Intercepting Mobile Communications: The Insecurity of 802.11 —DRAFT—

Nikita Borisov
UC Berkeley

Ian Goldberg
Zero-Knowledge Systems

David Wagner
UC Berkeley

Abstract

The 802.11 standard for wireless networks includes a Wired Equivalent Privacy (WEP) protocol, used to protect link-layer communications from eavesdropping and other attacks. We have discovered several serious security flaws in the protocol, stemming from misapplication of cryptographic primitives. The flaws lead to a number of practical attacks that demonstrate that WEP fails to achieve its security goals. In this paper, we discuss in detail each of the flaws, the underlying security principle violations, and the ensuing attacks.

1 Introduction

In recent years, the proliferation of laptop computers and PDA's has caused an increase in the range of places people perform computing. At the same time, network connectivity is becoming an increasingly integral part of computing environments. As a result, wireless networks of various kinds have gained much popularity. But with the added convenience of wireless access come new problems, not the least of which are heightened security concerns. When transmissions are broadcast over radio waves, interception and masquerading becomes trivial to anyone with a radio, and so there is a need to employ additional mechanisms to protect the communications.

The 802.11 standard [11] for wireless LAN communications introduced the Wired Equivalent Privacy (WEP) protocol in an attempt to address these new problems and bring the security level of wireless systems closer to that of wired ones. The primary goal of WEP is to protect the confidentiality of user data from eavesdropping. WEP is part of an international standard; it has been integrated by manufacturers into their 802.11 hardware and is currently in widespread use.

Unfortunately, WEP falls short of accomplishing its security goals. Despite employing the well-known and believed-secure RC4 [12] cipher, WEP contains several major security flaws. The flaws give rise to a number of attacks, both passive and active, that allow eavesdropping on, and tampering with, wireless transmissions. In this paper, we discuss the flaws that we identified and describe the attacks that ensue.

The following section is devoted to an overview of WEP and the threat models that it is trying to address. Sections 3 and 4 identify particular flaws and the corresponding attacks, and also discuss the security principles that were violated. Section 5 describes potential countermeasures. Section 6 suggest some general lessons that can be derived from the WEP insecurities. Finally, Section 7 offers some conclusions.

2 The WEP Protocol

The Wired Equivalent Privacy protocol is used in 802.11 networks to protect link-level data during wireless transmission. It is described in detail in the 802.11 standard [11]; we reproduce a brief description to enable the following discussion of its properties.

WEP relies on a secret key k shared between the communicating parties to protect the body of a transmitted

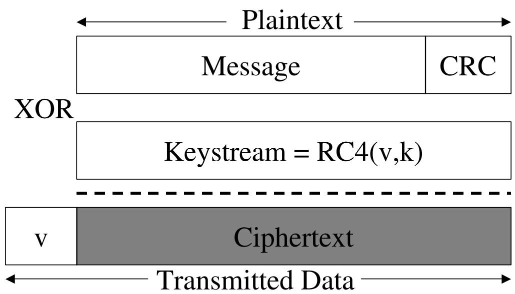

Figure 1: Encrypted WEP Frame.

frame of data. Encryption of a frame proceeds as follows:

Checksumming: First, we compute an *integrity checksum* $c(M)$ on the message M. We concatenate the two to obtain a plaintext $P = \langle M, c(M) \rangle$, which will be used as input to the second stage. Note that $c(M)$, and thus P, does not depend on the key k.

Encryption: In the second stage, we encrypt the plaintext P derived above using RC4. We choose an initialization vector (IV) v. The RC4 algorithm generates a *keystream*—i.e., a long sequence of pseudorandom bytes—as a function of the IV v and the key k. This keystream is denoted by $\mathrm{RC4}(v, k)$. Then, we exclusive-or (XOR, denoted by $\oplus$) the plaintext with the keystream to obtain the ciphertext:

$$C = P \oplus \mathrm{RC4}(v, k).$$

Transmission: Finally, we transmit the IV and the ciphertext over the radio link.

Symbolically, this may be represented as follows:

$$A \rightarrow B : v, (P \oplus \mathrm{RC4}(v, k)) \quad \text{where } P = \langle M, c(M) \rangle.$$

The format of the encrypted frame is also shown pictorially in Figure 1.

We will consistently use the term *message* (symbolically, M) to refer to the initial frame of data to be protected, the term *plaintext* (P) to refer to the concatenation of message and checksum as it is presented to the RC4 encryption algorithm, and the term *ciphertext* (C) to refer to the encryption of the plaintext as it is transmitted over the radio link.

To decrypt a frame protected by WEP, the recipient simply reverses the encryption process. First, he regenerates the keystream $\mathrm{RC4}(v, k)$ and XORs it against the ciphertext to recover the initial plaintext:

$$\begin{aligned} P' &= C \oplus \mathrm{RC4}(v, k) \\ &= (P \oplus \mathrm{RC4}(v, k)) \oplus \mathrm{RC4}(v, k) \\ &= P. \end{aligned}$$

Next, the recipient verifies the checksum on the decrypted plaintext P' by splitting it into the form $\langle M', c' \rangle$, re-computing the checksum $c(M')$, and checking that it matches the received checksum c'. This ensures that only frames with a valid checksum will be accepted by the receiver.

2.1 Security Goals

The WEP protocol is intended to enforce three main security goals [11]:

Confidentiality: The fundamental goal of WEP is to prevent casual eavesdropping.

Access control: A second goal of the protocol is to protect access to a wireless network infrastructure. The 802.11 standard includes an optional feature to discard all packets that are not properly encrypted using WEP, and manufacturers advertise the ability of WEP to provide access control.

Data integrity: A related goal is to prevent tampering with transmitted messages; the integrity checksum field is included for this purpose.

In all three cases, the claimed security of the protocol "relies on the difficulty of discovering the secret key through a brute-force attack" [11].

There are actually two classes of WEP implementation: classic WEP, as documented in the standard, and an extended version developed by some vendors to provide larger keys. The WEP standard specifies the use of 40-bit keys, so chosen because of US Government restrictions on the export of technology containing cryptography, which were in effect at the time the protocol was drafted. This key length is short enough to make brute-force attacks practical to individuals and organizations with fairly modest computing resources [1, 5]. However, it is straightforward to extend the protocol to use larger keys, and several equipment manufacturers offer a so-called "128-bit" version (which actually uses 104-bit keys, despite its misleading name). This extension renders brute-force attacks impossible for even the most resourceful of adversaries given today's technology. Nonetheless, we will demonstrate that there are shortcut attacks on the system that do not require a brute-force attack on the key, and thus even the 128-bit versions of WEP are not secure.

In the remainder of this paper, we will argue that none of the three security goals are attained. First, we show practical attacks that allow eavesdropping. Then, we show that it is possible to subvert the integrity checksum field and to modify the contents of a transmitted message, violating data integrity. Finally, we demonstrate that our attacks can be extended to inject completely new traffic into the network.

2.2 Attack Practicality

Before describing the attacks, we would like to discuss the feasibility of mounting them in practice. In addition to the cryptographic considerations discussed in the sections to follow, a common barrier to attacks on communication subsystems is access to the transmitted data. Despite being transmitted over open radio waves, 802.11 traffic requires significant infrastructure to intercept. An attacker needs equipment capable of monitoring 2.4GHz frequencies and understanding the physical layer of the 802.11 protocol; for active attacks, it is also necessary to transmit at the same frequencies. A significant development cost for equipment manufacturers lies in creating technologies that can reliably perform this task.

As such, there might be temptation to dismiss attacks requiring link-layer access as impractical; for instance, this was once established practice among the cellular industry. However, such a position is dangerous. First, it does not safeguard against highly resourceful attackers who have the ability to incur significant time and equipment costs to gain access to data. This limitation is especially dangerous when securing a company's internal wireless network, since corporate espionage can be a highly profitable business.

Second, the necessary hardware to monitor and inject 802.11 traffic is readily available to consumers in the form of wireless Ethernet interfaces. All that is needed is to subvert it to monitor and transmit encrypted traffic. We were successfully able to carry out passive attacks using off-the-shelf equipment by modifying driver settings. Active attacks appear to be more difficult, but not beyond reach. The PCMCIA Orinoco cards produced by Lucent allow their firmware to be upgraded; a concerted reverse-engineering effort should be able to produce a modified version that allows injecting arbitrary traffic. The time investment required is non-trivial; however, it is a one-time effort—the rogue firmware can then be posted on a web site or distributed amongst underground circles. Therefore, we believe that it would be prudent to assume that motivated attackers will have full access to the link layer for passive and even active attacks. Further supporting our position are the WEP documents themselves. They state: "Eavesdropping is a familiar problem to users of other types of wireless technology" [11, p.61]. We will not discuss the difficulties of link layer access further, and focus on cryptographic properties of the attacks.

3 The Risks of Keystream Reuse

WEP provides data confidentiality using a *stream cipher* called RC4. Stream ciphers operate by expanding a secret key (or, as in the case of WEP, a public IV and a secret key) into an arbitrarily long "keystream" of pseudorandom bits. Encryption is performed by XORing the generated keystream with the plaintext. Decryption consists of generating the identical keystream based on the IV and secret key and XORing it with the ciphertext.

A well-known pitfall of stream ciphers is that encrypting two messages under the same IV and key can reveal information about both messages:

$$\begin{array}{lrl} \text{If} & C_1 &= P_1 \oplus \mathrm{RC4}(v,k) \\ \text{and} & C_2 &= P_2 \oplus \mathrm{RC4}(v,k) \\ \text{then} & & \\ & C_1 \oplus C_2 &= (P_1 \oplus \mathrm{RC4}(v,k)) \oplus (P_2 \oplus \mathrm{RC4}(v,k)) \\ & &= P_1 \oplus P_2. \end{array}$$

In other words, XORing the two ciphertexts (C_1 and C_2) together causes the keystream to cancel out, and the result is the XOR of the two plaintexts ($P_1 \oplus P_2$).

Thus, keystream reuse can lead to a number of attacks: as a special case, if the plaintext of one of the messages is known, the plaintext of the other is immediately obtainable. More generally, real-world plaintexts often have enough redundancy that one can recover both P_1 and P_2 given only $P_1 \oplus P_2$; there are known techniques, for example, for solving such plaintext XORs by looking for two English texts that XOR to the given value $P_1 \oplus P_2$ [4]. Moreover, if we have n ciphertexts that all reuse the same keystream, we have what is known as a problem of *depth* n. Reading traffic in depth becomes easier as n increases, since the pairwise XOR of every pair of plaintexts can be computed, and many classical techniques are known for solving such problems (e.g., frequency analysis, dragging cribs, and so on) [14, 16].

Note that there are two conditions required for this class of attacks to succeed:

- The availability of ciphertexts where some portion of the keystream is used more than once, and
- Partial knowledge of some of the plaintexts.

To prevent these attacks, WEP uses a per-packet IV to vary the keystream generation process for each frame of data transmitted. WEP generates the keystream $\mathrm{RC4}(v, k)$ as a function of both the secret key k (which is the same for all packets) and a public initialization vector v (which varies for each packet); this way, each packet receives a different keystream. The IV is included in the unencrypted portion of the transmission so that the receiver can know what IV to use when deriving the keystream for decryption. The IV is therefore available to attackers as well[1], but the secret key remains unknown and maintains the security of the keystream.

The use of a per-packet IV was intended to prevent keystream reuse attacks. Nonetheless, WEP does not achieve this goal. We describe below several realistic keystream reuse attacks on WEP. First, we discuss how to find instances of keystream reuse; then, we show how to exploit these instances by taking advantage of partial information on how typical plaintexts are expected to be distributed.

Finding instances of keystream reuse. One potential cause of keystream reuse comes from improper IV management. Note that, since the shared secret key k generally changes very rarely, reuse of IV's almost always causes reuse of some of the RC4 keystream. Since IV's are public, duplicate IV's can be easily detected by the attacker. Therefore, any reuse of old IV values exposes the system to keystream reuse attacks. We call such a reuse of an IV value a "collision".

The WEP standard recommends (but does not require) that the IV be changed after every packet. However, it does not say anything else about how to select IV's, and, indeed, some implementations do it poorly. The particular PCMCIA cards that we examined reset the IV to 0 each time they were re-initialized, and then incremented the IV by one for each packet transmitted. These cards re-initialize themselves each time they are inserted into the laptop, which can be expected to happen fairly frequently. Consequently, keystreams corresponding to low-valued IV's were likely to be reused many times during the lifetime of the key.

Even worse, the WEP standard has architectural flaws that expose all WEP implementations—no matter how cautious—to serious risks of keystream reuse. The IV field used by WEP is only 24 bits wide, nearly guaranteeing that the same IV will be reused for multiple messages. A back-of-the-envelope calculation shows that a busy access point sending 1500 byte packets and achieving an average 5Mbps bandwidth (the full transmission

[1] Interestingly enough, some marketing literature disregards this fact: one manufacturer advertises 64-bit cipher strength on their products, even though only a 40-bit secret key is used along with a 24-bit public IV.

rate is 11Mbps) will exhaust the available space in less than half a day. Even for less busy installations, a patient attacker can readily find duplicates. Because the IV length is fixed at 24 bits in the standard, this vulnerability is fundamental: no compliant implementation can avoid it.

Implementation details can make keystream reuse occur even more frequently. An implementation that uses a random 24-bit IV for each packet will be expected to incur collisions after transmitting just 5000 packets[2], which is only a few minutes of transmission. Worse yet, the 802.11 standard does not even require that the IV be changed with every packet, so an implementation could reuse the same IV for all packets without risking non-compliance!

Exploiting keystream reuse to read encrypted traffic. Once two encrypted packets that use the same IV are discovered, various methods of attack can be applied to recover the plaintext. If the plaintext of one of the messages is known, it is easy to derive the contents of the other one directly.

There are many ways to obtain plausible candidates for the plaintext. Many fields of IP traffic are predictable, since protocols use well-defined structures in messages, and the contents of messages are frequently predictable. For example, login sequences are quite uniform across many users, and so the contents—e.g., the `Password:` prompt or the welcome message—may be known to the attacker and thus usable in a keystream reuse attack. As another example, it may be possible to recognize a specific shared library being transferred from a networked file system by analyzing traffic patterns and lengths; this would provide a large quantity of known plaintext suitable for use in a keystream reuse attack.

There are also other, sneakier, ways to obtain known plaintext. It is possible to cause known plaintext to be transmitted by, for example, sending IP traffic directly to a mobile host from an Internet host under the attacker's control. The attacker may also send e-mail to users and wait for them to check it over a wireless link. Sending spam e-mail might be a good method of doing this without raising too many alarms.

Sometimes, obtaining known plaintext in this way may be even simpler. One access point we tested would transmit broadcast packets in both encrypted and unencrypted form, when the option to control network access was disabled. In this scenario, an attacker with a conforming 802.11 interface can transmit broadcasts to the access point (they will be accepted, since access control is turned off) and observe their encrypted form as they are re-transmitted. Indeed, this is unavoidable on a subnet that contains a mixture of WEP clients with and without support for encryption: since broadcast packets must be forwarded to all clients, there is no way to avoid this technique for gathering known plaintext.

Finally, we remind the reader that even when known plaintext is not available, some analysis is still possible if an educated guess about the structure of the plaintexts can be made, as noted earlier.

3.1 Decryption Dictionaries

Once the plaintext for an intercepted message is obtained, either through analysis of colliding IV's, or through other means, the attacker also learns the value of the keystream used to encrypt the message. It is possible to use this keystream to decrypt any other message that uses the same IV. Over time, the attacker can build a table of the keystreams corresponding to each IV. The full table has modest space requirements—perhaps 1500 bytes for each of the 2^{24} possible IV's, or roughly 24 GB—so it is conceivable that a dedicated attacker can, after some amount of effort, accumulate enough data to build a full decryption dictionary, especially when one considers the low frequency with which keys are changed (see Section 3.2). The advantage to the attacker is that, once such a table is available, it becomes possible to immediately decrypt each subsequent ciphertext with very little work.

Of course, the amount of work necessary to build such a dictionary restricts this attack to only the most persistent attackers who are willing to invest time and resources into defeating WEP security. It can be argued that WEP is not designed to protect from such attackers, since a

[2] This is a consequence of the so-called "birthday paradox".

40-bit key can be discovered through brute-force in a relatively short amount of time with moderate resources [1, 5]. However, manufacturers have already begun to extend WEP to support larger keys, and the dictionary attack is effective regardless of key size. (The size of the dictionary depends not on the size of the key, but only on the size of the IV, which is fixed by the standard at 24 bits.)

Further, the dictionary attack can be made more practical by exploiting the behavior of PCMCIA cards that reset the IV to 0 each time they are reinitialized. Since typical use of PCMCIA cards includes reinitialization at least once per day, building a dictionary for only the first few thousand IV's will enable an attacker to decrypt most of the traffic directed towards the access point. In an installation with many 802.11 clients, collisions in the first few thousand IV's will be plentiful.

3.2 Key Management

The 802.11 standard does not specify how distribution of keys is to be accomplished. It relies on an external mechanism to populate a globally-shared array of 4 keys. Each message contains a key identifier field specifying the index in the array of the key being used. The standard also allows for an array that associates a unique key with each mobile station; however, this option is not widely supported. In practice, most installations use a single key for an entire network.

This practice seriously impacts the security of the system, since a secret that is shared among many users cannot stay very well hidden. Some network administrators try to ameliorate this problem by not revealing the secret key to end users, but rather configuring their machines with the key themselves. This, however, yields only a marginal improvement, since the keys are still stored on the users' computers. As anecdotal evidence, we know of a group of graduate students who reverse-engineered the network key merely for the convenience of being able to use unsupported operating systems.

The reuse of a single key by many users also helps make the attacks in this section more practical, since it increases chances of IV collision. The chance of random collisions increases proportionally to the number of users; even worse, PCMCIA cards that reset the IV to 0 each time they are reinitialized will all reuse keystreams corresponding to a small range of low-numbered IV's. Also, the fact that many users share the same key means that it is difficult to replace compromised key material. Since changing a key requires every single user to reconfigure their wireless network drivers, such updates will be infrequent. In practice, we expect that it may be months, or even longer, between key changes, allowing an attacker more time to analyze the traffic and look for instances of keystream reuse.

3.3 Summary

The attacks in this section demonstrate that the use of stream ciphers is dangerous, because the reuse of keystream can have devastating consequences. Any protocol that uses a stream cipher must take special care to ensure that keystream *never* gets reused.

This property can be difficult to enforce. The WEP protocol contains vulnerabilities *despite* the designers' apparent knowledge of the dangers of keystream reuse attacks. Nor is it the first protocol to fall prey to stream-cipher-based attacks; see, for example, the analysis of an earlier version of the Microsoft PPTP protocol [13]. In light of this, a protocol designer should give careful consideration to the complications that the use of stream ciphers adds to a protocol when choosing an encryption algorithm.

4 Message Authentication

The WEP protocol uses an integrity checksum field to ensure that packets do not get modified in transit. The checksum is implemented as a CRC-32 checksum, which is part of the encrypted payload of the packet.

We will argue below that a CRC checksum is insufficient to ensure that an attacker cannot tamper with a message: it is not a cryptographically secure authentication code. CRC's are designed to detect random errors in the message; however, they are not resilient against malicious attacks. As we will demonstrate, this vulnerability of CRC is exacerbated by the fact that the message payload is encrypted using a stream cipher.

4.1 Message Modification

First, we show that messages may be modified in transit without detection, in violation of the security goals. We use the following property of the WEP checksum:

Property 1 *The WEP checksum is a linear function of the message.*

By this, we mean that checksumming distributes over the XOR operation, i.e., $c(x \oplus y) = c(x) \oplus c(y)$ for all choices of x and y. This is a general property of all CRC checksums.

One consequence of the above property is that it becomes possible to make controlled modifications to a ciphertext without disrupting the checksum. Let's fix our attention on a ciphertext C which we have intercepted before it could reach its destination:

$$A \to (B) : \langle v, C \rangle.$$

We assume that C corresponds to some unknown message M, so that

$$C = \mathrm{RC4}(v,k) \oplus \langle M, c(M) \rangle. \tag{1}$$

We claim that it is possible to find a new ciphertext C' that decrypts to M', where $M' = M \oplus \Delta$ and Δ may be chosen arbitrarily by the attacker. Then, we will be able to replace the original transmission with our new ciphertext by spoofing the source,

$$(A) \to B : \langle v, C' \rangle,$$

and upon decryption, the recipient B will obtain the modified message M' with the correct checksum.

All that remains is to describe how to obtain C' from C so that C' decrypts to M' instead of M. The key observation is to note that stream ciphers, such as RC4, are also linear, so we can reorder many terms. We suggest the following trick: XOR the quantity $\langle \Delta, c(\Delta) \rangle$ against both sides of Equation 1 above to get a new ciphertext C':

$$\begin{aligned}
C' &= C \oplus \langle \Delta, c(\Delta) \rangle \\
&= \mathrm{RC4}(v,k) \oplus \langle M, c(M) \rangle \oplus \langle \Delta, c(\Delta) \rangle \\
&= \mathrm{RC4}(v,k) \oplus \langle M \oplus \Delta, c(M) \oplus c(\Delta) \rangle \\
&= \mathrm{RC4}(v,k) \oplus \langle M', c(M \oplus \Delta) \rangle \\
&= \mathrm{RC4}(v,k) \oplus \langle M', c(M') \rangle.
\end{aligned}$$

In this derivation, we used the fact that the WEP checksum is linear, so that $c(M) \oplus c(\Delta) = c(M \oplus \Delta)$. As a result, we have shown how to modify C to obtain a new ciphertext C' that will decrypt to $P \oplus \Delta$.

This implies that we can make arbitrary modifications to an encrypted message without fear of detection. Thus, the WEP checksum fails to protect data integrity, one of the three main goals of the WEP protocol (see Section 2.1).

Notice that this attack can be applied without full knowledge of M: the attacker only needs to know the original ciphertext C and the desired plaintext difference Δ, in order to calculate $C' = C \oplus \langle \Delta, c(\Delta) \rangle$. For example, to flip the first bit of a message, the attacker can set $\Delta = 1000 \cdots 0$. This allows an attacker to modify a packet with only partial knowledge of its contents.

4.2 Message Injection

Next, we show that WEP does not provide secure access control. We use the following property of the WEP checksum:

Property 2 *The WEP checksum is an unkeyed function of the message.*

As a consequence, the checksum field can also be computed by the adversary who knows the message.

This property of the WEP integrity checksum allows the circumvention of access control measures. If an attacker can get ahold of an entire plaintext corresponding to some transmitted frame, he will then able to inject arbitrary traffic into the network. As we saw in Section 3, knowledge of both the plaintext and ciphertext reveals the keystream. This keystream can subsequently be reused to create a new packet, using the same IV. That is, if the attacker ever learns the complete plaintext P of any given ciphertext packet C, he can recover keystream used to encrypt the packet:

$$P \oplus C = P \oplus (P \oplus \mathrm{RC4}(v, k)) = \mathrm{RC4}(v, k).$$

He can now construct an encryption of a message M':

$$(A) \rightarrow B : \langle v, C' \rangle,$$

where

$$C' = \langle M', c(M') \rangle \oplus \mathrm{RC4}(v, k).$$

Note that the rogue message uses the same IV value as the original one. Therefore, the attack works only because of the following behavior of WEP access points:

Property 3 *It is possible to reuse old IV values without triggering any alarms at the receiver.*

When we know an IV v along with its corresponding keystream sequence $\mathrm{RC4}(v, k)$, this IV-reuse property is what allows us to reuse known keystream and circumvent the WEP access control mechanism.

A natural defense against this attack would be to disallow the reuse of IV's in multiple packets, and require that all receivers enforce this prohibition. However, the 802.11 standard does not do this. While the 802.11 standard strongly recommends against IV reuse, it does not require it to change with every packet. Hence, every receiver must accept repeated IV's or risk non-interoperability with compliant devices. We consider this a flaw in the 802.11 standard.

In networking one often hears the rule of thumb "be conservative in what you send, and liberal in what you accept." However, when security is a goal, this guideline can be very dangerous: being liberal in what one accepts means that each low-security option offered by the standard must be supported by everyone, and is thus available to the attacker. This situation is analogous to the ciphersuite rollback attacks on SSL [17], which also made use of a standard that included both high-security and low-security options. Consequently, to avoid security at the least-common denominator level, we suggest that the 802.11 standard should be more specific about forbidding IV reuse and other dangerous behavior.

Note that in this attack we do not rely on Property 1 of the WEP checksum (linearity). In fact, substituting any unkeyed function in place of the CRC will have no effect on the viability of the attack. Only a keyed message authentication code (MAC) such as SHA1-HMAC [9] will offer sufficient strength to prevent this attack.

4.3 Message Decryption

What may be surprising is that the ability to modify encrypted packets without detection can also be leveraged to decrypt messages sent over the air. Consider WEP from the point of view of the adversary. Since WEP uses a stream cipher presumed to be secure (RC4), attacking the cryptography directly is probably hopeless. But if we cannot decrypt the traffic ourselves, there is still someone who can: the access point. In any cryptographic protocol, the legitimate decryptor must always possess the secret key in order to decrypt, by design. The idea, then, is to trick the access point into decrypting some ciphertext for us. As it turns out, the ability to modify transmitted packets provides two easy ways to exploit the access point in this way.

4.3.1 IP redirection

The first way is called an "IP redirection" attack, and can be used when the WEP access point acts as a IP router with Internet connectivity; note that this is a fairly common scenario in practice, because WEP is typically used to provide network access for mobile laptop users and others.

In this case, the idea is to sniff an encrypted packet off the air, and use the technique of Section 4.1 to modify it so that it has a new destination address: one the attacker controls. The access point will then decrypt the packet, and send the packet off to its (new) destination, where the attacker can read the packet, now in the clear. Note that our modified packet will be traveling *from* the wireless network *to* the Internet, and so most firewalls will allow it to pass unmolested.

The easiest way to modify the destination IP address is to figure out what the original destination IP address is, and then apply the technique of Section 4.1 to change it to the desired one. Figuring out the original destination IP address is usually not difficult; all of the incoming traffic, for example, will be destined for an IP address on the wireless subnet, which should be easy to determine. Once the incoming traffic is decrypted, the IP addresses of the other ends of the connections will be revealed, and outgoing traffic can then be decrypted in the same manner.

In order for this attack to work, however, we need to not only modify the destination IP address, but also to ensure that the IP checksum in the modified packet is still correct—otherwise, the decrypted packet will be dropped by the access point. Since the modified packet differs from the original packet only in its destination IP address, and since both the old and new values for the destination IP address are known, we can calculate the required change to the IP checksum caused by this change in IP address. Suppose the high and low 16-bit words of the original destination IP address were D_H and D_L, and we wish to change them to D'_H and D'_L. If the old IP checksum was χ (which we do not necessarily know, since it is encrypted), the new one should be

$$\chi' = \chi + D'_H + D'_L - D_H - D_L$$

(where the additions and subtractions here and below are one's-complement) [3, 10].

The trick is that we only know how to modify a packet by applying an XOR to it, and we don't necessarily know what we need to XOR to χ to get χ', even though we *do* know what we would need to *add* (namely, $D'_H + D'_L - D_H - D_L$).

We now discuss three ways to try to correct the IP checksum of the modified packet:

The IP checksum for the original packet is known: If it happens to be the case that we somehow know χ, then we simply calculate χ' as above, and modify the packet by XORing in $\chi \oplus \chi'$, which will change the IP checksum to the correct value of χ'.

The original IP checksum is not known: If χ is not known, the task is harder. Given $\xi = \chi' - \chi$, we need to calculate $\Delta = \chi' \oplus \chi$.

In fact, there is not enough information to calculate Δ given only ξ. For example, if $\xi =$ `0xCAFE`, it could be that:

- $\chi' =$ `0xCAFE`, $\chi =$ `0x0000`, so $\Delta =$ `0xCAFE`
- $\chi' =$ `0xD00D`, $\chi =$ `0x050F`, so $\Delta =$ `0xD502`
- $\chi' =$ `0x1EE7`, $\chi =$ `0x53E8`, so $\Delta =$ `0x4D0F`
- ...

However, not all 2^{16} values for Δ are possible, and some are much more likely than others. In the above example, there are four values for Δ (`0x3501`, `0x4B01`, `0x4D01`, `0x5501`) which occur more than 3% of the time each. Further, we are free to make multiple attempts—any incorrect guesses will be silently ignored by the access point. Depending on the value of ξ, a small number of attempts can succeed with high probability. Finally, a successful decryption of one packet can be used to bootstrap the decryption of others; for example, in a stream of communication between two hosts, the only field in the IP header that changes is the identification field. Thus, knowledge of the full IP header of one packet can be used to predict the full header of the surrounding packets, or narrow it down to a small number of possibilities.

Arrange that $\chi = \chi'$: Another possibility is to compensate for the change in the destination field by a change in another field, such that the checksum of the packet remains the same. Any header field that is known to us and does not affect packet delivery is suitable, for example, the source IP address. Assuming the source IP address of the packet to be decrypted is also known (we can obtain it, for example, by performing the attack in the previous item on one packet to decrypt it completely, and then using this simpler attack on subsequent packets once we read the source address from the first one), we simply subtract ξ from the low 16-bit word of the source IP address, and the resulting packet will have the same IP checksum as the original. However, it is possible that modifying the source address in this way will cause a packet to be dropped based on egress filtering rules; other header fields could potentially be used instead.

Highly resourceful attackers with monitoring access to an entire class B network can even perform the necessary adjustments in the destination field alone, by choosing $D'_L = D_H + D_L - D'_H$. For example, if the original destination address in a packet is 10.20.30.40 and the attacker holds control over the 192.168.0.0/16 subnet, selecting the address 192.168.103.147 results in identical IP header checksum values, and the packet will be delivered to an address he controls.

4.3.2 Reaction attacks

There is another way to manipulate the access point and break WEP-encrypted traffic that is applicable whenever WEP is used to protect TCP/IP traffic. This attack does not require connectivity to the Internet, so it may apply even when IP redirection attacks are impossible. However, it is effective only against TCP traffic; other IP protocols cannot be decrypted using this attack.

In our attack, we monitor the reaction of a recipient of a TCP packet and use what we observe to infer information about the unknown plaintext. Our attack relies on the fact that a TCP packet is accepted only if the TCP checksum is correct, and when it is accepted, an acknowledgement packet is sent in response. Note that acknowledgement packets are easily identified by their size, without requiring decryption. Thus, the reaction of the recipient will disclose whether the TCP checksum was valid when the packet was decrypted.

The attack, then, proceeds as follows. We intercept a ciphertext $\langle v, C \rangle$ with unknown decryption P:

$$A \to (B) : \langle v, C \rangle.$$

We flip a few bits in C and adjust the encrypted CRC accordingly to obtain a new ciphertext C' with valid WEP checksum. We transmit C' in a forged packet to the access point:

$$(A) \to B : \langle v, C' \rangle.$$

Finally, we watch to see whether the eventual recipient sends back a TCP ACK (acknowledgement) packet; this will allow us to tell whether the modified text passed the TCP checksum and was accepted by the recipient.

Note that we may choose which bits of C to flip in any way we like, using techniques from Section 4.1. The key technical observation is as follows: By a clever choice of bit positions to flip, we can ensure that the TCP checksum remains undisturbed exactly when the one-bit condition $P_i \oplus P_{i+16} = 1$ on the plaintext holds. Thus, the presence or absence of an ACK packet will reveal one bit of information on the unknown plaintext P. By repeating the attack for many choices of i, we can learn almost all of the plaintext P, and then deducing the few remaining unknown bits will be easy using classical techniques.

We explain later precisely how to choose which bits to flip. For now, the details are not terribly important. Instead, the main point is that we have exploited the receiver's willingness to decrypt arbitrary ciphertexts and feed them to another component of the system that leaks a tiny bit of information about its inputs. The recipient's reaction to our forged packet—either acknowledging or ignoring it—can be viewed as a side channel, similar to those exploited in timing and power consumption attacks [7, 8], that allows us to learn information about the unknown plaintext. Thus, we have used the recipient as an oracle to unknowingly decrypt the intercepted ciphertext for us. This is known as a *reaction attack*, as it works by monitoring the recipient's reaction to our forgeries.

Reaction attacks were initially discovered by Bellovin and Wagner in the context of the IP Security protocol, where their existence was blamed on the use of encryption without also using a MAC for message authentication [2]. As a result, Bellovin proposed a design principle for IP Security: all encryption modes of operation should also use a MAC. It seems that the same rule of thumb applies to the WEP protocol as well, for the presence of a secure MAC (rather than the insecure CRC checksum) would have prevented these attacks.

The technical details. We have deferred until now the technical details on how to choose new forged packets C' to trick the recipient into revealing information about the unknown plaintext P.

Recall that the TCP checksum is the one's-complement addition of the 16-bit words of the message M. Moreover, one's-complement addition behaves roughly equivalently to addition modulo $2^{16}-1$. Hence, roughly speaking, the TCP checksum on a plaintext P is valid only when $P \equiv 0 \bmod 2^{16}-1$.

We let $C' = C \oplus \Delta$, so that Δ specifies which bit positions to flip, and we choose Δ as follows: pick i arbitrarily, set bit positions i and $i+16$ of Δ to one, and let Δ be zero elsewhere. It is a convenient property of addition modulo $2^{16}-1$ that $P \oplus \Delta \equiv P \bmod 2^{16}-1$ holds exactly when $P_i \oplus P_{i+16} = 1$. Since we assume that the TCP checksum is valid for the original packet (i.e., $P \equiv 0 \bmod 2^{16}-1$), this means that the TCP checksum will be valid for the new packet (i.e., $P \oplus \Delta \equiv 0 \bmod 2^{16}-1$) just when $P_i \oplus P_{i+16} = 1$. This gives us our one bit of information on the plaintext, as claimed.

4.4 Summary

In this section, we have shown the importance of using a cryptographically secure message authentication code, such as SHA1-HMAC [9], to protect integrity of transmissions. The use of CRC is wholly inappropriate for this purpose, and in fact any unkeyed function falls short from defending against all of the attacks in this section. A secure MAC is particularly important in view of composition of protocols, since the lack of message integrity in one layer of the system can lead to breach of secrecy in the larger system.

5 Countermeasures

There are configuration options available to a network administrator that can reduce the viability of the attacks we described. The best alternative is to place the wireless network outside of the organization firewall. Instead of trying to secure the wireless infrastructure, it is simpler to consider it to be as much of a threat as other hosts on the Internet. The typical clients of a wireless network are portable computers that are mobile by their nature, and will frequently employ a Virtual Private Network (VPN) solution to access hosts inside the firewall when accessing via dial-up or from a remote site. Requiring that the same VPN be used to access the internal network when connected over 802.11 obviates the need for link-layer security, and reuses a well-studied mechanism. To provide access control, the network can be configured such that no routes to the outside Internet exist from the wireless network. This prevents people within radio range of the wireless infrastructure from usurping potentially costly Internet connection bandwidth, requiring VPN use for any outside access. (However, it may be desirable to allow visitors to access the Internet wirelessly without additional administrative setup.)

A useful additional measure is to improve the key management of a wireless installation. If possible, every host should have its own encryption key, and keys should be changed with high frequency. The design of a secure and easy-to-use mechanism for automated key distribution to all users is a good subject for further research. Note, though, that good key management alone cannot solve all of the problems described in this paper; in particular, the attacks from section 4 remain applicable.

6 Lessons

The attacks in this paper serve to demonstrate a fact that has been well-known in the cryptography community: design of secure protocols is difficult, and fraught with many complications. It requires special expertise beyond that acquired in engineering network protocols. A good understanding of cryptographic primitives and their properties is critical. From a purely engineering perspective, the use of CRC-32 and RC4 can be justified by their speed and ease of implementation. However, many of the attacks we have described rely on the properties of stream ciphers and CRC's, and would be rendered ineffective, or at least more difficult, by the use of other algorithms. There are also more subtle interactions of engineering decisions that are not directly related to the use of cryptography. For example, being stateless and being liberal in what a protocol accepts are well-established principles in network engineering. But from a security standpoint, both of these principles are dangerous, since they give an attacker more freedom to operate, and indeed, the traffic injection attacks capitalize on this freedom. Security is a property of an entire system, and every decision must be examined with security in mind.

The setting of WEP makes a secure design particularly difficult. A link-layer protocol must take into account interactions with many different entities at the same time. The IP redirection attack relies on collaboration between an agent injecting messages at the link-layer and a host somewhere the Internet. The complex functionality of a 802.11 access point makes it susceptible to such attacks from all sides. Faced with such difficulties, even the most experienced of security professionals can make serious errors. Recognizing this fact, the accepted practice is to rely on the expertise of others to improve the security of protocols. Two important ways to do this is to reuse past design and to offer new designs for public reviews.

Past designs should be reused whenever possible. A common tenet of protocol design is "don't do it." WEP could have benefitted from the experience gained in the design of the IP Security Protocol (IPSEC) [6]. Although the goals of IPSEC are somewhat different, it also aims to provide link-layer security, and as such needs to deal with many of the same issues as WEP. Even if the protocol could not be reused as-is, a review of its design and past analysis would have been very instructive. Some of the previously published problems in IPSEC [2] share many similarities with the attacks presented in this paper.

Public review is also of great importance. If WEP had been examined by the cryptographic community before it was enacted into an international standard, many of the flaws would have been almost surely eliminated. (For example, the dangers of using a CRC to ensure message integrity are well-known, see [15].) While we applaud the fact that the standard is open, there are still barriers to public review. A security researcher is faced with a financial burden to even attempt to examine the standard—the cost of the document is in the hundreds of dollars. This is the opposite of what should be—a working group developing a new security protocol should actively invite the security community to analyze it.

7 Conclusions

In this paper, we have demonstrated major security flaws in the WEP protocol and described several practical attacks that result. Consequently, we recommend that WEP should not be counted on to provide strong link-level security, and that additional precautions be taken to protect network traffic. We hope that our discoveries will motivate a redesign of the WEP protocol to address the vulnerabilities that we found. Our further hope is that this paper will expose important security principles and design practices to a wide audience, and that the lessons we identify will benefit future designers of both WEP and other mobile communications security protocols.

References

[1] Alan Beck. Netscape's export SSL broken by 120 workstations and one student. *HPCwire*, August 22 1995.

[2] Steven M. Bellovin. Problem areas for the IP security protocols. In *6th USENIX Security Symposium*, San Jose, California, July 1996. USENIX.

[3] B. Braden, D. Borman, and C. Partridge. Computing the internet checksum. Internet Request for Comments RFC 1071, Internet Engineering Task Force, September 1988.

[4] E. Dawson and L. Nielsen. Automated cryptanalysis of XOR plaintext strings. *Cryptologia*, (2):165–181, April 1996.

[5] Damien Doligez. SSL challenge virtual press conference. `http://pauillac.inria.fr/~doligez/ssl/press-conf.html`, 1995.

[6] Stephen Kent and Randall Atkinson. Security architecture for the Internet Protocol. Internet Request for Comment RFC 2401, Internet Engineering Task Force, November 1998.

[7] P. Kocher. Cryptanalysis of Diffie-Hellman, RSA, DSS, and other cryptosystems using timing attacks. In Don Coppersmith, editor, *Advances*

in cryptology, CRYPTO '95: 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27–31, 1995: proceedings, pages 171–183. Springer-Verlag, 1995.

[8] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proc. 19th International Advances in Cryptology Conference – CRYPTO '99*, pages 388–397, 1999.

[9] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication, February 1997. Status: INFORMATIONAL.

[10] T. Mallory and A. Kullberg. Incremental updating of the internet checksum. Internet Request for Comments RFC 1141, Internet Engineering Task Force, January 1990.

[11] LAN MAN Standards Committee of the IEEE Computer Society. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE Standard 802.11, 1999 Edition, 1999.

[12] R. L. Rivest. *The RC4 Encryption Algorithm*. RSA Data Security, Inc., March 12, 1992. (Proprietary).

[13] Bruce Schneier and Mudge. Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP). In *5th ACM Conference on Computer and Communications Security*, pages 132–140, San Francisco, California, November 1998. ACM Press.

[14] Simon Singh. *The code book: the evolution of secrecy from Mary, Queen of Scots, to quantum cryptography*. Doubleday, New York, NY, USA, 1999.

[15] S. G. Stubblebine and V. D. Gligor. On message integrity in cryptographic protocols. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 85–105, 1992.

[16] W.T. Tutte. FISH and I, 1998. A transcript of Tutte's June 19, 1998 lecture at the University of Waterloo.

[17] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce (EC-96)*, pages 29–40, Berkeley, November 18–21 1996. USENIX Association.