Attacking WPA3: New Vulnerabilities & Exploit Framework

Prof. Mathy Vanhoef
KU Leuven University (Belgium)

August 25, 2022, Singapore

Who am I?

- Prof. at KU Leuven (Belgium)
- Research: security of the whole network stack
- Hacker at heart! I try to bridge theory & practice

Noteworthy research:

- Key reinstallation attacks (KRACK)
- FragAttack against all Wi-Fi networks
- RC4 NOMORE against HTTPS

Early 2000 Wi-Fi Protected Access (WPA1/2)

Affected by various design flaws:

- Vulnerable to offline **dictionary attacks**: a captured handshake can be used to brute-force the password
- **No forward secrecy**: can decrypt previously captured traffic after learning the password
- **Unprotected management frames**: can spoof beacons, deauthentication frames, & other non-data frames

2017 Key reinstallation attacks (KRACK)

Practical impact:

- **Decrypt frames** sent by a vulnerable device
- **Replay frames** towards a vulnerable device

- Flaw in the standard → all devices affected
- Motivated standard bodies to improve Wi-Fi security

2018 Wi-Fi Protected Access 3 (WPA3)

1. Simultaneous Authentication of Equals (**SAE**) handshake
   - Also called the **Dragonfly** handshake

Provides mutual authentication

Negotiates session key

Forward secrecy & prevents offline dictionary attacks

Protects against router compromise

2018 Wi-Fi Protected Access 3 (WPA3)

1. Simultaneous Authentication of Equals (**SAE**) handshake
   - Also called the **Dragonfly** handshake

2. Usage of Management Frame Protection (**MFP**)
   - Protection of deauthentication and disassociation frames to **prevent denial-of-service attacks**
   - Protection of “robust” action frames (frames used to manage the connection)

2019 **Dragonblood** attacks against WPA3

Main flaw is a **side-channel leak** in the Dragonfly handshake

- Time it takes for the AP to respond to (partial) handshake leaks info about the password
- Leaked info enables **offline brute-force attack**
- Attack against Raspberry Pi **feasible in practice**!

Late 2020 Hotspots: SAE Public Key (**SAE-PK**)

Protocol to **secure hotspots** using a password

Context: **WPA2 is inadequate** since an attacker can clone the network based on the shared password

Protect hotspot using a **pre-shared password**

Idea: derive password from the network's public key

Agenda

HITBSecConf 2022 Singapore

- Wi-Fi history and WPA3
- **Management Frame Protection**
  - New “0-day” disconnection attacks[1,2]
  - Framework to implement & perform attacks

1. M. Vanhoef, P. Adhikari, and C. Pöpper. Protecting Wi-Fi Beacons from Outsider Forgeries. In *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2020.
2. D. Schepers, A. Ranganathan, M. Vanhoef. On the Robustness of Wi-Fi Deauthentication Countermeasures. In *15th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2022.

Types of frames in Wi-Fi

Background: in Wi-Fi there are three types of frames:

1. **Management** frames: scanning for networks, disconnecting, advanced sleep mode, etc.
2. **Control** frames: acknowledgements, request to send, etc.
3. **Data** frames: transporting higher-layer data

WPA2 didn't require Management Frame Protection (MFP)

- With WPA3 this is now mandatory!

Management Frame Protection (MFP)

Provides confidentiality, integrity, and replay protection for:

- **Deauthentication** frames
- **Disassociation** frames

Prevent Denial-of-Service attacks

- **Robust action** frames

Management frame protection has exisisted since 2009 (!!)

- Defined in 802.11w amendment and **optional under WPA2**
- Rarely used because supported was low & often buggy

How secure is MFP?

1. How secure is the MFP standard?
2. How secure are implementations?

Security of the standard:

› **Manual analysis**: are all frames protected? Are the rules complete, consistent, and secure?

Security of implementations:

› **Code inspection & tests**: looking for disconnection attacks

Analysis of standard (part one)

Several management frames are still left unprotected:

- **ATIM frames**: power management in ad hoc networks
- **Timing Advertisement frames**: used in vehicular networks
- **Beacons**: announces a network and its properties
  - Used by all Wi-Fi networks, including hidden ones
  - There's a recent beacon protection defense, but this defense is optional for WPA3 networks

Abusing beacons [VAP20]

Beacons contain the maximum allowed transmit power

- Abuse to **lower transmission power of victims**

Cisco extension to limit transmission power of clients

- Linux: can be abused to **forcibly disconnect a victim** by setting a negative transmission limit

Abusing beacons: part two [VAP20]

They contain transmission back-off parameters (CSMA/CA)

- Abuse to **lower the bandwith of clients**

4:30 PM Fri 3 Jul 81%

Speedtest by Ookla - Th ×

https://www.speedtest.net/result/970175464

https://www.speedtest.net/result/970175464
https://www.speedtest.net/result/970175464

https://www.speedtest.net/result/970175464

Speedtest by Ookla - The Global Broadband Speed Test
https://www.speedtest.net/result/9701754644

Connections

HOW LIKELY IS IT THAT YOU WOULD RECOMMEND

: - / .com

tab q w e r t y u i o p delete

caps lock a s d f g h j k l go

shift z x c v b n m ! ? shift

.?123 .?123

Other beacon attacks [VAP20]

Partial **machine-in-the-middle** position

› Bypasses channel operating validation in Linux

Battery depletion attacks

› Spoof beacons to **make clients stay awake**

→ Solution: use beacon protection!

Source: M. Vanhoef, P. Adhikari, and C. Pöpper. Protecting Wi-Fi Beacons from Outsider Forgeries. In *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2020.

Analysis of standard: part two [SRV22]

- There's ~10 main rules regarding MFP
- Rules rules are complex and conditional on:
  - Whether the client/AP is capable of MFP
  - Whether they advertised support
  - Whether keys are negotiated

Example: "A STA with **dot11RSNAProtectedManagementFramesActivated equal to true** and **dot11RSNAUnprotectedManagementFramesAllowed equal to true** shall transmit and receive unprotected individually addressed robust Management frames to and from any associated **STA that advertised MFPC = 0** and shall discard protected individually addressed robust Management frames received from any associated STA **that advertised MFPC = 0**."

Analysis of standard: part two [SRV22]

We discovered that this complexity leads to:

- Contradictory rules
  - For instance, **handling unprotected Deauthentication frames**: Some rules say to drop them, some say to start a protected SA Query
- Undefined scenarios
  - For instance, how a client should **handle broadcast management frames** when the AP doesn't support MFP

→ May lead to DoS vulnerabilities in implementations

2nd issue: insecure rules in the standard

"The receiver **shall process unprotected** individually addressed Disassociation and Deauthentication frames before the PTK and IGTK are installed."

- Adversary can **cause the handshake to fail** by spoofing Disassociation or Deauthentication frames

- **Defense: start a timer instead of disconnecting**
- Stop the timer if the handshake progresses
- Only disconnect when the timer expires

Addressing issues in the standard

Long-term fix: require MFP so the rules become simpler

- Avoids all the conditional statements in the rules

Short-term fix: we proposed updates to simplify the standard:

- Presented at TGm meetings of the IEEE 802.11
- Currently still being discussed by the IEEE

MFP security part two: implementations

We also **audited implementations** for **disconnection attacks**

- Goal: make the victim to instantly disconnect & reconnect
- This facilitates attacks that target the connection process
  - For instance, KRACK, FragAttacks, Dragonblood, etc.

MFP security part two: implementatios

Methodology:

1. Inspect **open-source** implementations (e.g., Linux, *BSD)
2. Look for code paths that lead to **disconnection calls**
3. Can these be triggered by spoofing plaintext frames?
4. Also **test** confirmed attacks against closed source platforms

Beacon with invalid bandwidth

- Beacons include the network channel & bandwidth
  - E.g.: primary channel 60 & secondary channel 64 (= 40MHz bandwidth)
- Spoof beacon with **invalid channel/bandwidth combination**
  - Primary channel 1 with secondary channel below the primary:

```
 Primary Channel: 1
˅HT Information Subset (1 of 3): 0x07
   .... ..11 = Secondary channel offset: Secondary channel is below the primary
   .... .1.. = Supported channel width: Channel of any width supported
```

  - This combination is disallowed in most regulatory environments
  - **Linux and Windows disconnect** when receiving the spoofed beacon.

Channel Switch Announcement

Beacon can contain a Channel Switch Announcement (CSA)

› Used when the AP changes channel due to radar detection

```
⌄ Tag: Channel Switch Announcement Mode: 1, Number: 11 , Count: 1
    Tag Number: Channel Switch Announcement (37)
    Tag length: 3
    Channel Switch Mode: 1
    New Channel Number: 11
    Channel Switch Count: 1
```

Abuse this to trick clients into switching to another channel

› Clients **will disconnect** since the AP isn't on the new channel

› Many clients vulnerable: **Linux, macOS, iOS, Windows**, etc.

Crashing an Access Point

Flooding many SAE (Dragonfly) handshake messages **crashes** the D-Link DIR-X1860

- Out-of-memory issue in the driver leading to a NULL pointer dereference

- All routers with this driver are likely affected
- Bug is in WPA3's useless anti-clogging defense…
  - …which was shown to be trivial to bypass in the Dragonblood attacks
- Avoid **useless defenses**, they **increase the attack surface**

Other attacks

Also found several other implementation vulnerabilities:

- Spoofing **plaintext handshake** messages
  - Accepted on Linux. Can abuse to disconnect client.
  - Two types: 4-way handshake and 802.1X handshake messages
- **Packet counter for group key** is not set after connecting
  - Abuse to replay management frames, e.g., Deauth frames

Testing & exploit framework

Created a framework easily to **test for all vulnerabilities**[1]

- Build on top of Linux's hostap daemon using Python

Allows for easy reuse of functionality of hostap and Linux:

- When targeting an AP: to **scan** for the target network
- When targeting a client: to periodically **transmit beacons**
- **Retransmits** injected frames if not acknowledged
- **Queues** injected frames until the client wakes up

1. D. Schepers, M. Vanhoef, A. Ranganathan. DEMO: A Framework to Test and Fuzz Wi-Fi Devices. In *WiSec*, 2021.

Framework: example test case

HITBSecConf 2022 Singapore

```python
emonstration(Test):
  """Simplified demonstration presenting the basic test case structure."""
  name = "example-demo"
  kind = Test.Supplicant

  def __init__(self):
        """Initialization of the sequential actions defining the test case."""
        super().__init__([
              # Once authenticated, we will inject a frame.
              Action( trigger=Trigger.AfterAuth, action=Action.Inject ),
              # Once connected, we will call a user-defined function.
              Action( trigger=Trigger.Connected, action=Action.Function )
        ])

  def ping(self, station):
        """User-defined function, giving us run-time access to the station."""
        # For example, we will send a command over the control interface.
        response = station.wpaspy_command("PING")
        if "PONG" in response:
              log(STATUS,'Received a PONG from the control interface.')

  def generate(self, station):
        """Generate the test case by configuring the defined actions."""

        # Create a Dot11-frame with dummy payload.
        frame = station.get_header() # Returns Dot11()-header.
        frame /= LLC()/SNAP()/Raw(b'A'*32)

        # Load the frame into our action, and let us sent it encrypted.
        self.actions[0].set_frame( frame , mon=True , encrypt=True )

        # Load our user-defined function, PING'ing the control interface.
        # (Optional) Automatically terminate the test case after our action.
        self.actions[1].set_function( self.ping )
        self.actions[1].set_terminate( delay=2 )
```

- Define triggers and actions
  - E.g., when connected inject a frame
- Custom functions and actions
  - E.g., monitoring raw Wi-Fi frames
- Generation function to specify details of each trigger/action
  - E.g., whether to encrypt injected frame

Framework extras

- Has a **library** for common Wi-Fi tasks (e.g., crypto functions)
  - And the framework also uses the Scapy library
- Supports **simulated Linux interfaces** (mac80211_hwsim)
  - Perform Linux experiments without Wi-Fi hardware ☺
- Built by relying on virtual interfaces
  - Network card acts as client/AP & simultaneously has monitor mode
  - Monitor mode: receiving and injecting raw Wi-Fi frames.

Test case vs. exploit case

The core goal of the framework is **tests and audits**

- Several triggers in a test case assume the password is known
- E.g., triggers when (dis)connected to / from the network

But can construct **exploit cases** by avoiding such triggers ☺

- E.g., by replacing "connected" trigger with "about to connect"
- Preliminary support available & will be further extended

→ github.com/domienschepers/wifi-framework

Agenda

HITBSecConf 2022 Singapore

- Wi-Fi history and WPA3
- Management Frame Protection
- **The SAE-PK “Hotspot” Protocol**

Introduction to SAE-PK

Goal of SAE Public Key:

- Authenticate a Wi-Fi hotspot using a password…
- …but prevent an adversary from cloning the network

→ Accomplished by using asymmetric crypto

High-level overview of SAE-PK

Based on public key crypto:

1. The Access Point (AP) generates a public/private key pair
2. The Wi-Fi password is derived from the public key
3. The public key is sent to the client when connecting
4. Clients use the password to verify the public key of the AP
5. AP proves possession of the corresponding private key

→ The **password forms a signature** of the public key

The SAE-PK password

The SAE-PK password is the truncated output of:

Hash(SSID || Modifier M || public key)

- SSID (Service Set Identifier): name of the Wi-Fi network
- Public key: point on an elliptic curve
- Modifier M: starts from a random value and is incremented until the output starts with 3 or 5 zero bytes.
  - Number of required zero bytes is controlled by a security parameter

The SAE-PK password

The SAE-PK password is the truncated output of:

Hash(SSID || Modifier M || public key)

Output is converted into a human-readable form

- Example password: `2udb-slxf-3ijn-dbu3`
- Password length is variable and decided by administrator
- Shortest allowed password length encodes 52 bits of the hash output (excluding the leading 3 or 5 zero bytes)

Attack: creating a clone of the network?

Find a modifier M & public key that result in the same password

- What is the complexity of this in the best case?

Hash(SSID || Modifier M || public key)

- Hash output must start with at least 3 zero bytes → 2^{24}
- Remaining output must equal the password → 2^{52}

Total time complexity of 2^{76} to perform a naïve attack

Time-memory trade-off attack

An attacker is essentially inverting the hash function:

- We can construct **rainbow tables** to optimize the attack.
- Every table only will work **against a specific network name**.
  - The SSID acts is a seed
- On verge of practicality based on theoretical estimates:
  - A table of ~6TB can break a password in ~2 weeks on AWS

→ Mitigate attacks using a **long password** or by making the truncated output start with at least **5 zero bytes**.

Network-based attacks

Security goal of SAE-PK:

- Prevent an adversary from cloning the AP
- = preventing an adversary from intercepting client traffic
- This protection occurs at the link layer

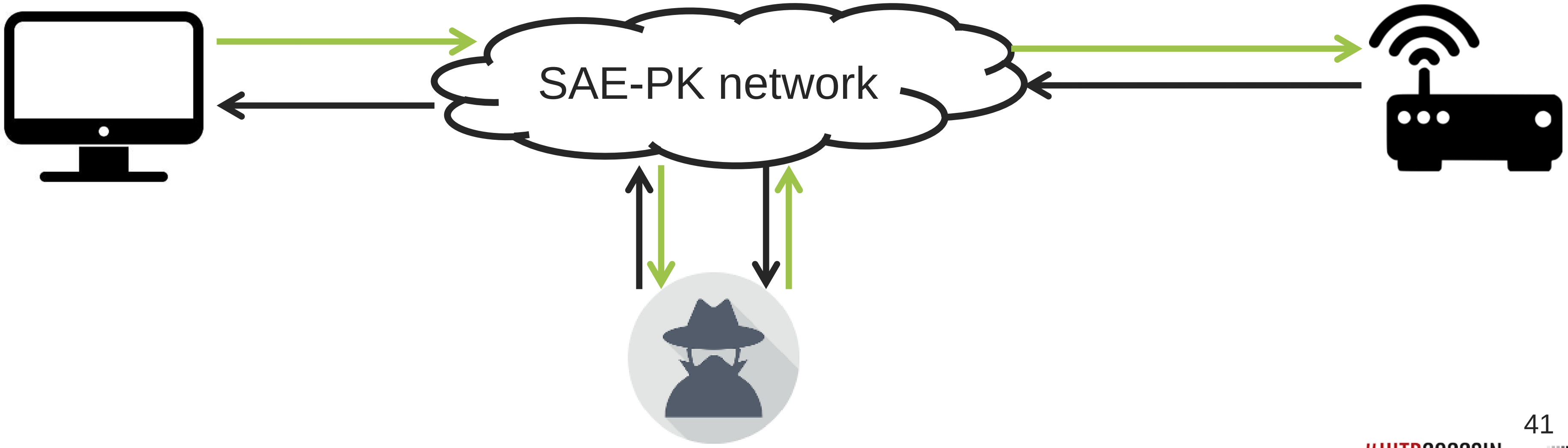
But we can still **intercept traffic at the network layer**!

1. Using ARP poisoning (a well-known attack method)
2. By abusing the shared Wi-Fi group key to inject packets

1st attack: ARP poisoning

Straightforward ARP poisoning attack:

› Make the victim client believe you are the gateway
› Make the gateway believe you are the victim



2nd attack: abusing the shared group key

Background:

- WPA1/2/3 encrypts broadcast traffic using a symmetric key
- All clients receive this key from the AP when connecting

Abuse of the group key

- Every client possess the group key
- This means every client and spoof broadcast traffic

Injecting unicast packets?

› Put unicast IP packet in a broadcast Wi-Fi frame?

Flags	Receiver			
to client	FF:⋯:FF	Source IP	Destination IP	Data

802.11 specific

This is similar to the “Hole 196” attack

› But now in a new threat model: hotspot networks

› Do devices nowadays (still) accept unicast IP packets in broadcast Wi-Fi frames?

Injecting unicast packets: experiments

We tested various clients. All the following were vulnerable:

- Windows 10
- Huawei Y6 Prime
- iPad
- Android Nexus 5X
- Linux on all recent kernels

Only one device wasn't vulnerable: Android Pixel 4XL

Sending broadcast frames to the AP?

HITBSecConf 2022 Singapore

Will **an AP** process frames with a broadcast receiver address?[1]

› Normally, an AP only transmits broadcast frames

› What if we set the "To AP" flag in the header?

Nearly all APs ignore frames with a broadcast receiver address

› Only the Asus RT-AC51U AP processes broadcast frames.

1. M. Vanhoef and F. Piessens. Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys. In USENIX Security Symposium, 2016.

Preventing abuse of the group key

Drop unicast IP packets in broadcast link-layer frames

More generally, in an SAE-PK hotspot:

1. The AP should give each client a **random group key** when it is connecting (see Passpoint hotspots)
2. Clients should **drop (most) broadcast packets**

→ In other words, *if possible*, disable broadcast traffic.

Recap: framework exploit / test cases

Disconnection despite MFP exploit/tests:

- Channel switch: `dos-beacon-csa`
- Invalid bandwidth: `dos-beacon-bandwidth`
- SAE flooding: `dos-sae-flood`
- Extra CVE in hostap: `example-pmf-deauth`

SAE-PK network-layer exploit/tests:

- Abuse group key against client: `group-hole196`
- Abuse group key against AP: `group-tods`

Thank you!

- WPA3 still affected by disconnection attacks
- Traffic in SAE-PK networks may still be intercepted using network-based attacks
- Check out our test/exploit framework!

Framework is on GitHub:

github.com/domienschepers/wifi-framework