UNIVERSITÀ
DI TRENTO

KRACK Attack

Network Security Laboratory Report

Academic year 2023/2024

Matteo Beltrami, Christian Sassi, Luca Pedercini

Submission date: 15/05/2024

Contents

1 Introduction

KRACK (**K**ey **R**einstallation **Attack**)[1] [2] is an attack that exploits a vulnerability in the Wi-Fi Protected Access 2 (WPA2) [3], a security protocol that guarantees the security of Wi-Fi networks. This attack allows cybercriminals to access encrypted data when in physical proximity to their victim. It was discovered for the first time by Mathy Vanhoef in 2017 and upset the perception of security related to WPA, revealing a widespread vulnerability that involved devices with different operating systems.

The KRACK Attack is performed through the attacker's manipulation of the messages exchanged during the handshake procedure. Handshakes are mutual authentication protocols which, through an exchange of messages between the client and access point, serve to establish a temporary key for encryption of communication.

However, KRACK Attack exploits a vulnerability whereby an attacker can create a cloned network that tricks the victim's device into reinstalling an already-in-use key, gaining the ability to decrypt encrypted data and access sensitive information transmitted through the compromised network. Handshakes vulnerable to this type of attack are: 4-way handshake, Fast BSS Transition handshake, Group Key handshake and PeerKey handshake.

2 WPA2

2.1 Introduction

The WPA2 (Wi-Fi Protected Access 2) protocol is a security standard for Wi-Fi networks designed to protect wireless data transmission [4]. Using advanced cryptographic algorithms such as AES (Advanced Encryption Standard), WPA2 ensures a secure connection between devices and the Wi-Fi access point. This increase in cryptographic strength makes it much more difficult for attackers to breach the security of a Wi-Fi network and gain unauthorized access to transmitted data. WPA2 performs several crucial functions. First of all, it authenticates devices trying to connect to the Wi-Fi network through a cryptographic key exchange process. It also establishes an encrypted channel for data transmission, preventing potential intruders from intercepting or manipulating information exchanged between devices and the access point.

2.2 Handshake

The handshake process within the WPA2 protocol is a critical element in ensuring the security of wireless connections [5]. In general, the handshake process involves exchanging information between a client device and a Wi-Fi access point in order to establish a secure connection and authenticate both parties. In particular, the 4-way handshake of the WPA2 protocol is a cryptographic procedure that occurs when a client device tries to authenticate and connect to a Wi-Fi access point.
This process involves four messages exchanged between the client and the access point and is intended to verify the authenticity of the client's credentials and establish a cryptographically secure session key for subsequent communication.

2.2.1 Initial Phase (also common to WPA)

- **Client Authentication Request**: During this phase, the client sends a connection request to the access point (router) to join the network.
- **Access Point Authentication Response**: The router responds to the client by specifying all the parameters necessary for authentication.

2.2.2 4-way handshake

- **Message 1 - Client Authentication and Key Exchange**: The access point generates the *Anonce* value and sends message one to the client, which includes the *Anonce* (a random number) and the value of a counter (used to prevent replay attacks). This message is unencrypted, has no integrity check and if someone messes with this message, the handshake will fail and that's it. The client receives message one and derives the Pairwise Transient Key (PTK). The client already had the Pairwise Master Key (PMK), the *Snonce*, the client MAC address and received the *Anonce* and the access point MAC address.
- **Message 2 - Access Point Authentication and Key Exchange**: The client responds to the access point by sending it the counter value, its *Snonce* and a Message Integrity Code (MIC) to ensure the integrity of the message. The access point, in turn, calculates the PTK and compares the received MIC to the calculated MIC.
- **Message 3 - Deriving Session Keys**: The access point sends the updated counter, and other parameters necessary for calculating the session keys. It sends the key installation request, the MIC and the Group Transient Key (GTK) for broadcast transmissions.

- **Message 4 - Confirmation of Key Establishment**: The client compares the received *MIC* with its own *MIC* for match, installs the *PTK* and *GTK*, and sends an *ACK* that completes the handshake. It also sends an EAPOL-key to confirm the installation of the keys.

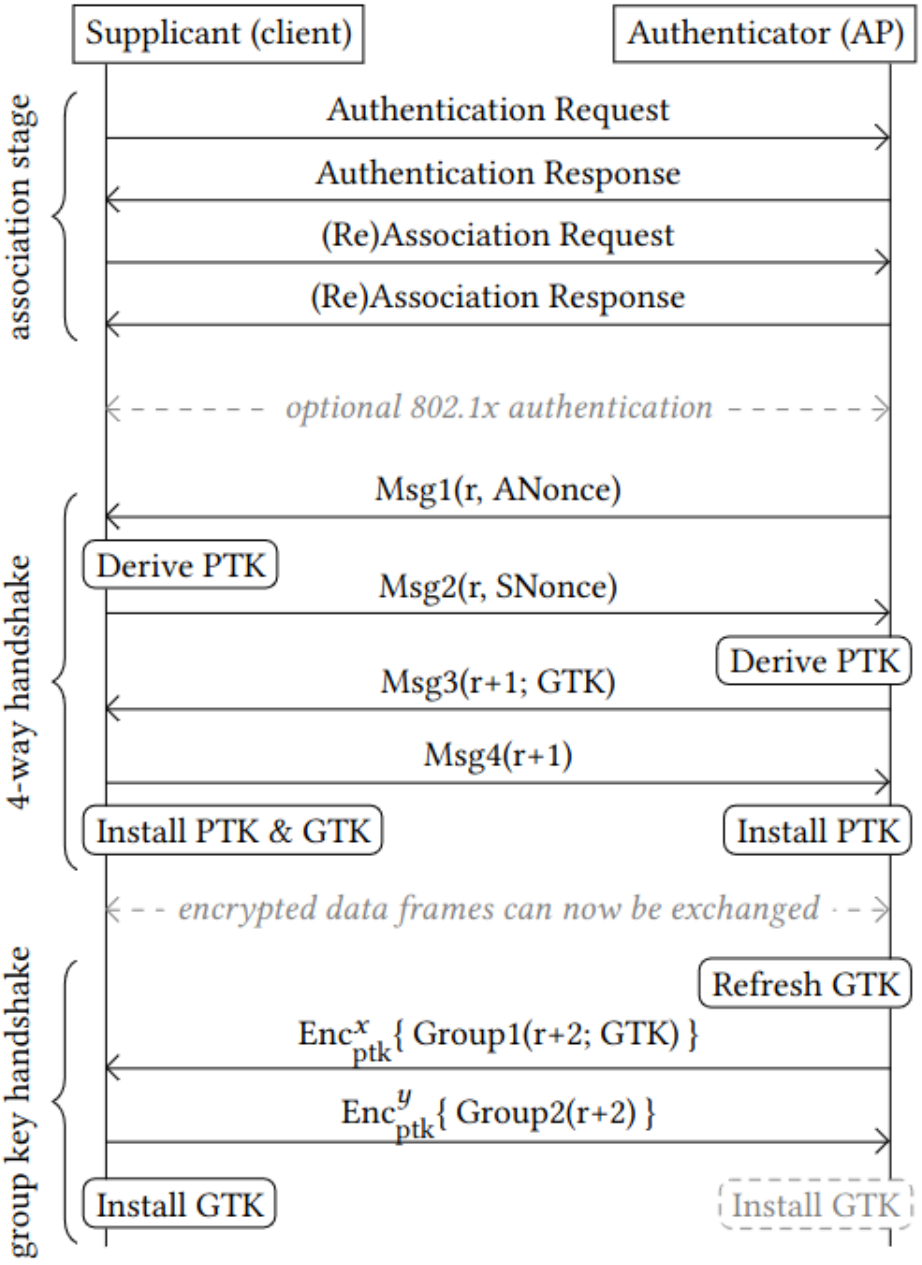


Figure 1: WPA2 4-way handshake

3 KRACK Attack

3.0.1 Introduction

The KRACK Attack was first discovered in 2017 by Mathy Vanhoef, a Belgian security researcher. This attack targets WPA2, the security protocol widely used in today's wireless networks. Specifically, the vulnerability exploits a weakness identified by researchers in the 4-way handshake used by the protocol. If exploited, an attacker gains the ability to decrypt all packages exchanged between two endpoints, such as a client and an access point. However, this only applies if both endpoints are vulnerable; otherwise, the attacker can only decrypt packages transmitted by the vulnerable endpoint. Vulnerable devices include access points, as well as Android and Linux devices, depending on the protocol or service used to manage WPA2. For instance, access points implementing the IEEE 802.11r protocol, Fast BSS Transition (FT), are vulnerable. Similarly, devices using specific versions of wpa_supplicant (up to 2.5) are considered vulnerable. However, this vulnerability has been patched, making it unlikely to encounter a device still affected by it.

3.0.2 How it works

Now, let's delve into this attack from the client's perspective, focusing on its actions during the 4-way handshake, which can be better understood by examining its state machine (Figure 2).

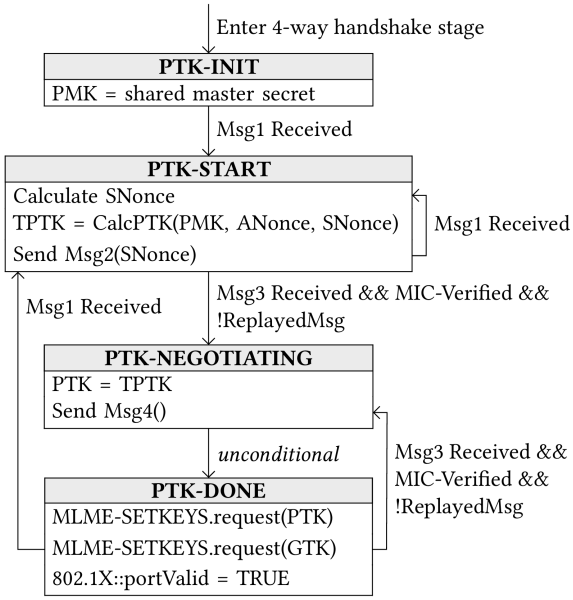


Figure 2: Client's state machine during the 4-way handshake

During the *PTK-NEGOTIATING* state, the client receives *Msg3* from the access point and installs the pairwise key (first time). Then, it finally goes to the *PTK-DONE* state. However, if an attacker manages to block the client's *Msg4*, causing the access point to resend the packet, the client in the *PTK-DONE* state will go back to the previous *PTK-NEGOTIATING* state and reinstall the key, regardless of whether it has done so before. Consequently, this resets the packet number causing a reuse of the nonces. With predictable nonces, an attacker who also knows the encryption key can decrypt packets sent by the client to the access point.

4 Laboratories

The choice of using Ubuntu 16.04.* is because the exploits and simulations created for this vulnerability are now outdated and rely on old versions of the used dependencies. Given that, the affected services are nearly all patched at this point and it is quite challenging to recreate this scenario in an environment with updated dependencies.
All the proposed laboratories can be easily accessed by using the `launcher.sh` provided in the laboratory folder, as all necessary setups are already in place.
However, if you wish to do a fresh new installation of the lab environment, just execute the `vm_setup.sh` script inside the `scripts` folder of the GitHub project [6].
The main operations performed by the setup are:

- Installation of the official krackattacks-script repo [7].
- Installation of mininet-wifi repository [8].
- Installation of all the required dependencies (python, scapy, pycryptodome ecc...).
- Download of the source files of wpa_supplicant version 2.5 and 2.10 (from [9])

For further information, refer to the `README` file in the project repository [6].
Overall, the main idea is to keep things as simple as possible and allow the user to perform each laboratory autonomously without problems so that he can focus on the core concepts.

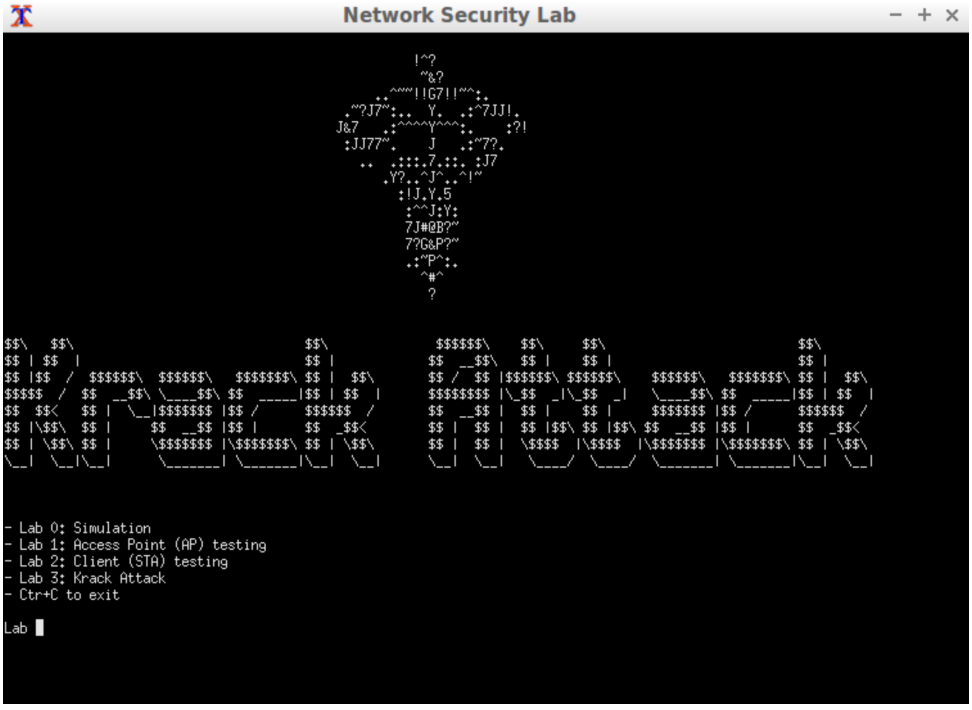


Figure 3: Menu of `launcher.sh`. From here you can access the desired laboratories

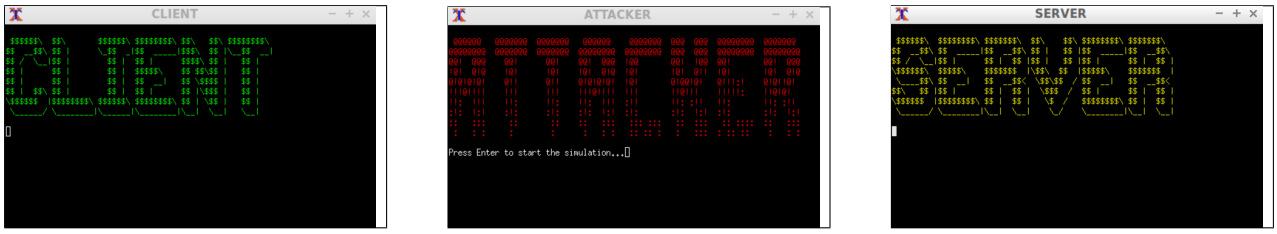
5 Laboratory 0: Simulation

5.1 Introduction

This laboratory will simulate a KRACK Attack from a theoretical point of view. Specifically, it is possible to interactively replicate step-by-step the 4-way handshake of the WPA2 protocol between a client and an access point (here labeled as *server*). However, there is an attacker that interferes with the normal handshake with the intent of exploiting the vulnerability.

5.2 Hands-on

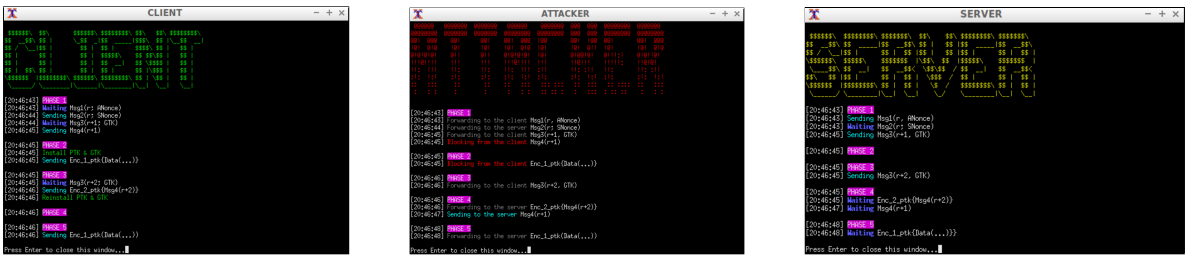
By typing `0` and then `Enter` on the menu, it is possible to execute this lab. These three terminal windows are representing the client, the attacker and the access point:



(a) Client (b) Attacker (c) Server

Figure 4: Initial view

By pressing `Enter` on the attacker terminal (which behaves as the master for this script) it is possible to execute each step of the 4-way handshake until reaching the end.



(a) Client (b) Attacker (c) Server

Figure 5: Final view

As it can be seen by looking at the figures above (Figure 5) it is possible to notice how the attacker forwards most of the messages except for `Msg4` (phase 1 of the attacker). By intercepting this particular message, as previously mentioned, the access point assumes `Msg3` was lost and resends it, resulting in the client's pairwise key being reinstalled (phase 3 of the client). Additionally, it is evident that after each key installation, the client resets the packet number, presenting an opportunity for exploitation by an attacker. Of course during this simulation, the reinstallation of the pairwise key happens once but it can be forced by the attacker an indefinite number of times.

6 Laboratory 1: Access Point (AP) testing

6.1 Introduction

This laboratory allows the user to check if an access point (*ap1*) is vulnerable or not by using a client (*sta1*) as the "attacker" that forces the victim to reinstall the pairwise key. For this lab, the scenario is composed by two access points (*ap1* and *ap2*) and a client (*sta1*) (Figure 6). Specifically, *ap2* is vulnerable because it implements the protocol 802.11r, Fast BSS Transition (FT) [10].

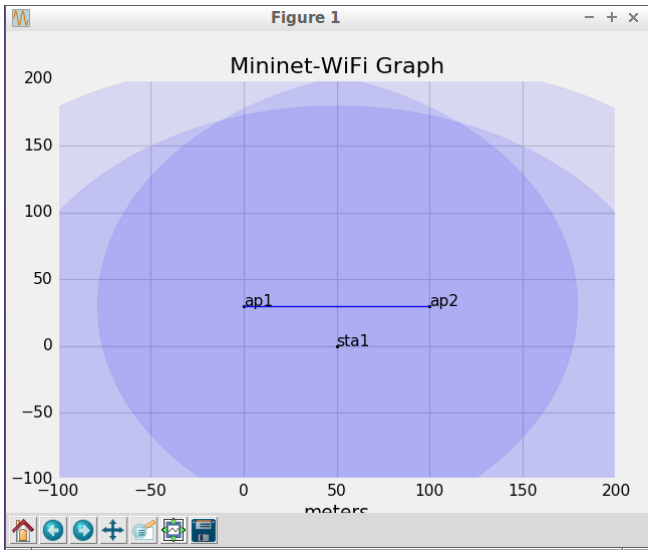


Figure 6: Scenario

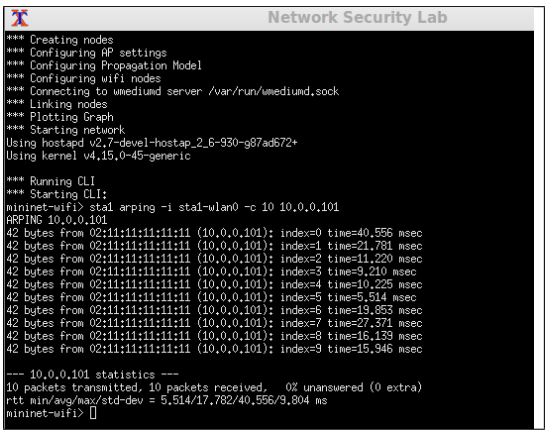
6.2 Hands-on

By typing `1` and then `Enter` on the menu, it is possible to execute this lab. When everything is loaded, the user will see five windows:

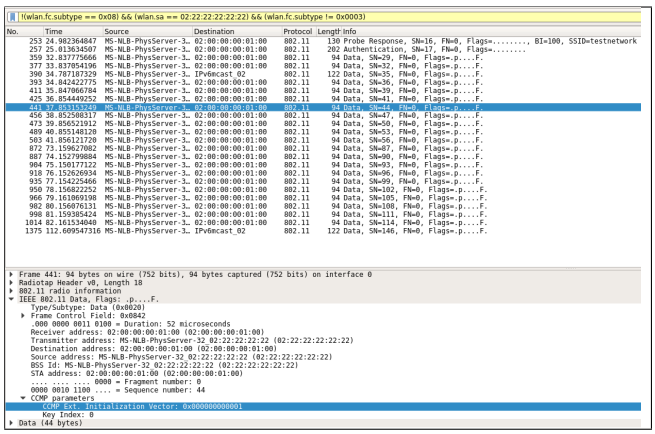
- Figure representing the scenario (Figure 6).
- Wireshark, to capture the transmitted packets.
- Three XTerm terminals.
  - Mininet console.
  - Two terminals to implement the KRACK Attack.

Use Wireshark to capture the packets coming from two interfaces: *sta1-wlan0* and *mon0*.
It is possible to inspect the network from the Mininet console by using commands like `nodes` and `links`. The `nodes` command displays the nodes present in the network, while the `links` command shows the current connections between nodes.
At the beginning the station is connected with *ap1* (MAC address = 02:11:11:11:11:11) and the association process is visible in the *KrackAttack: sta1* terminal. This can be checked by running the `sta1 arping -i sta1-wlan0 -c 10 10.0.0.101` command inside the Mininet terminal.
Additionally, the same thing can be visualized on Wireshark. It is recommended to use the filter `!(wlan.fc.subtype == 0x08)` to filter out beacon frames and keep only those exchanged with the arping command.

Conversely, executing the `sta1 arping -i sta1-wlan0 -c 10 10.0.0.102` command (targeting *ap1*) results in all packets being lost.



(a) AP response



(b) AP response in Wireshark

In the *KrackAttack: sta1* terminal, it can be noted that *ap1* transmits data using different Initialization Vectors (*IV*), suggesting that it is not vulnerable to a KRACK Attack (Figure 7). Additionally, the same result can be observed on Wireshark by inspecting the CCMP parameters of the 802.11 Data field.

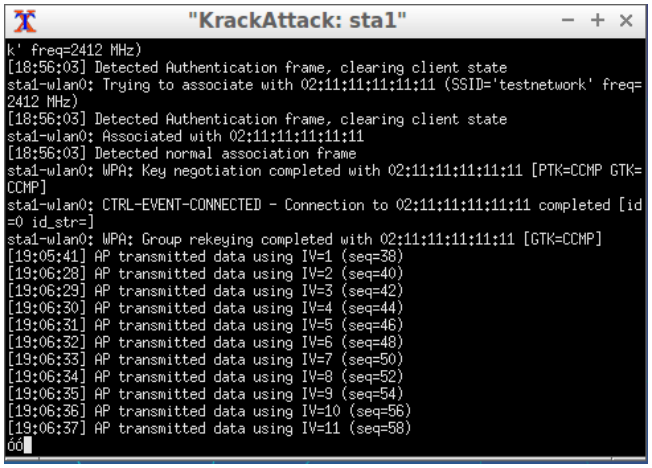


Figure 7: AP responses with incremental *IV* values

Now, it is possible to use these commands in order to connect the client to the vulnerable access point (*ap2*):

```
wpa_cli -i sta1-wlan0 #open WPA CLI on the station's interface
scan #scan the available access points
scan_results #inspect scan results
roam 02:22:22:22:22:22 #connect with AP2
```

Restart live capturing in Wireshark but this time it is necessary to adjust the filters in order to better understand what is happening: `!(wlan.fc.subtype == 0x08) && (wlan.sa == 02:22:22:22:22:22) && (wlan.fc.subtype != 0x0003)`. With this configuration, Wireshark will discard beacon frames (`0x08` subtype), reassociation responses (`0x0003` subtype), and keep only *ap2* packets.
Finally, start transmitting data using the same command as before `sta1 arping -i sta1-wlan0 -c 10 10.0.0.102` (now targeting *ap2*). This time, the client will be able to reinstall the pairwise key

on *ap2* as it can be seen by looking at the *KrackAttack: sta1* terminal (Figure 8). Additionally, the same result can be observed on Wireshark by inspecting the CCMP parameters of the 802.11 Data field.

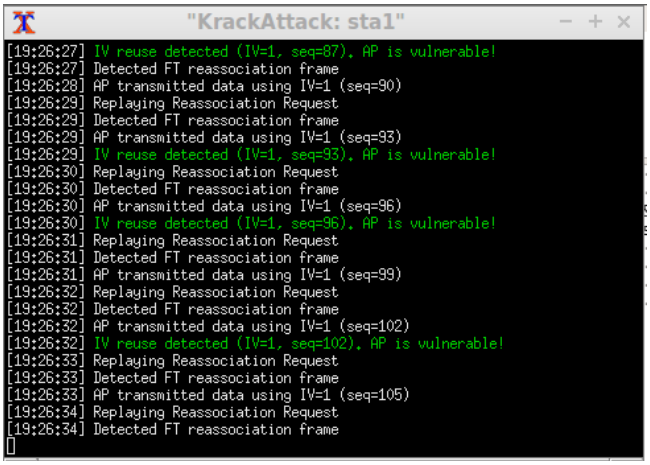


Figure 8: IV reuse in vulnerable AP responses

7 Laboratory 2: Client (STA) testing

7.1 Introduction

This laboratory allows the user to check if a client (*sta1*) is vulnerable or not by using an access point (*ap1*) as the "attacker" that forces the victim to reinstall the pairwise key. For this lab, the scenario is composed only by these two nodes (Figure 9).

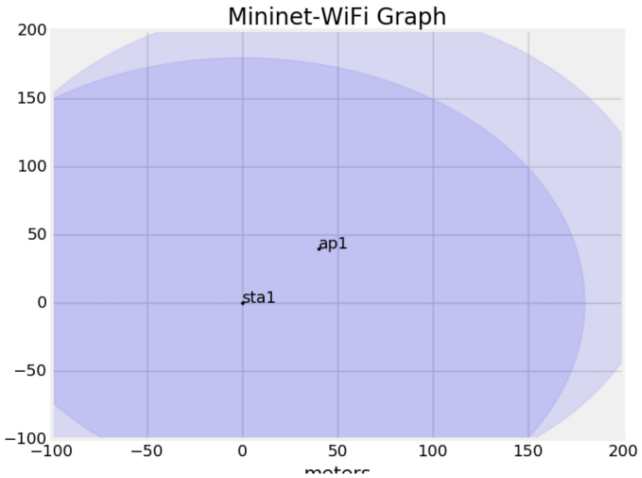


Figure 9: Scenario

7.2 Hands-on

By typing `2` and then `Enter` on the menu, it is possible to execute this lab. When everything is loaded, the user will see five windows:

- Figure representing the scenario (Figure 9).
- Wireshark, to capture the transmitted packets.
- Three XTerm terminals:
  - Mininet console.
  - Two terminals to implement the KRACK Attack.

In addition to the graph, the current network structure can be viewed by executing the `nodes` and `links` commands within the Mininet terminal. Initially, there is no connection between *sta1* and *ap1* confirmed by running the command `sta1 arping -i sta1-wlan0 -c 10 192.168.100.256` in the Mininet console. Since the nodes are not connected, all sent packets will return timeout.

7.2.1 Testing a non-vulnerable client

To establish a connection between *sta1* and *ap1*, use the *Connection: sta1* terminal by executing `wpa_cli sta1-wlan0 -c client_network.conf`. This command prompts the client to connect to the network specified in the *.conf* file using the designated interface. Inside the configuration file there are the SSID and the password of the network. Upon execution, observe in the *AP: ap1* terminal how the access point attempts to force the reinstallation of the pairwise key, ultimately failing. A green message will appear indicating the client is not vulnerable (Figure 10). This is attributed to the current wpa_supplicant version installed on the machine, identified as version 2.10, viewable in the Mininet terminal or by executing the command `wpa_supplicant -v`.

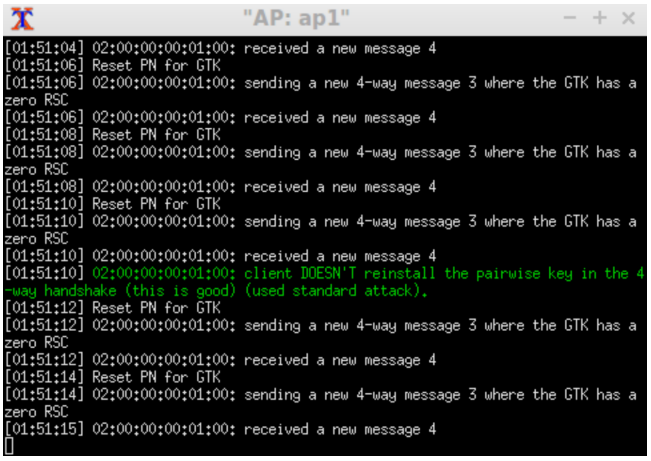


Figure 10: The access point warns that the client is not vulnerable

7.2.2 Testing a vulnerable client

To make the client vulnerable to the attack, exit the lab by typing `exit` in the Mininet terminal. Then, downgrade wpa_supplicant to a vulnerable version such as 2.5. To do this, navigate to the folder containing all the source files of wpa_supplicant v2.5 located at:

`network-security-lab/krackattacks-scripts/hostap-wpa_supplicant-2.5/wpa_supplicant`

Within this directory, install the service by executing `make clean`, then `make`, and finally `sudo make install`. Verify the installation success by running `wpa_supplicant -v`, which should now return v2.5.
Upon restarting lab 2 and ensuring everything is loaded, Mininet will display the newly installed version of wpa_supplicant. Apart from wpa_supplicant, the rest of the lab remains unchanged. This time, executing the same command as before, `wpa_cli sta1-wlan0 -c client_network.conf` in the *Connection: sta1* terminal, confirms that in the *AP: ap1* terminal, the access point identifies the client as vulnerable, as it successfully forces the client to reinstall the pairwise key (Figure 11).

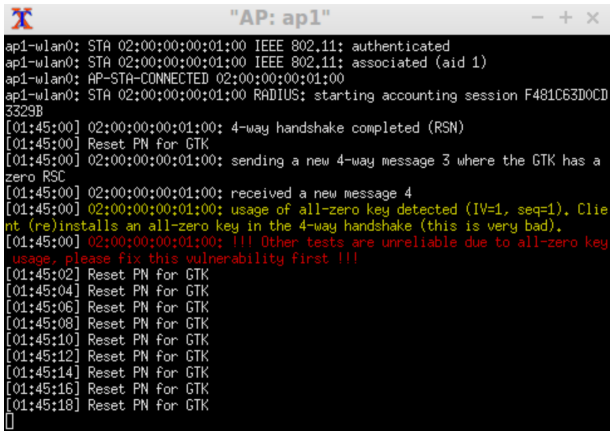


Figure 11: The access point warns that the client is vulnerable

7.2.3 Live test

The third part of lab 2 consisted in testing a real client device (an old Android phone with a vulnerable version of wpa_supplicant).

In order to do so, it is necessary to reconstruct the attack scenario proposed by Vanhoef in his paper [1]. In our case, the scenario was set as following:

- We opted to use a mobile phone as the access point since it is easier to manage. The network was configured with SSID="`testnetwork`" and password="`abcdefgh`".
- We used our laptop to impersonate a Man In The Middle. It executes the same script shown in this lab and used by *ap1* in order to test the vulnerability of the client.
- We use the old mobile phone as the client to connect to the wireless network.

In this scenario, unlike previous setups, three devices are required: the client, the MitM device, and the access point. This is because, while in the Mininet virtual environment we have complete control over the network, in a real-world test case, the attacker has to replicate the existing wireless network emitted by the access point on a different channel.
Finally, as we are operating in a real environment, the client must be positioned far enough from the access point, yet close enough to the MitM device, so it perceives the MitM device's signal as stronger and connects to it.

Figure 12: Android mobile phone used as client (Samsung s5 mini)

8 Laboratory 3: Real attack

In the third laboratory the intention was to implement a real attack that exploited the KRACK vulnerability.
In particular, the idea was to use some scripts developed by Vanhoef himself as proof-of-concept of the vulnerability (Video [11]), and published in the Github repository [12]. A key reinstallation attack is performed against an Android smartphone, and the attacker is able to decrypt all data that the victim transmits.
The attack exploits the fact that some particular Android and Linux devices can be tricked into reinstalling an all-zero encryption key, and therefore make it really simple to decrypt the messages sent.
The attack shows that we can recover data such as login credentials and, in general, any information that the victim transmits. Additionally, it is also possible to decrypt data sent towards the victim, as HTTPS protection could be bypassed in a worrying number of situations.
However, dealing with outdated and not maintained code, made our life harder, and ultimately, we did not manage to actually implement this type of attack. The demonstration was therefore left to the original video by Vanhoef.
The results achieved and the problems encountered are briefly presented below.

8.1 Achieved results and encountered problems

To be able to execute the scripts that implemented the KRACK attack, we tried to use two different approaches.

8.1.1 First solution: real world

The first solution was to run the scripts locally on our PC and carry out the attack in reality, with a vulnerable device, exactly as shown in the video.
In order to create the environment in which to run the scripts, we used a docker container with a Kali Linux image, in which all the various necessary dependencies were installed (Python2.7, Scapy 2.3.3, sslstrip etc..).
To have complete control over the network interfaces of our PC via the docker container, we used distrobox [13], which makes the created container tightly integrated with the host, allowing sharing of the HOME directory of the user, external storage, external USB devices and graphical apps (X11/Wayland), and audio.
The problem with this approach was to find a client device (Android phone) that was actually vulnerable to CVE-2017-13077 [14](the one with the greatest impact) and therefore could be forced to install an all-zero encryption key.
After testing numerous old phones (using the main repo client test script [7]), our search yielded no results.

8.1.2 Second solution: simulate scenario with Mininet

The second possible solution was to try to replicate the environment used by Vanhoef in his video, in the Mininet virtual network, creating a vulnerable client (with wpa_supplicant version 2.5), a MitM that would execute the attack script (with all the necessary dependencies) and an access point to generate the wireless network.
However, we encountered several difficulties that did not allow us to succeed with this approach.

9 Mitigation

Following the KRACK Attack vulnerability discovery, each affected device, including access points and particularly clients, received a crucial security patch which was proposed by the same discoverer of the vulnerability, as described in the official paper [1].
To solve KRACK Attack vulnerabilities, especially those related to the client side (*CVE-2017-13077* [14], *CVE-2017-13078* [15], *CVE-2017-13079* [16], *CVE-2017-13080* [17], *CVE-2017-13081* [18]), it is necessary to ensure that a particular key is installed only once during a handshake execution.
For example, the generated session key in a 4-way handshake should only be installed once. When the client receives a retransmitted *Msg3*, it should reply, but not reinstall the session key.
The idea is to add a boolean variable into the *wpa_supplicant* state machine (Figure 2), initialized as false and set to true when generating a fresh *PTK* in *PTK-START* state.
When entering the *PTK-DONE* state: if the boolean is `true`, the keys are installed and the boolean is set to `False`; if the boolean is `False`, keys are NOT installed.
An example of the "fixed" state machine can be observed in figure 13.

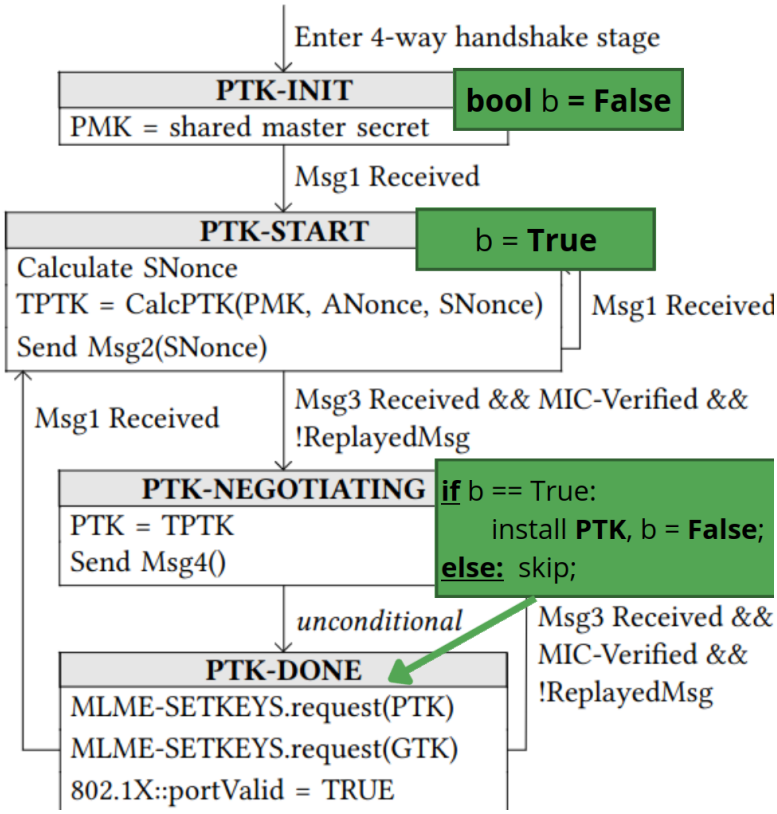


Figure 13: Fixed wpa_supplicant state machine

References

[1] *Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2*. https://papers.mathyvanhoef.com/ccs20

[2] *KRACK Attacks: Breaking WPA2 by forcing nonce reuse in WPA2*. https://www.krackattacks.com/.

[3] *Wi-Fi Protected Access - Wikipedia*. https://it.wikipedia.org/wiki/Wi-Fi_Protected_Access.

[4] *IEEE 802.11i*. https://it.wikipedia.org/wiki/IEEE_802.11i.

[5] *WPA and WPA2 4-Way Handshake*. https://networklessons.com/cisco/ccnp-encor-350-401/wpa-and-wpa2-4-way-handshake.

[6] *christiansassi/network-security-lab*. https://github.com/christiansassi/network-security-lab.

[7] *vanhoefm/krackattacks-scripts*. https://github.com/vanhoefm/krackattacks-scripts.

[8] *intrig-unicamp/mininet-wifi*. https://github.com/intrig-unicamp/mininet-wifi.

[9] *w1.fi git repositories*. https://w1.fi/cgit.

[10] *IEEE 802.11r-2008 - Wikipedia*. https://en.wikipedia.org/wiki/IEEE_802.11r-2008.

[11] *KRACK Attacks: Bypassing WPA2 against Android and Linux - Youtube*. https://youtu.be/Oh4WUF

[12] *vanhoefm/krackattacks-poc-zerokey*. https://github.com/vanhoefm/krackattacks-poc-zerokey/tree/re

[13] *Distrobox*. https://distrobox.it/.

[14] *NVD-CVE-2017-13077*. https://nvd.nist.gov/vuln/detail/CVE-2017-13077.

[15] *NVD-CVE-2017-13078*. https://nvd.nist.gov/vuln/detail/CVE-2017-13078.

[16] *NVD-CVE-2017-13079*. https://nvd.nist.gov/vuln/detail/CVE-2017-13079.

[17] *NVD-CVE-2017-13080*. https://nvd.nist.gov/vuln/detail/CVE-2017-13080.

[18] *NVD-CVE-2017-13081*. https://nvd.nist.gov/vuln/detail/CVE-2017-13081.