The Final Nail in WEP's Coffin

Andrea Bittau
University College London
a.bittau@cs.ucl.ac.uk

Mark Handley
University College London
m.handley@cs.ucl.ac.uk

Joshua Lackey
Microsoft
joshlack@microsoft.com

Abstract

The 802.11 encryption standard Wired Equivalent Privacy (WEP) is still widely used today despite the numerous discussions on its insecurity. In this paper, we present a novel vulnerability which allows an attacker to send arbitrary data on a WEP network after having eavesdropped a single data packet. Furthermore, we present techniques for real-time decryption of data packets, which may be used under common circumstances. Vendor produced mitigation techniques which cause frequent WEP re-keying prevent traditional attacks, whereas our attack remains effective even in such scenarios.

We implemented a fully automatic version of this attack which demonstrates its practicality and feasibility in real networks. As even rapidly re-keyed networks can be quickly compromised, we believe WEP must now be abandoned rather than patched yet again.

1. Introduction

Everyone knows that the *Wired Equivalent Privacy* (WEP) algorithm is broken, but only a minority manage to recover keys successfully in practice. The reason for this is partly because these attacks require long waiting times. As many believe that it is unlikely for an attacker to spend hours physically waiting around a network, people prefer to adopt WEP rather than seeking more sophisticated and possibly more difficult to manage security solutions. This will no longer be the case as with the *fragmentation attack* described in this paper, hours become minutes.

Although we evaluate the complexity of recovering a WEP key, the main scope of our work is to emphasize the threat of attacks which do not ever require the key. Some of these attacks were introduced in the past, although they were considered highly impractical. We provide the missing link, making these attacks even more practical (in the short-run) than ones which recover the key.

Region	WEP	WPA	802.11i
London	76	20	4
Seattle region	85	14	1

Table 1. Popularity (%) of encryption schemes used in encrypted networks.

1.1. Who Uses WEP?

There is no reason in scrutinizing WEP if it is no longer being used. However, contrary to popular belief, WEP is still highly deployed. To sustain our claims, we decided to provide some evidence that WEP really is out there.

Although *Wi-Fi Protected Access* (WPA) [32] has been available for some time, and 802.11i [19] is also now available, few networks use these more secure solutions. We wrote monitoring software to assess the current situation and conducted a survey of 400 wireless networks in London and 2,539 networks in the Seattle region. In both cases, about half of the networks used encryption. In London, 76% of the encrypted networks in our sample used WEP, and in Seattle 85% of them used WEP. Although vendors recommend upgrading to to WPA or 802.11i, only a minority of users seem to use these solutions. The complete results of our survey are summarized in Table 1.

Care was taken to allow legacy WEP hardware to support WPA, although in practice, new cards are frequently needed. This is especially true for WPA2 where previous cards were not fast enough to encrypt at full link speed using AES. Perhaps this is why WEP is still used—it is the only lowest common denominator that everyone supports.

For additional security, some vendors have recommended solutions that involve using WEP and dynamically re-keying it, in order to protect against its weaknesses. As we will show, such workarounds *do not* protect against all the attacks described in this paper. Thus, we believe that our discoveries on WEP have practical relevance even today, and should be cause for concern.

The rest of this paper is organized as follows. In the next sections we describe the operation of WEP followed

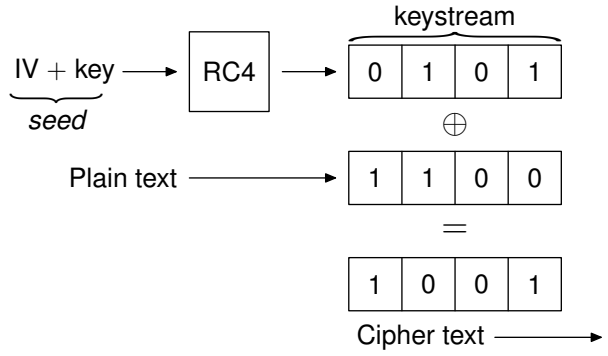

Figure 1. WEP encryption.

by a history of how it has been attacked. Our fragmentation attack is described in Section 2 which details how to transmit data, decrypt and ultimately recover the key. Section 3 analyzes the practical issues we needed to address to successfully implement the attack. Results, including cracking times, are presented in Section 4. In Section 5 we summarize the lessons learnt from WEP's history and we conclude in Section 6.

1.2. WEP Operation

WEP is a stream cipher used to encrypt the data portion of data packets in 802.11 networks. The 802.11 header is always in clear-text and both management and control frames are transmitted unscrambled. WEP uses RC4 [28] and the 802.11 standard [17] specifies a 40-bit pre-shared key, although 104-bit keys are commonly used today. Figure 1 shows the process of encryption. A *seed* is constructed by pre-pending a 24-bit *Initialization Vector* (IV) to the secret key. This seed is used to set up RC4's state via the *Key Scheduling Algorithm*. The output of RC4's *Pseudo Random Generation Algorithm* (known as a *keystream*) is XOR-ed with the clear-text to produce the cipher-text. Before encryption, a checksum (CRC32) of the data is appended to the message body and encrypted with it. Decryption is similar: the cipher-text is simply XOR-ed with the keystream, and the CRC is checked.

The IV chosen for producing RC4's seed is pre-pended in clear-text in the payload being sent. Normally, each data packet will have a different IV. In practice, implementations tend to use a linear counter for their IV generation. This counter increments by one, each time a packet is sent.

1.3. Keystream Details

Each different IV produces a different keystream, since the RC4 seed will have changed. A single secret key will therefore produce 2^{24} different keystreams. In essence, both WEP encryption and decryption are an XOR with one of these keystreams. To decrypt a packet, the keystream produced from the IV specified in the packet must be XOR-ed with the cipher-text. To encrypt a packet, the requirements are more relaxed: XOR the clear-text with *any* keystream. Although a normal station should not do so, it is possible to know a single keystream and transmit many different packets using the same keystream and IV. Thus, by recovering one keystream, an attacker may transmit any data and also decrypt payloads which use that keystream.

XOR (denoted as $\oplus$) has the property that $cipher \oplus clear = keystream$. Therefore, one way of recovering the keystream is by knowing the cipher-text and clear-text and by performing a XOR of the two. The cipher-text can be obtained by eavesdropping a packet. If the keystream is recovered from that cipher-text, then transmission may occur by always re-using the IV from the captured packet. That keystream may also be used to decrypt any future packets using that IV. However, the problem in calculating the keystream from the cipher-text is knowing the clear-text.

1.4. History of WEP Attacks

WEP has a long history of vulnerabilities and "fixes". Initial attacks did not seem very practical, so vendors preferred not to invest in shipping new security solutions, but rather, they provided patches to mitigate these difficult to achieve attacks. Attacks evolved over time, and new ones were discovered, posing more serious threats to WEP. Once again, the industrial response was in providing further mitigation techniques. In the following sections we summarize the main problems WEP faced and how vendors responded.

1.4.1. Brute-force

The most naive attack is a *brute-force* which tries all possible keys until the correct one is found. The 802.11 standard specifies a 40-bit WEP key. An exhaustive search on a single modern machine would require less than a month to complete—not impossible, especially if the task is distributed. It was noted that many implementations had an option of entering a human readable pass-phrase which would then be converted into a WEP key. Some of these implementations made use of a conversion algorithm which would generate keys with only 21 bits of entropy [30]. Other implementations would convert a pass-phrase into a hex key by using the ASCII values of its characters. These non-standard mechanisms for entering a pass-phrase often reduced considerably the difficulty of brute-force attacks. Perhaps, a standardized algorithm for hashing a pass-phrase into a WEP key would have prevented simple brute-force attacks, and would even have been useful (humans tend to prefer passwords rather than hex digits). Vendors provided support for 104-bit WEP which was effective against brute-force.

1.4.2. Keystream re-use

Cryptanalysis on WEP demonstrated that the algorithm's security is independent from its key [21, 11]. Early attempts to improve WEP's security by increasing its key size were therefore futile. If a keystream is recovered, it is possible to decrypt data which uses that keystream, and to transmit. The clear-text must be known in order to recover a keystream. There was no major concern for these vulnerabilities because it was considered impractical to know the clear-text of data.

Mechanisms for reliably discovering a keystream were later discovered [2, 8, 27]. The most practical relied on *Shared Key Authentication* to be enabled, which was a mechanism for preventing un-authorized access to a network. In this scheme, the AP sends a clear-text challenge to the authenticating peer. The peer authenticates by responding with the encrypted version of the challenge. By snooping this transaction, the attacker has a cipher-text and plain-text pair which he may XOR in order to recover a keystream. The 802.11 standard identifies this scenario and discourages stations for re-using the IV from this handshake, since future traffic using it may be decrypted. What it fails to mention is that an attacker may transmit indefinitely by using that keystream.

The response to this attack was to discourage the use of this authentication scheme and to use mechanisms such as SSID cloaking and MAC address filters. Of course, this approach is flawed—eavesdropping an association request and spoofing a MAC address are both trivial. There was no real pressure to eliminate the problem of keystream re-use. The main argument was that a network has 2^{24} keystreams making the complete attack too complex. Despite the observation made that in practice clients use a limited number of these keystreams, re-use still was not seen as a major threat.

1.4.3. Weak IV Attacks

Further study on WEP revealed that the key could be calculated [13]. This attack required gathering $\approx 1,000,000$ packets of which some used "weak" IVs. Actually, these RC4 properties were first noted four years before WEP became available [31]. A single weak IV reveals a correct key byte 5% of the time. By gathering a high number of statistics (IVs) the most probable key may be calculated. Weak IVs were seen as a major threat to WEP, perhaps because for the first time there was an automated tool which could recover the key. Now even technically unskilled hackers could compromise a network. The response was in building hardware which would filter these weak IVs. By patching this issue, vendors have just made the keystream re-use vulnerabilities more serious—there are now even fewer than 2^{24} keystreams. The weak IV attack was nevertheless mitigated, since only in certain circumstances and after having gathered a great deal of statistics, could the key be recovered. It could take days.

It turned out that there were more weak IVs than the ones originally published [15]. Vendors had to implement new filters although the problem was already becoming obvious. A single legacy host could compromise the entire network. Furthermore, weak IVs which yielded correct results with higher probability (13%) were noticed [29] although their details were never published. Vendors could not implement filters for IVs which were privately being exploited [5, 24] although their only salvation was the fact that these attacks still required long waiting times (≈500,000 packets). These IVs were later rediscovered and made public [23].

In practice, even after the public releases of tools which used high probability IVs, a large number of packets ($\approx 1,000,000$) could still be needed to recover the key. This is due to the fact that most hardware filters the old weak IVs, so the attacker can only use the new IVs. The hardware protections and the discovery of the new IVs effectively canceled each other out. We still have not seen hardware which filters the higher probability IVs. Perhaps, this is so because the public implementations which exploit these IVs do not use use filters, whereas the previous generation of tools did, allowing vendors to simply copy the filter.

1.4.4. Modern Attacks

There are two main problems with the attacks of the past. One is how to recover a keystream reliably and the other is how to speed up the weak IV attack. Both have been solved. It is possible to recover one byte of keystream after sending at most 256 packets [22, 1] (a method will be described in Section 2.4.2). To speed up the weak IV attack, it is possible to replay WEP packets. If a packet which elicits a response is replayed, traffic will be generated on the network and the attacker no longer needs to passively wait for data—he can actively cause traffic which may use a "weak IV".

At this point, vendors realized that WEP was dead. A skillful attacker can compromise a network in hours by using these vulnerabilities. However, the mitigation process continued as it was realized that if a WEP key is changed often enough, it was possible to eliminate the practical threat—attackers will not have enough time to compromise the network. EAP based solutions which frequently re-key emerged and are being recommended by vendors [10, 9, 18].

1.4.5. Our Contribution

We discovered a novel attack which may compromise a WEP network quickly and reliably. Our *fragmentation attack* is still applicable against networks which re-key frequently, since it may be performed almost instantly. By publishing this work we hope it becomes obvious that WEP

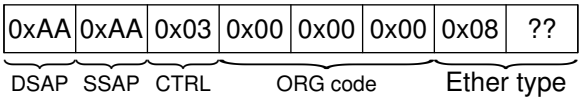

Figure 2. LLC/SNAP header contained in practically all 802.11 data frames.

must not be used and patches to it will not solve the core issues. Furthermore, we hope this paper serves as a case study as to what happens in practice when things go wrong. Even though attacks only seemed theoretical back in 2000, serious action had to be taken immediately. Until today, advances have been made in breaking WEP but there has not been such a push in trying to eliminate it completely.

2. The Attack

In the next sections, we describe a series of attacks which may be used to transmit and decrypt data on a WEP network. In general, transmission is simple and the complexity of decryption is what distinguishes attacks.

We begin by presenting a further design flaw in WEP—layer 2 fragmentation. We illustrate how it may be used on its own in order to compromise a network. Following this, we discuss how fragmentation can interplay with other attacks. We group attacks into two classes: those that fully compromise a network without recovering the WEP key, and those that recover the key.

2.1. Known Plain-text in Packets

Our fragmentation attack requires knowledge of a small portion of keystream. After that, transmission of arbitrary data may occur. Many keystream based attacks were thought impractical because they required plain-text knowledge. Our attack will have a different fate since such knowledge can be as minimal as a couple of bytes, which turn out to be readily available.

The initial portion of 802.11 packets is virtually constant. A packet commences with an LLC header followed by SNAP as shown in Figure 2. These two headers occupy the first eight bytes of a packet. The only "unknown" field is the *ethertype* which occurs at the end of the SNAP header. The ethertype will normally be either ARP or IP. ARP packets are easily distinguished by their fixed size of 36 bytes and are usually destined to a broadcast address. Some hardware pads short packets to a minimum length, making ARP packets longer. By inspecting the MAC address prefix of the AP, one can determine the hardware being used [16] and may judge whether or not short packets are being padded. Since we can differentiate between IP and ARP based on

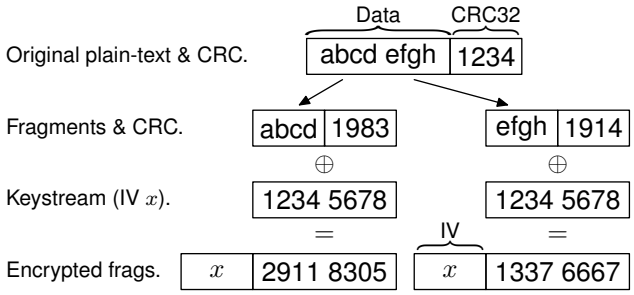

Figure 3. Transmitting a single logical packet in multiple 802.11 fragments.

the length of the packet, we assume that for each packet at least the first eight bytes of plain-text is known.[1]

By intercepting a packet and knowing the first eight bytes of plain-text, eight bytes of keystream may be calculated by performing clear-text $\oplus$ cipher-text. By using this keystream, it is now possible to send eight bytes of encrypted payload. The payload will be composed of four bytes of data followed by its CRC32. Nothing practical can be achieved by sending four data bytes since the LLC/SNAP header alone requires eight bytes. Therefore, this result was never considered to be an issue.

2.2. Fragmentation in 802.11

Little scrutiny has been done on how WEP interacts with the rest of the 802.11 protocol. Most of the past attacks focused solely on WEP's weak cryptographic properties. However, it is exactly in this interaction that WEP's largest flaw emerged—802.11 may be used against WEP.

The 802.11 standard specifies fragmentation at the MAC layer and each fragment is encrypted independently. It is possible to send multiple 802.11 fragments (up to a maximum of 16) each using the same keystream. By sending payloads in eight byte fragments, it is possible to inject $4 \times 16 = 64$ bytes of data (each payload requires a CRC32).

Therefore, after snooping one data packet, the attacker is able to recover at least eight bytes of keystream. By using 802.11 fragmentation, the attacker may immediately transmit up to 64 bytes of arbitrary data. Figure 3 illustrates 802.11 fragmentation.

The 802.11i standard motivates the TKIP MIC by noting that WEP was vulnerable to "Fragmentation attacks" [19]. We still wonder what exactly they referred to, and whether they knew about this attack. All we know is that we informed CERT about this issue back in 2003, although we were never properly followed up.

[1] In reality much more is known. For example, ARP packets contain plenty of known information, especially because MAC addresses are clear-text in the 802.11 header.

2.3. Pure Fragmentation Attack

We now analyze how the fragmentation technique may be used alone in attacking WEP networks connected to the Internet. Broadly speaking, an attacker desires two things: to be able to transmit and decrypt data. Fragmentation permits both.

Transmission is trivial and does not require Internet connectivity. The attacker needs to eavesdrop one data packet and recover eight bytes of keystream. This is easily accomplished since the first eight bytes of clear-text are known and the cipher-text has been intercepted. After that, the attacker may use 802.11 fragmentation for transmitting data of up to 64 bytes. IP fragmentation may be used on top for sending larger packets.

Traffic may also be decrypted in real-time using 802.11 fragmentation, provided that the 802.11 network is connected to the Internet. To do this, an attacker can use the AP to decrypt. Suppose that the attacker has eavesdropped an encrypted packet x and wishes to decrypt it. Clearly the AP knows the encryption key, and if the attacker replayed the packet, the AP would decrypt it. By itself, this would not be useful, as the replayed packet would simply be forwarded to its original destination. However, the attacker can use fragmentation to simply prepend an additional IP header to the front of the eavesdropped packet. With 802.11, only the data portion is encrypted, and there is no sanity check in order to ensure that what was originally a complete payload cannot be replayed as a fragment. Upon reception, the AP will decrypt both the new header and the original packet x, and reassemble them into a single packet. If the new header contains an Internet address, the AP will send the packet there in clear-text (WEP protects only the wireless link). If the attacker controls the Internet host the packet was sent to, he can recover the clear-text of x.

This may be accomplished by constructing an IP header in four byte fragments using the recovered eight bytes of a keystream, followed by a further larger fragment containing the entire unmodified encrypted payload x. The AP will decrypt, de-fragment, and send off the data to the Internet in the clear. Figure 4 illustrates this process, although for clarity, only one fragment is drawn for the IP header. In the following sections, we will address some of the details regarding the decryption strategy of this attack.

2.3.1. Forwarding to the Internet

To transmit to an Internet host, two pieces of information are required: the router's MAC address and a source IP address. Obtaining the router's MAC address is not difficult, especially since the 802.11 header which contains MAC addresses is always in clear-text. Often, the AP itself will act as a router and its MAC address may be obtained from the

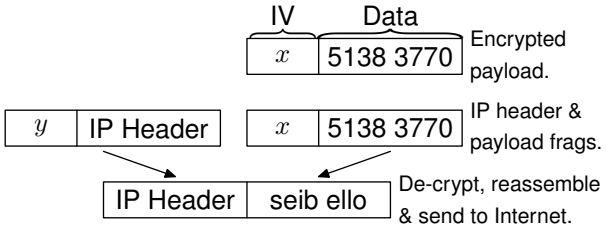

Figure 4. Decryption by using the AP to re-send data in clear-text to the Internet.

802.11 header in its Beacon frames. Another possibility is determining which MAC address seems the most popular and infer it being an Internet gateway.

Depending on the network configuration, a correct source IP address may not be needed for transmission. Some public IP networks allow IP spoofing, and many NATs translate all packets they route, regardless of their source address. In Section 2.4.2, we present other techniques for obtaining the router's MAC address and a source IP address.

2.3.2. Redirecting MTU Packets

Decrypting data by prepending an IP header will not work for packets which will exceed the *Maximum Transmission Unit* (MTU) size. Only packets no longer than 28 bytes less than the MTU may be re-sent, since at least 20 bytes are required for prepending the IP header, and the decrypted payload will include the LLC/SNAP header. In practice, this is not a limitation. Firstly, attackers will want to decrypt authentication data, which is normally transmitted in short packets (*e.g.* POP3 credentials). Secondly, many APs act as routers and will IP fragment the larger than MTU payload when forwarding it. For completeness though, we discuss techniques which may be used to re-direct even MTU-sized packets.

Bit-flip destination address. The first technique is to bit-flip the destination IP address in the original payload directly. The problem is knowing the original destination and IP checksum. For TCP flows, the attacker could intercept the SYN (≈ 40 bytes) and redirect it to the Internet using our technique. If the attacker has an out-of-band reverse channel, such as GPRS, then the original IP addresses are now known. The techniques presented by Borisov *et al.* may then be used to bit-flip the destination address of future packets [8].

It is also possible to receive this information back using covert signaling from the attacker's Internet "buddy" host. Although the attacker does not know the contents of the now-encrypted incoming messages, he

may see their length which can be used to encode data. To do this, the attacker must choose a valid source address—in Section 2.4.2 we will show how this too can be achieved.

Chop-Chop. There is a mechanism by which packets may be truncated while still keeping them valid [22]. This proceeds a byte at a time by dropping a byte, guessing its clear-text, and calculating the change to the payload which yields a valid message. The payload change is dependent only on the byte being chopped. For each ≤ 256 packets sent, the payload can be truncated by one byte. By chopping 28 bytes from the tail of the message, it is now possible to prepend an IP header and a new LLC/SNAP header.

ICMP "packet too big". With *Path MTU Discovery* [26], TCP packets are sent with the *Don't Fragment* bit set. If a router with a low MTU needs to fragment such a packet, it drops it and sends back an ICMP "packet too big" message. The TCP sender will reduce its *Maximum Segment Size* to the new path-MTU as indicated in the ICMP message and send future data in smaller chunks.

The ICMP message contains 64 bits of the TCP header of the packet that was too large. In theory, the sender might check some of these bits. However, in our experiments with Linux 2.6, FreeBSD 5.4, Windows XP SP1 and MacOS X Tiger, none of the implementations did any real sanity checking on this information, so long as the encapsulated destination IP address was correct. We can decrypt the IP header directly by relaying the small TCP SYN packet. None of the remaining TCP information in the ICMP message needs to be valid.

It is therefore trivial to spoof such an ICMP message and lower the MTU for a particular destination. Since the victim will now transmit arbitrarily smaller packets for that destination, an IP header may be prepended to them allowing them to be decrypted.

2.3.3. Summary of Fragmentation-only Attack

In short, after eavesdropping a single packet, it is possible to send arbitrary data immediately. In the case that the network is connected to the Internet, and the attacker knows the router's MAC address and the network's IP prefix, it is possible for the attacker to capture data and re-send it to the Internet, where it will arrive in clear-text. Therefore, frequent WEP re-keying, even on a per-minute basis, is not sufficient to prevent this attack completely since it may be performed in real-time. We summarize the attack as follows:

Requirements. For decryption, the network must be connected to the Internet. The router's MAC address and a source IP address are needed. Techniques for obtaining these parameters are discussed in Section 2.4.2.

Recover Keystream. Eavesdrop a packet. If the packet's length is 36 bytes,[2] its type is ARP, otherwise it is IP. Recover eight bytes of keystream by performing a XOR of the first eight bytes of cipher-text and the known plain-text for ARP or IP (depending on the packet's length).

Transmit. Transmit data by sending multiple eight-byte 802.11 fragments, using the keystream recovered. If a large IP packet needs to be sent, IP fragmentation may be used in conjunction.

Decrypt. Eavesdrop a packet to decrypt. Send an IP header, destined to a controlled Internet host, in multiple 802.11 fragments using a known keystream. Send the eavesdropped packet as the last fragment. The Internet host will receive the payload as clear-text.

2.4. Keystream Based Attacks

The limitation of the pure fragmentation attack is that it requires co-operation of an Internet host for decrypting data. It is not applicable to private networks. In the following sections, we illustrate how fragmentation may aid keystream based attacks. These allow compromising networks, even without Internet connectivity.

The basic idea is to discover either all possible keystreams (dictionary attack) or specific ones. If a packet is eavesdropped and the attacker knows its corresponding keystream, he may XOR the two and obtain the plain-text.

2.4.1. Discovering Keystreams

The practical problem with past keystream based attacks is that there was no immediate method for recovering a single full (1500 byte) keystream. Furthermore, since a network uses at most 2^{24} keystreams, attacks which built a dictionary of all these keystreams were considered impractical since one would need to know 16M cipher-text and plain-text pairs. With fragmentation, recovering a single full keystream takes seconds. One could then use this knowledge to recover other keystreams.

Consider sending a large broadcast frame in small fragments. The AP will reassemble it and relay it as a single large frame, since it has no need to fragment it. If the attacker originated such a frame, the clear-text is obviously known to him. By eavesdropping the frame relayed by the AP, the attacker can recover the keystream for the new IV chosen by the AP. This process is illustrated in Figure 5.

[2] See Section 2.1 regarding hardware which pads short packets.

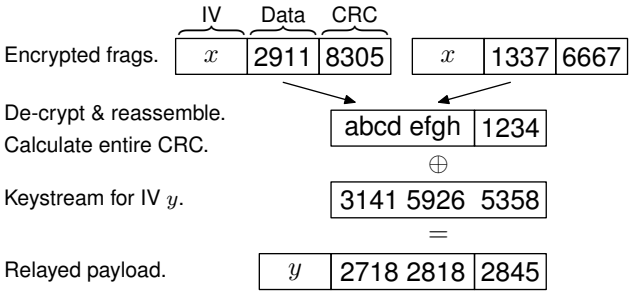

Figure 5. Discovering a keystream by causing the AP to relay broadcasts.

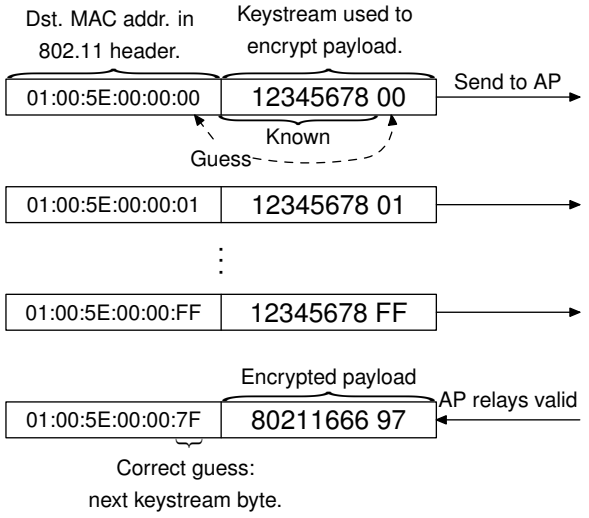

Figure 6. Expanding a specific keystream in a linear amount of time.

If 64 bytes of data were sent in 4-byte fragments, the AP will relay a single 68 byte payload (4 bytes for the CRC32). The attacker can now send 64 bytes in each of 16 fragments, resulting in a 1028 byte relayed frame. Therefore, by sending a total of 34 fragments, 1500 bytes of keystream are recovered. Fragmentation therefore enables an attacker to discover a 1500 byte keystream almost immediately, after having eavesdropped a single data packet.

To recover other keystreams, the attacker can now send 1500 bytes (without needing fragmentation) and snoop the relayed version from the AP, which will most likely use a different IV. By sending $\approx 2^{24}$ (16M) packets, a complete IV dictionary may be built. In practice, stations initialize their IV to zero and increment it by one for each packet sent. Therefore, if stations remain on a network for a limited amount of time, possessing a small number of keystreams may enable decryption of their traffic.

2.4.2. Discovering a Specific Keystream

When building the IV dictionary via broadcasts, a new keystream is recovered each time (since the AP chooses its next IV when relaying). At times, it is necessary to decrypt a specific packet. Doing so requires knowledge of a particular keystream which may not yet have been recovered.

One decryption mechanism is Chop-Chop [22] which discovers the keystream of a packet back to front. Our approach is similar but works front to back.[3] We find it more useful, especially when seeking for data in the initial portion of the payload.

The technique proceeds as follows. Suppose that an encrypted payload which uses an unknown keystream has been eavesdropped and needs to be decrypted. The initial eight bytes of its keystream may be recovered, since their plain-text is generally known (as previously described). Therefore, a broadcast packet with eight bytes of payload which uses that keystream may be sent. The AP will relay it since the packet is valid. One can guess the next byte of the keystream and send a longer broadcast packet by using the extended keystream. If the guess is correct, the AP will relay it. After at most 256 guesses, the AP will relay one of them. Therefore, one can determine the next keystream byte and proceed by recovering the byte following that. After the whole keystream has been recovered, the plain-text of the eavesdropped packet may be revealed with a XOR operation between the cipher-text and the newly discovered keystream.

Instead of using a timer to determine whether or not the AP relayed the current guess, one can exploit the fact that MAC addresses are in the clear in the 802.11 header. All 256 guesses for the next keystream byte may be sent "in parallel" to 256 different multicast addresses. The AP will relay only one and the attacker can read off the correct guess from the multicast MAC address. Therefore, after sending at most $256 \times 1488 = 380,928$ packets, an arbitrary packet may be decrypted. If the packet to decrypt is short, less traffic needs to be generated because less keystream needs to be recovered. Also, if a timer implementation (or hybrid) is used, half this number of packets will be required on average. This process is illustrated in Figure 6.

The main use we found for decrypting specific packets is for determining a source IP address in the network. ARP packets are a particularly good candidate. Their header is shown in Figure 7. The *type* field is either a request or reply. If the packet is a broadcast, then the type is a request, else it is a reply. The first real unknown value is the source IP

[3] We discovered this mechanism while implementing the fragmentation attack. We later found out that it is very similar, if not the same, to the attack described in [1].

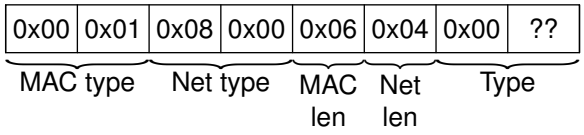

Figure 7. Format of the ARP header. This header is followed by the source MAC and IP addresses.

address, which follows the ARP header. Decrypting the first three bytes of the IP address will yield enough information for obtaining a source IP address. Therefore, after sending $\frac{256\times3}{2} = 348$ packets on average, the network address may be obtained. In practice, heuristics for common IP address ranges can further reduce this.

2.4.3. Summary of Keystream Attacks

Fragmentation buys us fast keystream recovery and real-time Internet-based decryption. With fragmentation, a 1500 byte keystream may be recovered after sending 34 packets. If fragmentation were disabled, the same length keystream would be recovered after $\frac{256\times1488}{2} = 190,464$ packets on average.

To decrypt by re-sending to the Internet, two packets (new IP header & original payload fragments) must be sent for each packet to be decrypted. To decrypt without Internet access would require $\frac{256l}{2}$ packets on average, where l is the length of the packet. Thus, decrypting 1500 bytes would require $\approx 190,464$ packets instead of two (in the case that the AP is a router). Frequent re-keying might mitigate such linear decryption attacks, but if fragmentation is used, real-time decryption is possible. We present a summary of how one might attack networks without Internet connectivity:

Requirements. Recover eight bytes of keystream. This may be easily achieved, as discussed, by eavesdropping a data packet.

Expand Keystream. Send large broadcasts in multiple smaller 802.11 fragments. Eavesdrop the relayed broadcast from the AP and XOR the cipher-text with the clear-text in order to recover a larger keystream. Repeat the process until a 1500 byte keystream is recovered.

Build IV Dictionary. Transmit 1500 byte broadcasts. The AP will most likely relay the data with a new IV. Recover the keystream for that IV and continue this process until all the keystreams have been recovered.

Decrypt. If a packet uses a known IV, lookup its keystream in the dictionary and XOR it with the cipher-text. Otherwise, XOR the first eight bytes of that packet with the known plain-text. Send 256 multicasts of nine bytes each, using a different guess for the ninth keystream byte. Use a different multicast address for each guess. Read the correct guess from the relayed multicast packet. Continue this process until the keystream for the whole packet has been recovered. The clear-text is obtained by a XOR with the keystream recovered.

2.5. IV Dictionary and Weak IV Attacks

The simplest way to enable receipt of arbitrary data is to build a dictionary of IVs and their corresponding keystreams by causing packets with known clear-text to be forwarded by the AP or by other wireless stations. The weak IV attack can also be performed on the gathered data to try and recover the key. In fact, recovering the key this way will usually take much less time than building a complete dictionary, but a small IV dictionary may still be effective if there are multiple stations on the wireless network using IVs from the same part of the IV space. Fragmentation is of no special assistance in performing either attack, other than quickly providing a means for sending arbitrary data.

The weak IV attack may only be performed after a large number (≈500,000–3,000,000) of data packets have been eavesdropped. A widespread optimization to the weak IV attack is replaying data in the hope of generating more traffic. We found that the ability to send arbitrary data can help achieve better results. The standard replay attack has two main problems. First, not all packets are "replayable"; many will not generate a response. Second, transmitting and receiving with the same wireless card is not always reliable, hence these attack implementations normally advise using two cards: one for replaying and the other for eavesdropping.

Being able to inject arbitrary data provides better ways for generating traffic in the network. A simple example is generating a broadcast ICMP echo request. If the LAN has many hosts, it is likely that our single request generates multiple replies.

If the network has Internet connectivity, an attacker might cause a remote controlled Internet host to flood the wireless network with UDP packets. This way, the attacker only needs to passively receive traffic on the wireless network, and can perform the attack with a single card. The only requirement is that the attacker needs to send an ARP and UDP packet once in a while to keep the router's ARP cache updated and to keep open firewall holes or the NAT (if applicable). The fragmentation attack provides the means to quickly bootstrap such a flood.

Another advantage is that the Internet host can flood using very small packets. The weak IV attack only uses the first few (1–3) bytes of payload. Sending a high volume

of small packets thus speeds up the data collection process. Without flooding, it could take weeks before enough data is gathered—real wireless networks are often rather quiet. By doing a simple replay attack, it is possible to recover a key within hours. With our more elaborate flooding techniques, the attack may be further accelerated significantly.

If flooding from the Internet is used to build an IV dictionary, care must be taken with the *Time To Live* (TTL) field in the incoming packets. Attackers do not normally care too much about correctly decrypting the TTL value (and the IP checksum which depends on it) unless they wish to send packets with these IVs. However, should they care about this for some reason, they could use the keystream expansion technique for decrypting the TTL. Since the TTL value for a given route is normally constant (varies upon routing changes) an attacker needs to decrypt its value only once. Decrypting the TTL byte will require the transmission of ≈ 128 packets. By knowing the TTL, the attacker may calculate IP checksums and correctly recover the keystreams of future flooded packets. If the attacker is concerned about the TTL changing, he may verify it by periodically sending one packet—a broadcast using a recently discovered keystream. If the AP relays it, then the TTL did not change.

2.6. Summary of all Attacks

The attacks presented so far allow transmission and decryption of data on a WEP network. In all cases, after eavesdropping a single data frame, fragmentation enables immediate arbitrary data transmission. Furthermore, if the network has Internet connectivity, thanks to fragmentation it is possible to decrypt traffic in real-time. This was not possible with past attacks.

Decryption of frames, without the aid of fragmentation or an Internet host, may be achieved in linear time ($\propto$ packet length). Each decrypted byte requires 128 packets to be sent on average. Putting all these techniques together, an attacker could attack a network in the following manner:

1. Eavesdrop a data packet.

2. Recover eight bytes of keystream by performing a XOR operation with the known plain-text. The IP or ARP variant of the known plain-text may be chosen based on the length of the eavesdropped packet. At this point, transmission of arbitrary data (up to 64 bytes) is possible via 802.11 fragmentation. IP fragmentation may be used for sending larger IP packets.

3. Recover 1500 bytes of keystream by sending large broadcasts in smaller fragments (requires the transmission of 34 packets). At this point, transmission of arbitrary data (of any length) is possible even without 802.11 fragmentation.

4. If an external communications channel is available, try to obtain the network's IP prefix and the router's MAC address directly. Attempt to re-send an eavesdropped data packet to a controlled Internet host by using fragmentation, a random source IP address, and assuming the AP is the router. The AP will decrypt the packet which is then received by the remote host, and returned to the attacker.

5. Otherwise, obtain the network's IP prefix by decrypting the IP address in a packet by using the linear keystream expansion technique. It requires the transmission of ≈ 384 packets if an ARP packet has been eavesdropped.

   It is now possible to communicate with hosts on the LAN.

6. Obtain the router's MAC address. This is only necessary if communication with the Internet is required:

   - Try using the AP's MAC address. Many APs are routers.
   - Try the most popular MAC address seen.
   - Send an ARP request for the IP address ending in ".1" (use heuristics). Intercept the ARP reply (if any) and read the MAC address off the 802.11 header (in clear-text).
   - Send a DHCP request and decrypt the response (if any) by using the linear keystream expansion technique.

   It is now possible to communicate with the Internet.

7. Decrypt "interesting" data.

   - If Internet connectivity has been established, resend data to the Internet buddy. Data packets may now be decrypted in real-time, with an overhead of sending two packets: an 802.11 fragment containing an IP header followed by another fragment with the original encrypted payload.
   - If the Internet is unavailable, use the linear keystream expansion technique. For each byte to decrypt, the overhead is sending 128 packets on average.

8. Generate traffic in the network. For example, instruct the Internet buddy to flood the wireless network or send broadcast ICMPs.

   - Build an IV dictionary.
   - Perform the weak IV attack.

   Data which uses an IV present in the dictionary may be decrypted without additional overhead. The weak IV attack will eventually reveal the WEP key.

3. Implementation

Like all other WEP attacks, this particular one seems great in theory. However, we were keen to see if it worked in practice. The main hurdle was to determine whether the hardware available on the market would allow the sending of raw 802.11 frames. It is well known that many wireless cards support the commonly called "monitor mode" which allows the driver to read all raw 802.11 frames. It is less obvious how to inject data.

3.1. Prism2-based cards

The first attempt was made using Intersil Prism2 cards [20]. These have a mode for creating an AP in software. The kernel is responsible for sending management frames and encapsulating data frames—exactly what is needed for this attack. The `airjack` [25] driver for Linux uses this mode to allow a user to inject raw 802.11 frames and receive traffic concurrently. Unfortunately, after experimentation, it became clear that the Prism2 firmware changes some fields in the 802.11 header before transmission, including the sequence and fragment number.

There is, however, a work-around for this problem, mentioned briefly by Bellardo *et al.* [4]. Prism2 cards have an auxiliary (AUX) debug port which provides raw access to the card's memory. The basic idea is to queue the packet in the normal way for transmission and locate its header via the AUX port. Just after instructing the card to transmit, one can busy wait reading a header byte (such as the *duration*) through the AUX port until it is modified. At that point, the firmware has done its processing and is about to send the packet off to the radio. Just before it is able to transmit the packet physically, the modified packet header bytes can be re-written via the AUX port. This is a race condition which is virtually always won in practice.

Our implementation was for the FreeBSD `wi` driver. We implemented the `airjack` functionality and also the AUX overwriting. There were also other subtle aspects which were encountered and needed to be resolved. For example, the More Fragments bit needs to be set only when the overwrite is performed and not when the packet is first queued. Another issue is that when the WEP bit is set, it is cleared after the firmware processing although the packet does indeed get transmitted using WEP. However, it is important not to set the WEP bit when performing the re-write or the card will not transmit. Obviously none of this is documented which caused the authors to have great fun.

The main limitation with our Prism2 implementation has to do with the fact that reception is difficult immediately after transmission. For example it is not possible to receive the 802.11 ACKs after each fragment is sent, reducing the reliability of the attack in noisy environments.

3.2. Atheros-based cards

Atheros [3] cards are mostly software radios, making them ideal for packet injection. We modified the FreeBSD `ath` driver in order to allow the sending and reception of raw frames. Reception is easily achieved by simply changing the RX filter to accept all frames, including control frames. Control frames turn out to be very useful since the card is able to see ACKs for data being sent, making it possible to implement re-transmissions in the attack.

Atheros will readily send out WEP fragments. It does mangle the fragment and sequence number, but the fix is simpler than with Prism2—the packet needs to be queued with a type of '2' (which indicates PS-Poll frames). In order to eliminate re-transmissions from the firmware, a flag indicating that the packet requires no ACKs exists.

The only limitation with our Atheros implementation is its inability to send ACKs in time. The temporary work-around is to connect another wireless card and use its MAC address while performing the attack. Its firmware will automatically respond to data packets with ACKs.

3.3. Proof of Concept: `wesside`

We implemented a proof-of-concept tool called `wesside` [6]. Its main purpose is to recover the WEP key, since we felt that was the long-term objective of most attackers, although we have already demonstrated that it is possible to decrypt traffic even without such knowledge. The tool will therefore use the fragmentation attack for generating a high volume of traffic in order to speed up the weak IV attack. When launched with no command line arguments, it does the following:

1. Channel hops looking for a WEP network.
2. It then tries to authenticate and associate. If authentication fails, it attempts to find a MAC address to spoof by eavesdropping associated clients.
3. After it eavesdrops a single data packet, it discovers at least 128 bytes of keystream by sending out larger broadcasts and intercepting the relayed packets.
4. After it eavesdrops an ARP request, it decrypts the IP address by guessing the next four bytes of the keystream using multicast frames.
5. It floods the network with ARP requests for the decrypted IP address.
6. It launches `aircrack` [12] (v2.1) every 100,000 packets captured for one minute and attempts to decrypt the key. If 3,000,000 packets have been captured so far, the cracking time is increased to ten minutes and cracking is started every 1,000,000 packets.

The tool could be much smarter, and it is, but only when launched with at least one command line argument: the IP address of an Internet host. The tool acts as described previously but with the following differences.

First, the address of the router is discovered. After the IP network address is decrypted, an ARP request for the IP address terminating with ".1" is sent and the reply is waited for (the tool assumes this will be the router). Although the ARP response is encrypted, the MAC address of the router is all we need, and this is in clear-text in the 802.11 header. The IP address ending in ".123" is assigned to the attacker. A full "product" could use smarter heuristics, but these are sufficient for a proof-of-concept and actually work in most cases. As he can send traffic, the dedicated attacker might send a DHCP request and decrypt the response via the keystream expansion technique.

Next, a pathway into the network from outside is opened. After the MAC address of the router is obtained, a UDP packet to the Internet host is sent every five seconds to open a hole in any firewall or NAT. ARP requests also continue to be sent to maintain the attacker's IP address in the router's ARP cache. This allows IP traffic to be forwarded back to the attacker. Finally, depending on the attacker's choice, the Internet host may either send short packets to perform the weak IV key attack, or it may send long packets to build a keystream dictionary.

The tool also binds to a TAP interface in order to implement the dictionary attack. A TAP interface is a virtual network interface, normally used by VPN software. If the user transmits via the TAP interface, `wesside` will encrypt the data using a known keystream and transmit it to the wireless network. If data with a known IV (an IV which has an entry in the dictionary) traverses the network, our tool will decrypt the packet and originate it from the TAP interface. This mechanism allows the attacker to use the wireless network (without knowing the key) as if it were connected via the TAP network interface. Transmission will always be possible although decryption is limited by the number of entries in the IV dictionary and their popularity.

3.3.1. Missing Enhancements

Only a limited subset of the attacks we described have been implemented in `wesside`. Our primary goal was to explore the complexity of building a fully automatic tool and its efficiency in recovering the key of a WEP network.

In fact, the attacks we believe to be the most dangerous and cunning are missing. For example there is no built-in support for decrypting data by re-sending it to the Internet. We tested this attack separately. An implementation could be built once again by using TAP interfaces. Transmission is achieved just like in `wesside`. For decryption, all data packets (or those which match a filter) on the wireless network may be re-sent to the Internet. If the attacker has an external Internet connection, his buddy may forward the data back to him. The data may then be originated from the TAP interface, just as if it were decrypted by using the IV dictionary. The TAP interface will look exactly like a normal wireless interface, although no WEP key is required.

The other main attack which is not directly available to the user is decryption via keystream expansion, although there is an API for it, as `wesside` uses it internally. Because of the nature of the attack ($\approx$ 128 packets sent per byte decrypted) filters are necessary in order to decrypt only very specific data packets which look "interesting". Decryption using this technique takes under five minutes per packet.

The only keystream based attack implemented is the dictionary attack. The main reason for including this attack was to explore how usable its implementation would be. Thanks to TAP interfaces and making `wesside` act in a similar way to a VPN client, the resulting implementation turned out to be very simple to use.

4. Evaluation

In this section, we show the performance that may be achieved when using `wesside`. This is important, as it establishes the degree to which real networks will be vulnerable. The extent to which the fragmentation attack is a serious threat in practice will be evident from the results.

Our tests use common everyday hardware. The setup for all the experiments uses a Linksys WRT54G AP. The "attacking" host is a Pentium IV 2.4GHz laptop with 512MB RAM and an Atheros 802.11g card. The "Internet" host is an old Celeron 400MHz laptop with 200MB RAM. This setup is best-case in terms of connectivity with the "Internet" since the flood host is connected via Ethernet to the WAN port, effectively having a 100Mbit/s connection. When presenting the results, we will calculate how much bandwidth is actually being used in order to demonstrate that this throughput could be achieved on most real Internet links.

In all cases, once the `wesside` tool is started, a *single* ARP request is generated by a host attached to the LAN port of the AP. This is the requirement to bootstrap the fragmentation attack. Although any packet type will allow keystream determination, IP discovery has only been implemented on ARP packets and for that reason such a packet is generated. This requirement does not distort results too much, as IP packet headers (and thus the source address) may be decrypted in the same way as ARP packets, without taking too long. Also, the tool could force a wireless client to disconnect by spoofing a de-authentication management frame. When the client (automatically) re-associates, it will most likely send an ARP request for its router.

The efficiency of `wesside` is measured in two stages. The first stage is bootstrapping, which involves recovering a keystream and determining the network parameters and configuration before flooding may commence. The metric used in this stage is time—the quicker the better.

The second part of the attack either involves cracking the key, building a dictionary, or both. In all cases, the more data packets received, the better. For this part of the attack, the metric used is packets received per second.

4.1. Bootstrap speed

The bootstrap of the attack includes determining a keystream, determining an IP and determining the router's MAC address. All of this is independent of the key size and its complexity, since no attacks are being performed on the key scheduling algorithm. Thus, the metric obtained for the bootstrapping procedure should be very similar across all networks and configurations.

The results are as follows. From when the first data packet is eavesdropped, it takes one second to determine 144 bytes of keystream (using the broadcast-relay technique with fragments) for the IV the AP used in relaying the packet. For revealing 1500 bytes of keystream, it takes less than two seconds. It takes about six seconds to decrypt a single byte of the IP address in a packet which uses an unknown IV (using the multicast-guess technique) and less than 30 seconds to decrypt the whole IP address. Finally, determining the router's MAC address takes less than a second if it has an IP address ending with ".1". This will work in many cases. Another way of instantly forwarding packets would be to use the AP's MAC address as a destination—often APs are also routers.

In short, after having eavesdropped a single ARP packet, it takes less than a minute for an attacker to be able to transmit any data and determine an IP address on a WEP wireless network. This metric is quite network independent as no assumptions are being made about the WEP key. In the next section, we describe how much traffic we were able to generate on a network after the bootstrap of the fragmentation attack has been completed.

4.2. Flood rate

The are two main ways in which a network may be flooded. First, we can send ARP requests. This emulates the simple ARP replay attack, which is good for establishing a baseline for comparison. Alternatively, we can flood from an Internet-connected host, which this tool makes possible. From an Internet host, we have the choice of flooding using short packets, which is useful for the weak IV attack, or flooding using MTU sized packets, which are useful for both building a dictionary and for weak IV attacks. However, the packet rate will be lower with larger packets. The results of our experiments, summarized in Table 2, are as follows.

Traffic source	$\approx$ p/s
Local 802.11b client FTP download.	150
ARP replay-like attack.	350
Internet flood (short packets).	1200
Internet flood (MTU sized packets).	250

Table 2. Approximate packet rates reached with different traffic sources.

- The simple replay attack can generate $\approx$ 350 unique (no re-transmissions) packets per second.
- When Internet-flooding using small packets, the maximum sustainable rate was $\approx$ 1200 p/s. These are UDP packets with 5 bytes of data, so each packet will be be 47 bytes in size (including Ethernet header). Thus the Internet traffic required to support this rate is approximately 440 Kb/s, which is an achievable downstream on a typical ADSL link.
- When Internet-flooding using MTU-sized packets for populating the dictionary, we can sustain a rate of $\approx$ 250 p/s. This corresponds to a data rate of 2.8 Mb/s, which is not always feasible. Such a data rate would yield a full dictionary in $\approx$ 17 hours. Lower link speeds would increase the time to construct the dictionary proportionately.

If the Internet link is slow, it may still be possible to obtain high flood rates. For example, the Internet host may send ICMP echo requests to a client on the wireless LAN. This will amplify the traffic by a factor of three—the AP relays the request to the client, the client replies and the AP relays the client's reply. Another technique would be sending broadcast or multicast ICMP echo requests on the local network. A single request can cause many replies, especially on large networks. Simple replay attacks cannot benefit from these methods.

4.3. Cracking time

Although key cracking using the weak IV attack is not the main purpose of this work, some results are presented here. Initially, all cracking is run for less than one minute and with the default "fudge" factor (the breadth with which the key is searched amongst the possible candidates) of two. Cracking is attempted every 100,000 packets captured. After three million packets have been captured, the cracking time is increased to ten minutes and cracking occurs every

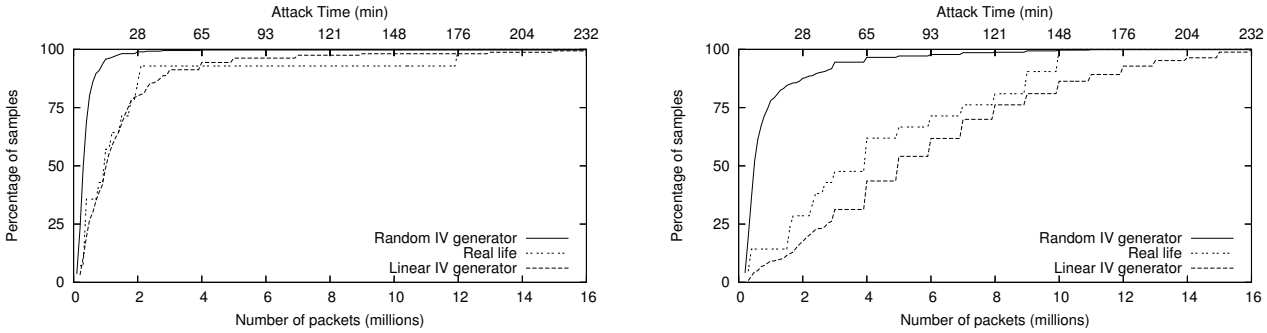

Figure 8. Cumulative distribution of packets required for cracking 40 bit (left plot) and 104 bit keys.

million packets because, as the number of packets increases, the time to load the cracking process and perform the calculations increases. Therefore, it is not sensible to crack for only one minute when a very large number of packets had to be processed.

To obtain a larger data set, we developed a simulator. It generates packets with IVs starting at zero, and initiates key cracking periodically as described earlier. For the results to be realistic, the simulator had to perform weak IV filtering just as our test AP did. After some experimentation, we discovered that the Linksys AP skips IVs according to the filter present in BSD airtools [14].

The results of these simulations are displayed in Figure 8, with label "Linear IV generator". With 40-bit keys, the median number of packets required to crack the key is one million. With two million packets, 80% of the 40-bit keys could be obtained. When cracking 104-bit keys, one would expect to need five million packets, with 80% of the 104-bit keys obtainable after ten million packets received. The plot for 104-bit keys contains evident steps after three million packets. This is because we model `wesside`'s behavior of initially attempting to crack the key every 100,000 packets, but after three million packets have been received, only attempting to crack the key every million packets.

The attack time (top axis) is estimated by dividing the number of packets required by the 1200 p/s flood rate figure. The key cracking time is also included in this value—ten minutes when more than three million packets are needed, one minute otherwise. According to the simulator, 50% of the 40-bit keys are obtained in under 15 minutes, and half of the 104-bit keys are recovered in less than 80 minutes.

An interesting result emerged when we changed the simulator to explore the behavior if IVs are chosen randomly, which might have been a potential workaround for the dictionary attack. This is shown in Figure 8, under the label "Random IV generator". It is clear that choosing random IVs greatly assists the cracking process. This is because weak IVs are scattered in clusters across the IV space, and so a random strategy provides more varied usable statistics than an incremental strategy.

We believe it should be possible to exploit this effect by causing both the AP *and* the clients to generate traffic. Assuming that the IV counters of the stations are at different phases, the packets generated will cover a larger distribution of IVs and speed up the cracking process. From a key recovery perspective, the most effective way of flooding may be generating local ICMP echos to broadcast or multicast addresses—all stations should generate a reply. Therefore, the distribution of packets required for recovering a key in practice probably lies somewhere between the two curves generated by our simulator, posing even a greater threat than the results we are presenting.

To validate our simulation results, we cracked real keys on our AP using `wesside`. The curves with labels "real life" in Figure 8 show the results, which indeed closely follow the simulation curves. From the 128-bit plot, it is evident that cracking keys in real-life requires fewer packets. We believe that this difference is due in part to packet loss—`wesside` sees more of the IV space for the same number of packets received than the simulator sees. Covering more IV space, even with missing packets, is a better strategy because of the clustering of weak IVs. Also the x-axis under reports slightly in the real-world case, as some additional packets are received while the cracking process loads data.

4.4. IP Redirection Experience

We also separately implemented decryption using the AP to graft a new IP header onto the front of an eavesdropped packet. The AP successfully decrypts the eavesdropped data, allowing the now clear-text packet to be forwarded to an Internet host. This technique works perfectly. In fact, we expected we might also have to add a TCP, UDP or ICMP header in the first fragment to satisfy the NAT in the AP, but this proved unnecessary with the WRT54G base station, which happily forwarded the packets anyway. A proper external firewall might restrict such traffic at some corporate sites, necessitating the addition of extra transport headers to allow this attack to succeed.

As IP redirection is performed without needing to know-

ing either the WEP key or the keystream for a specific IV, it may occur immediately. Thus, solutions which frequently re-key WEP are still vulnerable. This is the most serious threat that the fragmentation attack poses to WEP. Even though a 104-bit WEP key can be recovered in a couple of hours, a dedicated attacker may be able to decrypt traffic immediately.

5. Lessons

The WEP fiasco is a good example of how things can go wrong in the real world when the guidelines that theory teaches us are not followed. For example, preventing keystream re-use really does matter. If it is not prevented, it is possible to build a dictionary of keystreams and decrypt future traffic. Furthermore, it might be possible to use such a keystream to inject data. If a keystream could not be re-used in WEP, all the keystream based attacks presented would not apply, and it would not be possible to replay data.

It is well known that a strong message integrity check is a must, in order to avoid the possibility of forging messages [7]. The keystream expansion techniques would not be feasible if WEP messages included such a check rather than the simple 32-bit CRC.

Fragmentation is used in three ways in our attacks: to transmit arbitrary data, to expand few bytes of known keystream into an entire MTU, and to graft a new header onto an eavesdropped packet while using the AP to decrypt it. Preventing keystream re-use would prevent all of these. However, relying on a single defense is never ideal, as it only takes one oversight for the entire system to fail. A message integrity check that depends on the key as well as the fragment contents would also be effective against these specific attacks. However there might still be a risk of replay attacks that spliced together replayed fragments from different packets. Thus, an integrity check used to bind fragments together into a packet would also be needed. This too should be obvious though—it is the integrity of a packet that matters, not of an isolated fragment.

Finally, the key lesson is that the security of a protocol must be designed in the context of the protocol itself. We showed how a networking property of the 802.11 protocol, namely fragmentation, could be used to break 802.11's cryptography. The system must be seen as a whole—its security must not be designed in an isolated manner.

WEP is also a good example of how attacks evolve and mature over time. If the threat was eliminated when early researchers discovered WEP's flaws, there would not be issues today. Instead, because of the perceived impracticality of early attacks, WEP remained widespread. Getting rid of it now is much more difficult than it was in 2000, simply because there are many more networks today.

Walker's and Simon's attacks on keystream re-use were never considered a threat. A year later, Arbaugh, and Borisov *et al.* resurrected those attacks by noting the vulnerability in Shared Key authentication. With the weak IV attack, Fluhrer *et al.* resurrected Wagner's weak RC4 keys of 1995. In 2004, the high probability weak IVs went public and resurrected the Fluhrer *et al.* attack. Today, we resurrect the 2000 keystream re-use attacks once more. What will happen? Will vendors disable fragmentation? We presented a linear keystream expansion technique which is independent of fragmentation. Will vendors disable short packets? Chop-chop can be performed by decrypting from the tail of the packet. There may well be other approaches, perhaps already in use but not publicized. WEP is fundamentally flawed and needs to be totally abandoned rather trying to win a cat and mouse chase.

6. Conclusions

The fragmentation attack proves to be highly practical. From the point where an encrypted packet is eavesdropped, it takes less than a minute to bootstrap to the point where an attacker can send MTU-sized packets and know the IP address range for the local subnet. From this moment, he can redirect encrypted traffic to a host on the Internet, using the AP to decrypt the traffic. From the point of view of secrecy, the wireless network is already completely compromised.

If the ability to receive traffic via the network is also needed, active attacks bootstrapped using the fragmentation attack will recover 40-bit WEP keys in perhaps fifteen minutes and 104-bit WEP keys in an hour or two. WEP networks with low traffic were considered to be "safe" since it would require an attacker to wait many hours, perhaps days, before the key could be recovered. Similarly, solutions which frequently re-key WEP were thought sufficient since the attacker does not have enough time, even by replaying data, to recover the key before it changes. With our attack however, even in such conditions, traffic may still be injected and redirected almost instantly after a single data packet traverses the network. We believe that the fragmentation attack was the final missing link in providing an efficient and practical mechanism for breaking WEP.

Many lessons have already been learned from WEP's problems. For example, WPA's *Message Integrity Check*, a cryptographic hash function which depends on the key and payload, and its mandatory use of the IV as a sequence number provide improved protection against the sort of packet modification and replay attacks used in the fragmentation attack. In WPA, keystreams may not be re-used even when the IV space wraps, since a re-key will occur in that moment.

However, the interaction between fragmentation and encryption has not been widely discussed. In particular, some simple changes to WEP would have made this attack much

more difficult, despite WEP's flawed design. A wireless AP is especially helpful to an attacker when it relays broadcast packets. There is no good reason why the AP needs to relay these packets in any other form than that in which they were received:

- By de-fragmenting before relaying, the AP unnecessarily allows an attacker to bootstrap from a little knowledge to knowing an entire keystream.
- By relaying with a different IV from that generated by the attacker, the AP permits the attacker to expand his knowledge from one keystream to many.
- By de-fragmenting two fragments with unrelated IVs, the AP allows the attacker to use it to decrypt packets and relay them to arbitrary destinations.

Of course, WEP would still have been a flawed design without these elementary errors. However, we note that the first two errors are preserved in WPA. At least for now, no-one seems to have found a way to exploit them.

Acknowledgments

We are thankful to David Hulton, Anton Rager and Michael Lynn for providing us with information on this attack. Many thanks to David Wagner who assisted us in revising this paper, Vern Paxson who suggested our title, and Alex Lee who lent us his Atheros card.

References

[1] W. A. Arbaugh. An Inductive Chosen Plaintext Attack Against WEP and WEP2, 2001.

[2] W. A. Arbaugh, N. Shankar, and Y. J. Wan. Your 802.11 Wireless Network has No Clothes, 2001.

[3] Atheros Communications. Atheros chipset. `http://www.atheros.com`.

[4] J. Bellardo and S. Savage. 802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions. In *Proc. USENIX Security Symposium*, Washington D.C., August 2003.

[5] A. Bittau. Additional weak IV classes for the FMS attack. 2003. `http://www.cs.ucl.ac.uk/staff/a.bittau/sorwep.txt`.

[6] A. Bittau. `wesside`, 2005. `http://www.cs.ucl.ac.uk/staff/a.bittau/frag-0.1.tgz`.

[7] J. Black and H. Urtubia. Side-channel attacks on symmetric encryption schemes: The case for authenticated encryption, 2002.

[8] N. Borisov, I. Goldberg, and D. Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In *Proc. ACM Mobicom*, Rome, Italy, July 2001.

[9] Cisco Systems. *Protected Extensible Authentication Protocol (PEAP)*.

[10] Cisco Systems. *Lightweight Extensible Authentication Protocol (LEAP)*, 2000.

[11] D. Simon and B. Aboba and T. Moore. IEEE 802.11 security and 802.1X, 2000.

[12] C. Devine. `aircrack`, 2004. `http://www.cr0.net:8040/code/network/`.

[13] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. *Lecture Notes in Computer Science*, 2259:1–24, 2001.

[14] D. Hulton. `bsd-airtools`, 2002. `http://www.dachb0den.com/projects/bsd-airtools.html`.

[15] D. Hulton. Practical Exploitation of RC4 Weaknesses in WEP Environments. Feb 2002. Message to Bugtraq mailing list, `http://www.dachb0den.com/projects/bsd-airtools/wepexp.txt`.

[16] IEEE. MAC address prefix list (OUI). `http://standards.ieee.org/regauth/oui/oui.txt`.

[17] IEEE Computer Society. *ANSI/IEEE Standard 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999.

[18] IEEE Computer Society. *IEEE Standard 802.1X: Port-Based Network Access Control*, 2001.

[19] IEEE Computer Society. *IEEE Standard 802.11i: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 6: Medium Access Control (MAC) Security Enhancements*, 2004.

[20] Intersil. Prism2 chipset. `http://www.intersil.com/globespanvirata`.

[21] J. R. Walker. Unsafe at any key size; an analysis of the WEP encapsulation, 2000.

[22] KoreK. chopchop (Experimental WEP attacks), 2004. `http://www.netstumbler.org/showthread.php?t=12489`.

[23] KoreK. Next generation of WEP attacks?, 2004. `http://www.netstumbler.org/showpost.php?p=93942&postcount=35`.

[24] J. Lackey. An Implementation of all possible Weak IV Classes, 2002.

[25] M. Lynn. `airjack`, 2003. `http://sourceforge.net/projects/airjack/`.

[26] J. Mogul and S. Deering. Path MTU discovery. RFC 1191 (Draft Standard), Nov. 1990.

[27] A. Rager. `WEPWedgie`, 2003. `http://sourceforge.net/projects/wepwedgie/`.

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

[29] A. Stubblefield, J. Ioannidis, and A. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. In *Proc. Symposium on Network and Distributed System Security*, San Diego, California, Feb 2001. Internet Society.

[30] Tim Newsham. Cracking WEP Keys Applying known techniques to WEP Keys, 2001. `http://www.lava.net/~newsham/wlan/WEP_password_cracker.pdf`.

[31] D. Wagner. Weak Keys in RC4, 1995. `http://www.cs.berkeley.edu/~daw/my-posts/my-rc4-weak-keys`.

[32] Wi-Fi Alliance. Wi-Fi Protected Access (WPA). `http://www.wi-fi.org`.