



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

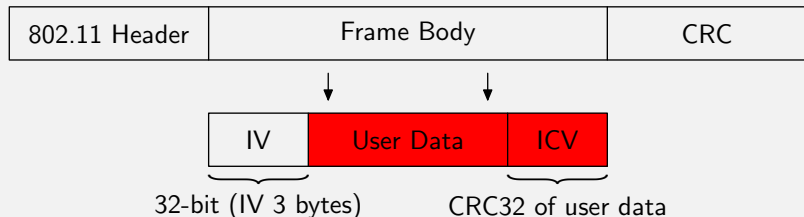
Wired Equivalent Privacy?

Overview



- 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



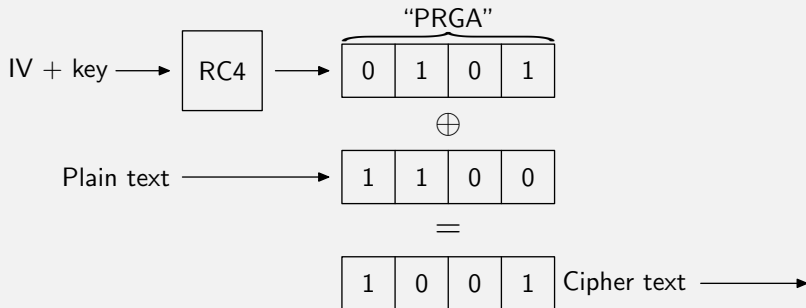
Wired Equivalent Privacy??

Encryption



- ① **Seed**: Choose IV (any 24-bit number) and prepend to key.
- ② **KSA**: Run RC4 Key Scheduling Algorithm on seed.
- ③ **PRGA**: Run RC4 Pseudo-Random Generation Algorithm.
- ④ **XOR**: XOR user data with PRGA.

WEP Encryption





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!



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!



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.

Sample PRGA

0	1	0	1	PRGA
0	0	1	1	Plain text
0	1	1	0	Cipher text

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.

Sample PRGA

0	1	0	1	PRGA
0	0	1	1	Plain text
0	1	1	0	Cipher text

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 \oplus 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 \oplus 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!

PRGA Discovery

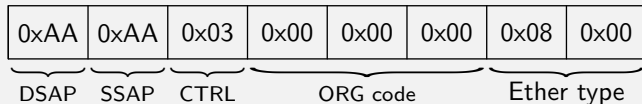
How much clear text do we know?



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



ARP packets have 0x0806 as ethernet type!

- Distinguishable by fixed and short length.

In general, we can recover at least 8 bytes of PRGA.

Fragmentation

Greets: Josh Lackey, h1kari, anton, abaddon

Introduction
Theory
Practice
Conclusion



10/24

802.11 supports fragmentation at a MAC layer.

- Each WEP fragment is encrypted independently.

Fragmentation

Greets: Josh Lackey, h1kari, anton, abaddon

Introduction
Theory
Practice
Conclusion



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.



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



- ① Eavesdrop a WEP packet.
- ② Recover 8 bytes of PRGA (clear \oplus 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 \oplus 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! (greet 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.

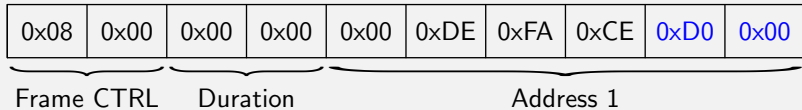


FreeBSD using `wi` driver.

- 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.





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				

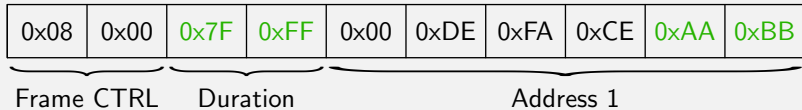


FreeBSD using `wi` driver.

- 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.**



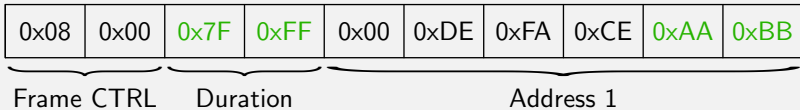


FreeBSD using `wi` driver.

- 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.



- Able to send any 802.11 frame and receive all frames.



- **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 *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 *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.**



- 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.



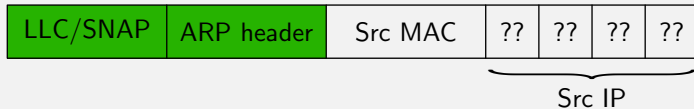
- Send ARP requests for common IP networks and await reply.
 - No luck—need to be smarter.
- **Eavesdrop ARP request/reply and try to decrypt it.**
 - Guess next unknown byte of PRGA and send data using it.
 - If correct, AP will relay data.
 - Can decrypt next byte of cipher text.
 - Instead of randomly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



- 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.



The Attack

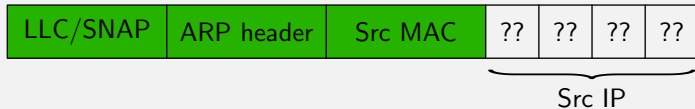
IP determination



- 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!

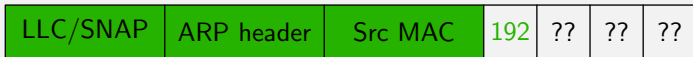




- Send ARP requests for common IP networks and await reply.
 - No luck—need to be smarter.
- Eavesdrop ARP request/reply and try to decrypt it.
 - Guess next unknown byte of PRGA and send data using it.
 - If correct, AP will relay data.
 - Can decrypt next byte of cipher text.
 - Instead of randomly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



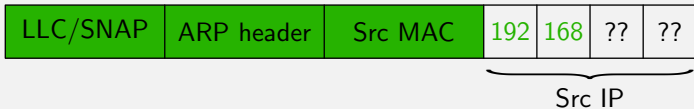
Src IP



- 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 second IP byte: 168.



The Attack

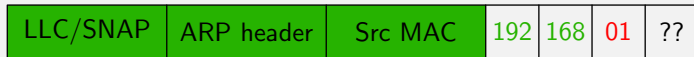
IP determination



- 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.



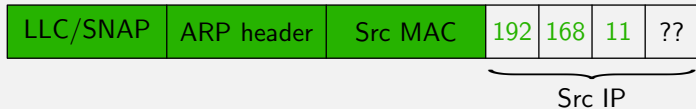
Src IP



- 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

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





- 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.

LLC/SNAP	ARP header	Src MAC	192	168	11	??
----------	------------	---------	-----	-----	----	----

Src IP



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 ≈ 200 packets/s of traffic.
- Key was actually 40-bit alpha-numeric ASCII.
 - Bruteforceable in ≤ 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 ≈ 200 packets/s of traffic.
- Key was actually 40-bit alpha-numeric ASCII.
 - Bruteforceable in ≤ 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 ...]



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.



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.



- 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.



- 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.



- 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.**



- 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.

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).



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

Flood source	\approx p/s
802.11b client FTP download.	150
LAN client ping -f (no replies).	550
Internet flood (MTU sized packets). ARP replay.	250 350
Internet flood (short packets).	950

Full dictionary requires $\approx \frac{2^{24}}{250} \times \frac{1}{3600} \approx 18.6$ hours of flooding.

The Tool: wesside

Key recovery time

Introduction
Theory
Practice
Conclusion



21/24

Total attack time for /dev/urandom keys

Key	Packets	Time (m)
2C:CE:FC:1D:2B	100,000	1.93
80:19:B8:3F:C8	200,000	3.83
6F:34:11:BC:A3	200,000	4.30
91:B7:C0:A7:F7	300,000	5.45
3B:07:DA:02:B7	300,000	5.60
EB:A6:50:D0:2B:DA:CC:B7:E1:B7:E8:50:59	1,700,000	30.77
D9:06:CA:9E:EA:B3:18:CD:24:9F:2E:5E:10	2,400,000	42.85
5E:02:F4:83:FE:F6:27:10:21:EC:8E:87:27	2,700,000	49.17
64:AC:EE:55:B7:7E:27:93:09:6B:78:00:78	9,000,000	156.58
41:0A:68:52:5B:BE:C7:64:D7:09:FC:CC:BB	10,000,000	181.28

The Tool: wesside

Screen shot

Introduction
Theory
Practice
Conclusion



23/24

```
# ./wesside -s 1.2.3.4
[10:49:50] Setting up ath0... done
[10:49:50] Opened tap device: tap3
[10:49:50] Set tap MAC to: 00:00:DE:FA:CE:0D
[10:49:50] Looking for a victim...
[10:49:53] Found SSID(sorbo) BSS=(00:06:25:FF:D2:29) chan=11
[10:49:53] Authenticated
[10:49:53] Associated (ID=3)
...
```

The Tool: wesside

Screen shot

Introduction
Theory
Practice
Conclusion



23/24

...

[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)

[10:49:58] Guessing PRGA 5f (IP byte=255)

[10:49:58] Got clear-text byte: 192

[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] Got clear-text byte: 100

[10:50:18] Got IP=(192.168.1.100)

[10:50:18] My IP=(192.168.1.123)

[10:50:18] Sending arp request for: 192.168.1.1

[10:50:18] Got arp reply from (00:06:25:FF:D2:27)

...

The Tool: wesside

Screen shot

Introduction
Theory
Practice
Conclusion



...

[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)

[10:49:58] Guessing PRGA 5f (IP byte=255)

[10:49:58] Got clear-text byte: 192

[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] Got clear-text byte: 100

[10:50:18] Got IP=(192.168.1.100)

[10:50:18] My IP=(192.168.1.123)

[10:50:18] Sending arp request for: 192.168.1.1

[10:50:18] Got arp reply from (00:06:25:FF:D2:27)

...

The Tool: wesside

Screen shot

Introduction
Theory
Practice
Conclusion



23/24

...

[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)

[10:52:41] KEY=(2C:CE:FC:1D:2B)

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!