Mathy Vanhoef

New Flaws in WPA-TKIP

@vanhoefm

Brucon 2012
Why listen to me?
Why listen to me?

universiteit hasselt

ERNST & YOUNG
Quality In Everything We Do

Gandanur

FWO VLAANDEREN

KATHOLIEKE UNIVERSITEIT LEUVEN
if (frame.isAgenda) 
print_callstack();

0x00  The WPA-TKIP protocol
0x04  Denial of Service
0x08  Demo
0x0C  Beck & Tews attack
0x10  Fragmentation attack
0x14  Performing a port scan
The WPA-TKIP Protocol

We will cover:

- Connecting
- Sending & receiving packets
- Quality of Service (QoS) extension

Design Constraints:

- Must run on legacy hardware
- Uses (hardware) WEP encapsulation
4-way handshake

- Defined by EAPOL and results in a session key
- What you normally capture & crack
4-way handshake

- Result of handshake is 512 bit session key
- Renewed after rekeying timeout (1 hour)
4-way handshake

- Result of handshake is 512 bit session key
- Renewed after rekeying timeout (1 hour)

- DataEncr key: used to encrypt packets
4-way handshake

- Result of handshake is 512 bit session key
- Renewed after rekeying timeout (1 hour)

<table>
<thead>
<tr>
<th>EAPOL protection</th>
<th>DataEncr</th>
<th>MIC1</th>
<th>MIC2</th>
</tr>
</thead>
</table>

- DataEncr key: used to encrypt packets
- MIC keys (Message Integrity Code):
  - Verify integrity of data. But why two?
Why two MIC keys?

- WPA-TKIP designed for old hardware
  - Couldn’t use strong integrity checks (CCMP)
- New algorithm called Michael was created
  - Weakness: plaintext + MIC reveals MIC key

- To improve security two MIC keys are used
  - MIC₁ for AP to client communication
  - MIC₂ for client to AP communication
Sending Packets

- Calculate MIC to assure integrity

Diagram: TSC | Data | MIC | CRC

Encrypted
Sending Packets

- Calculate MIC to assure integrity
- WEP Encapsulation:
  - Calculate CRC
Sending Packets

- Calculate MIC to assure integrity
- WEP Encapsulation:
  - Calculate CRC
  - Encrypt the packet using RC4
Calculating MIC to assure integrity.

WEP Encapsulation:
- Calculate CRC
- Encrypt the packet using RC4
- Add replay counter (TSC) to avoid replays
Calculate MIC to assure integrity

WEP Encapsulation:
  - Calculate CRC
  - Encrypt the packet using RC4
  - Add replay counter (TSC) to avoid replays
Receiving Packets

- WEP decapsulation:
  - Verify TSC to prevent replays
Receiving Packets

- WEP decapsulation:
  - Verify TSC to prevent replays
  - Decrypt packet using RC4
Receiving Packets

- WEP decapsulation:
  - Verify TSC to prevent replays
  - Decrypt packet using RC4
  - Verify CRC
Receiving Packets

- WEP decapsulation:
  - Verify TSC to prevent replays
  - Decrypt packet using RC4
  - Verify CRC
- Verify MIC to assure authenticity
WEP decapsulation:
- Verify TSC to prevent replays
- Decrypt packet using RC4
- Verify CRC

Verify MIC to assure authenticity
MIC Defense Mechanism

- Replay counter & CRC are good, but MIC not
  - Transmission error unlikely
  - Network may be under attack!
MIC Defense Mechanism

- Replay counter & CRC are good, but MIC not
  - Transmission error unlikely
  - Network may be under attack!

Defense mechanism on MIC failure:
- Client sends MIC failure report to AP
- AP silently logs failure
- Two failures in 1 min: network down for 1 min
Quality of Service (QoS)

- Defines several QoS channels
- Implemented by new field in 802.11 header
Quality of Service (QoS)

- Defines several QoS channels
- Implemented by new field in 802.11 header

<table>
<thead>
<tr>
<th>QoS</th>
<th>TSC</th>
<th>Data</th>
<th>MIC</th>
<th>CRC</th>
</tr>
</thead>
</table>

Encrypted
Quality of Service (QoS)

- Defines several QoS channels
- Implemented by new field in 802.11 header

<table>
<thead>
<tr>
<th>QoS</th>
<th>TSC</th>
<th>Data</th>
<th>MIC</th>
<th>CRC</th>
</tr>
</thead>
</table>

unencrypted    Encrypted
Quality of Service (QoS)

- Defines several QoS channels
- Implemented by new field in 802.11 header

<table>
<thead>
<tr>
<th>QoS</th>
<th>TSC</th>
<th>Data</th>
<th>MIC</th>
<th>CRC</th>
</tr>
</thead>
</table>

unencrypted ➔ Encrypted

- Individual replay counter (TSC) per channel
- Used to pass replay counter check of receiver!
For Example:

- Support for up to 8 channels
- But WiFi certification only requires 4

<table>
<thead>
<tr>
<th>Channel</th>
<th>TSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: Best Effort</td>
<td>4000</td>
</tr>
<tr>
<td>1: Background</td>
<td>0</td>
</tr>
<tr>
<td>2: Video</td>
<td>0</td>
</tr>
<tr>
<td>3: Voice</td>
<td>0</td>
</tr>
</tbody>
</table>
MIC = Michael(MAC dest, MAC source, MIC key, priority, data)

Rc4key = MixKey(MAC transmitter, key, TSC)
The previous slides contain all the info to find a denial of service attack, any ideas? 😊
Wait a minute...

- The previous slides contain all the info to find a denial of service attack, any ideas?

- Key observations:
  - Individual replay counter per priority
  - Priority influences MIC but not encryption key
  - Two MIC failures: network down
The previous slides contain all the info to find a denial of service attack, any ideas?

Key observations:
- Individual replay counter per priority
- Priority influences MIC but not encryption key
- Two MIC failures: network down

What happens when the priority is changed?
Changing the priority

- Capture packet, change priority, replay

On Reception:
- Verify replay counter
- Decrypt packet using RC4
- Verify CRC (leftover from WEP)
- Verify MIC to assure authenticity
Changing the priority

- Capture packet, change priority, replay

On Reception:
- Verify replay counter: OK
- Decrypt packet using RC4
- Verify CRC (leftover from WEP)
- Verify MIC to assure authenticity
Changing the priority

- Capture packet, change priority, replay

On Reception:
- Verify replay counter OK
- Decrypt packet using RC4 OK
- Verify CRC (leftover from WEP)
- Verify MIC to assure authenticity
Changing the priority

Capture packet, change priority, replay

On Reception:
- Verify replay counter: OK
- Decrypt packet using RC4: OK
- Verify CRC (leftover from WEP): OK
- Verify MIC to assure authenticity
Changing the priority

- Capture packet, change priority, replay

On Reception:
- Verify replay counter: OK
- Decrypt packet using RC4: OK
- Verify CRC (leftover from WEP): OK
- Verify MIC to assure authenticity: FAIL
Denial of Service Attack

- Capture packet, change priority, replay

On Reception:
- Verify replay counter: OK
- Decrypt packet using RC4: OK
- Verify CRC (leftover from WEP): OK
- Verify MIC to assure authenticity: **FAIL**

→ Do this twice: Denial of Service
If QoS is disabled?

- Disadvantage: attack fails if QoS is disabled
- Solution: Capture packet, add QoS header, change priority, replay
If QoS is disabled?

- Disadvantage: attack fails if QoS is disabled
- Solution: Capture packet, **add QoS header**, change priority, replay

On Reception:
- Doesn’t check whether QoS is actually used
If QoS is disabled?

- Disadvantage: attack fails if QoS is disabled
- Solution: Capture packet, add QoS header, change priority, replay

On Reception:
- Doesn’t check whether QoS is actually used
- Again bypass replay counter check
- MIC still dependent on priority
If QoS is disabled?

- Disadvantage: attack fails if QoS is disabled
- Solution: Capture packet, add QoS header, change priority, replay

On Reception:
- Doesn’t check whether QoS is actually used
- Again bypass replay counter check
- MIC still dependent on priority

[Cryptanalysis for RC4 and breaking WEP/WPA-TKIP]
Time for action: Demo!

Attacker: VMWare  vs.  Victim: Windows
Example: network with 20 connected clients

Deauthentication attack:
- Must continuously sends packets
- Say 10 deauths per client per second
- \((10 \times 60) \times 20 = 12\,000\) frames per minute

New attack
- 2 frames per minute
if (frame.isAgenda) {
    print_callstack();
}

0x00 The WPA-TKIP protocol
0x04 Denial of Service
0x08 Demo
0x0C Beck & Tews attack
0x10 Fragmentation attack
0x14 Performing a port scan
Beck & Tews Attack

- First known attack on TKIP, requires QoS
- Decrypts ARP reply sent from AP to client
Beck & Tews Attack

- First known attack on TKIP, requires QoS
- Decrypts ARP reply sent from AP to client

Simplified: each byte is decrypted by sending a modified packet for all 256 possible values:
  - Wrong guess: CRC invalid
  - Correct guess: CRC valid but MIC failure
Beck & Tews Attack

- First known attack on TKIP, requires QoS
- Decrypts ARP reply sent from AP to client

- Simplified: each byte is decrypted by sending a modified packet for all 256 possible values:
  - Wrong guess: CRC invalid
  - Correct guess: CRC valid but MIC failure

- MIC key for AP to client
Beck & Tews Attack

- Takes 12 minutes to execute
- Limited impact: injection of 3-7 small packets
What is needed to inject packets:
- MIC key
  - Result of Beck & Tews attack
What is needed to inject packets:

- MIC key
  - Result of Beck & Tews attack
- Unused replay counter
  - Inject packet on unused QoS channel
What is needed to inject packets:
- MIC key
  - Result of Beck & Tews attack
- Unused replay counter
  - Inject packet on unused QoS channel
- Keystream corresponding to replay counter
  - Beck & Tews results in only one keystream...
  - **How can we get more?** First need to know RC4!
Background: RC4 algorithm

- Stream cipher
- XOR-based

This means:

1. Ciphertext
2. Plaintext
3. Keystream

→ Predicting the plaintext gives the keystream
Predicting packets

Simplified:
- All data packets start with LLC header
- Different for APR, IP and EAPOL packets
- Detect ARP & EAPOL based on length
- Everything else: IP
Predicting packets

Simplified:
- All data packets start with LLC header
- Different for APR, IP and EAPOL packets
- Detect ARP & EAPOL based on length
- Everything else: IP

- Practice: almost no incorrect guesses!
- Gives us 12 bytes keystream for each packet
But is 12 bytes enough to send a packet?
No, MIC & CRC alone are 12 bytes.

If only we could somehow combine them...
But is 12 bytes enough to send a packet?
No, MIC & CRC alone are 12 bytes.

If only we could somehow combine them...
...well, title of this section is fragmentation
Using short keystreams

- But is 12 bytes enough to send a packet?
- No, MIC & CRC alone are 12 bytes.

If only we could somehow combine them...
...well, title of this section is fragmentation

- Using 802.11 fragmentation we can combine 16 keystreams to send one large packet
802.11 fragmentation
802.11 fragmentation

- MIC calculated over complete packet
MIC calculated over complete packet
802.11 fragmentation

- MIC calculated over complete packet
- Each fragment has CRC and different TSC
MIC calculated over complete packet
Each fragment has CRC and different TSC
12 bytes/keystream: inject 120 bytes of data
Fragmentation Attack

- Beck & Tews attack: MIC key AP to client
- Predict packets & get keystreams
- Combine short keystreams by fragmentation
- Send over unused QoS channel
Beck & Tews attack: MIC key AP to client
- Predict packets & get keystreams
- Combine short keystreams by fragmentation
- Send over unused QoS channel

What can we do with this?
- ARP/DNS Poisoning
- Sending TCP SYN packets: port scan!
Port scan on TKIP client

A few notes:
- Scan 500 most popular ports
- Detect SYN/ACK based on length
- Avoid multiple SYN/ACK’s: send RST

Port scan of internal client:
- Normally not possible
- We are bypassing the network firewall / NAT!
Demo: port scan
Random remark:

Building packets sucks... 😞

```c
int z;

if (((h80211[0] & 0x0C) != 8)
    return 0; //must be a data packet
if (((h80211[0] & 0x70) != 0)
    return 0;
if (((h80211[1] & 0x40) == 0)
    return 0;

// Get the header length
z = (((h80211[1] & 3) != 3) ? 24 : 30;
if (((h80211[0] & 0x80) == 0x80) /* QoS */
    z += 2;

// Must be a TKIP/CCMP frame
if (((h80211[z + 3] & 0x20) == 0)
    return 0;
```

```plaintext
- targetPortId: 1
- tlvType: Cancel unicast transmission (6)
- lengthField: 2

[Malformed Packet: PTP]
[Expert Info (Error/Malformed): Malformed]
[Message: Malformed Packet (Exception of...]
[Severity level: Error]
[Group: Malformed]
```
... until wireshark crashes ...

```
root@bt:~# wireshark &
[1] 6001
root@bt:~# *** buffer overflow detected ***: wireshark terminated
======== Backtrace: =======
/lib/tls/i686/cmov/libc.so.6(__fortify_fail+0x50)[0xb4885390]
/lib/tls/i686/cmov/libc.so.6(+0xe12ca)[0xb48842ca]
/usr/local/lib/libwireshark.so.2(+0x605980)[0xb561a980]
/usr/local/lib/libwireshark.so.2(+0x9a33f9)[0xb59b83f9]
/usr/local/lib/libwireshark.so.2(+0x9afefb)[0xb59c4efb]
/usr/local/lib/libwireshark.so.2(+0x9b5010)[0xb59ca010]
/usr/local/lib/libwireshark.so.2(+0x5ba986)[0xb55cf986]
/usr/local/lib/libwireshark.so.2(+0x5bb1e9)[0xb55d01e9]
/usr/local/lib/libwireshark.so.2(call_dissector+0x3a)[0xb55d03ea]
/usr/local/lib/libwireshark.so.2(+0x9b6fb0)[0xb59cbfb0]
/usr/local/lib/libwireshark.so.2(+0x5ba986)[0xb55cf986]
```
... and it’s reproducible

tcpdump -i mon0 -w crash.pcap
Can we pass the firewall?

- Target will send outgoing SYN/ACK
- Will this go through the firewall/NAT?
- *Normally* not...

<table>
<thead>
<tr>
<th>Device</th>
<th>SYN/ACK forwarded?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scarlet VDSL Box</td>
<td>No</td>
</tr>
<tr>
<td>WAG320N</td>
<td>No</td>
</tr>
<tr>
<td>OpenBSD/PF</td>
<td>No</td>
</tr>
<tr>
<td>DD-WRT</td>
<td>When SPI is disabled</td>
</tr>
</tbody>
</table>
If we can pass NAT

- Realistic in practice?
- Bidirectional traffic is possible
If we can pass NAT

- Realistic in practice?
- Bidirectional traffic is possible
If we can pass NAT

- Realistic in practice?
- Bidirectional traffic is possible
If we can pass NAT

- Realistic in practice?
- Bidirectional traffic is possible
If we can pass NAT

- Realistic in practice?
- Bidirectional traffic is possible
If we can pass NAT

- Realistic in practice?
- Bidirectional traffic is possible
If we can pass NAT

- Realistic in practice?
- Can connect to open ports
Worst case scenario

- Client running SSH server with weak password
- *Bypass firewall using fragmentation attack*
- Bidirectional communication is possible
- Connect to SSH server as root
Worst case scenario

- Client running SSH server with weak password
- *Bypass firewall using fragmentation attack*
- Bidirectional communication is possible
- Connect to SSH server as root
- Dump the network password!
Worst case scenario

- Client running SSH server with weak password
- *Bypass firewall using fragmentation attack*
- Bidirectional communication is possible
- Connect to SSH server as root
- Dump the network password!

Note: not been tested
Comparison

Beck & Tews:
- Inject 3-7 packets of 28 bytes

Fragmentation:
- Inject *arbitrary* amount of packets
- With a size up to 120 bytes
- Additionally, exploit IP fragmentation to transmit IP packets of arbitrary size
Fun with wireless adapters

Belkin F5D7053:
- Ignores TSC... you can simply replay a packet
- When connected to a protected network, it still accepts *unencrypted* packets
Belkin F5D7053:

- Ignores TSC... you can simply replay a packet
- When connected to a protected network, it still accepts *unencrypted* packets
Belkin F5D7053:
- Ignores TSC... you can simply replay a packet
- When connected to a protected network, it still accepts *unencrypted* packets
Conclusion

- Very efficient Denial of Service
- Use fragmentation to launch actual attacks

- Forced to use WPA-TKIP?
  - Use short rekeying timeout (2 mins)
  - Disable QoS and update drivers (if possible)

- Update to WPA2-AES
  - Specifically set encryption to AES only
Questions?

@vanhoefm

Brucon 2012