Wireless Insecurities

MICHAEL STHULTZ AND JACOB UECKER

Internet Forensics Laboratory
Center for Cybermedia Research
University of Nevada, Las Vegas
USA

HAL BERGHEL

University of Canterbury
New Zealand
and
Internet Forensics Laboratory
Center for Cybermedia Research
University of Nevada, Las Vegas
USA

Abstract

The popularity of wireless networking is a function of convenience. It addresses one of the most important goals in advanced computing technology: mobility. When viewed conceptually, wireless technology can be seen as having a contemporaneous, parallel evolutionary path with remote login, distributed, and nomadic computing in the area of computing and car portable, car and cellular phones in the area of telecommunications.

This chapter provides an overview of the wireless landscape and the security mechanisms that have been introduced in an effort to protect Wireless Local Area Networks (WLANs). It then gives a detailed description of these mechanisms including a discussion of their inherent weaknesses. The conclusion is there is no effective WLAN security available in today's environment.

ADVANCES IN COMPUTERS, VOL. 67
ISSN: 0065-2458/DOI 10.1016/S0065-2458(05)67005-2

1. The Wireless Landscape

Wireless networking is one of the most recent technologies whose usage has exploded in the last decade. The operational metaphor behind wireless technology is mobility. The concept of wireless networking dates back at least as far as ALOHANET in 1970. While this project is now of primarily historical interest, the online overview is still worth reading (http://en.wikipedia.org/wiki/ALOHA_network). The concept of ALOHANET spanned many of the core network protocols in use today, including Ethernet and Wireless Fidelity (aka WiFi®). ALOHANET was the precursor of first generation wireless networks.

Wireless technologies may be categorized in a variety of ways depending on their function, frequencies, bandwidth, communication protocols involved, and level of sophistication (i.e., 1st through 3rd generation wireless systems). For present purposes, we'll lump them into four basic categories:

(1) Wireless Data Networks (WDNs),
(2) Personal Area Networks (PANs),
(3) Wireless Local Area Networks (WLANs), of which the newer Wireless Metropolitan Area Networks (WMANs) and Wireless Wide Area Networks (WWANs) are offshoots, and
(4) satellite networks.

WDN is a cluster of technologies primarily related to, developed for, and marketed by vendors in the telephony and handheld market. This market covers a lot of ground from basic digital cellular phones to relatively sophisticated PDAs and tablet PCs that may rival notebook computers in capabilities. WDN includes protocols such as the Cellular Digital Packet Data (CDPD), an older 19.2 Kbps wireless technology that is still in use in some police departments for network communication with patrol cars; General Packet Radio Service (GPRS) and Code Division Multiple Access 2000 (CDMA2000) which are multi-user, combined voice and data 2.5 generation technologies that exceed 100 Kbps; and Wireless Application Protocol (WAP) which provides wireless support of the TCP/IP protocol suite and now provides native support of HTTP and HTML. If you're using a cellular phone with text messaging and Web support, you're likely using some form of WAP.

PANs began as "workspace networks." Bluetooth, for example, is a desktop mobility PAN that was designed to support cable-free communication between computers and peripherals. Blackberry (http://www.blackberry.com) is like Bluetooth on steroids. It integrates telephony, web browsing, email, and messaging services with PDA productivity applications. As such it blurs the distinction between PAN and WLAN.

WLAN is what most of us think of as Wireless technology. It includes the now-ubiquitous 802.11 family of protocols, as well as a few others. Table I provides a quick overview of some of the 802.11 protocol space. Note that all but the first are derivative from the original 802.11 protocol introduced in 1997.

We note in passing that both the 802 and 802.11 landscape is somewhat more cluttered than our table suggests. For example, 802 also allows for infrared support at the physical layer. In addition, proprietary standards for 802.11 have been proposed. In

TABLE I
THE 802.11 PROTOCOL FAMILY

Standard	802.11	802.11a	802.11b	802.11g	802.11n
Year	1997	1999	1999	2003	2006
Frequency	2.4 GHz	5 GHz	2.4 GHz	2.4 GHz	?
Band	ISM	UNII	ISM	ISM	?
Bandwidth	2 Mbps	54 Mbps	11 Mbps	54 Mbps	300+ Mbps
Encoding techniques	DSSS/ FHSS	OFDM	DSSS	OFDM	?

"Year" denotes approximate year of introduction as a standard (e.g., 802.11a and 802.11b were introduced at the same time, though 802.11a came to market later). The two bands used for "WiFi" are Industrial, Scientific and Medical (ISM) and Unlicensed National Information Infrastructure (UNII). Bandwidth is advertised maximum. Encoding, aka "spectrum spreading" techniques appear at the physical or link layer and include frequency-hopping spread-spectrum (HPSS), direct-sequence spread-spectrum (DSSS), and orthogonal frequency division multiplexing (OFDM). (Source: [2].)

2001 Texas Instruments proposed a 22 Mbps variation of 802.11b called "b+," and Atheros proposed a 108 Mbps variant of 802.11g called "Super G." Further, there are standards for enhanced QoS (802.11e) and enhanced security (802.11i) that are actually orthogonal to the traditional 802.11 family in the sense that they deal with limitations rather than the characteristics of the protocol suite. To make comparisons even more confusing, there are 802.1x protocols like 802.16 (2001), 802.16a (2003) that are designed for wider area coverage, the so-called "Metropolitan Area Networks" or MANs. The 802.11n specifications are thin as of this writing, although the current attention is on increasing throughput at the MAC interface rather than the physical layer.

1.1 The WLAN Environment

WLAN offers many advantages: e.g., the ease and reduced expense of not having to run cabling through an existing building, communication between buildings, or just the convenience of not having to find a wall jack to establish a network connection. But this convenience comes at a price. While great care may have been taken through the use of firewalls and intrusion detection systems to secure a network's connection to the outside world, a WLAN creates another entrance into the network that is typically behind the firewall.

Wireless signals cannot be easily confined to their area of intended use. In fact, wireless communications can be monitored and captured from a mile or more away. And this covert monitoring activity is virtually undetectable.

A number of wireless security mechanisms have been introduced to address these problems. The first of these is an encryption and authentication standard called the Wired Equivalent Privacy, or WEP. More recent encryption and authentication protocols include EAP, WPA, and VPNs. Unfortunately, each of these has security vulnerabilities [1].

On a positive note, the 802.11i standard includes the Counter Mode/CBC-MAC Protocol (CCMP). CCMP is based on the Advanced Encryption Standard (AES) and should provide stronger encryption and message integrity than anything available now. Unfortunately, since CCMP will require new hardware that is incompatible with the older WEP-oriented hardware, it will probably be some time before this security mechanism is widely implemented.

It is important to note that nothing that is covered here isn't already understood and put into practice by the hacker and criminal communities. The people in the dark tend to be law-abiding citizens. It is hoped that the information presented here will raise awareness so that the defender stands a chance of protecting his digital assets against WiFi intrusion.

2. Basic Wireless Security

Some mention should be made of some basic wireless features that are often described as security mechanisms even though they are actually ineffective as deterrents.

(1) Disabling the Service Set Identifier (SSID) broadcast, and changing the name of the SSID to something other than the widely known default values. This will only serve to deter the inexperienced or lazy attacker since SSIDs can still be sniffed from association packets.
(2) MAC address filtering. Although MAC-based authentication is not a part of the 802.11 standard, it is a feature on many APs. MAC filtering can be easily bypassed though since the network traffic can be sniffed to determine which MAC addresses are being used. All an attacker has to do at that point is to force a host off of the wireless network (or just wait) and then assume that host's MAC address.
(3) Protocol filtering. Even if implemented on an access point, the range of attack vectors is so large at this point, that there are vulnerabilities that apply to whatever protocols are supported [15].

2.1 Wired Equivalent Privacy (WEP)

WEP is an algorithm that was a part of the original IEEE 802.11 specification with the design goals of preventing disclosure and modification of packets in transit and providing access control for the network. It uses the RC4 algorithm from RSA Security which was first designed in 1987 and kept as a trade secret until it was leaked on a mailing list in 1994. RC4 is a symmetric cipher, i.e., the key that encrypts the traffic is the same key that decrypts the traffic. It is also a stream cipher, meaning that it creates a stream of bits that are XORed with the plaintext (original data) to create the ciphertext (encrypted data). When the data reaches the other end, the same stream of bits is XORed with the ciphertext to retrieve the plaintext. RC4 uses a pseudo-random generation algorithm (PRGA) to create a stream of bits that are computationally difficult for an attacker to discover. This same stream of bits is reproduced at the other end to decrypt the data. Since RC4 is not supposed to be reused with the same key, the WEP designers added an Initialization Vector (IV) which is a value that changes for each packet. The IV is concatenated with the WEP key to form the WEP seed. Figure 1 outlines this process visually.

When a user inputs a key to configure the client wireless card, he or she must configure the same key on the opposite end of the communication (most likely an access point). Users provide either 40-bits or 104-bits of information for the secret

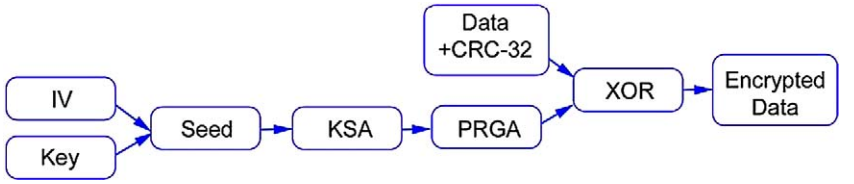

FIG. 1. The WEP encryption process.

key. While manufacturers may claim that the key is 64-bit or 128-bit, only 40 or 104 bits, respectively, are actually used for data. This remaining 24-bits are the IV which is pre-pended to the secret key before it is used in the key scheduling algorithm (KSA). This format is often symbolized by IV.SK, where IV symbolizes the 3 byte (24-bit) Initialization Vector, and SK symbolizes the 5 byte (40-bit) or 13 byte (104-bit) secret key from the user. This composite value is the input to the KSA which converts IV.SK into an initial permutation S of $\{0, \ldots, N-1\}$. The PRGA then uses this permutation to generate a pseudo-random output sequence. The algorithms can be seen in Fig. 2. Note that all additions are made modulo 256.

WEP contains a number of flaws in the implementation of RC4 that allows an attacker to completely compromise the intended security. This potential compromise is so well publicized and so complete, that WEP should never be considered a reliable form of security [3].

KSA(K)	PRGA(K)
Initialization: for $i = 0 \ldots N-1$ $S[i] = i$ $j = 0$	Initialization: $i = 0$ $j = 0$
Scrambling: For $i = 0 \ldots N-1$ $j = j + S[i] + K[i \bmod l]$ Swap($S[i]$, $S[j]$)	Generation loop: $i = i + 1$ $j = j + S[i]$ Swap($S[i]$, $S[j]$) Output $z = S[S[i] + S[j]]$
Where— $N = 256$ (for WEP) $K[x]$ = value of key (IV.SK) at index x l is the length of IV.SK	Where— z is the byte used to XOR the plaintext

FIG. 2. The pseudocode of the Key Scheduling Algorithm and Pseudo-Random Generation Algorithm.

2.2 Brute Force/Dictionary WEP Cracking

The secret key can usually be entered in hex format or ASCII format. Since a number of hexadecimal characters are much harder to remember then half as many ASCII characters, most people will choose a word or phrase that they can easily remember. This might make the key management much easier, but it also facilitates a brute force attack. There are utilities that try to decrypt WEP encrypted packets. decrypt, a utility that comes with AirSnort, is one such tool. It tries a list of words from a wordlist (aka dictionary) against a saved encrypted packet capture. With each possible key, it decrypts a packet using that key and computes a checksum on the newly decrypted data. If the checksum matches the checksum that was transmitted with the packet, a potential secret key has been found. If that same potential key works with another packet, the correct secret key has been revealed.

Brute force cracking requires more time and resources to find the correct key than dictionary attacks. This is the process of trying all possible combinations of values until the correct key is found. With a 40-bit secret key, there are a total of $2^{40} = 1{,}099{,}511{,}627{,}776$ possible secret keys. If a computer could check 50,000 different secret keys per second, it would take over 250 days to find the correct key. The amount of time that would be required to brute force a 104-bit key is measured in centuries.

The time required to brute force a 40-bit secret key can be brought down to under a minute due to a flaw in the random WEP key generation programs that was discovered by Tim Newsham [11]. Unfortunately, this flaw has been implemented in firmware by many wireless vendors.

These programs are supposed to create a set of random keys based upon ASCII input. The algorithm that is often used by these programs is the Neesus Datacom key generation algorithm. This algorithm takes a string of ASCII characters as input, arranges them in a two dimensional array with four characters in a row, and then XORs all of the column values together in sequence to get a 32-bit output value. The 4 byte (32-bit) output value is then fed through a PRGA which generates the 40-bit secret keys. See Fig. 3 for a visual representation.

Note that the most significant bit of each ASCII character is always zero and therefore the resulting bytes from the XOR operation also have a most significant bit of zero. This, combined with the PRGA algorithm that is used, only produces unique keys for seeds 00:00:00:00 through 00:7F:7F:7F. This greatly reduces the amount of effort required for a brute force attack. To prove this concept, Newsham created the toolkit wep_tools which will brute force keys that have been generated by this type of "random" WEP key generation utility.

A utility called WEPAttack also makes claims to WEP cracking efficiency using both brute-forcing and dictionary attacks against the key. The claim is that only one WEP encrypted data packet is necessary to start the attack. This is possible because

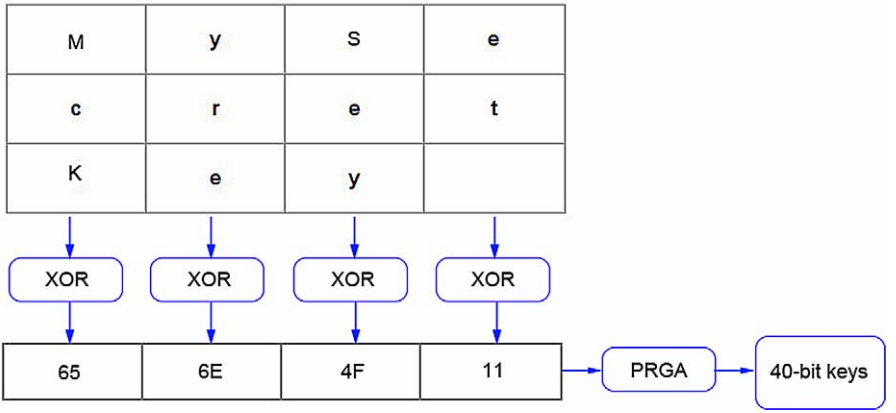

FIG. 3. XOR operation performed by the Neesus DataCom key generation algorithm.

each word in the dictionary is treated as the WEP key. WEPAttack uses the IV that is found in the encrypted data packet and prepends it to the words in the dictionary. This key is used to find the cipherstream which is XORed against the encrypted packet. If the decrypted packet starts with 0xAA (the standard SNAP header value), there is a good chance that the key has been found. More than one encrypted packet should be used for the processes since not all packets start with 0xAA. However, the chance that two packets, picked at random, are neither IP nor ARP (both start with 0xAA) is very unlikely.

2.3 The FMS Attack

The FMS attack is the most well known attack on WEP. It is derived from (and named after) Scott Fluhrer, Itsik Mantin, and Adi Shamir who published their research findings in a 2001 paper entitled "Weaknesses in the Key Scheduling Algorithm of RC4" [5]. The basis for this attack is a weakness in the way RC4 generates the keystream. Specifically:

1. The Initialization Vector (IV) that is always prepended to the key prior to the generation of the keystream by the RC4 algorithm is transmitted in cleartext.
2. The IV is relatively small (three bytes) which results in a lot of repetitions as the relatively small (16.78 million) number of unique IVs are re-used to encrypt of packets.
3. Some of the IVs are "weak" in the sense that they may be used to betray information about the key.

This weakness makes it possible, under certain conditions, to recover the input of RC4 (i.e., the key), knowing its first byte of output. This first byte of output is easy to determine since the first data to be encrypted in a WEP packet is usually the SNAP header (as with IP and ARP packets) and this header is almost always 0xAA.

A weak IV has a format of B+3:FF:X where B is the index of the shared key's byte currently being guessed, FF is 255, and X can be any number between 0 and 255.

Given a weak IV, the first several steps of the RC4 KSA that affect the leftmost positions of the table can be computed. There is then approximately a 5% chance that the remainder of the permutations of the KSA occur in the right part of the table. There is therefore a 5% chance that the portion of the table that was computed is the table that will be the input of the PRGA. Since the value determined for this shared key byte is only accurate 5% of the time, a number of weak IVs (usually about 60) with varying X's have to be used to compute guessed values for that byte. The value that is produced most often has a high probability of being the correct value. This process is then repeated to recover the remaining bytes of the shared key. As a rule of thumb, a few million packets generate enough weak IV traffic to recover 40-bit WEP keys. The attack is linear regardless of the key size so it does not take that much more traffic to recover a 104-bit key. A very good illustrated description of this process can be found in [6].

Since the IEEE standard of IV selection was so ambiguous, many wireless vendors use sequential IV generators that begin with 00:00:00 and wrap with FF:FF:FF. This is the worst of both worlds. Not only is this procedure guaranteed to generate weak IVs, but it does so predictably.

WEPCrack (http://wepcrack.sourceforge.net) was the first publicly released tool to use the FMS attack. Airsnort (http://airsnort.shmoo.com) is much better known and much easier to use. Since modern WiFi cards and appliances reduce the percentage of weak IVs that are generated (under the rubric of "WEP+" or "Advanced WEP Encryption," etc.), Airsnort is declining in importance as it takes an unreasonably long time to collect enough packets to break keys.

2.4 Enhancements to the FMS Attack

Subsequent to the original FMS research, a number of people have discovered that there are more ways that weak IV's can be used to speed up the WEP cracking process. "Using the Fluhrer, Mantin, and Shamir attack to break WEP," Stubblefield et al. [17] discusses an approach that deviates from the standard FMS algorithm methodology of finding all the previous values for B before finding the next value. The authors suggest that weak IVs associated with higher B values can be used to narrow down the beginning bytes of the secret key. This can be done by testing different values of the key and checking to see if the decrypted packet has a valid

checksum. This can be facilitated by making assumptions about the range of possible values for a WEP key (users will often use ASCII characters).

Another whitepaper, written by David Hulton, "Practical Exploitation of RC4 Weaknesses in WEP environments" [7] describes a number of alternate approaches for expanding the FMS concepts including additional methods of finding weak IVs for secret key bytes beyond the first. Hulton claims it would be best to devise an algorithm which can determine whether or not a particular IV can be considered weak. His algorithm (implemented in a utility called dwepcrack) is shown in Fig. 4:

Using this algorithm, the search time for a weak IV is roughly 1/20 of the time it would take using the unmodified FMS algorithm. Notice that the line of code that is bolded tests to see whether the first byte of the IV is the byte of the secret key that is currently trying to be determined and whether the second byte of the IV is 255. Hulton includes a number of other tests to determine weak IVs resulting in a shorter cracking time and an overall smaller number of packets that need to be captured.

```
 x = iv[0];
 y = iv[1];
 z = iv[2];

 a = (x + y) % N;
 b = AMOD((x + y) - z, N);

 for(B = 0; B < WEP_KEY_SIZE; B++)
 {
  /*
   * test to see if this key would apply to any of the bytes that
   * we're trying to crack.
   */
  if((((0 <= a && a < B) ||
   (a == B && b == (B + 1) * 2)) &&
   (B % 2 ? a != (B + 1) / 2 : 1)) ||
   (a == B + 1 && (B == 0 ? b == (B + 1) * 2 : 1)) ||
   (x == B + 3 && y == N - 1) ||
   (B != 0 && !(B % 2) ? (x == 1 && y == (B / 2) + 1) ||
   (x == (B / 2) + 2 && y == (N - 1) - x) : 0))
  {
   // It is a Weak IV
  }
}
```

FIG. 4. Dwepcrack's algorithm for determining weak IV's.

While these additions to WEP cracking algorithms can help to lower the time necessary to compromise a complete WEP key, they have not been widely implemented. This could be due to the overall low percentage of WEP enabled devices that are currently being implemented, or to the popularity of current WEP cracking utilities. These methods will most likely become more popular as newer devices avoid the usage of more commonly known weak IVs.

In August 2004, a hacker named KoreK published code in the NetStumbler forums that outlined expansions to the FMS attacks. These attacks have been implemented in both aircrack and WEPLab, both of which claim to crack WEP in record time. Essentially, these attacks work much like the FMS attack. The KSA is run as far as possible while looking at the values of the S array and the known keystream. Depending on the values that are found, the key bytes can be extracted. Appendix B contains KoreK's original code with comments added by the authors. Since the publication of this code on NetStumbler forums, KoreK has released a utility called chopper which expands upon this concept.

KoreK has since released another program called chopchop. This program exploits WEP in a different way to decrypt single packets. When an encrypted packet is captured, it can be decrypted one byte at a time by making a slight modification and attempting to retransmit it. The attacker will remove the last byte of the encrypted packet and replace it with a guess. To test to see if the guess was correct or not, the packet is sent to the access point. If the access point accepts the message and rebroadcasts it, the attacker can be sure that the guess was correct. The attacker can then use this byte and the corresponding cipherbyte to find the plaintext byte. Since there are only 256 choices for each byte, the packet can be decrypted in a relatively short period of time.

2.5 Injecting Packets for Faster WEP Cracking

The WEP cracking processes discussed so far require the capture of a large number of WEP encrypted data packets. For large wireless networks this requirement is easily met because of the large volume of traffic. However, there are other networks where the volume of traffic is not sufficient to allow the capture of enough packets in a reasonable period of time. It is possible to generate the necessary traffic on these networks by injecting packets to solicit responses. These packets do nothing to help the breaking of WEP, but the responses from a legitimate device on the network will increase the probability of generating weak initialization vectors. Some utilities like reinj and aireplay (part of the aircrack package) do this by capturing ARP requests. It then turns the packet around and injects it back into the network. Since the access point can't tell the difference between the injected packet and the original, the ARP request and response will give the attacker two more packets to work with. Other

utilities (WEPWedgie; see Section 2.6) will use ICMP echo requests to a broadcast address. This could generate echo replies from hosts on the network using a possible weak initialization vector. It is also possible to use this approach with other types of packets (TCP RSTs, TCP SYN/ACKs, etc.) [4].

These utilities allow a remote attacker the ability to create traffic on parsimonious networks. Home wireless networks and small offices often are included in this category. No longer can security professionals claim wireless networks of this size "secure" due to the amount of traffic. If the traffic does not naturally exist, an attacker can create it.

2.6 Encrypted Packet Injection

WEP does not have an effective mechanism to ensure data integrity. It does use a cyclic redundancy check (CRC) which is calculated for the data that is sent over the network. This verifies that a packet has not become corrupt during transmission but it does not protect data integrity.

When a packet is sent, its CRC is calculated from the plaintext. The output of the CRC32 algorithm is a 4 byte value which is called the integrity check value (ICV). The ICV is appended to the plaintext and encrypted using the WEP algorithm. The ICV is run again against the data after it has been decrypted by the receiver. If the ICV value that was sent in the packet matches the value that was calculated by the receiver, then the packet is considered legitimate. However, due to the nature of WEP and stream ciphers, it's possible to create an encrypted packet with a correct ICV, without knowing the actual WEP key. The attacker must only determine the ciphertext and plaintext combination for data that is sent with a particular Initialization Vector (IV).

The attacker first takes a legitimate known plaintext and known ciphertext combination and XORs the two values together to recover the keystream that was used to encrypt the plaintext. This same keystream is used for all packets that are sent over the wireless network with the same Initialization Vector and secret WEP key. Since it is unlikely that the WEP key will be changed in a reasonable amount of time, the keystream will be the same for all packets using the same IV.

That keystream value is then XORed with the plaintext and ICV that the attacker wishes to inject into the datastream. The resulting value is an encrypted packet to be prepended with that same IV and injected on the network. This technique is called PRGA Injection. An attacker can use a number of mechanisms to recover an encrypted packet and its corresponding plaintext value.

Tools have been created that perform injection attacks on WEP encrypted networks. reinj was a tool created for BSD that finds what it considers to be a TCP ACK packet or an ARP request and sends it back onto the network. While this is technically the injection of encrypted packets onto a wireless network, it doesn't

provide the attacker with the capability to customize the data. Another utility, WEPWedgie listens for WEP encrypted packets that are generated during a shared key authentication. This happens when a client wishes to connect to an access point. The access point will send the client a nonce which the client encrypts using the WEP key. This encrypted value is sent back to the access point for verification. If the value that the access point gets when it encrypts the nonce is the same as the value that the client returns, access is granted. This process provides a plaintext value and the corresponding ciphertext that is needed. WEPWedgie can now inject packets into a wireless network.

This attack can be expanded for even more malicious purposes. Once attackers gain the ability to inject packets into the wireless network, they can open connections to the internal servers, even if they are isolated by firewalls. The attacker will inject a packet into the wireless network asking for a connection to the server. This amounts to a TCP SYN packet with the source address of a computer that the attacker controls. The packet will be accepted into the network and sent to the unsuspecting server. The server will then do its part in opening the connection and send the attacker machine a TCP SYN/ACK packet. Assuming that the server can send a packet onto the Internet, it will send the packet through the firewall, onto the Internet and to the attacker's machine. The attacker's machine will then finish the 3-way handshake and a connection will be established. This approach can also be used to do port scans against the internal machines, do internal network mapping, etc. Being able to inject packets into a wireless network is an extremely powerful and dangerous tool.

2.7 802.1x Authentication

802.1x is a port authentication protocol that was originally created for wired networks. Switches, for example, can use 802.1x to authenticate a device before it allows the device access to a port. Once the authentication process has been successfully completed, access is granted to the device. There are three main components to the 802.1x authentication process: the supplicant, the authentication server, and the authenticator. The supplicant is the computer or device that wishes to have access on the network. The authentication server is the computer that performs the authentication. One of the most common types of servers that perform this task is a RADIUS server. The authenticator is the device that sits between the supplicant and the authentication server. In the example above, the authenticator is the switch. It is the point of access for the supplicant (Fig. 5).

Notice that there is two different sessions in Fig. 5. The supplicant starts the process by attempting to access the network resources. This triggers a request by

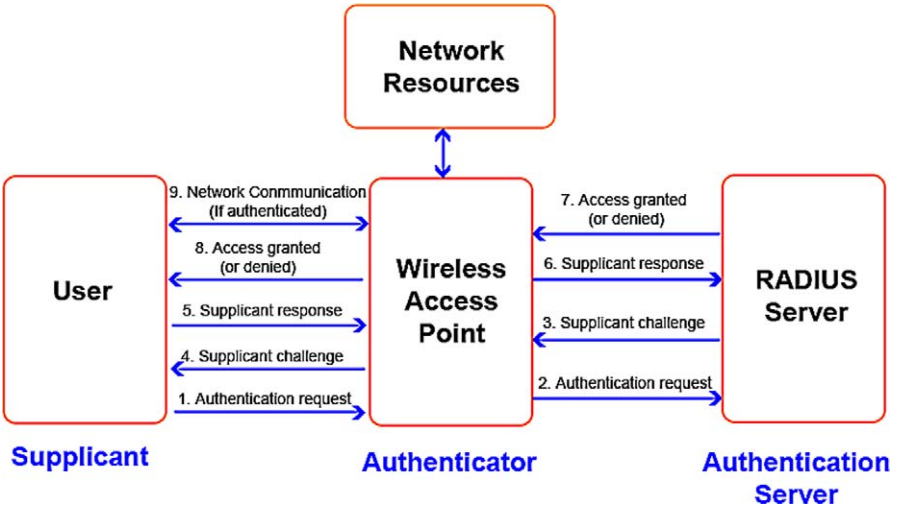

FIG. 5. 802.1x authentication process.

the authenticator for information about the identity of the supplicant. The supplicant provides this information and the authenticator forwards it to the authentication server. The authentication server then processes this information and usually sends a challenge to the supplicant through the authenticator. This challenge could be a nonce which needs to be encrypted or it could be some kind of token. The actual authentication mechanism is flexible and can vary between implementations. This is also the place where attacks can occur. When the supplicant responds to the challenge, the authenticator forwards this information on to the authentication server for processing. The authentication server then determines whether or not the supplicant should be granted access [16].

Although this process was originally developed for wired networks, it has been adopted by the 802.11i committee for use in wireless applications. In a wireless environment, the supplicant is a wireless client wishing to connect to the wireless network. The authentication server can still be a RADIUS server, but the authenticator is usually a wireless access point. Since one of the problems with WEP is key management, 802.1x can be very useful in a WEP environment. Any time a single key is used for an entire network, there will be security and scalability issues. The central server can provide clients with different keys, and even require a key change after a preset amount of time or data transmission. In a wireless environment, this is especially beneficial because it can change the secret key used by WEP and give different keys to clients [8].

2.8 EAP-LEAP Attack

802.1x uses a number of protocols to accomplish its goal of providing further security for wireless networks. The communication between the authentication server and the authenticator is logically separate from the communication between the authenticator and the supplicant. Extensible Authentication Protocol (EAP) is usually used for the authenticator/supplicant communication. EAP was created for point-to-point (PPP) authentication but has been adopted for use with wireless. EAP itself does not determine what method will be used to authenticate the supplicant, rather it allows the use of a server to facilitate the actual authentication. EAP-LEAP is one of the most popular types of EAP that is used today. It was developed by Cisco and has been implemented in a number of open-source RADIUS solutions [13].

EAP-LEAP is fundamentally flawed due to it's usage of MS-CHAPv2. This algorithm, and specifically the way that it was implemented in EAP-LEAP, allows an offline attack to be used to determine the password. When the usage of EAP-LEAP has been agreed upon, the authentication server sends the supplicant (by way of the authenticator) a nonce, or challenge text. Specifically it is an 8-byte random stream which the supplicant must encrypt. To encrypt the challenge text, the password is hashed using an NT hash and split up to generate three separate keys. The first key consists of the first seven bytes of the hashed password, the second key is the second seven bytes of the hashed password, and the third key is the final two bytes followed by five NULL values. These three keys are each used to encrypt the 8 byte challenge text. The three 8 byte results are then concatenated into one 24-byte value and this value is sent back to the authentication server for verification. Since EAP-LEAP supports mutual authentication, the process can be repeated in the opposite direction to authenticate the authentication server with the supplicant.

The problem with EAP-LEAP is that NT hashing does not use "salt." That means that the same plaintext value will hash to the same hashed value. So an attacker can hash a dictionary of plaintext passwords and store the corresponding hash values. If the password is one of the dictionary words, the hashes will match. Since the third hashed value that is used as a key to encrypt the 8 byte challenge consists of five null values, there is really only 2^{16} different possible values for the key. With so few possibilities, the two bytes can be found in less than a second. At this point, the last two NT hashed bytes of the password have been recovered. Using the precompiled dictionary, the attacker finds all hashed passwords where the last two bytes match what has been found. This usually narrows down the possible passwords to a number that can be brute forced against the authentication server. Now the attacker can achieve access to the wireless network.

There are a number of utilities that can perform this attack, the most famous of which is asleap, developed by Joshua Wright. Leapcrack and leap are two other

utilities which help an attacker perform this attack against EAP-LEAP authentication [19].

2.9 EAP-MD5 Attack

EAP requires that EAP-MD5 be implemented to serve as a fallback authentication mechanism. It has a number of vulnerabilities including susceptibility to man-in-the-middle attacks, lack of dynamic key distribution, and the plaintext/ciphertext combination. The EAP-MD5 process is somewhat similar to EAP-LEAP. The authentication server sends a challenge to the supplicant that is then hashed using MD5 and the password. That hashed value is sent back to the authentication server. This hashed value is then compared to the server's hash of the challenge text and if they are equivalent, access is granted.

Since the authentication process happens in only one direction, a man-in-the-middle attack can be performed against the authenticator. All that is required is a fake access point with authentication server software installed. When a supplicant requires access, it contacts the rogue AP instead of the authentic AP.

3. WPA Background and Introduction

Since there have been so many vulnerabilities discovered in WEP, an 802.11i standard committee was formed to find a new method of securing wireless communication. As the development of the standard progressed, some parts were ready to be deployed and other parts were not. In 2002, the Wi-Fi Alliance decided to deploy the parts that were deemed ready to help alleviate some of the security problems that existed. The parts that were released were named Wireless Protected Access (WPA). WPA still uses the WEP algorithm, but it adds a stronger integrity checking algorithm and better key management as in 802.1x. It can be implemented with a centralized authentication server or using pre-shared keys (PSK), like WEP. WPA offers two things of value: Temporal Key Integrity Protocol (TKIP) and 802.1x. TKIP is the encryption algorithm that was created to provide more security than WEP. It is essentially a shell that was placed around the WEP RC4 algorithm to address the following weaknesses: replay attacks, forgery attacks, key collision attacks, and weak key attacks. It does this by improving the integrity checking function, adding initialization vector sequencing rules, and creating a per-packet key.

TKIP uses a master key (MK) which is either distributed using 802.1x or as the PSK to derive a pairwise master key (PMK). In turn, the PMK is used to derive four more keys. These four keys are used during various parts of the encryption. One of the keys is called the temporal key (TK), which is primarily used for the encryption of data that is sent over the wireless link. The TK is XORed with the sender's MAC

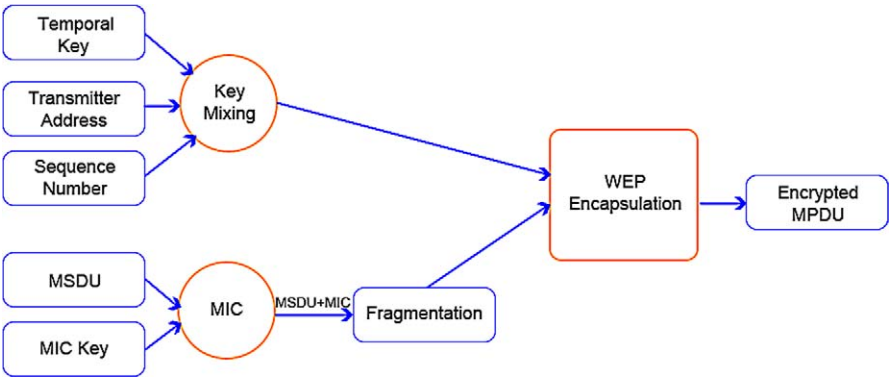

FIG. 6. Graphical diagram of TKIP Encapsulation. Note: MSDU is the MAC Service Data Unit and MPDU is the MAC Protocol Data Unit.

address, which is then mixed with a sequence number to produce a key that is used as input to the previous WEP algorithm. By adding all the extra steps the key is now much more secure against attack because it now depends on time and the sender's MAC address. TKIP also uses a sequence counter to prevent replay attacks and a Message Integrity Check (called MIC or Michael) to prevent message modification. Along with the sequence number, the source and destination addresses are added. Michael is also a one-way hash function rather than a simple linear CRC. This adds security to the algorithm because integrity verification is extremely difficult to forge. Figure 6 provides a graphical representation of TKIP [13,18].

More recently, the Wi-Fi alliance has announced the ratification of WPA2 which completely stops the use of WEP as an underlying protocol. Instead it uses the much stronger AES algorithm. As of this writing, WPA2 using AES does not share the same vulnerabilities as the original WPA.

3.1 The WPA-PSK Attack

There are two primary ways that TKIP can be used. The more secure way is to use 802.1x to distribute the keys and keep track of key management. The other way is to provide the client and access point with the same pre-shared key which is used as the pairwise master key (PMK) in the TKIP process. Although this shares many of the key management problems that WEP had, it does offer some "security through obscurity." One of the problems with TKIP is that if an attacker can determine what the PMK is for any one of the wireless clients, he or she can gain access to the network.

Researcher Robert Moskowitz discovered a problem with the WPA-PSK implementation when short passphrases are chosen [10]. When a pre-shared key is used,

it is hashed with the SSID and the SSID length to create a 256-bit PMK. This key is then used to derive all the other keys that are used in the TKIP algorithm. The pre-shared key is appended with the SSID and the SSID length which are then fed into the hashing algorithm defined by the PKCS #5 v2.0: Password-based Cryptography Standard. The string is hashed a total of 4096 times to create a 256-bit Pairwise Transient Key (PTK) that is used to derive all the other keys used in the algorithm. Moskowitz states that the 802.11i standard declares that there are approximately 2.5 bits of security per PSK character. The formula becomes: $2.5n + 12$ bits = Security Strength in bits, where n is the number of characters in the pre-shared key. This means that a typical password of 10 characters will only provide 37 bits of security. Because of this, Moskowitz claims, a dictionary attack on the hash can be performed to recover the password.

Recently a utility called coWPAtty has been released which uses a hash comparison based attack to recover the pre-shared key. coWPAtty captures the four way handshake authentication packets that are sent between the access point and the client. In these packets, coWPAtty finds the SSID of the network, the addresses of the access point and the client, and the nonces sent between the two parties. The SSID information and the passphrase from the dictionary are used to find a PMK. Using the other information from the exchange, the PTK is found. Using the PTK, the attacker can try to decrypt a message and see if the integrity check value found in the packet matches the calculated value. If so, the passphrase has been found. If not, the next passphrase in the dictionary is tried.

4. Other Attack Modes

4.1 VPNs: Point-to-Point Tunneling Protocol

A Virtual Private Network (VPN) allows an apparent "private" connection between two remote computers or networks. This is achieved by encrypting the traffic together with some sort of authentication. VPNs can be used by small office/home office users as well as large corporations.

Several standards have been developed to implement VPNs. One of these is the Point-to-Point Tunneling protocol (PPTP). This protocol allows a remote user to connect to another network using a VPN and it assures, or at least tries to assure, that the network that has been created is private. PPTP was created as an extension to point-to-point protocol (PPP) where dial-up users could connect to their Internet service provider (ISP) and then create a secure connection to the VPN server. Since it was pioneered by Microsoft, among others, PPTP is relatively simple to configure on Windows hosts and servers. As a result, a large number of VPN networks support-

ing PPTP are Windows networks. Like most other VPN protocols, PPTP employs authentication and encryption.

Wireless networking has adopted VPN protocols to add encryption and authentication security to its communications. Since WEP has been completely compromised and WPA has only recently been released as an upgrade to wireless devices, networks had to come up with another way to provide security. VPN's helped to fill this gap. Legitimate clients could be configured to connect to the wired network through a virtual private network which would encrypt the data that was being transferred. In this manner, wireless sniffers would only be able to capture encrypted data, which would then need to be decrypted before it became useful. Furthermore, the authentication process would help to shut out those people wishing to connect with the wireless network without legitimate access credentials [12].

Microsoft's original PPTP specified the use of Microsoft Challenge Handshake Authentication Protocol (MS-CHAP) for authentication. The design allowed someone to connect to a server, receive authentication based upon a password and gain access to the network resources. When a client connects to the network, it asks the authentication server for a login challenge. The server responds by sending the requesting client an 8 byte challenge nonce. The client then uses the LAN manager hash of the password that it has stored to derive three DES (Data Encryption Standard) keys. These keys are used to encrypt the challenge that the server sent. Each encryption results in an 8 byte encrypted string. All three encrypted strings are then concatenated to create a 24-bit reply. This same process is repeated with the Windows NT hash of the password. These two 24-bit blocks are sent to the server where they are compared. The server uses the stored client's password to decrypt the replies. If the decrypted blocks match the original challenge, access is granted. This process was used in the PPTP until vulnerabilities were found that compromise the authentication process. When these vulnerabilities became widely known, Microsoft re-worked the process and implemented a new version of MS-CHAP which was supposed to fix the problems that had been discovered.

MS-CHAP version 2, as it became known, added security to the process in the following ways: it stopped the use of the LM (LAN Manager) hash, introduced mutual authentication, replaced change password packets, and updated the encryption keys. To do this, it added a number of steps to the authentication process. When the client machine asks for a challenge, the server responds with a challenge of 16 bytes. The client then comes up with a 16-byte challenge itself which is called the "Peer Authenticator Challenge." The client then uses SHA-1 to hash the concatenation of the challenge from the server, the Peer Authenticator Challenge and the client's username. The first 8 bytes of the result then become the 8-byte challenge. Much like its predecessor, the 8 bytes are encrypted with the Windows NT hashed value of the password. This generates a 24-byte reply that is sent to the server where it is com-

pared. To provide mutual authentication, the Peer Authenticator Challenge is sent to the server, where the server concatenates it with the 24-byte response from the client and the string "magic server to client constant." This value is hashed using SHA to generate a 20-byte result that is then concatenated with the original challenge that was sent to the client and the afore-mentioned string padded, if necessary, to force more than one iteration and then hashed once again with SHA. This value is sent back to the client where it can be verified. If all values match, the session has been authenticated. While MS-CHAPv2 is much more complicated than MS-CHAPv1, it does very little to add to the security [14].

4.2 Attacks on PPTP

The attacks on PPTP are predominately attacks on the MS-CHAP authentication. As mentioned above, this is easier to do with the older version of MS-CHAP. In this version, a number of attacks were possible. One of the attacks involved spoofing a message from the server telling the client to change his or her password. If the client did change the active password, the password hashes could be picked up and cracked using a program such as L0phtcrack (http://www.insecure.org/sploits/l0phtcrack.lanman.problems.html). It was also possible for password cracking utilities, like L0phtcrack, to take advantage of the fact that the LM hash was being sent along with the NT hash. The LM hash is extremely easy to break and then could then be used to crack the NT hash and recover the password. With the password, an attacker could completely spoof the authentication process. Utilities such as anger (http://www.securiteam.com/windowsntfocus/2TUQBR5SAW.html) perform the attack on MS-CHAPv1 enabled PPTP VPNs. It collects the challenge and response packets that are exchanged for use in a cracking utility and it also provides the active attack using the change password messages.

In MS-CHAPv2, the change password message was altered to eliminate the vulnerabilities that tools like anger took advantage of. Since the LM hash is no longer sent along with the NT hash, it is more difficult to break. That is not to say that it is secure. The attack that can be performed on MS-CHAPv2 is described in the "EAP-LEAP" section. Ettercap (http://ettercap.sourceforge.net) is another utility that can be used to exploit weaknesses in PPTP. It has a number of plugins which automate the process of recovering passwords from PPTP MS-CHAP authentication.

4.3 Denial-of-Service Background and Attacks

Denial of service (DoS) and distributed denial of service (DDoS) attacks can render networks useless and are some of the hardest attacks to thwart. Though computers and networks have become faster and more reliable, they still have practical limits.

All available network bandwidth may be easily consumed by DoS and DDoS attacks, whether the network is wired or wireless.

The network clients count on the ability to access network resources. It is generally easier to perform a DoS attack on a wireless network than it is on a wired one. 802.11 networks broadcast data over a limited range of radio frequencies. All wireless networks within range compete for those frequencies. Attackers can take advantage of this fact by creating signals which saturate the network resources. This can be done with a powerful transmitter that broadcasts interfering signals or by low-tech approaches to RF-jamming like placing metal objects in microwaves that use the same frequency.

Other attacks can be performed at the link layer with disassociation and deauthentication frames that control the communication. If such frames were spoofed, connections could be manipulated without consent. Programs such as FakeAP (http://www.blackalchemy.to/project/fakeap), Void11 (http://forum.defcon.org/showthread.php?t=1427), and File2air (http://sourceforge.net/mailarchive/forum.php?thread_id=3164707&forum_id=34085) perform such attacks.

One strategy would be to send an authentication frame with an unrecognizable format which would cause the client to become unauthenticated because the access point would be confused. This attack has been implemented in the tool fata_jack (http://www.networkchemistry.com/news/whitepaper.pdf) which is meant to be used with AirJack (http://sourceforge.net/projects/airjack/). It sends an authentication frame to the access point with the sequence number and the status code both set to 0xFFFF. This frame is spoofed so that the access point believes it comes from a node that has already connected. This results in a fractured connection. If this attack is repeated, the real client will no longer have the ability to connect to the access point.

When a wireless client associates and authenticates with the access point, the access point must store information about the client in an internal state table. This includes the client's MAC address, IP address, etc. Since the memory of the access point is finite, it is possible to fake enough connections that the state table overflows. Depending on the access point, this could produce a crash or lock-up thereby blocking legitimate future authentications. Either way the attacker has successfully terminated the wireless connection. Joshua Wright wrote a Perl script called macfld.pl (http://home.jwu.edu/jwright/perl.htm) that will perform this attack. It works by flooding the access point with a large number of MAC addresses. Before WPA was implemented, the way that 802.11 checked the integrity of the packets that were received was through the CRC. If the CRC didn't match the value that was calculated by the wireless device, the packet was dropped. If, on the other hand, the packet was received correctly, an acknowledgement frame was sent so the sender could delete the transmitted frame from its send queue. This can be exploited by cor-

rupting a frame as it was transmitted so that the receiver would drop it. The attacker could then spoof an acknowledgement from the receiver saying that the frame was successfully received. The sender would then delete it from the queue and the frame would disappear from the datastream.

While DoS attacks don't allow an attacker to steal data or get access to the network, it does create significant chaos to the network resources. It is very difficult to stop and has disastrous consequences on affected networks.

4.4 Man-In-The-Middle Attacks

An effective man-in-the-middle (MITM) attack is one in which the attacker positions him/herself between the victim and the device with which the victim is trying to communicate. In this capacity, the attacker can control the information between the two devices. All traffic is re-routed through the attacker's computer where it can be manipulated or simply inspected. The attacker can then gather login information such as keys and passwords. In a wireless environment, the situation is more threatening because information that is transmitted over a wireless network is by definition available to all who have the ability to translate the RF signals to data. This might involve the injection of malicious code into the datastream to further compromise of the network and network nodes.

In the infrastructure mode of wireless networking, clients or stations all connect to central access points. One MITM strategy is to spoof an access point by de-authenticating and disassociating a client, neutralizing the AP with a DoS, and then re-authenticating the client with a clone under the control of the attacker. A number of MITM attack tools are widely available. Quite often the setup consists of a software access point and DoS software. An attacker's computer would usually have two separate wireless cards to handle both jamming and cloning functions. Some of the software access point programs that are available are: HostAP (http://hostap.epitest.fi) and HermesAP (http://hunz.org/hermesap.html).

5. Conclusion

It has been said that Wireless Networks will never be secure as long as radio frequencies fail to observe property lines. The validity of this claim lies in the fact that the physical security of the communication technology is for all intents and purposes absent in wireless environments (cf. also [9]). Though the physical security of a building is not a fault-proof barrier, it is at least a practical one. Wireless technology, even if properly configured, is not even a practical barrier. Even the most risk-averse can "sniff" transmissions with little chance of detection by using free software that is easily found on the Internet. The plain truth is that

wireless technology encourages digital eavesdropping. The World Wide Wardrive (http://www.worldwidewardrive.org) and Wigle (http://www.wigle.net) Websites are obvious illustrations. Wigle maintains a database of nearly four million active wireless networks in its online database. The World Wide Wardrives are network discovery contests.

In this chapter we have sought to identify some of the wireless insecurities that loom largest in today's increasingly wireless world. To the inattentive, the deployment of wireless can create the loss of trade secrets, the loss of personal privacy, theft of services, and unwitting participation in information warfare. For the attentive, the watch phrase must be "danger lurks within."

Appendix A: Wireless Security Utilities by Operating System and Function

Utility name	OS	Use
WEPcrack	Any platform (if Perl is available)	Finding WEP keys
AirSnort	Linux, Windows	Finding WEP keys
dwepcrack	BSD	Finding WEP keys
reinj	BSD	Packet Injection tool
WEPWedgie	Linux	Packet Injection tool
wep_tools	Linux	Finding WEP keys
asleap	Linux, Windows	Finding LEAP passwords
Leapcrack	Any platform (if Perl is available)	Finding LEAP passwords
leap	Linux	Finding LEAP passwords
L0phtcrack	Windows	Password cracking
ettercap	Linux	
fata_jack	Linux	Denial of service
macfld.pl	Linux	Denial of service
dinject	Linux	Denial of service
airjack	Linux	Drivers for MITM, DoS, etc.
File2air	Linux	Denial of service
Void11	Linux	Denial of service
omerta	Linux	Denial of service
HostAP	Linux	Software Access Point
HermesAP patch	Linux	Software Access Point
WPA Cracker	Linux	WPA-PSK password cracking
FakeAP	Linux, BSD	Random AP frame generation
WepAttack	Linux	Enhanced WEP cracking
aircrack	Linux	Enhanced WEP cracking
chopchop	Linux	Packet by packet decryption
chopper	Linux	POC WEP cracker
coWPAtty	Linux	WPA cracker

Appendix B: KoreK's New WEP Cracking Code

```
   /*
     This code was written by KoreK and published on the
NetStumbler.org forums
     (http://www.netstumbler.org/showthread.php?t=11869)
   */

   /* This code runs the KSA algorithm */
   for (i=0; i<256; i++) S[i]=i;    /* initialize the S[]
                                    array */
   for (i=0,j=0; i<p; i++)          /* run the loop until the
                                    unknown byte */
   {
       j=(j+K[i & 0x0f]+S[i]) & 0xff;
       swap(S+i,S+j);
   }
   /* at this point, the attacks can be run. Depending on how
   the weak IVs setup the S[] array, we can extract
information about the unknown key byte */
   /* Original FMS attack which has a 5% chance of working */
   if ((S[1] < p) && ((S[1]+S[S[1]]) == p))
   {
     jp=o1; /* o1 is the first byte of the keystream (z) */
     stat1[(jp-j-S[p]) & 0xff]++; /* extract the info and
                                  keep track of it */
   }

   /* The following attacks were developed by KoreK and work
in much the same way.
      That is, depending on how the S[] is setup, the values
give away information about the unknown key byte.
   */

   /* Second type of attack -- 13% chance of working */
   if (S[1] == p) /* S[1] is equal to the byte location we're
              looking for */
    {
      for (jp=0; S[jp]!=0; jp++); /* find the first index, i,
                                  where S[i] isn't 0 */
        if (o1 == p) /* if the unknown key byte is the same
                    as the first byte of the cipher (z) */
         {
         stat2[(jp-j-S[p]) & 0xff]++; /* extract the info and
                                         keep track of it */
         }
       // for statistical purposes
       pstat2[(jp-j-S[p]) & 0xff]++;
```

```
   }
    /* A third attack -- This one works if the second byte of
   the cipher (z) is 0, the value of the S[] array at
   location p (the unknown key byte were looking for) is 0,
      and the third (index=2) element of the S[] array isn't
   zero */
      if ((o2 == 0) && (S[p] == 0) && (S[2] != 0))
         {
          stat3[(2-j-S[p]) & 0xff]++;  /* extract the info and
                                       keep track of it */
        }
    // and two more (well there are still about 10 of 'em left:)
   /* The next two start with the same requirements. The
second element (index=1) of S[] must be greater than the
value of the unknown key byte we're searching for and
      the second (index=1) element of S[] plus the third
(index=2) minus the value of p (the unknown key byte
we're searching for) must be zero
   */
   if ((S[1] > p) && (((S[2]+S[1]-p) & 0xff) == 0))
     {
      /* once this condition is met, one of two more
   conditions are needed to qualify */
      /* The first is the second byte of the keystream (z)
   should be equal to S[1] */
      if (o2 == S[1])
        {
        /* Now find the first value in the S[] array that
     doesn't have a value
          of S[1] - S[2]. Save that index
        */
        for (jp=0; S[jp] != ((S[1]-S[2]) & 0xff); jp++);
          /* As long as that index isn't 1 or 2, we can do an
     extraction and save it */
        if ((jp!=1) && (jp!=2)) stat4[(jp-j-S[p]) & 0xff]++;
        }
      else
        {
        /* The other option is to see if the second byte of
     the keystream (z) is
           equal to 2 minus the third value in the S[] array
     (index=2)
        */
        if (o2 == ((2-S[2]) & 0xff))
          {
            /* Find the first index of the S[] array where
          the value isn't the same as the second byte of
          the keystream (z)*/
```

```
            for (jp=0; S[jp] != o2; jp++);
            /* As long as that index isn't 1 or 2, extract
          the byte and save it */
            if ((jp!=1) && (jp!=2)) stat5[(jp-j-S[p]) & 0xff]++;
            }
          }
        }
        // inverted attack
        /* This attack is useful in determining false positives. As
           the byte values get placed in the stat6[] bin, there is
           a chance that that value is not correct. KoreK placed a
      threshold on this at 32. If any byte appears in
      the stat6[] bin more than 32 times, it must not be the right
      value. These tests are pretty straight forward. Just
      remember that o1 is the first byte of the keystream.
       */
       if (S[2] == 0)
         {
           if ((S[1] == 2) && (o1 == 2))
             {
             stat6[(1-j-S[p]) & 0xff]++;
             stat6[(2-j-S[p]) & 0xff]++;
             }
           else if (o2==0)
             {
             stat6[(2-j-S[p]) & 0xff]++;
             }
  }
  if ((S[1] == 1) && (o1 == S[2]))
    {
      stat6[(1-j-S[p]) & 0xff]++;
      stat6[(2-j-S[p]) & 0xff]++;
    }
  if ((S[1] == 0) && (S[0] == 1) && (o1 == 1))
    {
      stat6[(-j-S[p]) & 0xff]++;
      stat6[(1-j-S[p]) & 0xff]++;
    }
```

REFERENCES

[1] Barkin L., *How Secure is Your Wireless Network?*, Prentice Hall PTR, Upper Saddle River, 2004.
[2] Berghel H., "Wireless infidelity I: War driving", *Commun. ACM* **47** (9) (2004) 21–26.
[3] Berghel H., Uecker J., "Wireless infidelity II: Air jacking", *Commun. ACM* **47** (12) (2004) 20–25.

[4] Berghel H., Uecker J., "WiFi attack vectors", *Commun. ACM* **48** (8) (2005) 21–27.
[5] Fluhrer S., Mantin I., Shamir A., "Weakness in the key scheduling algorithm of RC4", http://www.cs.umd.edu/~waa/class-pubs/rc4_ksaproc.ps, 2001.
[6] Giller R., Bulliard A., *Wired Equivalent Privacy*, Swiss Institute of Technology, Lausanne, 2004.
[7] Hulton D., "Practical exploitation of RC4 weaknesses in WEP environments", http://www.dachb0den.com/projects/bsd-airtools/wepexp.txt, 22 Feb. 2002.
[8] Meetinghouse Data Communications, "802.1X—still evolving as a standard", http://www.mtghouse.com/8021X.pdf, 28 Aug. 2004.
[9] Mikhailovsky A., Gavrilenko K., Vladimirov A., *Wi-Foo: The Secrets of Wireless Hacking*, Addison–Wesley, Boston, 2004.
[10] Moskowitz R., "Weakness in passphrase choice in WPA interface", http://wifinetnews.com/archives/002452.html, 04 Nov. 2003.
[11] Newsham T., "Cracking WEP keys", Presentation: Black Hat, 2001, http://www.blackhat.com/presentations/bh-usa-01/TimNewsham/bh-usa-01-Tim-Newsham.ppt.
[12] Peikari C., Fogie S., *Maximum Wireless Security*, Sams Publishing, Indianapolis, 2002.
[13] Sankar S., Sundaralingam S., Balinsky A., Miller D., *Cisco Wireless LAN Security*, Cisco Press, Indianapolis, 2005.
[14] Schneier B., Mudge, "Cryptanalysis of Microsoft's PPTP authentication extensions (MS-CHAPv2)", http://www.schneier.com/paper-pptpv2.html, 19 Oct. 1999.
[15] Stevens R.W., *TCP/IP Illustrated*, vol. 1, Addison–Wesley, Boston, 1994.
[16] Strand L., "802.1X port-based authentication HOWTO", http://ldp.hughesjr.com/HOWTO/8021X-HOWTO/intro.html, 18 Oct. 2004.
[17] Stubblefield A., Ioannidis J., Rubin A.D., "Using the Fluhrer, Mantin, and Shamir attack to break WEP", http://www.isoc.org/isoc/conferences/ndss/02/proceedings/papers/stubbl.pdf.
[18] Vladimirov A., Gavrilenko K., Mikhailovsky A., *Wi-Foo*, Addison–Wesley, Boston, 2004.
[19] Wright J., "Weaknesses in LEAP challenge/response", Presentation: Defcon, 2003, http://asleap.sourceforge.net/asleap-defcon.pdf.