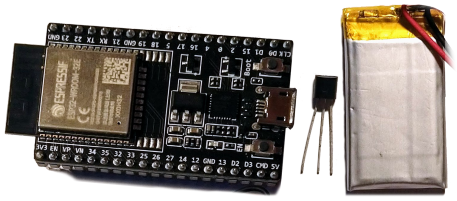
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2021
http://excel.fit.vutbr.cz

Wi-Fi attacks using ESP32

Richard Stehlík*



Abstract
This work explores possibilities of Espressif's ESP32 SoCs in combination with its official development framework ESP-IDF in terms of implementing well-known Wi-Fi attacks on them. Using ESP32 for such attacks may allow attackers to scale their malicious intentions more easily and cut cost and complexities of Wi-Fi attack executions to minimum. Being low powered device also opens ways to minimize size of necessary hardware for Wi-Fi attacks and can easily operate on battery while maintaining a low weight.
Proposed solution presented in this work covers attacks on WPA/WPA2 authentication and their variations like station deauthentication, WPS PIN brute-force attack or PMKID capture. An universal Wi-Fi penetration tool for ESP32 was introduced, that provides easy way to implement new attacks and their variants in the future. It shows how these attacks can be implemented purely by using ESP-IDF's public API or by bypassing closed source Wi-Fi Stack Libraries that have incorporated protection against misusing ESP32 for sending forged frames.
The outcome supports the need to mitigate some vulnerabilities in currently widely used Wi-Fi security features and give them more attention with higher priority.

Keywords: ESP32 — ESP-IDF — Wi-Fi attacks

Supplementary Material: Demonstration Video — Git Repository

*xstehl16@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

While Wi-Fi networks and its underlying 802.11 standard is widespread all around the world and today its usage is even more supported by raise of IoT devices and demand for portability, it still has its flaws in original design that were not yet fully mitigated. Besides some of the longtime well-known attacks like deauthentication attack that are exploiting parts of original 802.11 standard, new attacks are appearing even now, more than 20 years later after initial 802.11 standard release back in 1997. Some of them are exploiting newly found vulnerabilities like *KRACK* attack described in 2017 [1] or describing vulnerabilities themselves like *KR00K* first disclosed in 2020 [2]. There are also new methods being discovered that simplifies previously described attacks like *attack on PMKID* that greatly reduces requirements for classic handshake capture and brute force attack to crack network passphrase to gain access to target network or session Pairwise Transient Key to decrypt captured communication [3, 4].

Most of these attacks are already implemented and integrated in tools that usually rely on services provided by underlying operating system. One of the mostly referred tool for this kind of attacks is *aircrack-ng*[1] that operates on Windows, Linux, OS X and other UNIX based operation systems. They also require some insight into the topic and require specific hard-

[1] https://www.aircrack-ng.org/

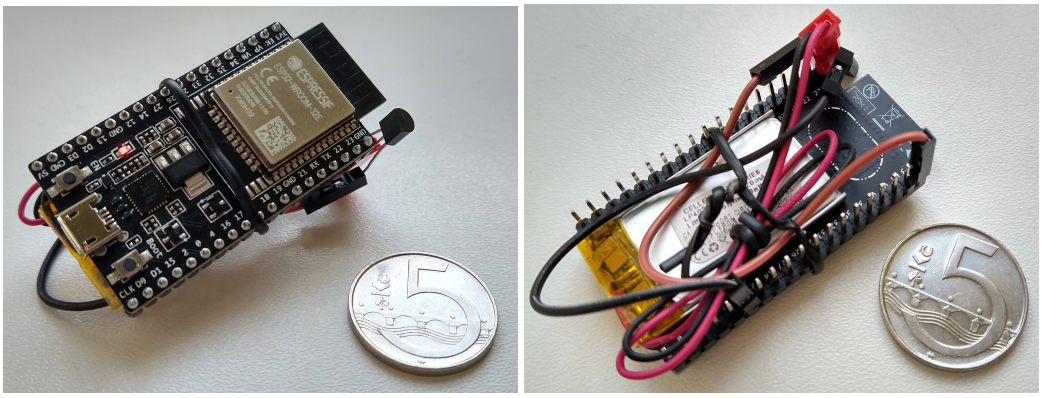


Figure 1. An example of *ESP32 DevKitC* board connected to a Li-Pol accumulator that altogether weights around 17 grams. Czech 5-koruna coin included for scale. This demonstrates how using ESP platform can be useful in terms of downsizing hardware requirements. It can be easily downsized further by using ESP32 module directly, voltage regulator and smaller accumulator to a really small piece of hardware that is easy to hide or transportable for example with small remotely controlled drone.

ware like Wi-Fi adapter with promiscuous mode and frame injection capability. Even though most of these systems are able to run on small computers like Raspberry Pi, in order to go further with reducing hardware size, weight and price needed for the attack, these operating systems are usually a limitation by themselves. By reducing hardware size and price, it can unleash new attack vectors that can lead to attacks in places, that for example were considered secure by physical measures. It can also simplify the overall usage for users by preprogrammed chips that then work in a plug and play way. Microcontrollers manufactured by Espressif Systems with integrated Wi-Fi interface are perfect adepts to be used for Wi-Fi penetration considering also their well documented Espressif IoT Developement Framework (ESP-IDF). ESP32 is currently their latest and most evolved MCU they produce. Figure 1 demonstrates one of the many possibile ways how to wire and operate ESP32 in a compact way.

Motivation behind this work is to explore possibilities of ESP-IDF and demonstrate that ESP32 platform is capable of executing malicious Wi-Fi attacks just by utilising ESP-IDF itself. Executing attacks on ESP32 can be an efficient alternative to existing solutions running on *Raspberry Pi Zero W* or similar micro-computers. Direct comparison with Raspberry Pi Zero W, that can be considered a closest competitor to ESP32, is presented in Table 1. As ESP32 requires low power and allows some low level configurations with Wi-Fi interface on MAC layer, it creates space for experimentation and may open new possibilities in attacker's scenarios.

This work explores capabilities of ESP-IDF and propose ways how to implement different well-known Wi-Fi attacks. It also takes advantage of existing projects, that already solved some of the obstacles introduced by ESP-IDF design and developer decisions and builds complex attacks on top of them. Practical outcome of this work is an universal Wi-Fi penetration tool that is easily extensible by adding new attack types and their methods and also includes some of the proposed attacks implementations to prove this concept.

1.1 Contribution

The main contribution of this work is proposal of implementations of various widely known Wi-Fi attacks using ESP-IDF and ESP32 platform. New approach to execute deauthentication attack by creating cloned rogue access point exploiting native behaviour defined in 802.11i standard is proposed that allows deauthentication attack even without frame injection capability. Some of these implementations are then realised by implementing a new tool to consolidate various attacks into one place that makes executing them simple alongside with easing addition of new attacks and their variants. To demonstrate usability, deauthentication attacks for denial of service attacks and for WPA/WPA2 handshake capture were included in the tool.

2. Common Wi-Fi attacks

Even though 802.11 standard was first introduced more than 20 years ago and is being actively amended by IEEE organisation, there are still vulnerabilities deep in its design that are hard to mitigate without reworking whole concept. These vulnerabilities are drawing attention of various attackers with all kinds of intentions. Attacks and exploits taking advantage of these vulnerabilities are being well described and are implemented in various forms. This section briefly describes common Wi-Fi attacks that have a potential to be implemented on ESP32 platform.

Table 1. Comparison of ESP32 with Raspberry Pi Zero W

	ESP32	Raspberry Pi Zero W
Monitor mode	native	requires custom firmware [5]
Frame injection	limited, can be unblocked	requires custom firmware [5]
Average current draw	∼100mA	∼150mA [6]
Boot current draw	<100mA	up to 200mA [6]
Voltage	3.3V or 5V	5V [6]
OS	—	e.g. Kali Linux
Weight	3.5g (module)	9g [6]
Dimensions	25.2x18x2.8mm (module)	65mm x 30mm x 5.4mm [6]
Price	∼80 CZK (module)	∼300 CZK
Other caveats		requires SD card often out of stock

2.1 Deauthentication attack

Deauthentication attack exploits a behaviour described in 802.11 standard, which allows access point (also referred to as *AP* in this work) or station (*STA*) to interrupt authenticated session by sending deauthentication management frame to its counterpart. If the station or access point receives this kind of frame, it will stop further communication with opposite side until STA authenticates itself again. In original version of 802.11 standard management frames are not encrypted nor they are authenticated [7], so potential attacker within the reach of target network can forge deauthentication frames and broadcast them to his vicinity. Due to lack of authentication or encryption of these frames, stations and access points cannot differentiate between forged and genuine frames and will assume they are valid — resulting into deauthenticating themselves from the network.

Even though this exploit was addressed in *802.11w* amendment in 2009 by forcing encryption of subset of management frames including deauthentication ones [8], it is not widely used and most of now-days APs does not have this feature enabled by default or even not available at all [9]. Hence this type of attack is one of the most used and is often a precondition of many other attacks.

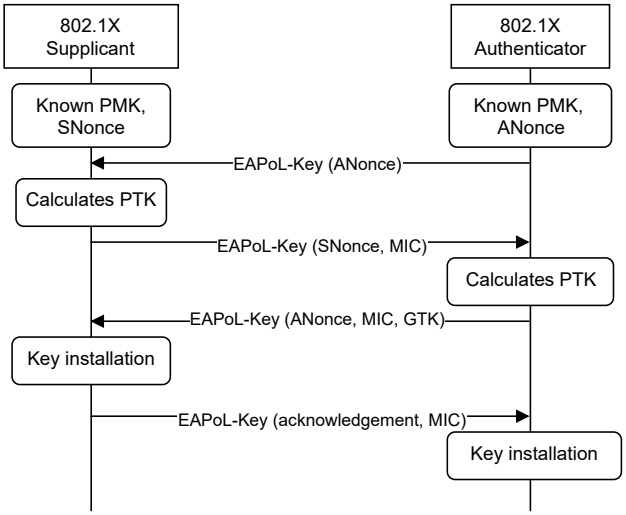


Figure 2. Simplified sequence diagram of WPA/WPA2 handshake exchange where both sides exchange four messages with parameters for calculating same Pairwise Transient Key that is then used for frame encryption. From the sequence diagram it's clear that only secret here is Pairwise Master Key that is used to calculate PTK and is not being transmitted over network. Hence PMK is main target of handshake capture and following brute-force attack

2.2 WPA handshake brute-force attack

Due to a way how session keys for authenticated sessions are being established in WPA/WPA2, it's possible to brute force network passphrase from the initial handshake exchange if captured. During handshake both involved parties are exchanging parameters from which they later calculate same *Pairwise Transient Key — PTK* in format `PTK = PRF (PMK, ANonce, SNonce, MAC(AA), MAC(SA))`. These parameters are random AP nonce, random STA nonce and both their MAC addresses as can be seen on figure 2. All of these parameters are not encrypted and can be easily eavesdropped. The unknown part here is *Pairwise Master Key — PMK*, that both sides know in advance and is never transmitted over network. But as *PMK* format is as following — `PMK = PBKDF2(Passphrase, SSID, 4096, 256)`, the only really unknown part is the network's *passphrase*. This is visualized on Figure 3. Once STA or AP has calculated *PTK*, it starts calculating *Message Integrity Code – MIC* using the *PTK* over subsequent messages and includes it in the message itself. Having all these information from captured handshake, attacker can brute-force the *passphrase* by assembling *PMK* with guessed passphrase, then calculating *PTK* that is then

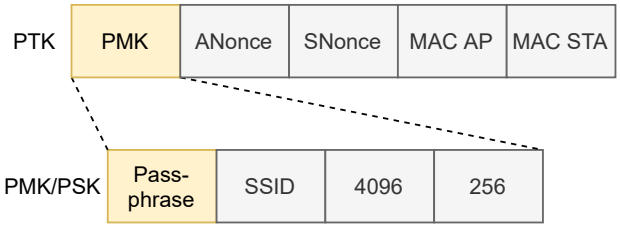


Figure 3. Hierarchy of keys used in WPA authentication. Blue color visualises fields, that are known to attacker as they are transmitted over wireless medium without any encryption. Orange color visualises the secrets, that are unknown to attacker.

used to calculate *MIC* and finally comparing it to actual *MIC* captured from original handshake message. If they match, the passphrase was found. This kind of brute-force attack can be automated by using for example password recovery tool *Hashcat*[2].

As this type of attack require some genuine station to join network at the time when the capture is happening, it's usually combined with deauthentication attack. This, in combination with an automatic rejoin feature that tries to connect device again to the network if it was disconnected, speeds up whole process of obtaining WPA/WPA2 handshake from target network.

2.3 PMKID capture and brute-force attack

Relatively new attack described in 2018 [3] aims on capturing *PMKID* from access point. The format of PMKID is as following `PMKID = HMAC-SHA1(PMK, "PMK Name", MAC_AP, MAC_STA)`. Again, the only unknown secret here is the *PMK*. The *PMKID* is usually being send by APs with roaming feature enabled in the first message of WPA handshake. Hence it can be brute-forced by checking calculated PMKIDs against the original one similarly to the brute-force attack on PMK in classic WPA/WPA2 handshake brute-force attack described in section 2.2. In contrast to classic WPA/WPA2 handshake capture and brute force attack, this approach does not require any STA to be active on the network.

2.4 WPS PIN attack

Another common attack that is still applicable on many networks is taking advantage of the poor design of *Wi-Fi Protected Setup* [10]. WPS was introduced by Wi-Fi Alliance in 2007 to ease setting up WPA2 secured networks and connecting new stations for non-technical users. When station is successfully authenticated, AP transmits WPA/WPA2 passphrase to station so it can then automatically proceed with WPA/WPA2 authentication as if user would provide passphrase manually. Besides other methods, it allows authentication by PIN code that is provided by AP and is the only method that does not require physical access to some authentication authority (for example physical push button or NFC reader) [11].

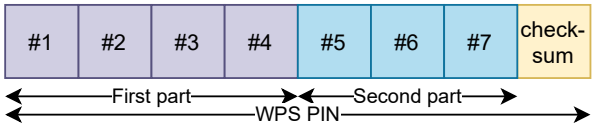


Figure 4. WPS PIN consists of eight digits, where the last digit is a checksum of the first seven digits. During WPS authentication where AP is in Enrollee role, AP is validating PIN in two phases. First it checks first four digits and only if they are correct, it moves forward to validate next three digits. Checksum is calculated over the first seven digits.

Although WPS PIN method can be considered secure in theory, the implementation of it, especially the PIN verification phase makes it vulnerable to brute-force attack. By design the eight-digits long PIN code is split in half. In a first authentication phase, AP validates first four digits and if they are not correct, authentication fails with NACK message from AP. When the first four digits are correct, it proceeds with validation of second half. Second half of PIN is split even further — last digit is checksum of first seven digits of the PIN. This division is visualised on Figure 4.

This validation flow drastically reduces complexity of brute-force attack as instead of 10^8 possible combination (if the full length of the PIN is not predictable and validation is done in a single run) it now requires only $10^4 + 10^3 = 11000$ attempts to exhaust whole space of possible combinations, making this type of attack very efficient. If AP is not protected against this attack by some cool-down period after given number of authentication failures, it may take up to approximately 4 hours to find the correct PIN [10].

2.5 KRACK attack

KRACK (which is an abbreviation of *Key Reinstallation Attack*), first described in 2017, exploits behaviour of stations after receiving third message of WPA handshake [1]. By design of 802.11i standard which is underlying protocol for WPA/WPA2 standard, when station receives third handshake message, it should install PTK calculated after receiving first handshake message from AP [12] — this can be seen as *Key installation* event on Figure 2. This event also resets session parameters like packet nonce or replay counter [1]. Due to presence of *Message Integrity Code* and packet nonce and replay counter in handshake messages, they

[2] https://hashcat.net/hashcat/

cannot be simply captured and replayed later. Man in the Middle (MitM) attack has to be setup to postpone third handshake message from AP with valid relay counter and MIC for later re-transmission to actual STA. At the time of this vulnerability disclosure, the vulnerability severity was multiplied by a bad design of Linux's `wpa_supplicant` in versions 2.4 and 2.5 used in Linux and Android distributions. This implementation caused reinstallation of all-zero key instead of the original PTK generated after first handshake message [1]. As a result, all following communication was encrypted by cryptographically weak all-zero key. This issue was addressed since version 2.6 of `wpa_supplicant` and hence only outdated devices may still be prone to this kind of attack [1]. There is still a space for cryptoanalysis considering reused parameters are being used, but those are out of scope of this work.

2.6 Kr00k attack

One of the most recent vulnerabilities discovered in Wi-Fi networks called *Kr00k* and disclosed in 2019, takes advantage of vulnerability in Broadcom and Cypress Wi-Fi chips design [2]. These chips implement transmit buffer from which some frames may be transmitted encrypted by all-zero key [2]. This occurs, when the vulnerable station is disassociated from AP and clears it's encryption key before all frames are transmitted from transmit buffer [2]. This is not an attack on its own, but if the disassociation is forced intentionally by for example deauthentication attack and frames encrypted by all-zero key are captured, it may target on specific victim on the network and can be considered an attack. Although this was patched on most of the affected chips, it can still be a possible attack vector on some outdated devices.

3. Possibilities of ESP-IDF

ESP-IDF provides powerful API for controlling embedded Wi-Fi interface. In this section the actual implementations of attacks briefly introduced in section 2 are proposed and for some variations are discussed. It also discusses existing projects that may be utilised for attack implementations. This section also covers ESP-IDF limitations that creates some unnecessary obstacles to implement these types of attacks.

3.1 Limitations

Main limitation for working with Wi-Fi interface on ESP32 are closed source *Wi-Fi Stack Libraries*[3]. These libraries incorporate a blocking mechanism that prevents sending arbitrary frames of specific types like deauthentication frames due to various undisclosed reasons. Based on posts from Espressif's employees on official Espressif forum, one of the reasons may be a potential risk of abusing this functionality for deauthentication attacks [13, 14].

Another limitation may be the fact, that ESP32 does not have sufficient power to run offline brute-force attack on captured WPA handshake or PMKID in efficient time. To compensate this limitation, further processing of captured handshake can be reduced by providing captured data directly in a format that is consumable by some third party password recovery tool. For example it can be HCCAPX[4] format, that can be directly passed to well-known password recovery tool *Hashcat*.

3.2 Existing solutions

The idea of utilising Espressif's micro-controllers for Wi-Fi attacks is not new and it was already done on ESP8266, which is predecessor of ESP32. Even though it's different platform incompatible with ESP32, it may still be beneficial to observe their implementation and take advantage of their findings while implementing similar attacks on ESP32 platform.

One of the most activate and well-maintained Wi-Fi attack project on ESP8266 platform is Spacehuhn Tech's *ESP8266 Deauther*[5]. This project implements various attacks, mostly by sending forged frames like deauthentication, probe or beacon frames. This project is limited by ESP8266 platform itself, as it is able to capture frames in promiscuous mode only up to 112 bytes and strips the rest [15]. Hence it cannot implement proper handshake capture.

First try to get better insight into Wi-Fi Stack Libraries on ESP32 was demonstrated in project by Jeija *esp32free80211*[6]. Author of this project partially decompiled Wi-Fi Stack Libraries and was able to figure out a function, that could send raw frames from buffer. Even though in project documentation it's being said that it may allow sending deauthentication packet, it was probably never possible as the blocking mechanism was still in place by Wi-Fi Stack Libraries. Espressif's developers also made limited sending function `esp_wifi_80211_tx` publicly available in ESP-IDF, hence this project is not maintained anymore.

[3] https://github.com/espressif/esp32-wifi-lib

[4] https://hashcat.net/wiki/doku.php?id=hccapx

[5] https://github.com/SpacehuhnTech/esp8266_deauther

[6] https://github.com/Jeija/esp32free80211

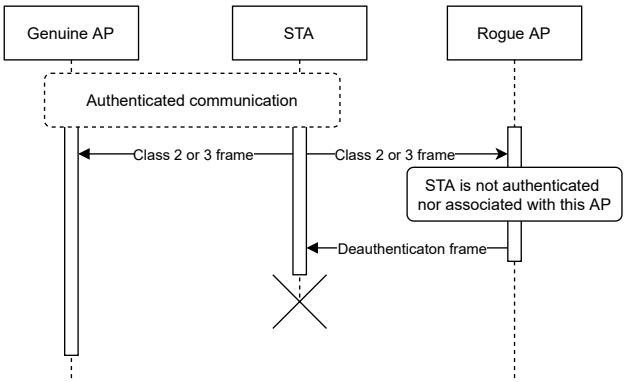


Figure 5. By utilising ESP-IDF option to change MAC address of interface used for AP, duplicated AP can be created that will act as common AP responding to stations that are sending frames to them. If unauthenticated (from AP point of view) station sends class 2 or class 3 frame with matching MAC address (BSSID), AP responds with deauthentication frame and appropriate reason as designed in 802.11 standard. For station receiving this deauthentication frame it's impossible to differentiate between genuine AP and rogue AP and accepts it as valid one disconnecting itself from network.

More recent project from GANESH ICMC/USP organization *esp32-deauther*[7] published in 2019 demonstrated a working solution that bypasses the blocking mechanism in Wi-Fi Stack Libraries by overriding function definition at compilation time. They were able to decompile Wi-Fi Stack Libraries and get a declaration of function that checks validity of frame in transmit buffer — `ieee80211_raw_frame_sanity_check`. Then by using linker flag `-z muldefs` [16] during compilation this method can be simply overridden and always return value `0`, that allows transmission of frame in transmit buffer. Despite the name of the project, this solution actually allows sending not only deauthentication frames, but anything stored in transmit buffer without further validation by Wi-Fi Stack Libraries.

3.3 Deauthentication using ESP-IDF libraries without modifications

To not rely on bypassing Wi-Fi Stack Libraries blocking mechanism by overriding guarding function explained in section 3.2, it's possible to utilise features provided by ESP-IDF only. Native behaviour of access points can be used to send deauthentication frames to stations that try to communicate with genuine AP within it's a range. This is demonstrated in Figure 5.

The main feature that supports this approach is an option to change MAC address of Wi-Fi interface of ESP32. If MAC address of target AP is obtained (it is usually BSSID that is present in most of the frames going through network), it can be configured to ESP32's Wi-Fi interface. If then also SSID is obtained, for example by using built in scanning functionality of `esp_wifi` component, it's possible to create exact copy of targeted AP. Considering an ongoing authenticated communication between targeted AP and some authenticated station, whenever ESP32 AP receives frame from this STA of class 2 or 3, it will automatically respond with deauthentication frame [12]. This is because from ESP32 AP point of view this station is not authenticated and have to authenticate first.

Deauhentication attack can be used for example to exploit Kr00k vulnerability or many other attacks where deauthentication of station is a prerequisite. This approach can be also used on various other systems, where Wi-Fi network interface has no frame injection capability. This approach requires only MAC spoofing option.

3.4 Capturing PMKID

Another possible attack that utilises ESP-IDF only without any modification is PMKID attack. To capture PMKIDs, only the first message of handshake has to be captured. This can be triggered by connecting to target AP with ESP32 in station mode. WPA handshake is always initiated by AP, so if AP has PMKIDs available, it will send them in the first message of the handshake without any prior authentication from the station.

3.5 WPS PIN brute-force attack

Running WPS PIN attack requires working WPS Registrar logic to be implemented on ESP32. In WPS Registrar mode, STA is authenticating itself using preshared PIN code provided by AP [11]. Unfortunately ESP-IDF provides only WPS Enrolee mode, hence to run WPS PIN attack, WPS Registrar mode has to be implemented first. This can be done by utilising *esp32-deauther* project introduced in section 3.2, that unblocks sending arbitrary 802.11 frames. It can be used to send all messages until the first part of PIN is confirmed by AP and then proceed with second part and finally get the WPA credentials. Even though the official documentation does not mention WPS Registrar mode in WPA supplicant component, source code exists for this mode in `wpa_supplicant` component[8] and can be used to implement WPS Registrar mode.

[7] https://github.com/GANESH-ICMC/esp32-deauther

[8] https://github.com/espressif/esp-idf/blob/master/components/wpa_supplicant/src/wps/wps_registrar.c

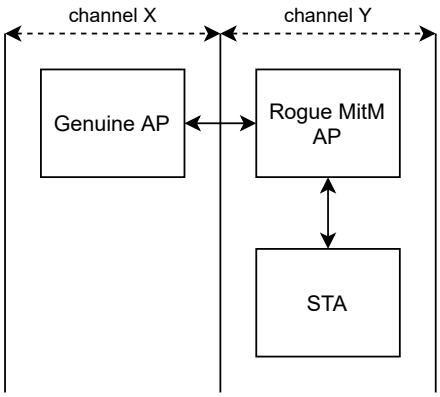


Figure 6. Visualisation of rogue AP in Man in the Middle position. Rogue AP fully duplicates genuine AP using MAC spoofing and using same SSID but operating on different channel. If some STA tries to connect to rogue AP, for example because it has better signal strength, rogue AP is able to manipulate and relay communication between genuine AP and this STA.

3.6 Man in the Middle AP

Wi-Fi interface channel of ESP32 can be switched by using ESP-IDF's function `esp_wifi_set_channel`. In combination with spoofing MAC address of genuine AP, this allows creation of rogue AP clone on different channel. By utilising *esp32-deauther* project mentioned in section 3.2, it's possible to forward and manipulate whole communication between genuine STA and AP. MitM position is demonstrated on Figure 6.

MitM AP can be used for example for KRACK attack described in 2.5. It can be used to postpone transmission of third handshake message that will cause key re-installation event later on STA.

3.7 Bypassing MAC filtering

Even though MAC filtering is not considered a proper security method alone, it's often used as another layer of security creating additional obstacle for potential attacker trying to break into the target network. Although changing MAC address of Wi-Fi interface is commonly available on laptops, it's often not possible on mobile devices. ESP32 can be used to provide an open AP giving any station nearby access to the target network even if they doesn't support changing of their Wi-Fi interface MAC address. ESP-IDF provides a function `esp_wifi_set_mac` that allows changing MAC address of interface. By taking advantage of promiscuous mode, automatic capture of connected STA MAC can be implemented and assuming the target network passphrase is already known, it can connect to network by spoofing captured MAC address.

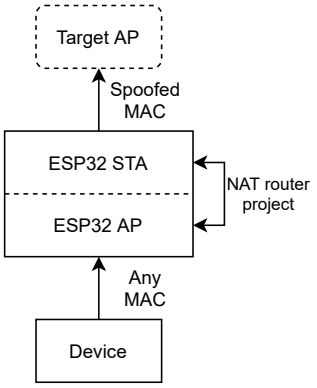


Figure 7. Visualisation of ESP32 configuration used to bypass MAC filtering on AP and open access into target network to devices without MAC changing capability. This approach assumes that the passphrase for target network is already known.

Deauthentication attack proposed in section 3.3 can be used to disconnect genuine stations of which MAC address is being used by ESP32. To setup an AP that then provides access to the target network, existing project *ESP32 NAT Router*[9] can be utilised. This configuration is visualised on Figure 7.

Direct comparison of what modifications different attacks are dependent on can be seen in Table 2.

4. ESP32 Wi-Fi Penetration Tool project

As it came out of the proposed implementations in section 3, various different attacks can be implemented on ESP32. This fact led to an idea to create a universal tool that would wrap shared logic and ease addition of new attack types and their methods to create a comprehensive but lightweight solution for Wi-Fi penetration testing. In this section I will introduce *ESP32 Wi-Fi Penetration Tool* that realizes this idea.

To demonstrate how attacks proposed in section 3 can be actually executed on ESP32, deauthentication attack was picked for that purpose and were implemented using both proposed methods. First method is by utilising *esp32-deauther* project that allows sending arbitrary frames including deauthentication ones. Second method is by creating duplicated rogue AP exploiting native behaviour standardised in 802.11i, where AP should automatically send deauthentication frames whenever it receives class 2 or 3 frame from unauthenticated station.

4.1 Project structure

The project is build on top of the components that ESP-IDF provides and recommends to use [17]. This way,

[9] https://github.com/martin-ger/esp32_nat_router

Table 2. Attack dependencies

Attack	Dependent on
Deauth	ESP-IDF only (using rogue AP) or WSL bypassing (forging deauth frames)
WPS PIN bruteforce	WPS Registrar mode
KRACK	MitM AP and WSL bypassing
Kr00k	Deauth attack
MAC filtering bypasser	ESP-IDF only (can utilise esp32-nat-router)

shared parts like Wi-Fi controller or frame analyzer are being implemented in standalone components that can be easily reused either directly in this project or even in other project by simply copying them and reusing them out of the box. ESP32 Wi-Fi Penetration Tool project consists of following components:

Wi-Fi controller Component that handles all Wi-Fi interface related operations and provides simplified interface. It's used to initialize Wi-Fi interface, to control access point and station configurations, to switch Wi-Fi interface into promiscuous mode and other similar operations.

Frame analyzer Main purpose of this component is to parse frames captured by Wi-Fi controller and do an analysis of these frames. For example it does parsing of PMKIDs from EAPOL packets, detected encrypted frames, filter frames by BSSID and passes the results into event pool. It also provides a parsing functionality for other components like HCCAPX serializer.

WSL Bypasser This component is used to unblock raw frame transmission by overriding blocking function in Wi-Fi Stack Libraries. This component is based on an existing project *ESP32-deauther*.

Webserver Webserver component provides an UI to control the tool itself and allows configuration of available attacks. It's build on top of ESP-IDFs *HTTP server* component. Sample of user interface is shown on Figure 8.

PCAP and HCCAPX serializers These two small components fromat captured frames into a common formats like PCAP for further analysis in Wireshark or other tools or HCCAPX for direct use with password recovery tool *Hashcat*.

Main The main component groups all the attack types and variations alongside with a universal attack handler, that takes care of timeouts, configurations and other shared operations.

4.2 Attack wrapper

The attack logic itself lives inside Main component. It handles shared attack operations like attack timeout and abort calls. It obtains configuration via Webserver component from user input in JavaScript powered web client and executes appropriate attacks and their methods based on this configuration. When the attack finishes, it returns results of the attack to the client that displays them to user.

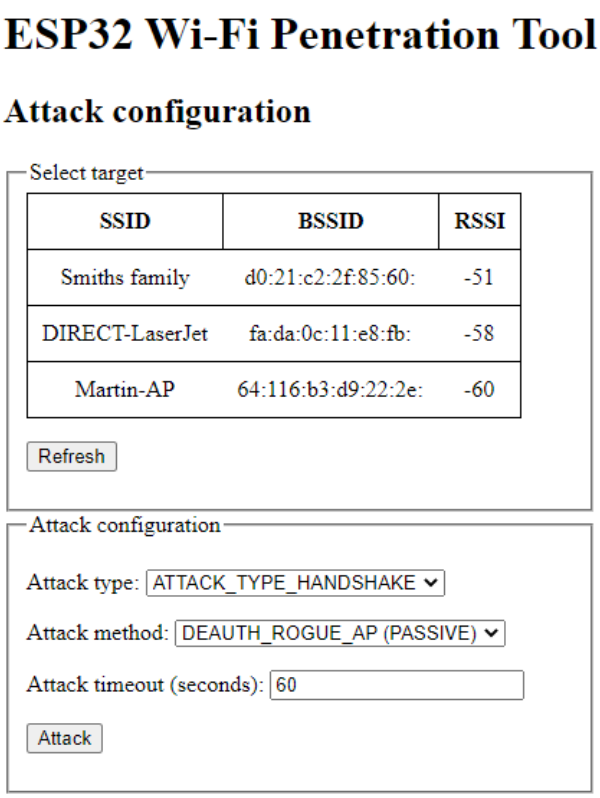


Figure 8. User interface for universal ESP32 Wi-Fi penetration tool that shows scanned APs in ESP32 surrounding and allows configuration of the attack itself by choosing attack type, method and timeout.

When adding new attack type or method, all the programmer have to do is add this new attack into enumeration of available attack types and methods in main component and in JavaScript client. Then it's a matter of calling appropriate functions from components that do required operations and implementing what is not already included.

4.3 General use-case scenario

To execute available attacks on surrounding APs with this tool, when user powers the ESP32, management AP is automatically started when ESP32 boot completes. User then can connect to this management AP using for example his cellphone. He's provided with a web application, that allows him to pick a target AP in his vicinity and choose attack type he wants to execute.

When the automated attack finishes, web application shows a result of this attack to user, from where he can for example download PCAP or HCCAPX files or see a reason for attack failure. This depends on specific attack type implementation.

4.4 Evaluation

ESP32 is not sufficient to run brute-force attacks so this has to be done on external machine with reasonable power. However this is not efficient even on standard laptops or computers. Brute-force cracking can be outsourced to some cloud service like AWS EC2 P3 instances[10].

ESP32 with ESP32 Wi-Fi Penetration Tool can reliably run deauthentication attacks. Some devices may ignore broadcast deauthentication frames or be protected against some types of attacks. It can be solved by combining different approaches when attacking against given network which is supported by this tool by its design. There was not detected any significant frame loss. Handshakes and PMKIDs were always captured.

Flashing this project onto ESP32 can be done by running simple binary included in ESP-IDF toolchain or by using standalone flashing tool with graphical user interface[11]. It can also be distributed pre-flashed, ready to work out of the box. In both cases it doesn't require almost any domain knowledge from the user. The attacks themselves are automated and everything the user has to do is choosing target AP/network and the type of attack he wants to run.

ESP32 as an ultra-low power platform can be efficiently powered by batteries, or smartphone which makes the solution portable and inconspicuous. Being less noticeable in the public by controlling the attacks using for example a smartphone can open new attack vectors to the attacker. Also thanks to the small dimensions and light weight of the ESP32-WROOM module alone, it makes it easily attachable to a small drone by which attacker can reach otherwise physically unreachable networks[12].

By experimental measurement while executing attacks implemented in Wi-Fi Penetration Tool, ESP32 consumes from 90 to 110 mA of power. Considering two standard Li-ion batteries 18650 with capacity of 2600 mAh, this allows approximately 24 hours[13] of active attack. That is enough to run long running online brute-force WPS attack proposed in section 3.5 taking up to 4 hours[10]. For practical example, on Figure 1, I have used a Li-Pol accumulator with capacity of 220 mAh, that is able to provide enough power for about two hours of active usage.

5. Conclusions

In this paper I have explored possibilities of portability of common Wi-Fi attacks to ESP32 platform. It was demonstrated that attacks mentioned in this work are possible to be implemented and executed on ESP32 thanks to capabilities like changing Wi-Fi interface MAC address, configuring access points and stations, promiscuous mode with an optional bypass of Wi-Fi Stack Libraries. An universal ESP32 Wi-Fi Penetration Tool was introduced which acts as a wrapper for further attack implementations.

Outcome of this work supports arguments why current Wi-Fi vulnerabilities has to be taken seriously as they are easily executable on cheap and accessible hardware. Considering that resources for Wi-Fi attacks may be really lightweight both in terms of price, weight and size it allows anyone with minimal knowledge about the topic — often referred to as *script kiddie* — to run potentially harmful attacks against Wi-Fi networks in their vicinity. It may raise awareness of how easily some of the attacks can be implemented and why one should not rely on obsolete 802.11 implementations and outdated security features and should upgrade their hardware (routers, cellphones and other Wi-Fi capable devices) and software to include latest security measures like WPA3 or 802.11w amendment.

As an demonstration of utilising official ESP-IDF framework for Wi-Fi attack implementations I have implemented deauthentication attack with handshake frames capture by creating rogue access point that duplicates genuine one and rely on native AP behaviour defined in 802.11 standard. I have also implemented other variants of these attack by assembling and sending deauthentication frames directly from code and an PMKID capture attack by simply connecting to

[10] https://aws.amazon.com/ec2/instance-types/p3/

[11] Windows OS application — https://www.espressif.com/en/support/download/other-tools

[12] e.g., networks in some buildings behind fence or in higher floors in office buildings that cannot be accessed by public

[13] excluding step-up voltage regulator power consumption

target AP and parsing PMKIDs from first handshake message.

This work provides a tool for further extensions and experimentation with Wi-Fi attacks on cheap and low powered ESP32 SoC. It shows that attacks can be done on small hardware powered with small battery which opens new ways how the attackers can execute attacks. Components structure in the project provides reusability and may simplify further researches of new attacks. Next steps may be implementation of WPS attack or use ESP32 to bypass MAC address filtering on AP.

ESP32 Wi-Fi Penetration Tool was published as an open-source project on GitHub[14]. The repository includes also more detailed description of the components and overall functionality available also in form of GitHub Pages[15].

Acknowledgements

I would like to thank my supervisor Ing. Jan Pluskal for his valuable inputs during my work.

References

[1] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in wpa2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1313–1328, New York, NY, USA, 2017. Association for Computing Machinery.

[2] Róbert Lipovský, Svorenčík, and Miloš Čermák. Kr00k - cve-2019-15126. *We live security*, page 9, 2020.

[3] New attack on wpa/wpa2 using pmkid. online, https://hashcat.net/forum/thread-7717-post-41427.html#pid41427.

[4] Michael Asante and Kwabena Akomea-Agyin. Analysis of security vulnerabilities in wifi protected access pre shared key (wpa psk/ wpa2 psk). *International Research Journal of Engineering and Technology (IRJET)*, 6(1):537 – 545.

[5] Kody. Enable monitor mode & packet injection on the raspberry pi. online, https://null-byte.wonderhowto.com/how-to/enable-monitor-mode-packet-injection-raspberry-pi-0189378/.

[6] RASPBERRY PI FOUNDATION. Faqs. online, https://www.raspberrypi.org/documentation/faqs.

[7] *IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements- Part 11*. New York, USA, 1999 ed. edition, 2003.

[8] *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 11*. USA, 2009 ed. edition, 2009. 802.11w.

[9] Hans Matthé. 802.11w – does it work? online, https://www.boostyourwifi.be/2018/07/04/802-11w-does-it-work/.

[10] Stefan Viehböck. Brute forcing wi-fi protected setup. 2011.

[11] *Wi-Fi Protected Setup Specification*, 1.0h edition, 2006. online, https://www.wi-fi.org/discover-wi-fi/wi-fi-protected-setup.

[12] *IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11*. New York, USA, 2004 ed. edition, 2004. 802.11i.

[13] Angus Gratton. esp_wifi_internal. online, https://www.esp32.com/viewtopic.php?t=586.

[14] Ivan Grokhotkov and Jeroen Domburg. Please open source this library. online, https://github.com/espressif/esp32-wifi-lib/issues/2.

[15] *ESP8266EX*, 6.6 (2020.10) edition, 2020. online, https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[16] *ld(1) - Linux man page*. online, https://linux.die.net/man/1/ld.

[17] *ESP32*, 2020. online, https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.

[14] https://github.com/risinek/esp32-wifi-penetration-tool

[15] https://risinek.github.io/esp32-wifi-penetration-tool/