KRACKing WPA2 and Mitigating Future Vulnerabilities

Mathy Vanhoef — @vanhoefm

HackPra, Ruhr-Universität Bochum, 18 July 2018
Overview

Key reinstall in 4-way handshake

Practical impact

Misconceptions

FACTS

MYTHS

Channel validation
Overview

Key reinstall in 4-way handshake

Misconceptions

Practical impact

Channel validation
The 4-way handshake

Used to connect to any protected Wi-Fi network

› Provides mutual authentication
› Negotiates fresh PTK: pairwise transient key

Appeared to be secure:

› No attacks in over a decade (apart from password guessing)
› Proven that negotiated key (PTK) is secret
› And encryption protocol proven secure
4-way handshake (simplified)

optional 802.1x authentication
4-way handshake (simplified)
4-way handshake (simplified)

PTK = Combine(shared secret, ANonce, SNonce)
4-way handshake (simplified)

PTK = Combine(shared secret, ANonce, SNonce)
4-way handshake (simplified)

optional 802.1x authentication

Msg1(r, ANonce)

Derive PTK

Msg2(r, SNonce)

Derive PTK
4-way handshake (simplified)

optional 802.1x authentication

Msg1(r, ANoncex)

Derive PTK

Msg2(r, SNoncex)

Msg3(r+1; GTK)

Derive PTK

Msg4(r+1)
4-way handshake (simplified)

optional 802.1x authentication

Msg1(r, ANonce)

Derive PTK

Msg2(r, SNonce)

Msg3(r+1; GTK)

Derive PTK

Msg4(r+1)

Install PTK & GTK

Install PTK
4-way handshake (simplified)

optional 802.1x authentication

(Msg1, ANonce)

Derive PTK

(Msg2, SNonce)

Derive PTK

(Msg3(r+1; GTK))

(Msg4(r+1))

PTK is installed

Install PTK & GTK

Install PTK
4-way handshake (simplified)

optional 802.1x authentication

Msg1(r, ANonce)

Derive PTK

Msg2(r, SNonce)

Msg3(r+1; GTK)

Derive PTK

Msg4(r+1)

Install PTK & GTK

Install PTK

encrypted data frames can now be exchanged
Frame encryption (simplified)

Nonce (packet number)

PTK (session key)

Mix

Packet key

Plaintext data

Keystream

encrypted data

Nonce

→ Nonce reuse implies keystream reuse (in all WPA2 ciphers)
4-way handshake (simplified)

optional 802.1x authentication

(Msg1(r, ANonce))

Derive PTK

(Msg2(r, SNonce))

Installing PTK initializes nonce to zero

Install PTK & GTK

Install PTK

Encrypted data frames can now be exchanged
Reinstallation Attack

Channel 1

Channel 6
Reinstallation Attack
## Reinstallation Attack

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Msg1</td>
<td>(r, ANonce)</td>
</tr>
<tr>
<td>Msg2</td>
<td>(r, SNonce)</td>
</tr>
<tr>
<td>Msg3</td>
<td>(r+1; GTK)</td>
</tr>
</tbody>
</table>

**optional 802.1x authentication**
Reinstallation Attack

```
optional 802.1x authentication

Msg1(r, ANonce)  Msg1(r, ANonce)
Msg2(r, SNonce)  Msg2(r, SNonce)
Msg3(r+1; GTK)   Msg3(r+1; GTK)
Msg4(r+1)        Block Msg4

Install PTK & GTK
```
Reinstallation Attack

Install PTK & GTK

\[ \text{Enc}^1_{\text{ptk}} \{ \text{Msg4}(r+2) \} \]

\[ \text{Msg3}(r+2; \text{GTK}) \]

\[ \text{Msg4}(r+1) \]
Reinstallation Attack

Install PTK & GTK

\[ \text{Enc}^1_{\text{ptk}} \{ \text{Msg4(r+2)} \} \]

\[ \text{Msg3(r+2; GTK)} \]

\[ \text{Msg4(r+1)} \]
Reinstallation Attack

In practice Msg4 is sent encrypted

$\text{Enc}_{\text{ptk}}^1 \{ \text{Msg4}(r+2) \}$
Key reinstallation!

nonce is reset
Reinstallation Attack

- Install PTK & GTK
  - Msg4(r+1)
  - Msg(3r+2; GTK)
  - Enc$_{ptk}^{-1}$ { Msg4(r+2) }
  - Reinstall PTK & GTK
  - Enc$_{ptk}^{-1}$ { Data(...) }

- Msg3(r+2; GTK)
  - Enc$_{ptk}^{-1}$ { Data(...) }
Reinstallation Attack

Msg4(r+1)

Install PTK & GTK

Msg3(r+2; GTK)

Enc_{ptk}^1 \{ Msg4(r+2) \}

Reinstall PTK & GTK

Enc_{ptk}^1 \{ Data(… \})

Msg3(r+2; GTK)

Enc_{ptk} \{ Data(… \})

Same nonce is used!
Reinstallation Attack

Install PTK & GTK

Msg3(r+2; GTK)

Enc^{1}_{ptk}\{ Msg4(r+2) \}

Reinstall PTK & GTK

Enc^{1}_{ptk}\{ Data(...) \}

Keystream

Msg4(r+1)

Msg3(r+2; GTK)

Decrypted!
Key Reinstallation Attack

Other Wi-Fi handshakes also vulnerable:
› Group key handshake
› FT handshake
› TDLS PeerKey handshake

For details see our CCS’17 paper:
› “Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2”
Overview

Key reinstalls in 4-way handshake

Practical impact

Misconceptions

Channel validation
General impact

Transmit nonce reset

Decrypt frames sent by victim

Receive replay counter reset

Replay frames towards victim
Cipher suite specific

AES-CCMP: No practical frame forging attacks

WPA-TKIP:
› Recover Message Integrity Check key from plaintext
› **Forge/inject** frames sent by the device under attack

GCMP (WiGig):
› Recover GHASH authentication key from nonce reuse
› **Forge/inject** frames in **both directions**
Handshake specific

Group key handshake:

› Client is attacked, but only AP sends real broadcast frames
Handshake specific

Group key handshake:
› Client is attacked, but only AP sends real broadcast frames
› Can only replay broadcast frames to client

4-way handshake: client is attacked $\rightarrow$ replay/decrypt/forge

FT handshake (fast roaming = 802.11r):
› Access Point is attacked $\rightarrow$ replay/decrypt/forge
› No MitM required, can keep causing nonce resets
Implementation specific

iOS 10 and Windows: 4-way handshake not affected
› **Cannot decrypt unicast traffic** (nor replay/decrypt)
› But group key handshake is affected (replay broadcast)
› Note: iOS 11 does have vulnerable 4-way handshake

wpa_supplicant 2.4+
› Client used on Linux and Android 6.0+
› On retransmitted msg3 will **install all-zero key**
initial stage of 4-way handshake
Android (victim)
initial stage of 4-way handshake

Derive PTK

Msg3(r+1; GTK)  Msg3(r+1; GTK)

Msg4(r+1)       Msg4(r+1)
initial stage of 4-way handshake

Derive PTK

Msg3(r+1; GTK) ← Msg3(r+1; GTK)

Msg4(r+1) ← Msg4(r+1)

Install-key(PTK) ← Install-key(PTK)

Clear PTK ← Install PTK
Install-key(PTK)

Clear PTK

Msg3(r+2; GTK)

Msg4(r+2)

Install-key(zeros)

Install all-zero PTK

Install PTK

Msg3(r+2; GTK)

Enc$_{ptk}^1$ {Msg4(r+2)}
Now trivial to intercept and manipulate client traffic
Is your devices affected?

github.com/vanhoefm/krackattacks-scripts

› Tests clients and APs
› Works on Kali Linux

Remember to:
› Disable hardware encryption
› Use a supported Wi-Fi dongle!
Countermeasures

Many clients won’t get updates…

AP can prevent (most) attacks on clients!
› Don’t retransmit message 3/4
› Don’t retransmit group message 1/2

However:
› Impact on reliability unclear
› Clients still vulnerable when connected to unmodified APs
Overview

Key reinstalls in 4-way handshake

Practical impact

Channel validation

Misconceptions

FACTS MYTHS
Misconceptions I

Updating only the client or AP is sufficient
› Both vulnerable clients & vulnerable APs must apply patches

Need to be close to network and victim
› Can use special antenna from afar

No useful data is transmitted after handshake
› Trigger new handshakes during TCP connection
Misconceptions III

Obtaining channel-based MitM is hard
› Can use channel switch announcements

Using (AES-)CCMP mitigates the attack
› Still allows decryption & replay of frames

Enterprise networks (802.1x) aren’t affected
› Also use 4-way handshake & are affected

Image from “KRACK: Your Wi-Fi is no longer secure” by Kaspersky
Overview

Key re installs in 4-way handshake

Practical impact

Misconceptions

Channel validation
Background: new attacks require MitM

Traffic Analysis
- **Capture all** encrypted frames
- **Block** certain encrypted frames

Attacking broadcast WPA-TKIP
- **Block** MIC failures
- **Modify** encrypted frames
Background: new attacks require MitM

Exploit implementation bugs
› **Block** certain handshake messages
› E.g. bugs in 4-way handshake

Other attack scenarios
› See WiSec’18 paper [VBDOP18]
› E.g. **modify** advertised capabilities
Threat model & defense

- Attacker manipulates channel and bandwidth
- No low-layer attacks (e.g. beamforming)
- No relay attacks (e.g. AP and client out of range)

Want to make attacks harder, not impossible
≈ stack canaries.

Solution: verify operating channel when connecting
### Verify Operating Channel Information (OCI)

**Operating Channel Information (OCI) element:**

<table>
<thead>
<tr>
<th>Operating class</th>
<th>Channel number</th>
<th>Segment index 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defines regulatory domain &amp; bandwidth</td>
<td>Defines secondary channel for 80+80 MHz networks</td>
<td>Defines primary channel</td>
</tr>
</tbody>
</table>
Problem: Channel Switch Announcements (CSAs)

Unauthenticated CSAs
› Need to verify securely

Authenticated CSAs
› May not arrive → verify reception!

Solution: authenticate CSA using SA query
Limitations

Other (partial) MitM attacks still possible:
› Adversary can act as repeater
› Physical-layer tricks (e.g. beamforming)

So why use this defense?
› Remaining attacks are harder & not always possible
› Straightforward implementation
Standardization & implementation

Will be part of the new 802.11 standard 😊

<table>
<thead>
<tr>
<th>March 2018</th>
<th>doc.: IEEE 802.11-17/1807r10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IEEE P802.11</td>
</tr>
<tr>
<td></td>
<td>Wireless LANs</td>
</tr>
<tr>
<td></td>
<td>Defense against multi-channel MITM attacks via Operating</td>
</tr>
<tr>
<td></td>
<td>Channel Validation</td>
</tr>
</tbody>
</table>

PoC: github.com/vanhoefm/hostap-channel-validation
Conclusion

› Flaw is in WPA2 standard

› Proven correct but is insecure!

› Update all clients & check Aps

› New defense: Channel Validation
Thank you!

Questions?

krackattacks.com