The Fragmentation Attack in Practice

Andrea Bittau
a.bittau@cs.ucl.ac.uk

September 17, 2005
Transmit arbitrary WEP data without knowing the key.

- Only requirement: Eavesdrop a single WEP packet.
1. Introduction
   - WEP
   - Common Attacks

2. Theory
   - PRGA & WEPWedgie
   - Fragmentation

3. Practice
   - Hardware & Software Limitations
   - Real-life Attack Example
   - Script-kiddie Tool

4. Conclusion
Bogus implementation of RC4 with a 40-bit shared key.

Only data portion of data packets is encrypted.

Initialization Vector (IV) prepended to key on each encryption.

- IV is transmitted in clear within WEP packets.

---

**Data frame format**

<table>
<thead>
<tr>
<th>802.11 Header</th>
<th>Frame Body</th>
<th>CRC</th>
</tr>
</thead>
</table>

- IV: 32-bit (IV 3 bytes)
- User Data
- ICV: CRC32 of user data
**Wired Equivalent Privacy??**

**Encryption**

1. **Seed**: Choose IV (any 24-bit number) and prepend to key.
2. **KSA**: Run RC4 Key Scheduling Algorithm on seed.
4. **XOR**: XOR user data with PRGA.

**WEP Encryption**

```
IV + key → RC4
```

```
| 0 | 1 | 0 | 1 |

Plain text

| 1 | 1 | 0 | 0 |

⊕

| 1 | 0 | 0 | 1 |
```

Cipher text
Common Attacks

1. **Bruteforce**
   - 40-bit key!
   - ASCII Passphrase.
     - Microsoft Windows XP requires *exactly* 5 or 13 characters.

2. **KSA**
   - The weak IV attack (aka FMS).
   - Requires $\approx 300,000$–$3,000,000$ unique IVs.
     - Many networks don’t have much traffic.
     - 13% probability IVs improve the attack a lot.
     - aircrack is a good implementation.

3. **PRGA**
   - WEP-wedgie: Shared key authentication networks.
   - PRGA discovery: Bit-flipping, IV collisions, etc.
   - Fragmentation: Not (yet) public!
Common Attacks

1. **Bruteforce**
   - 40-bit key!
   - ASCII Passphrase.
     - Microsoft Windows XP requires *exactly* 5 or 13 characters.

2. **KSA**
   - The weak IV attack (aka FMS).
   - Requires $\approx 300,000$–$3,000,000$ unique IVs.
     - Many networks don’t have much traffic.
     - 13% probability IVs improve the attack a lot.
     - aircrack is a good implementation.

3. **PRGA**
   - WEP-wedgie: Shared key authentication networks.
   - PRGA discovery: Bit-flipping, IV collisions, etc.
   - Fragmentation: Not (yet) public!
# Common Attacks

1. **Bruteforce**
   - 40-bit key!
   - ASCII Passphrase.
     - Microsoft Windows XP requires *exactly* 5 or 13 characters.

2. **KSA**
   - The weak IV attack (aka FMS).
   - Requires $\approx 300,000–3,000,000$ unique IVs.
     - Many networks don’t have much traffic.
     - 13% probability IVs improve the attack a lot.
     - aircrack is a good implementation.

3. **PRGA**
   - WEP-wedgie: Shared key authentication networks.
   - PRGA discovery: Bit-flipping, IV collisions, etc.
   - Fragmentation: Not (yet) public!
If we had PRGA for an IV:

- Decrypt all packets which use that IV (cipher text $\oplus$ PRGA).
  - With PRGAs for different IVs, we can decrypt more packets (IV dictionary).
- Encrypt user data with that IV (data $\oplus$ PRGA).
  - Can always use same IV.

If we intercept cipher text and somehow know the clear text:

- Discover PRGA for that IV (cipher text $\oplus$ clear text).
If we had PRGA for an IV:

- Decrypt all packets which use that IV (cipher text $\oplus$ PRGA).
  - With PRGAs for different IVs, we can decrypt more packets (IV dictionary).
- Encrypt user data with that IV (data $\oplus$ PRGA).
  - Can always use same IV.

If we intercept cipher text and somehow know the clear text:

- Discover PRGA for that IV (cipher text $\oplus$ clear text).
Shared key authentication:

1. Access point (AP) sends 128 byte challenge.
2. Client replies with encrypted version of challenge.
Shared key authentication:

1. Access point (AP) sends 128 byte challenge.
2. Client replies with encrypted version of challenge.

Have 128 bytes of PRGA!

(challenge ⊕ encrypted challenge) reveals PRGA for IV client used.

- Can encrypt 128 – 4 (ICV) arbitrary bytes of data.
- Can decrypt first 128 bytes of packets which use that IV.
Shared key authentication:

1. Access point (AP) sends 128 byte challenge.
2. Client replies with encrypted version of challenge.

Have 128 bytes of PRGA!

(challenge ⊕ encrypted challenge) reveals PRGA for IV client used.

- Can encrypt 128 – 4 (ICV) arbitrary bytes of data.
- Can decrypt first 128 bytes of packets which use that IV.

Optimization

Force clients to disconnect by spoofing de-authentication requests—management frames not encrypted!
All data is *Logical Link Control* (LLC) encapsulated.
- Commonly (always) followed by SNAP.
  - Most likely followed by IP.
  - At times followed by ARP.

**LLC/SNAP header for IP packet**

<table>
<thead>
<tr>
<th>0xAA</th>
<th>0xAA</th>
<th>0x03</th>
<th>0x00</th>
<th>0x00</th>
<th>0x00</th>
<th>0x08</th>
<th>0x00</th>
</tr>
</thead>
</table>

- **DSAP**
- **SSAP**
- **CTRL**
- **ORG code**
- **Ether type**

ARP packets have 0x0806 as ethernet type!
- Distinguishable by fixed and short length.

In general, we can recover at least 8 bytes of PRGA.
802.11 supports fragmentation at a MAC layer.

- Each WEP fragment is encrypted independently.
802.11 supports fragmentation at a MAC layer.
  - Each WEP fragment is encrypted independently.

**The Fragmentation Attack**

Send arbitrarily long data in 8 byte fragments!
802.11 supports fragmentation at a MAC layer.
  - Each WEP fragment is encrypted independently.

**The Fragmentation Attack**

Send arbitrarily long data in 8 byte fragments!

Some details:
  - Each fragment needs ICV. Only $8 - 4 = 4$ bytes for real data.
  - Fragment No. field is 4 bits. Only 16 fragments possible.
    - Max data length $= 2^4 \times 4 = 64$.
    - Can use IP fragmentation too.
  - Can generate traffic for which response is known, revealing more PRGA.
Outline of Attack

1. Eavesdrop a WEP packet.
2. Recover 8 bytes of PRGA (clear $\oplus$ WEP).
3. Transmit data in 8 byte fragments using same IV.
Outline of Attack

1. Eavesdrop a WEP packet.
2. Recover 8 bytes of PRGA (clear ⊕ WEP).
3. Transmit data in 8 byte fragments using same IV.

**Speed up other attacks**

1. Send data which generates traffic.
2. Collect weak IVs.
3. Perform KSA attacks (FMS).

**Pure PRGA attack**

1. Send data for which reply is known.
2. Recover PRGA for more IVs.
3. Slowly build an IV dictionary.
1. Eavesdrop a WEP packet.
2. Recover 8 bytes of PRGA (clear ⊕ WEP).
3. Transmit data in 8 byte fragments using same IV.

**Speed up other attacks**
1. Send data which generates traffic.
2. Collect weak IVs.
3. Perform KSA attacks (FMS).

**Pure PRGA attack**
1. Send data for which reply is known.
2. Recover PRGA for more IVs.
3. Slowly build an IV dictionary.
Prism2 (Intersil) based cards.

- Host-AP mode. Can send (almost) raw 802.11 frames.
- Monitor mode. Firmware passes all frames to kernel.

Firmware overwrites 802.11 header fields such as fragment & sequence number!
Prism2 (Intersil) based cards.

- Host-AP mode. Can send (almost) raw 802.11 frames.
- Monitor mode. Firmware passes all frames to kernel.

Firmware overwrites 802.11 header fields such as fragment & sequence number!

Re-write the fields via debug port! (greets to h1kari)

1. Queue the packet on the card for TX via the normal interface.
2. Locate the packet on the card’s memory via AUX port.
3. Instruct the card to begin TX.
4. After the firmware processed the header, but before it is sent, overwrite it.
   - In practice, we always win the race!
FreeBSD using wi driver.

- Added much of airjack’s (Linux driver) functionality.
FreeBSD using wi driver.

- Added much of airjack’s (Linux driver) functionality.

**AUX overwrite implementation**

1. Queue and locate packet with 2 random bytes in MAC addr.
2. Busy wait reading duration until it changes.
3. Overwrite header.

```
| 0x08 | 0x00 | 0x00 | 0x00 | 0x00 | 0xDE | 0xFA | 0xCE | 0xD0 | 0x00 |
```

Frame CTRL  Duration  Address 1
FreeBSD using wi driver.

- Added much of airjack’s (Linux driver) functionality.

**AUX overwrite implementation**

1. Queue and locate packet with 2 random bytes in MAC addr.
2. Busy wait reading duration until it changes.
3. Overwrite header.

```
0x08  0x00  0xD5  0x00  0x00  0xDE  0xFA  0xCE  0xD0  0x00
```

- Frame CTRL
- Duration
- Address 1
FreeBSD using \texttt{wi} driver.

- Added much of airjack’s (Linux driver) functionality.

**AUX overwrite implementation**

1. Queue and locate packet with 2 random bytes in MAC addr.
2. Busy wait reading duration until it changes.
3. Overwrite header.

\begin{tabular}{|cccccccc|}
\hline
0x08 & 0x00 & 0x7F & 0xFF & 0x00 & 0xDE & 0xFA & 0xCE & 0xAA & 0xBB \\
\hline
\end{tabular}

Frame CTRL \hspace{1cm} Duration \hspace{1cm} Address 1
FreeBSD using wi driver.

- Added much of airjack’s (Linux driver) functionality.

**AUX overwrite implementation**

1. Queue and locate packet with 2 random bytes in MAC addr.
2. Busy wait reading duration until it changes.
3. Overwrite header.

```
0x08 0x00 0x7F 0xFF 0x00 0xDE 0xFA 0xCE 0xAA 0xBB
```

Frame CTRL  Duration  Address 1

- Able to send any 802.11 frame and receive all frames.
The Attack
PRGA determination

- Eavesdrop WEP packet and determine 8 bytes of PRGA.
- Transmit ARP request (36 bytes) in 9 fragments of 4 data bytes.
  - Who has 192.168.0.1 tell 192.168.0.123.
- Didn’t get any reply.
  - Wrong IP network.
  - But AP relayed the packet (since it’s a broadcast).
  - Re-encrypted by the AP.
  - Knowing the contents, we discover 36 bytes of PRGA.
- Send ARP request padded with $x$ 0s (in larger fragments).
  - AP relays the longer ARP request.
  - Discover $36 + x$ bytes of PRGA.
  - Repeat until, say, 1504 bytes of PRGA are known.
- Can send 1500 bytes of data *without* fragmenting.
Eavesdrop WEP packet and determine 8 bytes of PRGA.
Transmit ARP request (36 bytes) in 9 fragments of 4 data bytes.
  - Who has 192.168.0.1 tell 192.168.0.123.
  - Didn’t get any reply.
    - Wrong IP network.
    - But AP relayed the packet (since it’s a broadcast).
    - Re-encrypted by the AP.
    - Knowing the contents, we discover 36 bytes of PRGA.
Send ARP request padded with $x$ 0s (in larger fragments).
  - AP relays the longer ARP request.
  - Discover $36 + x$ bytes of PRGA.
  - Repeat until, say, 1504 bytes of PRGA are known.
Can send 1500 bytes of data \textit{without} fragmenting.
Eavesdrop WEP packet and determine 8 bytes of PRGA.

Transmit ARP request (36 bytes) in 9 fragments of 4 data bytes.
- Who has 192.168.0.1 tell 192.168.0.123.

Didn’t get any reply.
- Wrong IP network.
- But AP relayed the packet (since it’s a broadcast).
- Re-encrypted by the AP.
- Knowing the contents, we discover 36 bytes of PRGA.

Send ARP request padded with $x$ 0s (in larger fragments).
- AP relays the longer ARP request.
- Discover $36 + x$ bytes of PRGA.
- Repeat until, say, 1504 bytes of PRGA are known.

Can send 1500 bytes of data without fragmenting.
Eavesdrop WEP packet and determine 8 bytes of PRGA.
Transmit ARP request (36 bytes) in 9 fragments of 4 data bytes.
  - Who has 192.168.0.1 tell 192.168.0.123.
Didn’t get any reply.
  - Wrong IP network.
  - But AP relayed the packet (since it’s a broadcast).
  - Re-encrypted by the AP.
  - Knowing the contents, we discover 36 bytes of PRGA.
Send ARP request padded with \( x \) 0s (in larger fragments).
  - AP relays the longer ARP request.
  - Discover 36 + \( x \) bytes of PRGA.
  - Repeat until, say, 1504 bytes of PRGA are known.
Can send 1500 bytes of data \textit{without} fragmenting.
The Attack
PRGA determination

- Eavesdrop WEP packet and determine 8 bytes of PRGA.
- Transmit ARP request (36 bytes) in 9 fragments of 4 data bytes.
  - Who has 192.168.0.1 tell 192.168.0.123.
- Didn’t get any reply.
  - Wrong IP network.
  - But AP relayed the packet (since it’s a broadcast).
  - Re-encrypted by the AP.
  - Knowing the contents, we discover 36 bytes of PRGA.
- Send ARP request padded with $x$ 0s (in larger fragments).
  - AP relays the longer ARP request.
  - Discover $36 + x$ bytes of PRGA.
  - Repeat until, say, 1504 bytes of PRGA are known.
- Can send 1500 bytes of data without fragmenting.
The Attack
IP determination

- **Send ARP requests for common IP networks and await reply.**
  - No luck—need to be smarter.
- Eavesdrop ARP request/reply and try to decrypt it.
  - Guess next unknown byte of PRGA and send data using it.
    - If correct, AP will relay data.
    - Can decrypt next byte of cipher text.
  - Instead of randomly guessing PRGA, make educated guess on clear text and calculate PRGA from it.
Send ARP requests for common IP networks and await reply.
  - No luck—need to be smarter.

Eavesdrop ARP request/reply and try to decrypt it.
  - Guess next unknown byte of PRGA and send data using it.
    - If correct, AP will relay data.
    - Can decrypt next byte of cipher text.
  - Instead of randomly guessing PRGA, make educated guess on clear text and calculate PRGA from it.
The Attack
IP determination

- Send ARP requests for common IP networks and await reply.
  - No luck—need to be smarter.
- Eavesdrop ARP request/reply and try to decrypt it.
  - Guess next unknown byte of PRGA and send data using it.
    - If correct, AP will relay data.
    - Can decrypt next byte of cipher text.
  - Instead of randomly guessing PRGA, make educated guess on clear text and calculate PRGA from it.

ARP decryption

Know whether its ARP request/reply depending on whether its a broadcast or not.

<table>
<thead>
<tr>
<th>LLC/SNAP</th>
<th>ARP header</th>
<th>Src MAC</th>
<th>??</th>
<th>??</th>
<th>??</th>
<th>??</th>
</tr>
</thead>
</table>

\[ Src IP \]
The Attack
IP determination

- Send ARP requests for common IP networks and await reply.
  - No luck—need to be smarter.
- Eavesdrop ARP request/reply and try to decrypt it.
  - Guess next unknown byte of PRGA and send data using it.
    - If correct, AP will relay data.
    - Can decrypt next byte of cipher text.
  - Instead of randomly guessing PRGA, make educated guess on clear text and calculate PRGA from it.

ARP decryption

Know source MAC—transmitted in clear in 802.11 header!

<table>
<thead>
<tr>
<th>LLC/SNAP</th>
<th>ARP header</th>
<th>Src MAC</th>
<th>??</th>
<th>??</th>
<th>??</th>
<th>??</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Src IP
Send ARP requests for common IP networks and await reply.
  - No luck—need to be smarter.

Eavesdrop ARP request/reply and try to decrypt it.
  - Guess next unknown byte of PRGA and send data using it.
    - If correct, AP will relay data.
    - Can decrypt next byte of cipher text.
  - Instead of randomly guessing PRGA, make educated guess on clear text and calculate PRGA from it.

ARP decryption

Guess first IP byte: 192. Calculate PRGA and send data with it. If it’s relayed, we are correct.
The Attack
IP determination

- Send ARP requests for common IP networks and await reply.
  - No luck—need to be smarter.
- Eavesdrop ARP request/reply and try to decrypt it.
  - Guess next unknown byte of PRGA and send data using it.
    - If correct, AP will relay data.
    - Can decrypt next byte of cipher text.
- Instead of randomly guessing PRGA, make educated guess on clear text and calculate PRGA from it.

### ARP decryption

Guess second IP byte: 168.
The Attack
IP determination

- Send ARP requests for common IP networks and await reply.
  - No luck—need to be smarter.
- Eavesdrop ARP request/reply and try to decrypt it.
  - Guess next unknown byte of PRGA and send data using it.
    - If correct, AP will relay data.
    - Can decrypt next byte of cipher text.
  - Instead of randomly guessing PRGA, make educated guess on clear text and calculate PRGA from it.

**ARP decryption**

Guess third IP byte: 1.

<table>
<thead>
<tr>
<th>LLC/SNAP</th>
<th>ARP header</th>
<th>Src MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>192 168 01 ??</td>
</tr>
</tbody>
</table>

Src IP
The Attack
IP determination

- Send ARP requests for common IP networks and await reply.
  - No luck—need to be smarter.
- Eavesdrop ARP request/reply and try to decrypt it.
  - Guess next unknown byte of PRGA and send data using it.
    - If correct, AP will relay data.
    - Can decrypt next byte of cipher text.
  - Instead of randomly guessing PRGA, make educated guess on clear text and calculate PRGA from it.

ARP decryption

Obtain third IP byte (after at most 256 tries): 11.

<table>
<thead>
<tr>
<th>LLC/SNAP</th>
<th>ARP header</th>
<th>Src MAC</th>
<th>Src IP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>192 168 11 ??</td>
<td></td>
</tr>
</tbody>
</table>

The Attack
IP determination

- Send ARP requests for common IP networks and await reply.
  - No luck—need to be smarter.
- Eavesdrop ARP request/reply and try to decrypt it.
  - Guess next unknown byte of PRGA and send data using it.
    - If correct, AP will relay data.
    - Can decrypt next byte of cipher text.
  - Instead of randomly guessing PRGA, make educated guess on clear text and calculate PRGA from it.

ARP decryption

Send ARP who has 192.168.11.1 tell 192.168.11.123.

- Got reply! IP network is 192.168.11.0.

<table>
<thead>
<tr>
<th>LLC/SNAP</th>
<th>ARP header</th>
<th>Src MAC</th>
<th>Src IP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>192 168 11 ??</td>
<td></td>
</tr>
</tbody>
</table>
By sending ARP request for 192.168.11.1

- Know MAC of router (clear in 802.11 header).
- Router knows our MAC/IP pair (ARP backward learning).

Send ICMP echo to a host we own on Internet.

- Use “our” source MAC/IP pair.
- Use router MAC as destination.
- Obtain network’s public IP address from Internet box.
By sending ARP request for 192.168.11.1

- Know MAC of router (clear in 802.11 header).
- Router knows our MAC/IP pair (ARP backward learning).

Send ICMP echo to a host we own on Internet.

- Use “our” source MAC/IP pair.
- Use router MAC as destination.
- Obtain network’s public IP address from Internet box.
Generate traffic to speed up KSA attacks

- Cause controlled host on Internet to flood network.
- Send ARP requests and ICMPs to broadcast IP.
  - Could generate \( \approx 200 \) packets/s of traffic.
- Key was actually 40-bit alpha-numeric ASCII.
  - Bruteforcable in \( \leq 5 \) minutes . . .

Login to AP and clean up

- Default passwords work great. (root without password here.)
- Clear the logs.
  - Obtain ISP login and send e-mail to customer advising him to use a VPN. [password is recoverable too . . . ]
Generate traffic to speed up KSA attacks

- Cause controlled host on Internet to flood network.
- Send ARP requests and ICMPs to broadcast IP.
  - Could generate $\approx 200$ packets/s of traffic.
- Key was actually 40-bit alpha-numeric ASCII.
  - Bruteforcable in $\leq 5$ minutes . . .

Login to AP and clean up

- Default passwords work great. (root without password here.)
- Clear the logs.
  - Obtain ISP login and send e-mail to customer advising him to use a VPN. [password is recoverable too . . .]
The Tool: wesside

Hardware

- Designed for Atheros based cards.
- Queue the packet and it shall be sent—No firmware hacks!
- Supports 802.11 a/b/g.
- FreeBSD ath driver patched to support injection.
  - Problem with sending 802.11 ACKs. Possibly they are sent too late—DIFS rather than SIFS.
  - Work around: Have another card in range with the same MAC as the attacker. The card will respond to data with ACKs.
The Tool: wesside

Hardware

- Designed for Atheros based cards.
- Queue the packet and it shall be sent—No firmware hacks!
- Supports 802.11 a/b/g.
- FreeBSD ath driver patched to support injection.
  - Problem with sending 802.11 ACKs. Possibly they are sent too late—DIFS rather than SIFS.
  - Work around: Have another card in range with the same MAC as the attacker. The card will respond to data with ACKs.
The Tool: wesside
Operation

1. Finds a WEP network and associates—spoofs MAC if AP does filtering.
2. Eavesdrops a single data packet and discovers at least 128 bytes of PRGA via broadcast relays.
3. Upon capturing an ARP request it discovers the network IP. Sends 256 PRGA guesses in parallel to different multicast addresses. Correct guess is in address of relayed packet.
4. Obtains router’s MAC by ARP request to “.1” IP.
5. Contacts Internet host which will flood.
6. Launches aircrack (v2.1—old!) periodically.
1. Finds a WEP network and associates—spoofs MAC if AP does filtering.

2. Eavesdrops a single data packet and discovers at least 128 bytes of PRGA via broadcast relays.

3. Upon capturing an ARP request it discovers the network IP. Sends 256 PRGA guesses in parallel to different multicast addresses. Correct guess is in address of relayed packet.

4. Obtains router’s MAC by ARP request to “.1” IP.

5. Contacts Internet host which will flood.

6. Launches aircrack (v2.1—old!) periodically.
The Tool: wesside

Operation

1. Finds a WEP network and associates—spoofs MAC if AP does filtering.

2. Eavesdrops a single data packet and discovers at least 128 bytes of PRGA via broadcast relays.

3. Upon capturing an ARP request it discovers the network IP. Sends 256 PRGA guesses in parallel to different multicast addresses. Correct guess is in address of relayed packet.

4. Obtains router’s MAC by ARP request to “.1” IP.

5. Contacts Internet host which will flood.

6. Launches aircrack (v2.1—old!) periodically.
The Tool: wesside

Operation

1. Finds a WEP network and associates—spoofs MAC if AP does filtering.
2. Eavesdrops a single data packet and discovers at least 128 bytes of PRGA via broadcast relays.
3. Upon capturing an ARP request it discovers the network IP. Sends 256 PRGA guesses in parallel to different multicast addresses. Correct guess is in address of relayed packet.
4. Obtains router’s MAC by ARP request to “.1” IP.
5. Contacts Internet host which will flood.
6. Launches aircrack (v2.1—old!) periodically.
The Tool: wesside

Operation

1. Finds a WEP network and associates—spoofs MAC if AP does filtering.
2. Eavesdrops a single data packet and discovers at least 128 bytes of PRGA via broadcast relays.
3. Upon capturing an ARP request it discovers the network IP. Sends 256 PRGA guesses in parallel to different multicast addresses. Correct guess is in address of relayed packet.
4. Obtains router’s MAC by ARP request to “.1” IP.
5. Contacts Internet host which will flood.
6. Launches aircrack (v2.1—old!) periodically.
The Tool: wesside

Operation

1. Finds a WEP network and associates—spoofs MAC if AP does filtering.

2. Eavesdrops a single data packet and discovers at least 128 bytes of PRGA via broadcast relays.

3. Upon capturing an ARP request it discovers the network IP. Sends 256 PRGA guesses in parallel to different multicast addresses. Correct guess is in address of relayed packet.

4. Obtains router’s MAC by ARP request to “.1” IP.

5. Contacts Internet host which will flood.

6. Launches aircrack (v2.1—old!) periodically.
The Tool: wesside

Operation

- Finds a WEP network and associates—spoofs MAC if AP does filtering.
- Eavesdrops a single data packet and discovers at least 128 bytes of PRGA via broadcast relays.
- Upon capturing an ARP request it discovers the network IP. Sends 256 PRGA guesses in parallel to different multicast addresses. Correct guess is in address of relayed packet.
- Obtains router’s MAC by ARP request to “.1” IP.
- Contacts Internet host which will flood.
- Launches aircrack (v2.1—old!) periodically.

IV dictionary built in parallel!

Binds to a TAP interface allowing transmission and reception (if PRGA is known).
After a single ARP request is eavesdropped:

- 144 bytes of PRGA are recovered in 1 second.
- IP is decrypted in < 30 seconds.
- Internet host is contacted in < 1 minute (total time).
The Tool: wesside
Bootstrap time & flood rate

After a single ARP request is eavesdropped:

- 144 bytes of PRGA are recovered in 1 second.
- IP is decrypted in < 30 seconds.
- Internet host is contacted in < 1 minute (total time).

### Traffic generation rate

<table>
<thead>
<tr>
<th>Flood source</th>
<th>≈ p/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.11b client FTP download.</td>
<td>150</td>
</tr>
<tr>
<td>LAN client ping -f (no replies).</td>
<td>550</td>
</tr>
<tr>
<td>Internet flood (MTU sized packets).</td>
<td>250</td>
</tr>
<tr>
<td>ARP replay.</td>
<td>350</td>
</tr>
<tr>
<td>Internet flood (short packets).</td>
<td>950</td>
</tr>
</tbody>
</table>

Full dictionary requires \( \approx \frac{2^{24}}{250} \times \frac{1}{3600} \approx 18.6 \) hours of flooding.
## The Tool: wesside

Key recovery time

<table>
<thead>
<tr>
<th>Key</th>
<th>Packets</th>
<th>Time (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2C:CE:FC:1D:2B</td>
<td>100,000</td>
<td>1.93</td>
</tr>
<tr>
<td>80:19:B8:3F:C8</td>
<td>200,000</td>
<td>3.83</td>
</tr>
<tr>
<td>6F:34:11:BC:A3</td>
<td>200,000</td>
<td>4.30</td>
</tr>
<tr>
<td>91:B7:C0:A7:F7</td>
<td>300,000</td>
<td>5.45</td>
</tr>
<tr>
<td>3B:07:DA:02:B7</td>
<td>300,000</td>
<td>5.60</td>
</tr>
<tr>
<td>EB:A6:50:D0:2B:DA:CC:B7:E1:B7:E8:50:59</td>
<td>1,700,000</td>
<td>30.77</td>
</tr>
<tr>
<td>64:AC:EE:55:B7:7E:27:93:09:6B:78:00:78</td>
<td>9,000,000</td>
<td>156.58</td>
</tr>
<tr>
<td>41:0A:68:52:5B:BE:C7:64:D7:09:FC:CC:BB</td>
<td>10,000,000</td>
<td>181.28</td>
</tr>
</tbody>
</table>
The Tool: wesside

Screen shot

# ./wesside -s 1.2.3.4
[10:49:50] Setting up ath0... done
[10:49:50] Set tap MAC to: 00:00:DE:FA:CE:0D
[10:49:50] Looking for a victim...

...
...[10:49:54] Got ARP request from (08:00:46:9E:AF:CD)
[10:49:54] Got 8 bytes of prga IV=(42:bc:00)
[10:49:54] Got 36 bytes of prga IV=(43:bc:00)
[10:49:55] Got 144 bytes of prga IV=(52:bc:00)
[10:50:00] Guessing PRGA 2d (IP byte=175)
[10:50:00] Got clear-text byte: 168
[10:50:09] Guessing PRGA f7 (IP byte=0)
[10:50:09] Got clear-text byte: 1
[10:50:18] Guessing PRGA f7 (IP byte=102)
[10:50:18] Sending arp request for: 192.168.1.1
...
The Tool: wesside

Screen shot

...  
[10:51:28] WEP=000100460 (next crack at 100000) (rate=1448)
[10:51:28] Starting crack PID=17410
[10:52:28] WEP=000185271 (next crack at 200000) (rate=1426)
[10:52:28] Stopping crack PID=17410
[10:52:39] WEP=000201124 (next crack at 200000) (rate=1433)
[10:52:39] Starting crack PID=17412
[10:52:40] WEP=000203778 (next crack at 300000) (rate=1365)

Owned in 2.85 minutes
#


Able to transmit arbitrary data on most (all?) 802.11 WEP networks after having eavesdropped a single data packet.

Can potentially recover a WEP key in a couple of hours.

Future Work:

- Develop method for higher flood rates (p/s).
- Study how IV generator can be reset—smaller dictionaries.
- Implement a more sophisticated tool and make a Live CD!

A final thought for the adventurous...

Assume the AP uses default password for WWW interface.

- Connect to WWW and request WEP configuration page.
- Decrypt TCP sequence number for connection ACK.
- Decrypt contents of page returned—may contain WEP key!