution was not the most elegant, it did do the job needed. We opened as many ping windows as we could. In a Dell Inspiron 8200 (2Ghz, 1GB RAM) the limit seemed to be a little over 450 ping windows, so we settled for around 430. The maximum rate captured achieved at 430 ping windows was 380 per second. To make our program simpler we only captured packets from one source. Consequently, it took about 14 hours to create a capture file of 17,000,000 packets.

Figure 2: Ping in action

The Program

Once we had access to the raw wireless capture files, we could start writing a program implementing the FMS attack. The program was written in C# using Visual Studio 2003 .NET.

All of the capture data was stored in pcap files. Determining the structure of the pcap file was straightforward. It consisted of a 24 byte file header, and a 16 byte header for every packet. An important field in the packet header was the size of the captured packets.

Once the ability for reading the capture file was completed, we needed functionality to first determine if the packet was encrypted and then to verify if the IV for the packet was weak. This functionality was included in the Packet class. Upon finding a weak key, we saved the IV and the first two encrypted bytes into a two-dimensional array of EncryptInfo objects.

The next important piece of the program was to write the RC4 encryption algorithm. Since, the algorithm is symmetric, writing the code to encrypt data is the same as writing the code to decrypt it. Although the RC4 algorithm is considered a trade secret of RSA Labs Inc., it was anonymously released to the public back in 1994. Now it is easily available in several books and papers. We confirmed the correctness of our implementation with a small test.

At this point, the program did not have the functionality to attempt the FMS attack, but we had the ability to test a brute force attack. Brute force attacks on a 104-bit key are infeasible even on computer clusters. Brute force attacks on a 40-bit WEP encryption can take many days to produce a key. We ran a small test where we restricted the key characters to lower case ASCII. A brute force attack did work.

Finally it was time to implement the FMS attack in our program. FMS discuss the attack in abstract form. While this is good for theoretical understanding, it is not helpful when attempting to implement the specific details. Fortunately, we did find a specific example of an IV and key that resolves a key byte in the book *Wireless: Maximum Security* [5]. This was extremely helpful for testing.

Learning from our past mistakes we made captures that we could use for the FMS attack. With the proper capture files, we then tested our program and broke the key first for 40-bit encryption and later for 104-bit encryption. Capturing 17 million packets takes several hours, but the actual cracking process takes only a few minutes.

The program itself uses six classes for storing data, performing encryption, and for the implementation of the FMS attack. To start the cracking process, a user needs to select an input and output file. Next, the program reads the file and collects instances of weak packets. Once all weak packets in a capture have been identified and organized, the FMS attack can be attempted. If the key is found, it displays it in both hexadecimal and the

corresponding ASCII; otherwise, the program displays a message saying the key was not found. In this case, calculated but incorrect key will be displayed.

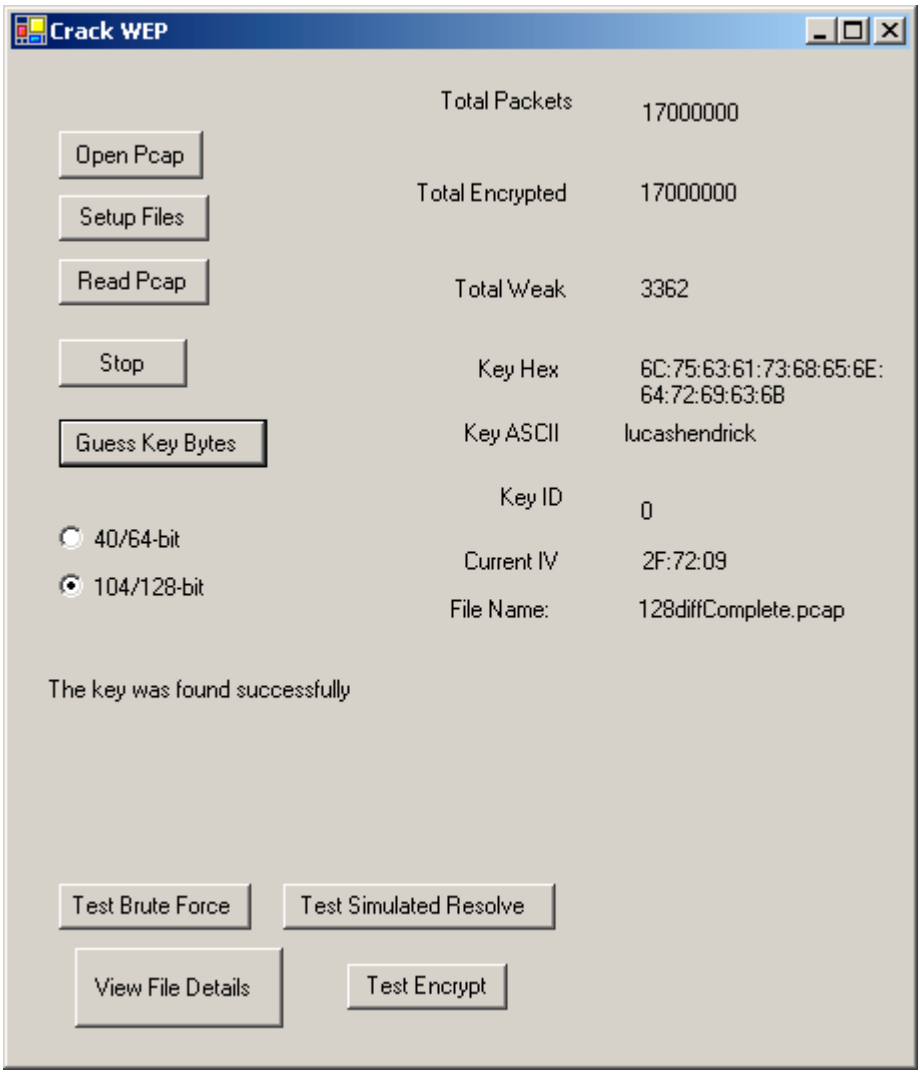

Figure 3: Reading 104-bit Capture

WPA and 802.11i (RSN)

After 802.11b was released, wireless networks became very popular. While WEP has its flaws as described earlier, it works well enough for small private networks where the risk is lower and updating the keys is easier. However, with a company that has hundreds of employees, changing keys becomes much more complex. This is why many wireless networks are used without WEP or with infrequent key changes. While there were proprietary extensions to WEP to solve these problems, the Institute of Electrical and Electronics Engineers (IEEE) is currently working on a new standard whose goal is to improve the security of wireless networks. This standard will be called 802.11i, also known as Robust Security Network (RSN).

In the first half of 2001 there was an overwhelming critique of WEP with the publishing of several papers [2, 3, 6]. An interim fix was urgently needed. Thus the Temporal Key Integrity Protocol (TKIP) was created by the 802.11i working group as a way to fix the security problems of WEP. Approving a new standard is a time consuming process, but TKIP needed to be introduced as quickly as possible. Therefore, the Wi-Fi Alliance, an organization of wireless manufactures that deal with issues of interoperability between vendors, introduced Wi-Fi Protected Access (WPA) that could be implemented with those technologies of 802.11i that were available as of late 2002. So, WPA is basically a subset of the functionality that will be available in the future 802.11i standard. The first devices that complied with WPA started appearing in late spring of 2003.

TKIP is basically a wrapper for WEP that fixes most of WEP's weaknesses. To eliminate IV collisions, it expands the IV space to 48 bits. With the new size, it would take thousands of years to completely exhaust the IV space. TKIP also uses this newly expanded IV space as sequence numbers to protect against replay attacks. If a host receives a packet that has an out of sequence IV, it is dropped. At the same time, weak IVs are also removed to prevent FMS based attacks. TKIP also includes what is called Per-Packet Key Mixing. What this basically does is change the encryption key for every packet based on the IV, a Pair-wise Master Key (PMK) known only by a client and an access point, and other parameters. Since the PMK is unique for every client, two people connected to the same access point cannot decrypt each others traffic. The CRC is also replaced with the Michael algorithm that uses a hash to create a more secure integrity check value for protection against message modification. In WPA, TKIP is required and Advanced Encryption Standard (AES) is optional. The future standard 802.11i will require the use of AES encryption but will allow the optional use of TKIP.

WPA also includes provisions for vastly improved authentication through 802.1X port based authentication combined with a Remote Dial-In User Service (RADIUS) running with one or more versions of the Extensible Authentication Protocol (EAP) to verify the identity of a client. In WEP, we only authenticate the client, but in WPA, the access point must be authenticated as well to eliminate the problem of rouge access points.

WPA is not a standard and in some aspects is not much more than a guideline [9]. For instance, WPA uses EAP, but does not specify what EAP implementation to use. For example, Cisco uses its Light EAP (LEAP), and Microsoft uses it Microsoft Challenge Handshake Authentication Protocol version 2 (EAP-MS-CHAP v2). Hopefully, once 802.11i is complete these issues will be resolved.

While WPA will work its best with the supporting infrastructure, the average user does not have a RADIUS server, so the concept of WPA-Preshared Key (PSK) was created. You enter a passphrase and the hardware or software takes care of the details. However, it has been suggested [13] that PSK is vulnerable to a dictionary attack. While no one has come out publicly declaring that they have succeeded in implementing this attack, it is something that should be considered.

A more detailed description of WPA is out of the scope of this paper, but we recommend the book by [1] for those interested in learning more about WPA and 802.11i.

Summary

WEP is insecure. Others have broken it, and we have shown that implementing the FMS attack is not difficult. WEP, while flawed, can still be used if precautions are made and keys are swapped regularly. It is also important to note that newer wireless cards do not initialize their IVs to zero, and they do not use weak IVs. This makes them immune to FMS attacks. However, by removing these weak keys, we are reducing the already limited number of available IVs and increasing the number of IV collisions, making us more vulnerable to other attacks.

There is some debate on whether or not it is ethical to publish information about security weaknesses. Some argue that this sensitive data should be shared to inform people about the risks and to learn from past mistakes. The opposite argument is that by keeping weaknesses secret once discovered leaves us safer overall. If you did not know that Achilles weakness was his heel, chances are you would not be able to beat him. Our implementation is not the first, but our program could still be potentially used to discover a WEP key. The source code for this project can be found on the Liberal Arts Information Assurance Curriculum website at http://www.laiac.org/wireless/crack.htm.

References

1. W. A. Arbaugh, and J. Edney. *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*. Boston: Addison-Wesley, 2001.
2. W. A. Arbaugh, N. Shankar, and J. Wang. *Your 802.11 Network has no Clothes.* In: Proceedings of the First IEEE International Conference on Wireless LANs and Home Networks, December 2001.
3. N. Borisov, I. Goldberg, and D. Wagner. *Intercepting Mobile Communications: The Insecurity of 802.11.* In: Proceedings of the Seventh Annual International Conference on Mobile Computing And Networking, pp. 180–188, 2001.
4. R. Flickenger. *Wireless Hacks: 100 Industrial-Strength Tips & Tools*. United States of America: O'Reilly & Associates, 2003.
5. S. Fogie, and C. Peikari. *Maximum Wireless Security*. Indianapolis: Sams Publishing, 2003.
6. S. Fluhrer, I. Mantin, and A. Shamir. *Weaknesses in the Key Scheduling Algorithm of RC4* Eighth Annual Workshop on Selected Areas in Cyptography, August 2001.
7. M. S. Gast. *802.11 Wireless Networks: The Definitive Guide*. United States of America: 0'Reilly & Associates, 2002.
8. A. Stubblefield, J. Ionnidis, and A. D. Rubin. *Using the Fluhrer, Mantin and Shamir Attack to Break WEP.* ATT Labs Technical Report TD-4ZCPZZ, August 2001.
9. J. Girard. *Risk-free Wireless LANs with End-to-End Security & Management: "Beyond Rogue Access Points"*, AirDefense Webcast, February 26, 2004.
10. http://www.rsasecurity.com/rsalabs/technotes/wep.html
11. http://www.netstumbler.com/nation.php
12. http://www.wildpackets.com/
13. http://www.eweek.com/article2/0,1759,1375027,00.asp