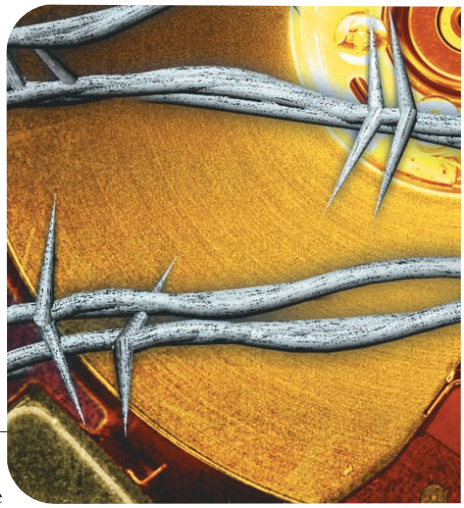
The Dangers of Mitigating Security Design Flaws:

A Wireless Case Study

Mitigating design flaws often provides the only means to protect legacy equipment, particularly in wireless local area networks. A synchronous active attack against the wired equivalent privacy protocol demonstrates how mitigating one flaw or attack can facilitate another.



NICK L. PETRONI JR. AND WILLIAM A. ARBAUGH
University of Maryland, College Park

Mitigating design flaws in security architectures is a difficult, if not impossible, task. Yet, organizations and vendors must often resort to mitigation to protect their installed bases. While mitigating implementation flaws, such as buffer overflows, usually involves only a few lines of code, mitigating design flaws often involves the wholesale redesign of the system architecture.

As wireless networks become more ubiquitous, security concerns related to these networks likewise increase. For many security researchers, wireless networks have proven to be an archetype of a fundamental premise in security design: adding security or fixing poorly designed security after the fact is often impossible. While the IEEE 802.11 Task Group I design team is advancing the state of wireless local area network security, legacy deployments will continue to operate for several more years, and these networks will predominantly operate without the infrastructure required for current best practice.

In a case study involving wireless LAN security, we found that mitigating one known security flaw or attack created a dramatic speed-up in a second known attack. Thus, the system's overall security saw little improvement. As part of our study, we developed and implemented a novel active attack against the *wired equivalent privacy* protocol. Our inductive attack lets any attacker with access to the wireless medium synchronously recover a full or partial dictionary for any static WEP deployment. Although other researchers have documented WEP properties that make partial dictionary recovery possible, previous attacks have been asynchronous and therefore incur significant operational difficulties in practice. Our implementation is both the first synchronous attack against WEP and, more importantly, a classic example of how a decision to mitigate one design flaw can greatly accentuate another.

Wired equivalent privacy protocol

IEEE 802.11 designers recognized the inherent differences between the wired and wireless environments, particularly with regard to medium access. Because the transport medium is shared, any client in transmission range of another client can process packets originating from that host. WEP was designed to protect data at the link layer and prevent unauthorized access to 802.11 data frames.[1] Although the WEP design goals called for "reasonably strong" protection, recent work has demonstrated the protocol's failure to meet that goal (see the sidebar "WEP Design Flaws").

To provide data confidentiality at the link layer, WEP uses the symmetric stream cipher RC4 to encrypt all network data traffic, as Figure 1 illustrates.[1] The system is based on a shared secret k that is distributed out of band. In most network configurations, this out-of-band method is the manual distribution of a single key shared by all parties in the network. RC4 uses the key to generate a stream of pseudorandom bytes equal in length to the target plaintext. It then combines this stream with the plaintext (P), using the bitwise "exclusive or" function (XOR, or $\oplus$) to produce ciphertext C.[1]

$$C = \mathrm{RC4}(k) \oplus P \quad (1)$$

Similarly, WEP decrypts network traffic by producing the pseudorandom stream used for encryption and combining it with the ciphertext. Use of the combiner ⊕ reveals the original plaintext.

$$P' = \mathrm{RC4}(k) \oplus C = \mathrm{RC4}(k) \oplus \mathrm{RC4}(k) \oplus P = P \quad (2)$$

Because the sender and receiver must generate the same key stream, the same key must always generate the same pseudorandom stream. Thus, identical plaintexts encrypted with the same key produce identical ciphertexts. This is an extreme weakness, particularly when the cipher is used to encrypt highly redundant messages, such as network protocols.[2] To address this weakness, WEP's set of *initialization vectors* attempt to add entropy to the key space by using an IV. WEP uses a 3-byte (24-bit) IV and a 40- or 104-bit shared secret to produce the encryption key k. The sender transmits the IV with the encrypted ciphertext so the receiver can produce the full k and decrypt.

In addition to its cryptographic protection of the data, WEP uses a 4-byte integrity check value (ICV), computed over the original plaintext of the data portion of the 802.11 frame. The 802.11 transmitter computes the ICV as a CRC-32 checksum and appends it to the plaintext before encrypting.[1]

Dictionary attacks

Historically, the term *dictionary attack* refers to a class of attacks against password systems whereby the attacker uses a large set of inputs—the dictionary—to gain access by trying all possibilities. Over time, the phrase has been used more generally to refer to any brute-force attack in which a large table is used or generated. In the case of RC4, each key has an associated key stream (pseudorandom stream) used for encryption and decryption. As previously mentioned, WEP utilizes a set of IVs to increase the total number of streams associated with a single secret. An attacker who can build a dictionary of all such streams (one per IV) has no need for the key, assuming the dictionary is a manageable size. An IV's relatively small size results in a total of 2^{24} possible key streams, each with a length of the maximum transport unit (MTU) for 802.11 (2,312 bytes). A quick calculation shows that with less than 40 Gbytes of storage, we could build a dictionary for a single key and all possible IVs. While many researchers have noted the potential for dictionary attacks, these attacks have relied on an attacker's ability to determine the plaintext or inject known data from a wired interface.

In the first approach, an attacker can use a network protocol structure to determine many bytes of the plaintext, but must resort to traffic analysis to guess the remaining fields. This analysis can be difficult without additional information about the network such as usernames, server addresses, and active protocols. Moreover, the recovered stream's length will depend on the size of packets for which the attacker can correctly predict the plaintext. An attacker who can only guess the beginning of the message will not be able to decrypt longer messages.

In the second approach, injecting data into the network, the attacker knows the entire plaintext and can control the injected packet's length to maximize the recovered stream. However, this approach relies on the attacker's ability to inject such information into the network via a wired interface or to persuade a wireless client to request predetermined content. Clearly, an attacker will not always have such influence over the network or its users. Additional problems arise from the asynchronous nature of these approaches. Attacks such as IP redirection or plaintext injection from an outside host make timing a difficult task for the attacker.

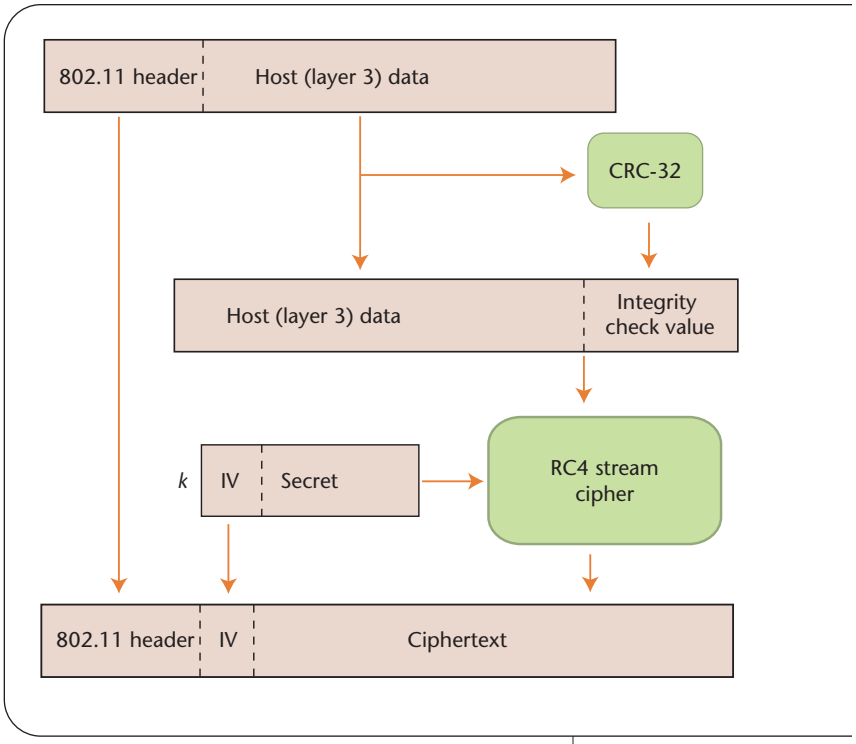


Figure 1. WEP data encryption and encapsulation. In WEP, the symmetric stream cipher RC4 encrypts network data traffic. Using a shared secret key, RC4 generates a stream of pseudorandom bytes equal in length to the target plaintext.

Inductive attack

The inductive attack aims to give the attacker full network access (both encryption and decryption) without knowledge of the secret key. As stated previously, the lack of integrity protection for wireless packets lets an attacker with knowledge of the key stream inject arbitrary packets without being detected by other clients.

Overview

The inductive attack maximizes the advantages of a completely known plaintext attack, while minimizing the at-

WEP design flaws

Researchers have identified threats against each of the primary security services (availability, integrity, and confidentiality) and have implemented attacks for a variety of network configurations.[1–7] While we don't give a comprehensive list of these vulnerabilities, we overview past work to compare it with the inductive attack we present.

Key stream reuse

Jesse Walker's analysis of WEP's use of RC4 was one of the earliest works to study 802.11 network insecurity.[7] Walker examined the initialization vector mechanism as a prevention against key stream reuse. He concluded that the way in which WEP uses RC4 causes high IV reuse and therefore key stream reuse that renders it ineffective. First, a moderately busy network will exhaust the relatively small IV space in a matter of hours, sometimes minutes. The likely configuration of multiple access points sharing the same key magnifies this result. Second, because WEP does not prevent collisions among multiple access points, access points using the same IV production algorithm are more likely to choose duplicate IVs in a smaller interval. As Walker points out, even if you employ a system to avoid collisions, the IV space is simply too small to guarantee such avoidance.

Nikita Borisov, Ian Goldberg, and David Wagner made similar observations about the dangers of key stream reuse and further analyzed the WEP checksum (that is, its integrity check value, or ICV).[3] Having reached the same conclusion about the likelihood of IV collisions, they described methods attackers could use to recover plaintext or inject new traffic into the network through key stream reuse. In terms of recovery, the authors used known characteristics of network and application protocols to make predictions about the plaintext of sniffed messages—for example, header structure, common fields, easily identifiable remote library calls, and frequently used strings such as prompts. Furthermore, they suggested using a wired Internet connection to make controlled IP requests of a wireless host or, if a permanent connection is not available, sending a known wireless user an email and simply waiting for the user to update his or her mail. While both methods are feasible, they rely on an attacker's knowledge of the network being attacked or ability to externally influence the environment asynchronously.

Message modification

Borisov, Goldberg, and Wagner also describe how an attacker can arbitrarily change or "bit flip" parts of a WEP-encrypted message to let the receiver decrypt the message without recognizing any error. This directly results from WEP's use of CRC-32 as its integrity checking function. Because CRC-32 is linear, its checksum distributes over the XOR operation. That is, $\mathrm{CRC}(x) \oplus \mathrm{CRC}(y) = \mathrm{CRC}(x \oplus y)$. Furthermore, because RC4 uses the XOR operation as its combining function, an attacker can arbitrarily modify the encrypted message while maintaining a valid checksum. Because the checksum lacks a keyed-input, an adversary who knows the plaintext of the message can compute the checksum. Thus, an adversary can inject a packet without knowing the cryptographic key (assuming he or she has the key stream).

tacker's reliance on external knowledge about and influence on the network. Furthermore, it does this synchronously to minimize attack complexity. The inductive attack has three phases.

Recovering an initial pseudorandom stream. The base case requires an attacker to recover n bytes of a pseudorandom stream for any IV. Generally, the attacker can accomplish this using network protocol structure and external information such as a media access control address and message size to guess the plaintext. Perhaps the simplest example is network traffic for a dynamic host configuration protocol session. DHCP messages, particularly `discover` and `request`, are generally easy to identify even when encrypted and have highly predictable fields including all of the IP header and most of the user datagram protocol (UDP) header.

We have successfully mounted the inductive attack with as few as 11 correctly guessed initial bytes. Given that the logical link control (LLC) and subnetwork access protocol (SNAP) combine for eight highly predictable bytes at the beginning of every 802.11 packet and that the first three bytes of an IP packet are also highly predictable, an attacker could perform this step on almost any network.

Extending the pseudorandom stream size. The attacker uses the redundant information provided by the ICV to extend the pseudorandom stream one byte at a time to obtain a stream for the entire MTU. Because an access point or other wireless host will only pass a WEP-encrypted packet if the ICV is computed correctly, an attacker can use the device to filter out false guesses at the pseudorandom stream. If, for example, the attacker recovers n bytes of a key stream from the above step, he or she can inject any message of length $n - 4$ by computing the correct ICV, appending it to the plaintext and combining it with the known key stream. To extend the known key stream, the attacker

1. Generates a message of length $n - 3$ that, if received cor-

One of us, William Arbaugh, extended these findings into a practical synchronous attack against WEP (and at the time a potential replacement for WEP called WEP2), the implementation of which we describe in the main article.[2]

Passive attacks

Not long after researchers identified the weaknesses of applying cryptographic primitives within WEP, Scott Fluhrer, Itsik Mantin, and Adi Shamir published work on weaknesses in the RC4 cipher.[1] They presented the FMS probabilistic attack, which leverages a class of weak keys in the underlying cryptographic logic used in IEEE 802.11 wireless LANs. This devastating attack can recover the encryption key in approximately 30 minutes when combined with an active attack to generate enough traffic. Previously, the best-known attack against WEP required almost 54 hours.[2] Under the right conditions, a vulnerability in RC4's key-scheduling algorithm provides information about the key generating the stream. Unfortunately for WEP, these conditions, a constant secret combined with multiple different exposed key bytes (the IV), are consistent with how WEP uses RC4. As a result, a passive attacker can recover the entire secret key with relatively few sniffed messages.

Explanations of the theory behind the attack are available elsewhere.[1,4,6] The FMS attack is entirely passive and relies on the use of a specific IV class that, when used with the associated secret, is likely to encrypt with a value that reveals information about the secret. By processing enough packets encrypted with these weak IVs, an attacker can determine the value and recover a piece of the key. This attack has multiple independent implementations and works quite well.[4,6] Some access point vendors have begun to filter weak IVs to mitigate the effectiveness of the passive attack. Without access to data encrypted using weak IVs, the attack cannot succeed.

References

1. S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," *Proc. 8th Ann. Workshop Selected Areas in Cryptography*, Springer, 2001, pp. 1–24.
2. W. A. Arbaugh, "An Inductive Chosen Plaintext Attack Against WEP/WEP2," IEEE 801.11, Verlag, May 2001, www.cs.umd.edu/~waa/wepwep2-attack.html.
3. N. Borisov, I. Goldberg, and D. Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11," *Proc 7th Ann. Int'l Conf. Mobile Computing and Networking*, ACM Press, 2001, pp. 180–188.
4. A. Stubblefield, J. Ioannidis, and A.D. Rubin, "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP," *Proc. Network and Distributed System Security Symposium* (NDSS), Internet Society, 2002; www.isoc.org/isoc/conferences/ndss/02/proceedings/papers/stubbl.pdf.
5. W. Arbaugh, et al., "Your 802.11 Wireless Network Has No Clothes," *IEEE Wireless Comm.*, vol. 9, no. 6, IEEE Press, 2002, pp. 44–51.
6. D. Hulton, "Practical Exploitation of RC4 Weaknesses in WEP Environments," white paper, Dachb0den Labs, Feb. 2002, www.dachb0den.com/projects/bsd-airtools/wepexp.txt.
7. J. Walker, "Unsafe at Any Key Size: An Analysis of the WEP Encapsulation," *IEEE 802.11-00/362*, IEEE Press, 2000, www.netsys.com/library/papers/walker-2000-10-27.pdf.

rectly, generates a predictable response from some network host (possibly the access point). If not received, or if received incorrectly, the message fails to produce such a response.
2. Computes the ICV for the new message and appends the first three bytes, saving the final byte for later use.
3. Combines the resulting n-byte message with the known key stream.
4. Transmits an $n + 1$-byte message (n bytes from the previous step and a guess at the final byte). The attacker repeats this step with a different, untested final byte until he or she detects the expected network response for a valid packet.
5. XORs the $n + 1$th transmitted byte with the last byte of the correct ICV computed in the second step. The resulting value is the $n + 1$th key stream byte.

Figure 2 illustrates this process. The completed procedure extends the known key stream by a single byte. The attacker can repeat the procedure until he or she recovers a pseudorandom stream up to the MTU.

Building the dictionary. Now that the attacker has a complete key stream for a particular IV, he or she can inject arbitrary packets at will using that IV. With no information about other IVs, however, the attacker cannot yet participate as a full network member. To recover key streams for the remaining IVs, the attacker simply makes a series of requests that elicit predictable, long responses from a host or hosts. By encrypting these requests using the recovered pseudorandom streams (as in the previous step), the attacker can recover additional full key streams. Although similar to a dictionary attack in which the attacker injects data into the network from outside connections, the attacker uses a key stream recovered during the attack rather than outside access to the wired network. Because the attacker is a participating member of the local network, he or she can easily synchronize requests and responses for key stream recovery.

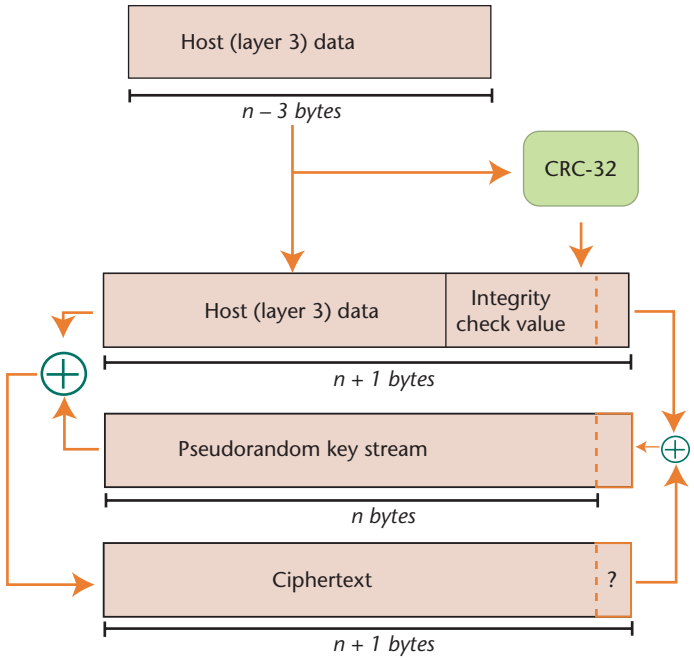


Figure 2. The inductive step. An attacker guesses the *n* + 1th byte until he or she receives a response and recovers the next byte of the pseudorandom stream.

Our attack implementation

We implemented a relatively simple inductive attack to demonstrate the feasibility of conducting such an attack on commercially available and widely deployed networks. The attack has modest hardware requirements:

- A wireless card that can enter RF Monitor mode—a mode in which all detected 802.11 traffic from any network in the area is passed to the operating system—to recover the initial pseudorandom stream in the base case
- A wireless card that lets the device driver transmit frames with arbitrary data portions so the attacking software rather than the card hardware encrypts the data
- Enough disk space (up to 40 Gbytes)

To meet the first two requirements, we used a single card based on the Intersil Prism2 chipset. This card has monitoring capabilities and lets the driver generate arbitrary data packets when using the correct transmit flags. Most currently available laptops meet the third requirement.

We performed the implementation on a system running OpenBSD 3.1. Because the `if_wi` driver already supports RF Monitor mode, it only required a simple change to be able to send arbitrary data being passed through the `wi` device interface as raw data in an 802.11 packet. We then implemented the attack as a user–space program, using the inductive attack methods described in the previous section.

Recovering an initial pseudorandom stream. In theory, we could use any packet long enough to have an ICV (4 bytes) to perform the inductive step. In practice, the attack relies on the use of a protocol that will cause a predictable response under certain conditions. If an attacker can recover more bytes in the first step, he or she will need to recover fewer bytes inductively (thereby reducing the overall time of attack) and can use more complex protocols earlier in the attack. An attacker who only recovers 11 bytes, for example, will not be able to inject packets with correct IP headers. Our implementation successfully identifies and recovers 34 bytes of a pseudorandom stream for the address resolution protocol (ARP) and DHCP packets and as many as 12 bytes for arbitrary, unidentifiable IP packets (see the sidebar "Recovering an Initial Pseudorandom Stream with DHCP").

Extending the pseudorandom stream size. Many protocols allow single-byte length increases, enabling the attacker to be creative in his or her selection of protocols. We successfully implemented our attack using two different packet types for induction: Internet control message protocol echo packets and arbitrary length, malformed ARP packets. Other possibilities include sending user datagram protocol packets to a known closed port.[3]

Because of their flexibility, widespread implementation, and response characteristics, ICMP echo (ping) requests are a natural candidate for key stream induction.[4] Any network host, wireless or wired, is a potential message destination. Moreover, the messages are easy to generate because the payload can contain arbitrary data.

To use ICMP, an attacker must know an active IP address for a network host. If the initial pseudorandom stream is at least 28 bytes long, the attacker can easily recover IP addresses for one or more network hosts (IP address are always in the first 20 bytes of any IP message and IVs are guaranteed to repeat). Because this is really a layer-2 attack, however, a valid layer-3 protocol—all of which require the attacker to identify a valid address—is unnecessary.

Our second implementation of the inductive attack uses packets with arbitrary data and a SNAP header that indicates the data is actually part of the ARP protocol. While the ICMP version relies on a reply from the destination host, the ARP version only uses the access point to filter out packets with invalid checksums. Because the packet data is invalid, the host will not reply. Instead, the attacker chooses a wireless host as the destination and sniffs the medium to determine if the access point has forwarded the injected packet to that host. Although the choice of ARP might seem arbitrary, many access points are increasingly intelligent about transmitted data. Tests indicated that some vendors' access points didn't forward IP packets with invalid IP headers. None of the tested access points were as discriminant with ARP packets.

Building the dictionary. We also use ICMP packets or malformed ARP packets to build a dictionary of remaining pseudorandom streams. By injecting maximal-length ICMP echo requests destined for a known host, we receive responses of known content that are long enough to recover entire pseudorandom streams for alternate IVs. Each time we receive data with a new IV, we use the known plaintext to recover the pseudorandom stream for that IV and then store the information in the dictionary. When the attacker sees a duplicate IV, he or she simply discards the data. As with arbitrary data injected into ARP packets, the access point will often forward the packet using its next IV and not the IV chosen by the initial sender. The attacker simply sniffs the forwarded packet and recovers the pseudorandom stream for the new IV. Other protocols can be used in this step as well.

Attack performance

The inductive attacker's primary goal is to become a peer on the network as quickly as possible. In general, the inductive step (phase two of the attack) is the most technically difficult to implement, while phase three takes the most attack time.

Several factors contribute to phase two's speed, in particular network latency. In the inductive step, the attacker sends packets that might or might not be the correct form and waits for a response. If a response comes, the attacker immediately knows he or she can move on to the next byte, but how long should an attacker wait for a response? Depending on network conditions, responses could take from a couple of milliseconds to a few seconds. Waiting too long obviously wastes time, but not waiting long enough means the attackers might not know which request induced the response. Fortunately for the attacker, this delay is a tunable parameter that can be updated each time the attack runs, or even dynamically during a single attack, depending on the implementation's sophistication.

Our implementation used a manually tuned static value, but also included a form of backtracking to handle the case where the wait period is usually optimal, but inconsistent. Although we determined that the implementation could have further reduced its wait time, limitations in the packet capture library used prevented delays of less than seven milliseconds. Still, as Table 1 shows, the implementation averaged a per-byte recovery rate of 1.83 seconds, resulting in an average phase-two length of just over one hour for the full 802.11 MTU and only 45 minutes for the 802.3 limit of 1,500 bytes.

The high expense of building a dictionary is due to the large amount of data that must be collected. While conservative estimates have exceeded 46 hours, fast-packet throughput on a commercial access point let us build the entire dictionary in just less than 18 hours. Furthermore, a number of properties of the inductive attack considerably lessen the importance of this number.

Table 1. Time characteristics of our attack implementation over 10 runs on a single access point.

ATTACK CHARACTERISTIC	TIME
Minimum single-byte induction	1.75 seconds
Average single-byte induction	1.83 seconds
Maximum single-byte induction	2.26 seconds
Minimum phase two (MTU 1500)	42.845 minutes
Average phase two (MTU 1500)	45.217 minutes
Maximum phase two (MTU 1500)	55.682 minutes
Estimated phase two (MTU 2300)	1 hour, 6.161 minutes

First, in the inductive attack, the attacker has far more power before the attack's completion than in the passive key-scheduling algorithm attack. In a KSA attack, the attacker cannot decrypt or inject messages until the entire key is known. Conversely, inductive attackers will already have partial key streams to use to their advantage. One way to capitalize on this advantage is to maintain a dictionary of partial pseudorandom streams during the inductive step. For each byte recovered there will be a corresponding response from a host, generally encrypted using a previously unseen IV. The attacker can disregard partial streams and wait for the entire dictionary or use this information to decrypt smaller messages.

Using partial dictionaries of full streams before the completion of phase three can also provide immense advantages in some WEP implementations. In the test environment, an access point from a major access point vendor proved to reuse IVs consistently in a short period of time (on the order of minutes). The clear advantage of such reuse is that the attacker can begin using the network as a full member after recovering only a subset of key streams.

The second, and more powerful, feature of the inductive attack is that it can be easily parallelized. During the third phase, an attacker can add an arbitrary number of hosts or interfaces without complicating the attack and thus gain almost linear speedup. Tightly coupled hosts could share a dictionary, or loosely coupled hosts could simply take the union of independently held dictionaries after working for a predetermined amount of time.

WEP implementations

Walker and Borisov, Goldberg, and Wagner were the first to point out the wide variance of possible WEP implementations left open by the standard. Early on, many vendors made poor implementation decisions such as the repeated use of a single IV or a repeating counter that resets on reboot. Most major vendors subsequently increased

Recovering an initial pseudorandom stream with DHCP

Recovering an initial pseudorandom stream is easy for any adversary who can sniff wirelessly. Because network headers are highly predictable, almost every packet can become an initial pseudorandom stream that can be the base for induction. Attackers benefit from highly recognizable messages that maximize the number of continuous bytes they can guess. An extremely common and easy-to-recognize message is the dynamic host configuration protocol request. DHCP is used to pass information to hosts on a TCP/IP network.[1] In general, to recover an initial pseudorandom stream, an attacker listens to packets sent from wireless clients, which he or she filters for certain properties. For example, an attacker might use DHCP in these ways:

- A wireless host sends a packet to the broadcast MAC address ff:ff:ff:ff:ff:ff
- The packet contains the 802.11 ToDS and WEP flags set, indicating that the broadcast is destined for the host (wired) network and that it is an encrypted packet.
- The packet is larger than the minimum DHCP request size (802.11 header + LLC (logical link control) + IP header + UDP header + DHCP request data).
- The packet came from a host whose MAC address recently appeared on the network (making it more likely that host will request an IP address).

Once the attacker has recovered what seems to be an encrypted DHCP request, he or she simply makes a best guess at the plaintext request and combines that guess with collected ciphertext, as Figure A (next page) shows. Table A enumerates the potentially guessed fields from a DHCP packet that an attacker could use. Many of the fields are predetermined, while others simply have common values used in well-known implementations.

Reference

1. R. Droms, Dynamic Host Configuration Protocol, Internet Engineering Task Force RFC 2131, Mar. 1997, www.ietf.org/rfc/rfc2131.txt.

Table A. Predictable fields in DHCP requests.

FIELD	BYTES	COMMENT
Logical link control	8	Same for all IP packets
IP Header		
Version/header length	1	Same for all IPv4 packets with no options
Differentiated service	1	Default 0 × 00 common
Length	2	Provided by packet characteristics
Identification	2	0 × 00 0 × 00 for many clients
Flags/frag offset	2	0 × 00 0 × 00 (no flags, no fragments)
Time to live	1	Commonly used values are 128 and 16
Protocol	1	User datagram protocol (UDP), 0 × 11
Header checksum	2	Calculable for any guessed header
Source address	4	0.0.0.0 (client yet to have an address)
Destination address	4	255.255.255.255 (broadcast)
UDP header		
Source port	2	DHCP, 68
Destination port	2	DHCP, 67
Length	2	Provided by packet characteristics
Total	34	

their use of the already small IV space. After the publication and subsequent implementation of the Fluhrer, Mantin, and Shamir passive attack, vendors quickly turned their attention to attack mitigation.

The only way to prevent the FMS passive attack is to prevent the attacker from seeing a sufficient number of packets encrypted using weak IVs. For many vendors, this means filtering out all potentially weak IVs, thereby reducing the total number of IVs available for transmission. Depending on the number of IVs being filtered, this increases the potential for IV reuse.

In the test environment of our implementation, a major vendor's access point filtered an entire class of IVs, thereby reducing the size of the dictionary needed by the attacker and hastening the attack by a factor of more than 45. Initial tests showed that when the vendor received many attack packets in a short time period, the vendor's access point began reusing IVs with an alarmingly high frequency in packets sent to the attacker. In this environment, an attacker can begin to use the network and can likely decrypt arbitrary responses. Furthermore, we obtained high reuse to a single interface each time we ran the attack, prompting a full analysis of the vendor's IV selection.

We conducted tests on different access points from the same vendor using the vendor's three most recently released firmware versions. We also took statistics for environments with different numbers of hosts and a variety of network traffic patterns. Analysis revealed a simple pattern in all cases: the vendor does not use IVs with a second byte larger than 0×03, reducing the IV space to 2^{18}, a remarkably low 15.625 percent of the total 2^{24} possible IVs. Reuse at such a high level gives an overwhelming advantage to an attacker.

Table 2 compares the effectiveness of a varying number of hosts where there is high IV reuse and little or no IV reuse. The estimates for the multihost columns are best-case in that they assume full parallelism. The two tested access points differed drastically in recovery rates (time to recover a single IV). Because it had much faster packet throughput, the nonfiltering access point's recovery time far exceeded the expected 45 hours. Thus, increasing access point performance significantly speeds up the attack. Had the faster, nonfiltering access point used the same filtering as the slower access point, the phase three time would have been an alarming 17 minutes.

Attack mitigation

Among the possible techniques for mitigating attacks are proprietary message integrity checks, limitation or detection of packets with failed ICVs, protocol filtering, and dynamic rekeying. As always, the best solution for a network depends on the network's requirements.

WEP's ICV does not ensure data integrity and is directly responsible for the success of bit-flip and dictionary attack techniques against WEP.[5] One possible solution is to use an additional cryptographically sound message integrity check that uses a keyed input. Simply replacing the ICV with a cryptographically sound hash would not protect against the inductive attack, because the attacker could still compute the function over arbitrary data and inject it without knowing a shared secret.[5] While some vendors have begun to implement keyed MICs, the systems remain proprietary, thereby limiting interoperability.

Because the attack is active, a number of aspects make its detection feasible in some instances. During phase two, a single source generates numerous packets with incorrect ICVs. Many access points provide statistics for administrators, who could identify an extreme rise in ICV failures and conclude that something anomalous had occurred. In practice, although many ICV failures occur during phase two, the inductive step takes a relatively short time, and, without an automated system, audit logs will probably not provide enough information to identify the attack. Even if the administrator could detect the attack, it is unclear what action he or she could take. The simplicity of MAC spoofing makes simply blocking traffic from a particular host useless. Rekeying is perhaps the most useful course of action, but without automation it is often a tedious process. Detection, therefore, is likely not

continued from p. 34

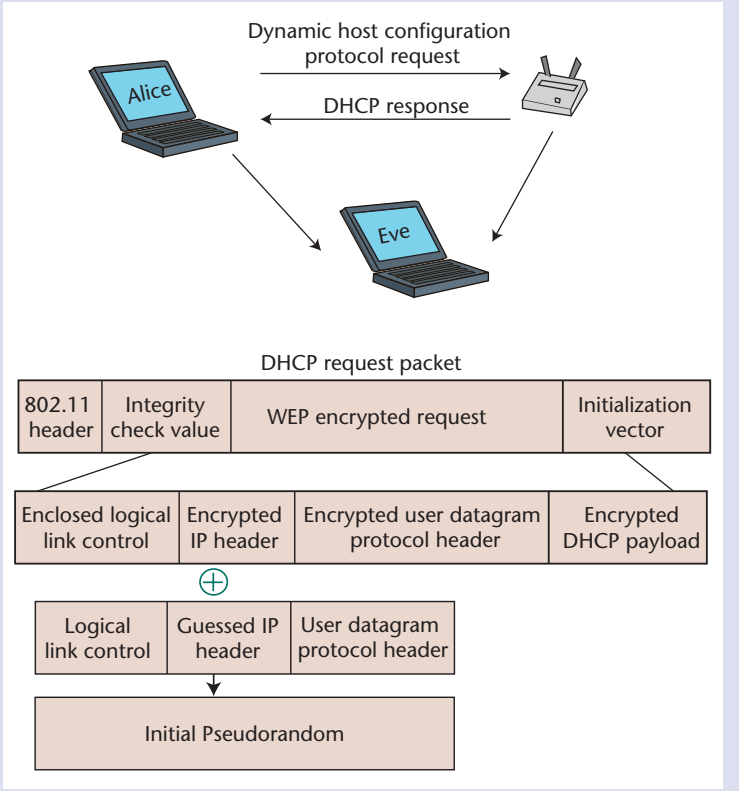


Figure A. Recovery of an initial pseudorandom stream from a dynamic host configuration protocol (DHCP) request.

Table 2. The effects of IV reuse on attack time (phase three only).

NUMBER OF ATTACKERS	1	2	1	2
Number of IVs	2^{18}	2^{18}	2^{24}	2^{24}
Recovery rate (msec/IV)	11.527	11.527	3.800	3.800
Total time (hours)	.839	.420	17.609	8.805
Tested	Yes	No	Yes	No

the most effective means of mitigation.

Packet filtering is a common method for mitigating active attacks. Several commercial access points provide this functionality and likely candidates for filtering are the protocols (such as ICMP) that make phase two easy. The problems with this approach are twofold. First, to prevent MAC spoofing, an administrator would have to unilaterally block such protocols, which would deny hosts access to services they might find useful or necessary. Second, and more importantly, it is difficult to identify which protocols can and cannot be used for the attack. Creative attackers can likely find a way to use unblocked protocols to their advantage. There are simply too many possibilities for key stream induction and no reasonable way to limit protocol usage. Thus, protocol filtering is not an option for mitigating the inductive attack.

Protocols using dynamic rekeying provide excellent resistance to the inductive attack. An attacker's ability to build a dictionary relies on the limited number of possible pseudorandom streams. With a properly implemented dynamic key system, an attacker cannot build a dictionary for any one key in use. While partial dictionary building and other forms of key stream attacks are still possible, dynamic rekeying eliminates an inductive attacker's ability to become a peer on the network. The primary disadvantage to dynamic keying systems is the amount of administration and infrastructure necessary in most implementations. Some systems require proprietary hardware, while others rely on public key infrastructure.

To address the issues surrounding interoperability of rekeying designs, the WiFi Alliance recently announced WiFi Protected Access, a new security architecture that would be available as a firmware upgrade to much of the currently deployed equipment. WPA provides for per-packet keys and a lightweight message authentication code—message integrity code in IEEE vernacular. Combining these security mechanisms would prevent our inductive attack as well as all currently known integrity and confidentiality attacks against 802.11-based networks. You can find more information at http://cedar.intel.com/media/pdf/security/80211_part2.pdf.

As we have shown in this article, the improper design of a system can have critical consequences on its overall security. While adequate design is difficult and expensive, the mitigation of flaws arising from poor design can be nearly impossible. More often than not, the only true solution is wholesale system redesign. To this end, WPA should not be considered a panacea. WPA mitigates WEP's security problems and, as we have demonstrated, mitigation can sometimes have unwanted effects. WPA should therefore be viewed as only an interim solution until the AES-based RSN is released in early 2004. □

References

1. "LAN MAN Standards of the IEEE Computer Society. Wireless LAN medium access control (MAC) and physical layer (PHY) specification," *IEEE Standard 802.11*, IEEE Press, 1997.
2. J. Walker, "Unsafe at Any Keysize: An Analysis of the WEP Encapsulation," *IEEE 802.11-00/362*, IEEE Press, 2000; www.netsys.com/library/papers/walker-2000-1027.pdf.
3. R. Braden, ed., *Requirements for Internet Hosts—Communication Layers*, Internet Engineering Task Force RFC 1122, Oct. 1989, www.ietf.org/rfc/rfc1122.txt.
4. J. Postel, ed., *Internet Control Message Protocol*, IETF RFC 792, Sept. 1981, www.ietf.org/rfc/rfc0792.txt.
5. N. Borisov, I. Goldberg, and D. Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11," *Proc. 7th Int'l Conf. Mobile Computing and Networking*, ACM Press, 2001, pp. 180–188.

Nick L. Petroni Jr. *is a second-year graduate student in the Department of Computer Science at the University of Maryland, College Park. His research interests include information security and wireless networks. He received a BS in computer science from the University of Notre Dame. He is a member of the IEEE and the ACM. Contact him at npetroni@cs.umd.edu.*

William A. Arbaugh *is an assistant professor in the Department of Computer Science at the University of Maryland, College Park. His research interests include information systems security and privacy with a focus on wireless networking, embedded systems, and configuration management. He received a BS from the United States Military Academy at West Point, an MS in computer science from Columbia University, New York, and a PhD in computer science from the University of Pennsylvania, Philadelphia. He is a member of the IEEE. Contact him at waa@cs.umd.edu.*