A Key Recovery Attack on the 802.11b Wired Equivalent Privacy Protocol (WEP)

ADAM STUBBLEFIELD
Johns Hopkins University
JOHN IOANNIDIS
AT&T Labs – Research
and
AVIEL D. RUBIN
Johns Hopkins University

In this paper, we present a practical key recovery attack on WEP, the link-layer security protocol for 802.11b wireless networks. The attack is based on a partial key exposure vulnerability in the RC4 stream cipher discovered by Fluhrer, Mantin, and Shamir. This paper describes how to apply this flaw to breaking WEP, our implementation of the attack, and optimizations that can be used to reduce the number of packets required for the attack. We conclude that the 802.11b WEP standard is completely insecure, and we provide recommendations on how this vulnerability could be mitigated and repaired.

Categories and Subject Descriptors: E.3 [**Data**]: Data Encryption

General Terms: Security

Additional Key Words and Phrases: Wireless security, wired equivalent privacy

1. INTRODUCTION

Wireless networking has taken off, due in large part to the availability of the 802.11b standard. A combination of high-speed access coupled with ease of use makes 802.11b appealing for a number of venues—office buildings, conferences, and even many residences now offer 802.11b connectivity. Because wireless networks are inherently easier for an adversary to monitor, a security protocol called Wired Equivalent Privacy (WEP) was included as a part of the standard.

The goal of WEP was to raise the level of security for WEP-enabled wireless devices to that of traditional wired networks. Data protected by WEP is

An earlier version of some portions of this paper appeared as *Using the Fluhrer, Mantin, and Shamir Attack to Break WEP* [Stubblefield et al. 2002].
Author's addresses: Adam Stubblefield, Aviel D. Rubin, John Hopkins University, 3100 Wyman Park Building, 4th Floor, Baltimore, MD 21211; email: rubin@jhu.edu; John Ioannidis, AT&T Labs—Research.

encrypted to provide confidentiality, checksummed to prevent active attackers from modifying packets, and authenticated so that only authorized users are provided with access. WEP was designed to be easy to implement with inexpensive hardware, even in access points that must provide several simultaneous WEP protected connections.

WEP was also designed for ease of administration. Each device using the WEP is configured with a key that in practice usually consists of a password or a key derived from a password. The same key is generally deployed on all the devices that are allowed to communicate with a given network access point. The idea is to protect the wireless communication from all devices that do not know the key.

The remainder of this paper is organized as follows: In Section 2, we discuss related work; in Section 3, we describe the WEP algorithm; in Section 4, we present the Fluhrer, Mantin, and Shamir. attack on RC4; in Section 5, we describe our implementation of the attack; in Section 6, we discuss our improvements to the attack; in Section 7, we discuss the issues that this attack raises; in Section 8, we present some thoughts on how to fix WEP, and in Section 9, we conclude.

2. RELATED WORK

Borisov et al. [2001] demonstrated some security flaws in WEP. They explained that WEP fails to specify how initialization vectors (IVs) for RC4 are specified. Several PC cards reset IVs to zero every time they are initialized, and then increment them by one for every use. This results in high likelihood that keystreams will be reused, leading to simple cryptanalytic attacks against the cipher, and decryption of message traffic. The authors verified this experimentally and describe other weaknesses as well. For example, the space from which IVs are chosen is too small, virtually guaranteeing reuse, and leading to the same cryptanalytic attacks just described. The paper also shows that message authentication in WEP is broken.

Newsham [2001] used this flaw in the message authentication protocol to build an offline dictionary attack against WEP. His attack engine was able to recover a random 40-bit key in about 210 days on a 500 MHz laptop. Some implementations of WEP use a weak hash function to map the password entered by the user into the actual WEP key. In this case, the key can be brute-forced in 35 s.

Arbaugh et al. [2001] showed how to break each of the authentication modes provided by WEP. Arbaugh [2001] also demonstrated an inductive chosen plaintext attack against WEP traffic.

Fluhrer et al. [2001] describe a passive partial key exposure attack against RC4. In their paper, the authors conjecture that their attack could be applied to WEP but state: "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." In this paper, we show that the attack can indeed be used to break WEP, describe the engineering hurdles that must be overcome to do so, and present some improvements that can speed up the key recovery.

```
RC4KeySetup(k)                    RC4GeneratePseudorandomByte
For i = 0 ... N − 1               i = i + 1
  S[i] = i                        j = j + S[i]
j = 0                             Swap(S[i], S[j])
For i = 0 ... N − 1               Output S[S[i] + S[j]]
  j = j + S[i] + K[i mod l]
  Swap(S[i], S[j])
i = j = 0
```

Fig. 1. The key-setup and pseudorandom byte generation algorithms for RC4.

3. OVERVIEW OF THE WEP PROTOCOL

In this section, we present an overview of the WEP protocol; for a detailed description of WEP we refer the reader to the official 802.11 standard [L.M.S.C. of the IEEE Computer Society 1999].

Encryption in WEP uses a secret key, k, shared between an access point and a mobile node. To compute a WEP frame, the plaintext frame data, M, is first concatenated with its (noncryptographic) checksum $c(M)$, to produce $M \cdot c(M)$ (where $\cdot$ denotes concatenation). Then, a per-packet 24-bit IV is prepended to the secret key, k, to create the packet key, $IV \cdot k$. In the original WEP standard, k was 56-bits. It has been extended to 128-bits. Note that the actual secret portions of the key are only 40- and 104-bits, respectively.

The RC4 stream cipher is then initialized using this packet key, and the output bytes of the cipher are exclusive-ored (denoted $\oplus$) with the checksummed plaintext to generate the ciphertext:

$$C = (M \cdot c(M)) \oplus RC4(IV \cdot k)$$

The actual WEP data is the per-packet IV prepended to this ciphertext, C.

4. PARTIAL KEY EXPOSURE ATTACK OF FLUHRER ET AL

Given many prefixes of an RC4 key, the attack of Fluhrer et al.'s efficiently recovers the next key byte. By iterating the attack, an adversary is able to recover a full key. In this section, we describe the attack in relation to WEP.

To begin, we describe the structure of the RC4 stream cipher (shown algorithmically in Figure 1). RC4 consists of two parts, a key scheduling algorithm (KSA) and an pseudorandom byte generator. In WEP, the KSA takes as input the per-packet WEP key (the 24-bit IV followed by either 40- or 104-bits of secret key data. The KSA uses this key to set up the RC4 state array, S, which is a permutation of $\{0, \ldots, 255\}$. The output generator then uses the state array S to create a pseudorandom sequence.

The attack utilizes only the first byte of output from the pseudorandom sequence, so we focus our attention there. Since the byte generation algorithm only executes once, we can write the equation for this first byte of output as $S[S[1] + S[S[1]]$. Thus, after the key setup stage, this first byte depends on only three values of the state array ($S[1]$, $S[S[1]]$, and $S[S[1] + S[S[1]]$). The attack is based on our ability to derive information about the key by observing this value. We defer the discussion of how to recover the value of this first byte from a WEP ciphertext stream until Section 5.

To mount the attack, we search for IVs that place the key setup algorithm into a state that leaks information about the key. Using the terminology of Fluhrer et al., we refer to these key-leaking cases as *resolved*. It is simple to test whether a particular packet provides an IV and output byte that result in a resolved condition, though we refer the reader to the Fluhrer et al. paper for the conditions under which they occur.[1] Each resolved packet leaks information about only one key byte, and we must correctly guess each key byte before any packet gives us information about a later key byte.

With WEP, when we are in a resolved state, the value of the next key byte is (with high probability) given by the equation:

$$K[B] = S_{B+2}^{-1}[Out] - j_{B+2} - S_{B+2}[B+3]$$

where B is the byte current being guessed, Out is the first output from the pseudorandom number generator, and S^{-1} is the position in S where its argument occurs.

In order to obtain values of S and S^{-1} the attacker must simulate the key setup algorithm. He is able to do this perfectly for the first B iterations, since he already knows the key bytes used. The equation is only correct when the attacker's simulation after B steps matches the final state of the true key setup algorithm on the three positions in the state array used to output the first pseudorandom byte. Modeling each swap in the true key setup algorithm as random, each resolved packet gives us a 5% chance of guessing a correct key byte and a 95% chance of guessing incorrectly. However, by looking at a large number of these resolved cases, we can expect to see a bias toward the true key bytes.

5. MOUNTING THE ATTACK

In implementing this attack, we had three goals. First and foremost, we wanted to verify that the attack could work in the real world. Second, we were interested in how cheaply and easily the attack could be launched. Lastly, we wanted to see what improvements could be made to both the general RC4 attack and the WEP attack in particular. In this section, we report on our success at the first two goals, while reserving discussion about attack optimizations to Section 6.

5.1 Simulating the Attack

Before trying to break WEP, we created a simulation of the RC4 attack both to verify our understanding of the weakness and to gather information about how many resolved packets we could expect would be required when mounting the actual attack. The coding of the simulated attack took under 2 h, including a few optimizations. The simulation showed that the attack was always able to recover the full key when given 256 probable resolved cases.[2] We also observed that although 60 resolved cases (the number recommended in the Fluhrer et al.

[1]It is important to use the criteria given in Section 7 rather than the criteria given in Appendix A. The IVs listed in Appendix A are only a subset of the IVs, which can resolve. We return to this in section 6 of this paper.

[2]Cases corresponding to IVs of the form $(B+3, 255, N)$ as in the Fluhrer et al. paper.

paper) were usually enough to determine a key byte, there were instances in which more were required. Because at this point we had not thoroughly investigated how accurately we would be able to determine the first output byte of the RC4 pseudorandom sequence, we also simulated the effect that sometimes guessing wrong would have on the attack. We were pleased to see that as long as the number of incorrect guesses was kept small, the correct key byte would still be returned, though sometimes more resolved cases were needed.

5.2 Capturing the Packets

Surprisingly, capturing WEP encrypted packets off of our wireless network proved to be the most time consuming part of the attack. There are a number of commercial software programs that are able to both capture and decode 802.11 packets, such as NAI's "Sniffer" and Wildpacket's "AiroPeek," though both products cost thousands of dollars. Because we wanted to show that the attack could be done by an adversary with limited resources, we purchased a $100 Linksys wireless card, based on the Intersil Prism II chipset. We made this choice because the Prism II allows much of its computation to be completed in software and because there was a Linux driver available that could grab raw WEP encrypted packets. Though we did not know it at the time, this chipset has been used by others to mount dictionary and brute force attacks against WEP.[3]

We used both the `linux-wlan-ng prism2` driver[4] and a modified version of Tim Newsham's patch to reenable raw packet monitoring,[5] to get the card working in Linux. We were then able to use a modified version of the packet sniffer `ethereal`[6] to capture raw WEP encrypted packets and to decode the data necessary for our attack tool.

There is one problem with using this card as opposed to a more sophisticated solution. The Prism II chipset does request a transmission time-slot even when in monitor mode. Many inexpensive base stations do not report this, though a software hack can allow Linux computers running as access points to register an SNMP trap each time that a node joins or leaves the network [Hamrick 2001]. This information does not directly indicate likely attackers, but could be combined with other information in an IDS to locate users who register with a base station but not with whatever network level access controls exist. Also, we know of no practical reason why this "registration" with the network is necessary; there may even exist consumer 802.11 chipsets that support listening without registering (perhaps even the Prism II chipset in some other undocumented mode).

Even with the hardware and software problems, from the time that we first decided to look at this problem, it took less than a week for the card to be ordered and shipped, the test-bed to be set up, the problems to be debugged, and a full key to be recovered.

---

[3]See Blackhat '01 presentation at `http://www.lava.net/newsham/wlan/WEP_password_cracker.ppt`.

[4]Available from `http://www.linux-wlan.com/`.

[5]Available from `http://www.lava.net/newsham/wlan/`.

[6]Available from `http://www.ethereal.com/`.

5.3 Mounting the Attack

The last piece in actually mounting the attack was determining the true value of the first plaintext byte of each packet, so that we could infer the first byte of the pseudorandom sequence from the first ciphertext byte. We originally looked at `tcpdump` output of decrypted traffic (using a correctly keyed card),[7] and were planning on using packet length to differentiate between ARP and IP traffic (both of which have well known first bytes in their headers) as these were by far the two most common types of traffic on our network. After implementing this, however, we discovered that the attack did not seem to work. We then tried hand decrypting packets to determine whether `tcpdump` was working correctly and discovered that an additional 802.2 encapsulation header is added for both ARP and IP traffic.[8] This discovery actually made the attack even easier, as all IP and ARP packets would now have the same first plaintext byte (0xAA, the SNAP designation).[9] If the network in question also carries legacy IPX traffic, the first plaintext byte will not be 0xAA for these packets. However, as we showed in our simulation, as long as the IP and ARP packets greatly outnumber the IPX packets, the attack is still possible. If the network carries mostly IPX traffic, the attack should be modified to use either 0xFF or 0xE0 instead of 0xAA.

Although our actual attack used the improvements discussed in the next section, we present an outline of how a naive attack could work here. It is interesting to note that even this baseline version of the attack would still be successful in a short period of time (a day or two at most) and with an even smaller amount of computation when compared to the improved implementation, assuming that the wireless network in question had a reasonable amount of traffic.

To begin, we collected a large number of packets from our wireless network. To speed the process up for some of our experiments late at night when network volume was low, we artificially increased the load on the wireless network by ping flooding a wireless node. (We could have waited until more traffic was created; this is not an active attack.) Because we are able to predict the value of the first byte of any plaintext, the fact that we changed the makeup of the network traffic did not affect these experiments. In looking at the IVs of these collected packets, we discovered that the wireless cards use a simple counter to compute the IV, wherein the first byte is incremented first.[10]

Figure 2 shows the basic attack used to recover a WEP key. In Section A.1 of Fluhrer et al., the authors postulate that 4,000,000 packets would be sufficient with this baseline attack; we found the number to be between 5,000,000 and

[7]Note that a correctly keyed card is not needed; we simply used one to design the attack.
[8]We eventually traced this back to RFC 1042 [Postel and Reynolds 1988].
[9]Some vendors, such as Cisco use a proprietary OID [Cafarelli 2001]. Fortunately, it also beings with 0xAA.
[10]Other cards have been reported to choose IVs at random, to count in big endian order, or to switch between two IVs. This last class cards are not vulnerable to the attack in this paper, although they break badly under the attacks of Borisov et al. [2001].

```
RecoverWEPKey()
    Key[0...KeySize] = 0
    for KeyByte = 0...KeySize
        Counts[0...255] = 0
        foreach packet → P
            if P.IV ∈ {(KeyByte + 3, 0xFF, N) | N ∈ 0x00...0xFF}
                Counts[SimulateResolved(P, Key, KeyByte)] += 1
        Key[KeyByte] = IndexOfMaximumElement(Counts)
    return Key

SimulateResolved(P, Key, KeyByte)
    K = P.IV · Key
    For i = 0...N − 1
        S[i] = i
    For i = 0...KeyByte
        j = j + S[i] + K[imodl]
        Swap(S[i], S[j])
    return S^{-1}_{B+2}[P.Out] − j_{B+2} − S_{B+2}[B + 3]
```

Fig. 2. The basic attack on WEP. Depending on the actual key used, this attack can take between 4,000,000 and 6,000,000 packets to recover a 128-bit key. The `SimulateResolved` function computes the value described in Section 7.1 of Fluhrer et al.

6,000,000 for our key. This number is still not unreasonable, as we were able to collect that many packets in a few hours on a partially loaded network.

6. IMPROVING THE ATTACK

In this section, we discuss several modifications that can be made to improve the performance of the key recovery attack on WEP. While not necessary for the compromise to be effective, they can decrease both time and space requirements for an attacker.

6.1 Testing for suitable IVs

In the baseline attack (the one described in Appendix A of Fluhrer et al.), only IVs of a particular form are considered (those corresponding to (*KeyByte*+3, 0*xFF*, *N*) where *KeyByte* is the current KeyByte we are guessing and *N* is unrestricted). However, there are other IVs that can result in a resolved state. Testing all IVs instead of only the subset suggested by the Fluhrer et al. paper can be done in parallel with receiving packets. This conclusion was verified by Shamir [2001], who also noted that these packets appear more often for higher key bytes. The algorithm used to test whether a generic IV is resolving is shown in Figure 3.

6.2 Guessing Early Key Bytes

As the Fluhrer et al. attack works by building on previously discovered key bytes, recovering early key bytes is critical. There are two approaches that we tried both separately and together. The first utilized the way that the IVs were generated, namely that we would receive packets that resolved for lots of different key bytes before necessarily receiving enough resolving packets to

```
Resolved?(IV,CurrentKeyGuess)
    K = IV · CurrentKeyGuess
    For i = 0 ... N − 1
        S[i] = i
    j = 0
    For i = 0 ... |CurrentKeyGuess|
        j = j + S[i] + K[i mod l]
        Swap(S[i], S[j])
    if (S[1] > |CurrentKeyGuess| + 1 and S[1] + S[S[1]] = |CurrentKeyGuess| + 1)
        return true
    else
        return false
```

Fig. 3. Pseudocode for determining if a particular IV is resolving. There are more resolving IVs of unexpected forms as more key bytes are found.

predict the early key bytes.[11] We would therefore use the resolving cases that we had received to narrow down the possibilities for the early key bytes. We were then able to test candidate keys by determining if the WEP checksum on a decrypted packet turned out correctly.

The second approach exploited the poor key management available in WEP implementations. Since WEP keys have to be entered manually, we assumed that instead of giving clients a long string of hex digits, a user-memorable passphrase would be used. After examining the test wireless cards at our disposal, we determined that the user-memorable passphrase is simply used raw as the key (i.e., the ASCII is used; no hashing is done). Although hashing does not protect against a dictionary attack, it would have helped in this circumstance, as we were able to determine directly whether each key byte was likely to be part of a user-memorable passphrase by checking whether the byte value corresponded to an ASCII letter, number, or punctuation symbol.

This pair of optimizations turned out to provide an astounding decrease in the number of packets required. In parallel with receiving packets (on another machine, though this is not really necessary), we were continually attempting to guess the key by choosing the most likely candidates based on the resolved cases we had already gathered. In the event of "ties" for the next most likely byte, we gave priority first to (in order): lowercase letters, uppercase letters, numbers, symbols, and other byte values. Pseudocode for these two optimizations is shown in Figure 4.

6.3 Special Resolved Cases

As Shamir pointed out to us, there are cases when a resolved case can provide an even better indication as to a particular key byte. If there is a duplication among the three values at positions $S[1]$, $S[S[1]]$, and $S[S[1] + S[S[1]]$ (i.e., these are only two distinct values), then the probability that these positions in the S permutation remain unchanged jumps from $e^{-3} \approx 5\%$ to $e^{-2} \approx 13\%$. We can thus treat the evidence from these cases as about three times more convincing as a standard resolved case.

[11] See Figure 6 of Fluhrer et al.; resolved cases are much more likely to occur for later key bytes.

```
CheckChecksums(Guess, P)
    k = P.IV · Guess
    (plaintext, checksum) =RC4(k) ⊕ P.ciphertext)
    return c(plaintext) == checksum

SelectMaximalIndicesWithBias(Counts)
    Biased = Counts
    For i ∈ {'a', ..., 'z'}
        Biased[i] = Biased[i] · LowercaseWeight
    For i ∈ {'A', ..., 'Z'}
        Biased[i] = Biased[i] · UppercaseWeight
    For i ∈ {'0', ..., '9'}
        Biased[i] = Biased[i] · NumberWeight
    For i ∈ Symbols
        Biased[i] = Biased[i] · SymbolWeight
    return max(Biased^-1)
```

Fig. 4. The `CheckChecksums` function checks to see if the current key guess correctly decrypts the current packet by computing and comparing the checksum. The `SelectMaximalIndicesWithBias` function returns the most likely key byte, biasing the result toward the characters that are likely in a human-entered password. In our implementation, *LowercaseWeight* > *UppercaseWeight* > *NumberWeight* > *SymbolWeight* > 1.

6.4 Combining the Optimizations

Figure 5 shows the key recovery algorithm after all of the improvements described above. The improvements drop the number of packets required from around 5,000,000 to around 1,000,000.

7. DISCUSSION

There are many variables that can affect the performance of the key recovery attack on WEP. In this section, we summarize the effect of some of these variables and look at how the WEP design could be slightly altered to prevent this particular attack.

7.1 IV Selection

Since the WEP standard does not specify how IVs should be chosen, there are a variety of IV generation in use in current 802.11 cards. The majority of cards seem to use one of three methods: counters, random selection, or value-flipping (i.e., switching between two IV values). This attack is possible with either of the first two types of IV selection. Value-flipping prevents this attack at the expense of reusing the pseudorandom stream every other packet. This is not a reasonable trade-off.

Counter modes are the most accommodating of this attack. In these cards, the IV is incremented with each packet sent (starting either at 0 or at some random value when the card is powered on). With counter mode cards, an attacker is practically guaranteed a nice distribution of resolving packets among the key bytes. Random selection of each IV is not much better, as there are enough expected resolved cases that although the distribution might not be quite as good as the counter modes, it would not be much worse.

```
RecoverWEPKeyImproved(CurrentKeyGuess, KeyByte)
    Counts[0...255] = 0
    foreach packet → P
        if Resolved?(P.IV, CurrentKeyGuess)
            Counts[SimulateResolved(P, CurrentKeyGuess)] += Weight(P, CurrentKeyGuess)
    foreach SelectMaximalIndexesWithBias(Counts) → ByteGuess
        CurrentKeyGuess[KeyByte] = ByteGuess
        if Equal?(KeyByte, KeyLength)
            if CheckChecksums(CurrentKeyGuess, P)
                return CurrentKeyGuess
        else
            Key = RecoverWEPKeyImproved(CurrentKeyGuess, KeyByte + 1)
            if notEqual?(Key,Failure)
                return Key
        return Failure

SimulateResolved(P, Key, KeyByte)
    K = P.IV·Key
    For i = 0...N − 1
        S[i] = i
    For i = 0...KeyByte
        j = j + S[i] + K[imodl]
        Swap(S[i], S[j])
    return S^{-1}_{B+2}[P.Out] − j_{B+2} − S_{B+2}[B + 3]
```

Fig. 5. The improved attack on WEP. Depending on the actual key used, this attack can take between 1,000,000 and 2,000,000 packets to recover a 128-bit key. The `SimulateResolved` function computes the value described in Section 7.1 of Fluhrer et al., the `CheckChecksums` checks to see if a key causes the checksums in the WEP packets to come out correctly, and the `Resolved?` predicate checks to see if a given packet results in a resolved condition. The `SelectMaximalIndicesWithBias` function corresponds to the optimization in section 6.2. The `Weight` function returns 3 if the resolved case corresponds to a special resolved case as described in section 6.3, and 1 otherwise.

7.2 Key Selection

The lack of key management in WEP certainly contributes to the ease of the key recovery attack. Most networks use a single shared key between the base station and all mobile nodes. Besides the suite of "disgruntled ex-employee who knows the key" style attacks, there is also the problem of distributing this key to the users. Many sites use a human memorable password to easy this key distribution. There is however no standard way of mapping these passwords to a WEP key. The current solution is mapping the ASCII value directly to a key byte. We would recommend switching to either using a secure (nonmemorable) WEP key or having the key setup software hash the password to the key using a cryptographic hash function. Note that neither of these solutions prevent the attack, only make it slightly more difficult.

There do exist protocols that allow each mobile node to use a distinct WEP key, most notably IEEE 802.1x. 802.1x can be used to set up a per-user, per-session WEP key when a user first authenticates to the network. This complicates the attack, but does not prevent it so long as the user does not rekey often enough. We would recommend securely rekeying each user after every approximately 10,000 packets. Note that this solution does not address other previously discovered problems with WEP.

7.3 RC4

RC4 is an efficient stream cipher that can be used securely. The implementation of RC4 in SSL is not affected by the Fluhrer et al. attack. The reasons are that SSL preprocesses the encryption key and IV by hashing with both MD5 and SHA-1 [Dierks and Allen 1999]. Thus different sessions have unrelated keys. In addition, in SSL, RC4 state from previous packets is used in future packets, so that the algorithm does not rekey after each packet.

A further recommendation (RSA Security Inc.'s standard recommendation) is for applications to discard the first 256 bytes of RC4 output. This may be a bit expensive for very small packets, but if session state is maintained across packets, that cost is amortized.

In summary, RC4 can be used as part of a security solution. However, care must be taken when implementing it so that key material is not leaked. One of the risks of algorithms that have such caveats is that protocol designers without a strong grounding in cryptography and security may not be aware of the correct way to implement them, and this is exactly what happened in the case of WEP.

8. REPAIRING WEP

Ultimately, WEP must be redesigned using a strong block cipher and a cryptographic MAC function. However, between now and then, a temporary "best effort" fix is desirable. In this section, we discuss the challenge of putting together a system that is as secure as possible under the limitations of previously deployed WEP-based hardware. We address not only the attack of this paper, but other attacks on WEP in the literature.

8.1 Confidentiality

There are two main attacks on the confidentiality of WEP encrypted packets: the attack discussed in this paper and the IV reuse attack of Borisov et al. [2001]. To prevent the IV reuse attack, the system must guarantee that each IV is only used once with each key. Besides a method of rekeying after all the IVs are exhausted, a scheme for ensuring that IVs are not used more than once by both a base station and a mobile node is required. Two simple solutions are the partitioning of the IV space between the two communicating nodes, and the use of a separate key for each direction of the link.

These solutions do not, however, address the attack of this paper. To that end, many have suggested simply ignoring the RC4 abilities of the 802.11 hardware and preencrypting the data portion of the frame on the host operating system, in the style of a "mini-VPN." While this might work for a software base station composed of a computer with an 802.11 PCI card, it would be impossible for the many deployed hardware-based solutions. For that reason, the only cryptographic algorithm available is RC4. There are two ways in which RC4 can be used more securely. An easy (but not perfect) solution would be for all WEP hardware to throw out the first few output bytes of the cipher stream. This would prevent the attack of this paper, and prevent the use of other distinguishers for RC4. A better solution would be to instead hash the IV and the key

together using a strong cryptographic hash function. Thus, the per-packet key K would become

$$K = hash(IV \cdot k)$$

where k represents the long-term WEP key. Of course, rekeying would still be required in both of these instances to prevent the keystream reuse attacks.

If IV-mode WEP must be used for technical reasons, the best solution is to rekey often and attempt to avoid choosing bad IVs. Unfortunately, there may be other bad IVs beyond the ones discussed in this paper, and so this solution should be considered temporary at best.

8.2 Integrity

As noted in Section 3, WEP's only data integrity mechanism is a noncryptographic checksum (CRC-32) that is appended to the data and encrypted similarly. As was previously shown by Borisov et al. [2001], this checksum can be easily forged. For the same reason that RC4 cannot be replaced on existing base stations, a strong cryptographic hash-based MAC (e.g., HMAC-SHA1) cannot replace the CRC calculation. Fortunately, assuming that the keystream can be secured, a cryptographic hash is not required. Under a secure pseudorandom keystream any keyed universal hash function can serve as a MAC [Wegman and Carter 1981; Brassard 1982]. These functions, unlike cryptographic hashes, are relatively simple to compute and so could be added to existing base stations through a firmware upgrade. Besides changing to a keyed hash function, the length of the MAC would also need to be increased to prevent collisions.

8.3 Authentication

While confidentiality and integrity are sufficient conditions for WEP's shared-key authentication mechanism to function, a more powerful authentication scheme is required. Most notably, the authentication mechanism needs to provide a rekeying mechanism for both the confidentiality and integrity functions. However, just as important is the ability for the authenticated parties to negotiate the "best" version of WEP that they both support. In this way, a man-in-the-middle could not convince two nodes that supported a patched version of WEP to use the original broken version.

9. CONCLUSIONS AND RECOMMENDATIONS

We implemented the attack described by Fluhrer et al. in several hours. It then took a few days to figure out which tools to use and what equipment to buy to successfully read keys off of 802.11 wireless networks. Our attack used off of the shelf hardware and software, and the only piece we provided was the implementation of the RC4 attack, along with some optimizations. We believe that we have demonstrated the ultimate break of WEP, which is the recovery of the secret key by observation of traffic.

Since the initial report of our attack appeared, others have duplicated our results. Although we did not release our code, there are now two publicly available tools for breaking WEP keys. As always, once security attacks become known,

exploits are available to *script kiddies*, who do not need to understand the technical details to break systems. The two tools that we know of are `Airsnort` and `WEPCrack`.

Given this attack, we believe that 802.11 networks should be viewed as insecure. We recommend the following for people using such wireless networks.

—Assume that the link layer offers no security.
—Use higher-level security mechanisms such as IPsec [Kent and Atkinson 1998] and SSH [Ylonen 1996] for security, instead of relying on WEP.
—Treat all systems that are connected via 802.11 as external. Place all access points outside the firewall.
—Assume that anyone within physical range can communicate on the network as a valid user. Keep in mind that an adversary may utilize a sophisticated antenna with much longer range than found on a typical 802.11 PC card.

The experience with WEP shows that it is difficult to get security right. Flaws at every level, including protocol design, implementation, and deployment, can render a system completely vulnerable. Once a flawed system is popular enough to become a target, it is usually a short time before the system is defeated in the field.

ACKNOWLEDGMENT

We thank Bill Aiello, Steve Bellovin, Scott Fluhrer, Bob Miller, Ron Rivest, Adi Shamir, Dave Wagner, and Dan Wallach for helpful comments and discussions.

We informed Stuart Kerry, the 802.11 Working Group Chair, that we successfully implemented the Fluhrer et al. attack. Stuart replied that the 802.11 Working Group is in the process of revising the security, among other aspects, of the standard and appreciates this line of work as valuable input for developing robust technical specifications.

REFERENCES

ARBAUGH, W. A. 2001. An inductive chosen plaintext attack against wep/wep2. IEEE Document 802.11-02/230.

ARBAUGH, W. A., SHANKAR, N., AND WAN, Y. C. J. 2001. Your 802.11 wireless network has no clothes. In *IEEE International Conference on Wireless LANs and Home Networks*.

BORISOV, N., GOLDBERG, I., AND WAGNER, D. 2001. Intercepting mobile communications: The insecurity of 802.11. In *MOBICOM 2001*.

BRASSARD, G. 1982. On computationally secure authentication tags requiring short secret shared keys. In *Crypto '82*. 79–86.

CAFARELLI, D. 2001. Personal communications.

DIERKS, T. AND ALLEN, C. 1999. *The TLS Protocol, Version 1.0*. Internet Engineering Task Force. RFC-2246, `ftp://ftp.isi.edu/in-notes/rfc2246.txt`.

FLUHRER, S., MANTIN, I., AND SHAMIR, A. 2001. Weaknesses in the key scheduling algorithm of RC4. In *Eighth Annual Workshop on Selected Areas in Cryptography*.

HAMRICK, M. 2001. Personal communications.

KENT, S. AND ATKINSON, R. 1998. Security architecture for the Internet protocol. Request for Comments 2401, Internet Engineering Task Force (Nov.).

L. M. S. C. OF THE IEEE COMPUTER SOCIETY. 1999. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Standard 802.11, 1999 Edition*.

NEWSHAM, T. 2001. Cracking WEP keys. Available from `http://www.lava.net/wlan/`.

POSTEL, J. AND REYNOLDS, J. K. 1988. Standard for the transmission of IP data grams over IEEE 802 networks. Request for Comments 1042, Internet Engineering Task Force (Feb.).

SHAMIR, A. 2001. Personal communications.

STUBBLEFIELD, A., IOANNIDIS, J., AND RUBIN, A. D. 2002. Using the Fluhrer, Mantin, and Shamir attack to break WEP. In *Symposium on Network and Distributed System Security*.

WEGMAN, M. N. AND CARTER, J. L. 1981. New hash functions and their use in authentication and set equality. *Journal of Computer System Science 22*, 265–279.

YLONEN, T. 1996. SSH—secure login connections over the Internet. In *USENIX Security Conference VI*. 37–42.

Received May 2002; revised October 2002, March 2003, November 2003, March 2004; accepted March 2004