informIT

Security

Byte-Sized Decryption of WEP with Chopchop, Part 1

Last updated Jun 9, 2006.

WEP contains a flawed encryption scheme as a result of a poor implementation of the RC4 algorithm. This weakness is fairly well known and exploited. However, what many people do not realize is that WEP has several other significant problems that can also lead to decryption attacks. In this section we will take a look at these issues and learn how they can be abused to decrypt a single captured packet in just a few seconds.

The Other WEP Vulnerabilities

WEP is not secure for many reasons. As previously mentioned, RC4 was not properly implemented in the encryption routine. In addition to this, WEP-protected packets can be reinserted or replayed back into the wireless network. Finally, the encrypted packets can be forged to allow spoofing attacks. In other words, an attacker can pretend to be a valid client and inject traffic into the network that the AP will accept and retransmit. By combining all three of these attacks together, an attacker can successfully crack the shared key in a few minutes.

To help mitigate this threat and other WEP related problems, wireless vendors implemented dynamic WEP keying. In theory, this should change the key after a set number of packets have been sent, thus preventing an attacker from collecting enough packets that were encrypted with the same key. However, most vendors implemented dynamic keying on a per session basis, not a per packet number basis. This means the key only changes after the user has disconnected and reconnected to the wireless network. The end result is that most users will transmit enough data to have their 'dynamic' key cracked. However, if the user is only online for a short period of time, the use of dynamic keying will protect them from the popular statistical attacks.

In theory, the use of dynamic keys seemed like a potential way to stop malicious hackers. At least until KoreK deduced a method for cracking one packet at a time, regardless of the key. With this approach, an attacker never has to know the shared key used to encrypt the data, but instead focuses on two other elements of the data transfer process.

ICV

WEP includes an integrity check known as the integrity check value (ICV). When a packet is ready for transmission, the data is first passed through a CRC-32 algorithm, which creates a four byte value that represents the original data. This CRC-32 value is then appended to the end of the data. Finally, this whole value (data + ICV) is XOR'd with the streaming key created by the RC4 algorithm. This ciphertext is then sent out to the access point (or client) that then XORs the ciphertext with its own streaming key to recreate the original plaintext + ICV. The plaintext is then passed into a CRC-32 algorithm once again to create a new ICV. If the new ICV matches the one in the original packet, the data is considered valid. Figure 1 shows how the CRC-32 algorithm fits into the decryption process.

Figure 1 The ICV process

The problem is that CRC-32 is a widely known integrity validation function that is generally used to protect against data corruption. The algorithm was not built with security in mind, and as such can be tricked and abused. In the case of a WEP's use of CRC-32, an attacker can capture an encrypted packet, change the content of that encrypted data in that packet, recreate a new encrypted ICV

that validates the altered encrypted data, and reinject that packet back into the wireless network. As the wireless guru KoreK explains, this entire process is based upon some relatively simple XOR math (at least relative to some other KoreK work).

First, the entire message (M) is actually two parts — the plaintext (P) and the ICV value. If an attacker wants to change the message, they have to alter both parts. Ignoring the fact that the message is encrypted for the moment, creating a new valid packet is fairly simple. All an attacker has to do is create a bitmask for the modification they want, and then XOR the original data with that mask. Next, an attacker calculates a new ICV for the bitmask, which is then XOR'd with the original ICV. Finally, the modified data and new ICV are concatenated together and injected. Mathematically, it looks like this (Mod=bitmask, + denotes concatenation).

```
Mnew = Pnew + ICVnew = Porg XOR Mod + ICVorg XOR ICVmod
```

Creating a new valid encrypted packet is not any more difficult because the relationship is maintained whether or not the message is encrypted. The reason is that the encryption itself is an XOR process that basically cancels itself out for this particular attack, which you can see below in a step by step outline (M = encrypted message).

```
Morg = (Porg + ICVorg(Porg) ) XOR RC4
```

Therefore...

```
Mnew = Pnew + ICV(Pnew)=Porg XOR Mod + ICVorg(Porg) XOR ICV(Mod)
Mnew = Pnew + ICV(Pnew) = ( Porg + ICV(Porg) ) XOR ( Mod + ICV (Mod))
Mnew = Pnew + ICV(Pnew) = Morg XOR (Mod+ICV(Mod)
Mnew = Morg + (Mod+ICV(Mod))
```

If all this algorithm stuff is not your thing, just know that you can XOR the original encrypted message with an equally long modified data string to create a new valid packet. This process is not really all that complex and can be done very quickly with a computer.

While changing the data in a WEP packet is a viable attack, it is probably not going to result in much of anything by itself because the attacker has no idea what they are changing. Does the packet contain email information? Is it part of a web page? The attacker will probably not know, and as a result, the packet mangling will manifest as corrupt application data and probably be rejected by the final destination.

In the next section we will look at how Chopchop uses this ICV flaw and another more dangerous vulnerability, to crack WEP protected packets — one byte at a time.
