802.11 Security Series

Part I: The Wired Equivalent Privacy (WEP)

Agenda and Series Roadmap

This is the first of a three-part series on the security of the 802.11 WLAN (Wireless Local Area Network) standard. This month's installment reviews 802.11 and WEP (Wired Equivalent Privacy), the original WLAN security protocol. Next month's installment describes the mechanisms that will probably be implemented to recover from the mistakes made in the design of WEP. The third month's installment describes more long-term 802.11 security solutions currently under discussion.

What is 802.11?

IEEE Standard 802.11-1999 [1] specifies 802.11, a suite of protocols defining a wireless local area network (WLAN). The intent of this standard is to define an Ethernet-like communication channel using radios instead of wires. As such, it provides an unreliable datagram medium. Because of radio interference and other physical phenomena, packet loss rates up to 20% are not uncommon. However, utilizing robust communication protocols like TCP/IP over the 802.11 medium, together with 802.11's high available bandwidth, a WLAN deployment can mask these problems in most environments. Because it is simpler and less costly than running cables, the medium is enjoying explosive growth.

The 802.11 standard slices the WLAN into two logical protocol layers: the Media Access Control (MAC) sub-layer, and the Physical media sub-layer (PHY). Different variations of 802.11 correspond to different PHY sub-layers. 802.11b, or WiFi, uses a PHY sub-layer operating in the 2.4 GHz spectrum, while 802.11a, or WiFi5, uses a PHY operating in the 5.2 GHz spectrum. 802.11b has been deployed for several years and offers a maximum bandwidth of 11 Mbps, while 802.11a devices are only coming on-line now, and have a maximum bandwidth of 54 Mbps.

All the variations of 802.11 use the same MAC sub-layer, and 802.11 security resides entirely in the MAC sub-layer, so all 802.11 devices enjoy identical security properties. Hence, it is both unnecessary and misleading to distinguish between security for 802.11a and 802.11b. There are two types of 802.11 WLANs: *infrastructure networks* and *ad hoc networks*. An infrastructure network consists of *station*s (*STA*s) and *access point*s (*AP*s), whereas an ad hoc network consists solely of stations.

An **infrastructure WLAN** consists of one or more access points and one or more stations. Each access point serves as an attachment point for STAs to the network. Each STA has exactly one link at a time, via a unique AP, connecting it to the infrastructure network. The link is a session-oriented structure called an *association*. A STA communicates with other STAs in the network by exchanging packets with its associated AP, which the AP then routes to the appropriate destination. That is, in an infrastructure network, an AP mediates all communication, and STAs do not communicate directly with one another. An infrastructure network allows access to external networks. Figure 1 depicts an infrastructure-based WLAN.

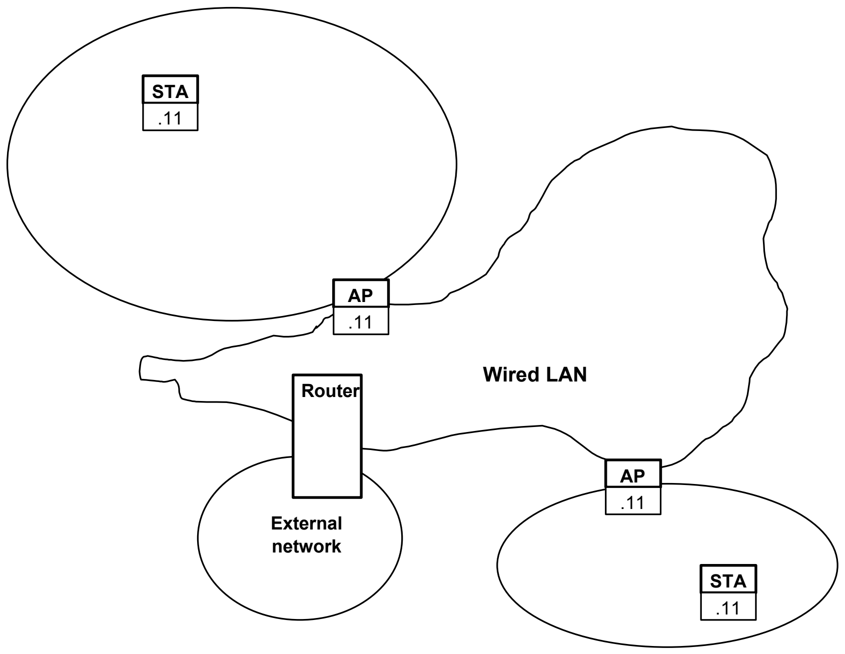

Figure 1: Typical Wireless LAN Configuration.

An **ad hoc WLAN** has no APs and supports no infrastructure, and therefore it has no ability to exchange packets with external networks (although a station may have separate links to both an ad hoc and an infrastructure network, and relay packets between them using layer 3 forwarding). Thus, an ad hoc network is a self-contained set of communicating entities. STAs communicate directly with one another in an ad hoc WLAN, and there is no notion of a session. STAs can communicate with as many other STAs in the same ad hoc WLAN as desired.

In either an infrastructure or an ad hoc network, 802.11 offers a data transfer service. The service primitives allow the caller to send and receive packets called *media access control service data units* (*MSDU*s). 802.11, in turn, transfers *MAC protocol data units* (*MPDU*s), or packet fragments across the radio medium. To send an MSDU, the 802.11 implementation first determines whether the MSDU must be fragmented into MPDUs to efficiently use the channel, or whether the MSDU can be sent as a single MPDU. The receiver uses control fields in the arriving MPDUs to discover whether they can be delivered as-is or require reassembly.

What is WEP?

An obvious problem with a communications medium based on radio is that anyone with a receiver can eavesdrop! The 802.11 designers understood this and they attempted to correct for it by inventing *Wired Equivalent Privacy*, or *WEP*. As its name suggests, WEP's goal was to create the level of privacy experienced on a wired LAN. This section describes how the original 802.11 designers sought to accomplish this level of privacy. We will examine the limitations of this construction in the next section.

WEP uses a pre-established shared secret key called the *base key*, the RC4[1] encryption algorithm and the CRC-32 checksum algorithm as its basic building blocks. WEP supports up to four different base keys, identified by *KeyIDs* 0 thorough 3. Each of these base keys is a group key called a *default key*, meaning that the base keys are shared among all the members of a particular wireless network. Some implementations also support a set of nameless per-link keys called key-mapping keys. However, this is less common in first generation products, because it implies the existence of a key management facility, which WEP does not define. The WEP specification does not permit the use both key-mapping keys and default keys at the same time, and most deployments share a single default key across all of the 802.11 devices.

The WEP tries to achieve its security goal in a very simple way. WEP operates on MPDUs, the 802.11 packet fragments. To protect the data in an MPDU, WEP first computes an *integrity check value* (ICV) over to the MPDU data. This is the CRC-32 of the data. WEP appends the ICV to the end of the data, growing this field by four bytes. The ICV allows the receiver to detect if data has been corrupted in flight or the packet is an outright forgery.

Next, WEP selects a base key and an *initialization vector* (IV), which is a 24-bit value. WEP constructs a per-packet RC4 key by concatenating the IV value and the selected shared base key. WEP then uses the per-packet key to RC4-encrypt both the data and the ICV. The IV and KeyID identifying the selected key are encoded as a four-byte string and pre-pended to the encrypted data. Figure 2 depicts a WEP-encoded MPDU.

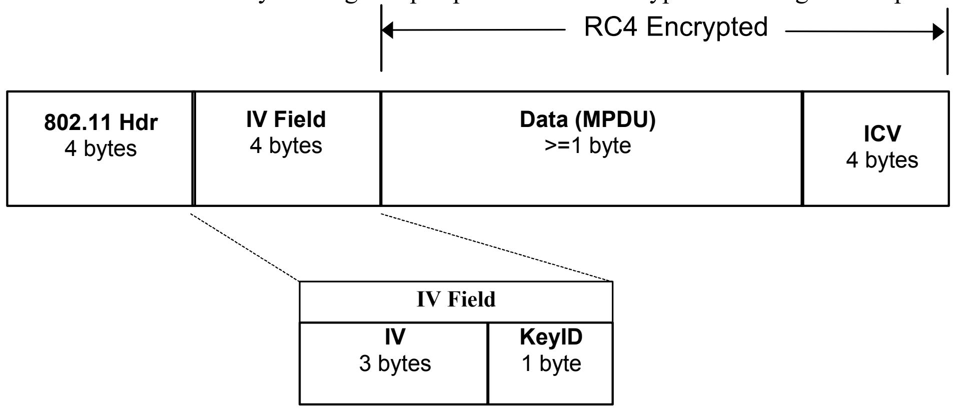

Figure 2: WEP Data Unit.

The IEEE 802.11 standard defines the WEP base key size as consisting of 40 bits, so the per-packet key consists of 64 bits once it is combined with the IV. Many in the 802.11 community once believed that small key size was a security problem, so some vendors modified their products to support a 104-bit base key as well. We will see that this difference in key length makes no difference whatsoever in overall security. The WEP construction is flawed, and an attacker can compromise its privacy goals with comparable effort regardless of the key size used.

Review of WEP's Limitations

This section reviews the major problems of the WEP design, summarizing the work reported in [2], [3], [4], and [5]. As with nearly any other human endeavor, the WEP design exhibits both sins of commission and sins of omission.

Foremost among the sins of commission is the misuse of the RC4 stream cipher. RC4 is an excellent cipher, used in a range of modern security applications, largely because of the high degree of privacy it affords with a relatively low performance penalty. However, stream ciphers in general and RC4 in particular are questionable choices in an unreliable datagram environment like WEP, because stream ciphers are notoriously difficult to use properly with packet protocols. To understand this, it is useful to review how stream ciphers operate.

By definition, a stream cipher generates a pseudo-random stream called a *key stream*. The cipher operates by requiring the encryptor to "exclusive-Ors" (XORs) the generated key stream with the plaintext to produce ciphertext. To recover the plaintext data, the decryptor generates the same key stream and XORs it with the ciphertext.

[1] RC4 is a stream cipher designed by Rivest for RSA Data Security (now RSA Security). To learn more about it, visit www.rsasecurity.com

This property of stream ciphers suggests a thought experiment. What happens if the encryptor XORs the same key stream with two different plaintexts? That is, suppose that there are two plaintext byte sequences $p_1, p_2, p_3, \ldots$ and $q_1, q_2, q_3, \ldots$, and a stream cipher C produces the key stream k_1, k_2, k_3. The corresponding ciphertexts are:

$$p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, \ldots \qquad \text{and} \qquad q_1 \oplus k_1, q_2 \oplus k_2, q_3 \oplus k_3, \ldots$$

If both of these ciphertext streams are exposed to an attacker, a catastrophic failure of privacy results, since:

$$(p_i \oplus k_i) \oplus (q_i \oplus k_i) = p_i \oplus q_i.$$

That is, using a stream cipher to encrypt two plaintexts under the same key stream trivially leaks a great deal of information about plaintext. Stream ciphers come with a warning label to *never* reuse a particular key stream. It implies that **a stream cipher cannot be safely used in a datagram environment without some sort of key management to replace keys before they can be reused.**

A second problematic stream cipher characteristic is that the encryptor and decryptor must remain synchronized. That is, the decryptor must know the relative offset of each byte of ciphertext. Otherwise it will use the wrong byte of key stream for decryption. Since IEEE 802.11 MAC is neither reliable nor delivers in-order at the level at which WEP operates, the design really requires a cipher with the *random access property*. A cipher with the random access property can generate any specified byte of the key stream in O(1) time, i.e., with approximately a constant number of instructions. This property is useful over an unreliable communications channel, because a packet can transport context with the encrypted data to properly synchronize the decryptor with the encryptor. **The problem is that, RC4, WEP's encryption algorithm, does not have the random access property!**

The WEP designers had some intimation of these stream cipher limitations and they tried to compensate for them. In an attempt to avoid these problems, they defined the per-packet RC4 key. Without much examination, this appears a reasonable enough strategy, but the naïve way in which they attempted to construct the per-packet key reintroduced the same evils they were trying to avoid.

The first per-packet key problem is the manner in which WEP constructs the per-packet key. Recall that WEP constructs the per-packet key by concatenating the IV with the selected base key, and that the IV is transmitted as plaintext with the encrypted data. The decryptor uses the IV to construct the same per-packet key. This construction should have been suspect at the outset, as it exposes three bytes of the per-packet encryption key. In the paper [3] presented in August 2001, Scott Fluhrer, Itsik Mantin, and Adi Shamir investigated the RC4 key schedule when a portion of the RC4 key is known. They showed that this kind of construction leads to a class of RC4 *weak keys*. The key schedule construct reflects patterns in the keys themselves by producing patterns at the beginning of the generated key stream. **If the first two bytes of enough key streams can be observed, then the RC4 key can be recovered. This exploit is called an FMS[2] attack.**

Using such a weak method to construct per-packet keys would be a problem in any case, but the WEP design compounds this by an unrelated flaw: the first few bytes of encrypted data in every packet are known. The SNAP-SAP header encapsulating some higher layer protocol is always the first eight bytes of encrypted data in every packet, so the first two bytes of the generated key stream can be recovered from every packet by simply "XOR"ing this known plaintext against the first two bytes of the encrypted packet payload. This is precisely the information needed to mount the FMS attack to recover the RC4 key. A public domain hacker tool called AirSnort has implemented the FMS attack. The first version of AirSnort appeared in August 2001, and it had to examine about 1 million packets to recover the WEP RC4 key. **By January 2002, the AirSnort implementation had reportedly been improved to require only about 20,000 packets to recover the RC4 key; which requires access to less than 11 seconds of IEEE 802.11b traffic under normal conditions.**

As bad as this is, WEP's IV usage also allows an attacker to recover all the plaintext without ever learning the RC4 key. WEP uses a 24-bit IV, which means that there can be no more than $2^{24} \approx 16$ million per-packet keys associated with any base key. Thus, to avoid duplication, the base key must be replaced at least once every 2^{24} packets. Since an IEEE 802.11b channel can sustain an average of about 1800 data messages per second, the base key must be replaced at least every 2.5 hours, even if there were no AirSnort program.

The collision of the 24-bit IVs suggests some very simple, low-tech attacks. A patient eavesdropper can record all the WEP encrypted traffic, group recorded packets by IV, and XOR packets encrypted under the same IV to learn a significant amount about the data. It is often feasible to use pattern recognition techniques to disentangle the two "XOR"d plaintext packets and, once this is accomplished, the generated key stream can be exposed. This permits the eavesdropper to directly decrypt all subsequent packets until the WEP key changes. A less patient attacker can send known plaintext, such as SPAM e-mail, into the network to directly recover the key stream.

A third security problem with WEP is IV selection. The specification mandates no rules for IV selection. Instead, it only recommends that implementations change the IV "frequently"—an undefined term. As a result, each vendor implements whatever IV selection strategy it chooses. Some implementations operate with a fixed IV, employing the same RC4 key to encrypt every packet**. Such implementations necessitate a key change after *every* packet**! Other vendors selected the IV at random. After n packets, the probability of an IV collision under this strategy is $P_n = 1/2^{24}$ after $n = 2$ packets, and $P_n = P_{n-1} + (n-1)(1-P_{n-1})/2^{24}$ for $2 < n < 2^{24}$; $P_n = 1$, of course, for $n \geq 2^{24}$. Therefore, after only 4823 packets there is a 50% chance of collision**. Such implementations require a key be change about every three seconds.** Other vendors have pursued a third strategy: using the IV space as a circular counter, always starting at zero upon boot. **This strategy guarantees a collision after two stations transmit a single packet, since it is common to use only default keys - i.e. the same key on every 802.11 device.**

[2] So named for the investigators who discovered the class of RC4 weak keys.

The problems discussed thus far require only a passive eavesdropper, but an active attacker can do more damage. The ICV mechanism does not provide effective data authenticity, making it easy to forge packets without every learning the key. The idea behind the ICV is that the receiver can detect data modifications or forgeries by decrypting the data and the ICV, then verifying that the decrypted ICV matches the data. However, this algorithm does not prevent undetected data modification. An attacker can record a valid packet, create a zero pad with the same length of the encrypted data, flip one or more bits, and compute the ICV of this bit-flipped zero pad. Then the attacker can create a valid forgery by "XOR"ing both the bit flipped zero pad with the encrypted data in the recorded packet, and the ICV of the bit-flipped pad with the encrypted ICV of the data. This works because the CRC-32 construction and XOR-based encryption commute - i.e. the same value results, regardless of the order of the operations. Furthermore, the CRC-32 is linear over combinations of data it protects. Under decryption, the modified ICV in the forgery will validate correctly, and WEP will accept the packet as genuine. If combined with a little packet analysis, it is not too difficult to use this technique to construct packets with correct application data. **This implies that the receiver of a WEP packet cannot reliably detect packet forgeries and therefore cannot rely on WEP encrypted data as genuine.**

Even it if the encrypted CRC-32 construction had worked effectively, the data integrity mechanism does not protect all the information that needs to be protected from modification. As an example, the ICV mechanism does not protect the packet destination address. An eavesdropper can record a packet from a station to an access point, and then retransmit the packet after changing the destination address of the recorded packet from that of a station authorized to participate in the communication to one that does not. When the access point receives this forgery, it dutifully decrypts the packet and forwards it to the "wrong" address. This permits the attacker to use the infrastructure itself to decrypt any WEP packet sent from a station via an access point. A similar alteration of the source address for packets from the access point to a station allow the adversary to masquerade as any other station.

The reason why this works leads to the last problem with the intended data protection mechanisms: **WEP provides no replay protection**. An eavesdropper can record any packet and then retransmit it later, with or without any alteration. Since each packet so constructed is encrypted under a valid key, all will be accepted at the IEEE 802.11 level as valid. Traffic analysis can reveal the use of various connectionless protocols, which are especially susceptible to replay. This leads to a very counter-intuitive conclusion: it is infeasible to make any effective privacy guarantees unless the protocol design also prevents forgery attacks.

All of these problems arise when an eavesdropper can collect a sufficient number of packets encrypted under the same base key. If the WEP base key were changed sufficiently often, these attacks might afford an adversary significantly fewer options to compromise security. However, **IEEE 802.11 provides no mechanism to replace keys, practically requiring customers to use static, manually configured keys.** It is infeasible to manually change keys often enough to provide protection from these attacks. So, compromise is the norm rather than the exception. Even if this were not the case, analysis indicates that **the WEP key should be changed at least every 256 packets to derive any security benefits from rekeying, and this appears to be beyond the capabilities of existing hardware.**

The WEP architecture compounds all these problems in two ways. First, WEP uses the same key to protect data in both directions over a link. When, as is very common, implementations use a counter to generate the next IV, this guarantees immediate IV collision and data exposure as a result. Second, IEEE 802.11 only provides a way to name group keys, thereby encouraging the use of a single group key within a WLAN, since it is infeasible to manage quantities that cannot be named.

Summary

The WEP design is seriously flawed, and given the emergence of automated tools like AirSnort, it really has very little utility. People often ask, "What is the one thing that needs to be fixed in WEP to increase its security the most?" This question implies a misunderstanding of the way in which security algorithms work. **Addressing only one or some of these problems is similar to closing only some of the hatches on a submarine before submerging!** The WEP design includes many design flaws. Fixing one or two does not render the protocol any more secure than fixing none of them. Security is possible only if all of the core deficiencies are addressed.

All hope is not lost, however. The IEEE 802.11 Working Group recognized the gravity of the issues with the current WEP design and created Task Group "i" (TGi) to resolve them. TGi has designed two solutions to the problem. One is intended as a short-term patch for currently deployed equipment. The other will address the long-term. In the installment next month, we will examine the TGi work to develop the short-term patch for already deployed equipment, called the *Temporal Key Integrity Protocol*, or TKIP. TKIP is designed to counter the security threats posed by the WEP design.

For Further Reading

[1] IEEE Std. 802.11, Standards for Local and Metropolitan Area Networks: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999.

[2] Borisov, N., I. Goldberg, and D. Wagner, "Intercepting mobile communications: the insecurity of 802.11," in *Proc. International Conference on Mobile Computing and Networking*, ACM, July 2001, pp 180-189.

[3] Fluhrer, S., I. Mantin, and A. Shamir, "Weaknesses in the key schedule algorithm of RC4," in *Proc. 4th Annual Workshop on Selected Areas of Cryptography, 2001*.

[4] Stubblefield, A., J. Ioannidis, and D. Rubin, "Using the Fluhrer, Mantin, and Shamir attack to break WEP", AT&T Labs Technical Report TD-4ZCPZZ, AT&T Labs, August 2001.

[5] Walker, J., "Unsafe at any key size: an analysis of the WEP encapsulation," IEEE 802.11 doc 00-362, October 27, 2000.

About the Author

Jesse Walker is the network security architect for Intel's Platform Networking Group. He is the technical editor for the 802.11 security enhancements, and was the first person to publicly identify the security flaws in the original 802.11 WLAN protocol. He joined Intel as part of the Shiva acquisition. Jesse has also been active in other industry standardization efforts, including IPSec. Jesse holds a Ph.D. in mathematics from the University of Texas at Austin. Jesse may be contacted at jesse.walker@intel.com.