Alin Tomescu, CSE408
Tuesday, February 3rd, Lecture #2

Hacking WEP (Wired Equivalent Privacy)

Why WEP?

Businesses and individuals need privacy over their wireless network to protect their secrets.

Amongst the different kinds of protection, the most important probably are:

- **Secrecy** or confidentiality
- Authentication
- **Integrity** of the data (laptop should receive the same data that the base station sent)
- **Availability** (protection against DoS attacks, since a microwave can take your wireless network down availability is an inherent issue)

How WEP works

WEP is used to encrypt and decrypt link-layer frames sent over an 802.11 wireless network.

WEP uses **a secret key** k shared among the communicating parties (a laptop and wireless access point) to protect the transmitted frames.

Encrypting a message m means going through a series of steps:

- *Integrity check-summing:* using the CRC32 algorithm a 32bit checksum $CRC(m)$ is computed for the message m. This checksum is appended to the message forming the **plaintext** $p = \langle m, CRC(m) \rangle$
- *RC4 keystream generation:* for each transmitted frame, a random **initialization vector** (IV) v is generated. Using the IV and the secret key k as input to the **RC4 algorithm** a **keystream** $K_s = RC4(v, k)$ is generated.
- *Actual encryption of the plain text:* finally the plaintext p and the keystream K_s are XORed together to obtain the **ciphertext** $c = p \oplus K_s$
- *Transmitting the frame:* in order for the receiver to decrypt the ciphertext c the sender will prepend the IV v to the ciphertext c, obtaining the **frame** $f = \langle v, c \rangle$ which is finally sent over the network.

Once the receiver gets a frame, he has the IV v appended to this frame, he has the secret key k, so he can use RC4 to get the keystream K_s, which he can XOR with the ciphertext c to get the plaintext $p = \langle m, CRC(m) \rangle$. He can now compute the CRC of m and ensure it matches the one sent along with the message. Job done.

What is the point of the IV?

If the plain-text p was merely XORed with the secret key k and sent over the network as a frame $f = p \oplus k$, then an attacker could get two frames $f_1 = p_1 \oplus k$ and $f_2 = p_2 \oplus k$, XOR them together $f_1 \oplus f_2 = (p_1 \oplus k) \oplus (p_2 \oplus k) = p_1 \oplus p_2$ and now he can potentially crack $p_1 \oplus p_2$ using trial-and-error (same problem as reusing a key with the one-time pad).

We can see that it is essential for each frame to be encrypted with a unique keystream. Therefore, a randomly generated IV v and the secret key k are used together with the RC4 algorithm to seed a unique keystream K_s. In reality, we will see that the likelihood of "IV collisions," having two frames with the same IV sent over the network, is quite high and is instrumental in breaking WEP.

Alin Tomescu, CSE408
Tuesday, February 3rd, Lecture #2

WEP attacks

WEP is susceptible to many attacks for a couple of reasons:

- The secret shared key k is rarely changed (it could take weeks or months before someone thinks about changing it), which means we can consider it a constant in our attacks.
- The small IV space (24 bits only) pretty much guarantees there will be IV collisions, if IVs are chosen randomly, which means some frames will be XORed using the same keystream and could be cracked.
- The CRC32 check-summing algorithm used by the underlying 802.11 layer is not cryptographically secure, allowing attackers to intercept and modify sent frames
- The ACKs sent by the underlying 802.11 layer upon receival of a corrupt frame can be used to build a **reaction attack** for guessing remaining bytes of an incomplete keystream

The birthday attack

After the transmitter has sent k frames, it will have made k choices of IVs from a bucket of 2^{24} IVs. What is the probability that the i^{th} IV is the same as the j^{th} IV? After the i^{th} IV is picked there is just one choice for the j^{th} IV such that it $IV_i = IV_j$

$$P[IV_i = IV_j] = \frac{1}{2^{24}}$$

$$E[\# \, of \; i, j's \; st. IV_j = IV_i] = \binom{k}{2}\frac{1}{2^{24}} = \frac{k^2}{2^{25}}$$

When $k = 2^{12.5}$, we have $\frac{k^2}{2^{25}} = 1$, therefore we can see that after approximately 6000 frames, an IV collision is bound to occur.

How does an IV collision help us decrypt traffic?

Unique IVs seed unique keystreams using the RC4 algorithm. We know it is essential for each transmitted frame to be encrypted with a unique keystream $K_S = RC4(v, k)$, where v is the randomly picked IV and k is the shared secret key, therefore it is essential for each frame to have an unique IV.

If we have two frames $f_1 = p_1 \oplus RC4(v_{col}, k)$ and $f_2 = p_2 \oplus RC4(v_{col}, k)$ which were encrypted using the same IV v_{col}, then that means the same keystream $K_s = RC4(v_{col}, k)$ was used to XOR the plaintexts in both frames. We can now XOR f_1 and f_2 together, which will cancel the keystream out effectively giving us $p_1 \oplus p_2$ which can be broken using traditional methods.

Once we have p_1 or p_2 we can obtain the keystream $K_s = RC(v_{col}, k)$ associated with the IV v_{col} by computing $f_1 \oplus p_1 = K_s = RC4(v_{col}, k)$. In this manner we can build a decryption table that associates IV's with their corresponding keystreams. Since a frame is 1500 bytes and there are 2^{24} possible IVs, about 24GB of space will suffice to build it. We can now inject and decrypt all WEP-encrypted traffic.

WEP tampering

CRC32 is a linear function, this means it has the following property:

$$CRC(p_1 \oplus p_2) = CRC(p_1) \oplus CRC(p_2)$$

This linearity of CRC is a dangerous property, because it allows us to flip bits (or XOR bits) in an encrypted frame and then alter the CRC so that the 802.11 integrity check succeeds for the modified frame.

Suppose we had an encrypted frame $f = p \oplus K_s$, where $p = \langle m, CRC(m) \rangle$ is the plaintext. We can modify the plaintext in the encrypted frame f using the properties of XOR and the linearity of the CRC as follows:

Suppose instead of the message m, we would like the receiver to get $m' = m \oplus \Delta$. We can modify f into f' as follows:

First we obtain a modified plaintext p' that contains m' and $CRC(m')$ by doing XOR operations on the frame f:

$$p' = p \oplus \Delta = \langle m \oplus \Delta, CRC(m) \oplus CRC(\Delta) \rangle = \langle m \oplus \Delta, CRC(m \oplus \Delta) \rangle = \langle m', CRC(m') \rangle$$

We did two things here:

- We obtained $m' = m \oplus \Delta$
- We altered the CRC of m, and obtained $CRC(m') = CRC(m) \oplus CRC(\Delta) = CRC(m \oplus \Delta)$

Note that even though we only have the encrypted frame f and not the actual plaintext p or the message m, we are using XOR's commutativity and associativity to modify m's bits by XORing part of the encrypted frame f with Δ and with $CRC(\Delta)$.

Now that we have altered the plaintext p as p' inside the encrypted frame f we effectively obtained f' which can be sent over the network.

WEP reaction attack

Imagine the attacker obtained the IV and a short corresponding keystream (like 100 bytes) and he would now like to learn more of that keystream. The attacker can use 802.11 acknowledgements to guess the next bytes of the keystream.

When the base station receives a packet which passes the CRC check it sends a short ACK packet back, letting the sender know that the packet arrived intact. If the CRC check fails, the receiver does nothing causing the sender to resend the packet. The attacker can use the ACKs in his reaction attack as follows:

Suppose the attacker has n bytes of a keystream K_s associated with the IV v and he wants to guess byte # $n + 1$. The attacker can try all possibilities for that byte ($2^8 = 64$ possibilities), for each possibility sending a frame with a dummy plaintext p consisting of a random message m and of $CRC(m)$. This plaintext will be XORed with the derived keystream $K_s' = K_s + guessed\ byte$ to from the frame.

The victim will receive and decrypt the frame, obtaining the message m and the CRC of m. However, the last byte of the CRC will have been decrypted using the guessed byte of the keystream K_s'. If the keystream $K_s' = K_s + guessed\ byte$ was the correct one, then the decrypted CRC from the frame will match the one computed by the receiver, causing him to send an ACK, letting the attacker know the $n + 1$ byte is correct. If there was a mismatch, this means the part of the keystream that was guessed was incorrect since it resulted in a bad CRC being decrypted on the receiver's side.