Article

WPAxFuzz: Sniffing Out Vulnerabilities in Wi-Fi Implementations †

Vyron Kampourakis [1], Efstratios Chatzoglou [2], Georgios Kambourakis [3,*], Apostolos Dolmes [2] and Christos Zaroliagis [4]

1 Department of Information Security and Communication Technology, Norwegian University of Science and Technology, 2802 Gjøvik, Norway
2 Department of Information and Communication Systems Engineering, University of the Aegean, 83200 Karlovasi, Greece
3 European Commission, Joint Research Centre (JRC), 21027 Ispra, Italy
4 Department of Computer Engineering and Informatics, University of Patras, 26504 Patras, Greece
* Correspondence: georgios.kampourakis@ec.europa.eu or gkamb@aegean.gr

† This is a short communication article based in part on a M.Eng. student thesis.

Abstract: This work attempts to provide a way of scrutinizing the security robustness of Wi-Fi implementations in an automated fashion. To this end, to our knowledge, we contribute the first full-featured and extensible Wi-Fi fuzzer. At the time of writing, the tool, made publicly available as open source, covers the IEEE 802.11 management and control frame types and provides a separate module for the pair of messages of the Simultaneous Authentication of Equals (SAE) authentication and key exchange method. It can be primarily used to detect vulnerabilities potentially existing in wireless Access Points (AP) under the newest Wi-Fi Protected Access 3 (WPA3) certification, but its functionalities can also be exploited against WPA2-compatible APs. Moreover, the fuzzer incorporates: (a) a dual-mode network monitoring module that monitors, in real time, the behavior of the connected AP stations and logs possible service or connection disruptions and (b) an attack tool used to verify any glitches found and automatically craft the corresponding exploit. We present results after testing the fuzzer against an assortment of off-the-shelf APs by different renowned vendors. Adhering to a coordinated disclosure process, we have reported the discovered issues to the affected vendors, already receiving positive feedback from some of them.

Keywords: IEEE 802.11; Wi-Fi; WPA3; WPA2; wireless network security; fuzz testing

check for updates

Citation: Kampourakis, V.; Chatzoglou, E.; Kambourakis, G.; Dolmes, A.; Zaroliagis, C. WPAxFuzz: Sniffing Out Vulnerabilities in Wi-Fi Implementations. *Cryptography* **2022**, *6*, 53. https://doi.org/10.3390/cryptography6040053

Academic Editor: Josef Pieprzyk

Received: 3 September 2022
Accepted: 18 October 2022
Published: 20 October 2022

1. Introduction

On the eve of the seventh generation of Wi-Fi technology, IEEE 802.11 networks are omnipresent in our daily life. Despite the advances brought with each generation of the 802.11 standard, security still remains a hot issue. That is, while the newest Wi-Fi Protected Access 3 (WPA3) certification by Wi-Fi Alliance mandates security features introduced in previous IEEE 802.11 amendments or standards, including 802.11w, 802.11-2016 and 802.11-2020 [1], recent work in the topic has shown that vulnerabilities are not scarce. For instance, the work in [2,3] showcased several Denial of Service (DoS) attacks against the so-called Simultaneous Authentication of Equals (SAE), while others [4,5] indicated that the Protected Management Frames (PMF), introduced with 802.11w, are insufficient when it comes to persistent deauthentication attacks. Nevertheless, in spite of several works on 802.11 security [6–8], the literature lacks a full-fledged solution that can be used as a reference test platform for detecting possible latent vulnerabilities in Wi-Fi Access Points (AP) and Stations (STAs). Fuzz testing, a.k.a. fuzzing, is a straightforward choice for fulfilling the aforementioned need; it comprises a black-box automated software testing technique, aiming to discover implementation bugs through malformed or semi-malformed data injection [9,10].

Our Contribution: In this context, the work at hand introduces a publicly available, open-source fuzzing tool that can be used to scrutinize any AP implementation. Specifically, the tool, dubbed *WPAxFuzz* [11], can be exploited against both WPA2 and WPA3-certified or compatible products and is currently able to fuzz 802.11 management and control frames, as well as the SAE handshake. Moreover, WPAxFuzz incorporates a dual-mode monitoring functionality for detecting communication disruptions between the AP and an STA, which may indicate a latent misconfiguration or vulnerability in the AP's (or STA's) implementation. No less important, as a side contribution, we offer an "attack module", which takes advantage of the log files produced by the fuzzing tool to directly assault the AP and aid the investigator in easily verifying the results of the tool. To the best of our knowledge, the implemented tool, essentially a stateless black-box mutation/grammar-based fuzzer, is the first of its kind in the literature and, therefore, is anticipated to fuel further research efforts in this timely and important field. Overall, the novelty of this work spans two main axes:

- Based on an empirical methodology, we contribute, to our knowledge, the first fuzz testing tool for both WPA2 and WPA3 Wi-Fi 802.11 implementations. The tool is provided to the community for further study and experimentation.
- Through the same tool, we expose several zero-days existing in contemporary 802.11 access point implementations. This fact alone suggests that the implementation of the protocol by chipset vendors is not yet mature enough.

The rest of the paper is structured as follows. The next section details the related work. Preliminaries regarding the setup and dependencies of the tool are given in Section 3. Section 4 delves into the specifics of WPAxFuzz, explaining its components and functionality. The results after using the tool against a variety of AP by different vendors are given in Section 6. The last section concludes and presents directions for future work.

2. Related Work

The topic of 802.11 fuzzing has received limited attention in the literature until now. The first SAE fuzz testing tool, dubbed *DragonFuzz* was contributed by [12]. It fuzzes the authentication commit and authentication confirm frames, which are being transmitted throughout the SAE exchange. However, the tool does not exhaustively check all relevant frame fields that can be received against every possible value, and, in addition, it does not implement an efficient way of detecting any kind of connectivity disruption in the communication between an STA and the AP. That is, by detecting and logging this kind of permanent or temporary interruptions in the communication, say, the STA does not respond to *ICMP echo requests* (ping) requests, the tester can easily detect misconfigurations toward finding the root cause of the issue.

Another study addressing the endurance of the SAE protocol was performed in [2]. The authors presented a variety of Denial of Service (DoS) attacks against the protocol and identified several vulnerabilities. Specifically, they concentrated on more than a dozen different attacks, concluding that SAE is susceptible to all of them if the attacker is persistent enough.

The authors of [13] contributed a tool dubbed *WPA3Fuzz* in an effort to discover latent vulnerabilities or implementation flaws in both SAE and the Protected Management Frames (PMF) mechanism. Through the tool, they managed to point out three vulnerabilities that can be used for initiating DoS attacks against several WPA3 devices.

Apart from the SAE authentication and key agreement protocol, PMF is a mandatory feature of the WPA3 certification. Precisely, PMF was originally introduced in amendment 802.11w, targeting the protection of specific, sensitive in terms of security management frames, including disassociation, deauthentication, and robust action frames. Put simply, PMF protects against deauthentication or disassociation assaults, which aim to violently disconnect a client from the network. The robustness of PMF is examined and assessed by the work in [5], where it is demonstrated that even with PMF, a DoS attack is feasible. This result also comes as a verification of the experiments conducted in [4]. On the downside, both the aforementioned works do not provide a tool to automatize any assessment process regarding the 802.11 pertinent management frames.

Based on the above literature analysis, it becomes clear that thus far, all methodologies or tools aim at the SAE password-authenticated key agreement method, omitting the rest of the 802.11 management frames; recall that SAE is performed over *Authentication* frames. On top of that, the hitherto contributions are able to fuzz a limited set of seeds to the target devices, mostly aiming at specific misconfigurations. No less important, none of the above-mentioned works contributes an adjacent monitoring mechanism or attack tool to ease the detection and logging of device disconnection events. Such functionality is of equal importance to the fuzzing process because it allows the analyst to determine the underlying root cause of the problem, possibly leading to a vulnerability.

3. Setup and Dependencies

The installation and operation of the fuzzer assumes a Linux environment. Prior to describing WPAxFuzz in detail, it is essential to mention some requirements and software dependencies. First, before initializing the fuzzer, the user has to probe the local network to discover any potential targets, i.e., STAs and APs; to do so, they can utilize external utilities such as *nmap*. For example, this can be achieved through the below command:

nmap -sP "ip_prefix."*, where *ip_prefix* is a local IP address, e.g., 192.168.1.

Naturally, the target STA has to be reachable and already associated with the target AP. In the case where the fuzz testing is executed on a Virtual Machine (VM), and the target STA happens to also run on the host machine, it may lead to mistaken deductions. Therefore, the target STA should preferably run on a separate host, e.g., a smartphone or laptop. If the target STA is an MS Windows Operating System (OS) machine, it may be required to modify the firewall to allow "pinging" within the local network. This is a straightforward restriction, considering that the fuzzer has to constantly monitor the aliveness of the target STA by probing it through ping requests or otherwise.

Moreover, for initializing the fuzz tool, the user has to fill out or update a JavaScript Object Notation (JSON) configuration file with essential information, namely, the MAC addresses of the target AP and the associated STA, the name of the network interface that will be used to inject the frames into the wireless channel, and the Service Set Identifier (SSID) of the target AP. The user must also configure the attacking Wireless Network Interface Card (WNIC) to operate in monitor mode and set the radio channel to that of the target AP. For instance, assuming a Linux environment, the user can find the radio channel the AP transmits using the below command, with the parameter ATT_INTER being replaced with the utilized wireless interface, say, *wlan0*.

sudo airodump-ng ATT_INTER.

To alter the WNICs channel, one can use the following command, with "ATT_INTER" being the name of the wireless interface and "CH" the desired radio channel, respectively.

sudo iw ATT_INTER set channel CH HT20.

Last but not least, it is also necessary to select one of the two currently implemented "fuzzing modes" as follows; the mode is configured through the fuzzer's Command-Line Interface (CLI).

1. *Standard*: All the 802.11 frame fields, including the ones being produced with "Blab" (explained below), carry a value length that abides by the 802.11 standard. This way, the frame will not risk being characterized as malformed and dropped.
2. *Random*: The fields produced via the seed generator have a random value length, which can be either lesser or greater than that defined by the 802.11 standard.

For generating the testing seeds that are injected into certain frame fields during the fuzzing process, WPAxFuzz utilizes the commonly accepted *Blab* [14] open-source tool. Blab is able to generate data according to grammars, thus, in our case, providing virtually unrestricted and random fabrication of the testing seed per frame field. In this respect, an obvious dependency is that the user has to install the grammar seed generator to create the injected frame content.

4. Methodology and Operation of WPAxFuzz

This section details the implemented fuzzing tool both from a methodological and operational standpoint. Given its more generic nature, we first refer to the fuzzing of the management and control frames, while SAE-specific fuzzing is presented in Section 4.4. This section assumes that the reader has a basic knowledge of the IEEE 802.11 standard.

4.1. Management Frames

WPAxFuzz considers eight different management frames, as presented in Table 1. Note that we purposefully excluded the deauthentication and disassociation management frames from this initial version of *WPAxFuzz* because they are well known to be straightforwardly exploited by attackers to kick STAs out of the network [4,5]. The table also contains the fields the fuzzer "seeds" during the fuzz process; these fields refer either to the frame's payload or MAC header and are typically seeded by Blab unless stated otherwise.

Table 1. Fields manipulated per management frame subtype.

Subtype	**List of Fields**		
	MAC Header	**Payload**	
		Fixed Parameters	**Tagged Parameters**
Beacon	Source MAC address	Timestamp and interval time, Capabilities	SSID, Supported rates, Extended supported rates, DSset, TIM, RM-enabled capabilities, HT capabilities, HT information, Extended capabilities, RSN
Probe request	Source MAC address	NONE	Supported rates, Extended supported rates, DSset, HT capabilities, RSN
Probe response	Source MAC address, Reverse destination and source addresses	Timestamp and interval time, Capabilities	SSID, Supported rates, Extended supported rates, DSset, RM-enabled capabilities, HT capabilities, HT information, Extended capabilities, RSN
Association request	Source MAC address	Capabilities	Supported rates, Extended supported rates, Power capabilities, Supported channels, RSN, HT Capabilities, Extended capabilities
Association response	Source MAC address, Reverse destination and source addresses	Capabilities	Supported rates, Extended supported rates, HT Capabilities, HT information, Overlapping BSS scan parameters, Extended capabilities
Reassociation request	Source MAC address	Capabilities	Current AP, Supported rates, Extended supported rates, Power capabilities, Supported channels, RSN, RM enabled capabilities, HT Capabilities, Extended capabilities
Reassociation response	Source MAC address, Reverse destination and source addresses	Capabilities	Supported rates, Extended supported rates, HT Capabilities, HT information, Overlapping BSS scan parameters, Extended capabilities, RM enabled capabilities
Authentication	NONE	Authentication algorithm, Authentication sequence, Status code	NONE

The fuzzing method follows two basic principles. First, the frame subtype selected by the user via the CLI, say, Beacon, is being transmitted in batches of 128 frames. The first batch only contains the MAC header without a payload. Second, for all the subsequent batches, every odd frame in the same batch is identical to the rest except for the currently tested field, which contains a different seed; this means that each batch tests 64 diverse seeds. Precisely, for monitoring reasons, i.e., easily distinguishing the seed that caused a connection disruption, every pair of subsequent frames contain the same seed. Taking a beacon frame as an example, the batch that the fuzzer transmits comprises the same seed except for, say, the SSID, which receives random values generated by *Blab*.

For reasons of completeness, after all the fields have been seeded sequentially (for an *Authentication* frame, which with reference to Table 1 carries three fields, this process will require 4 × 128 frames), the fuzzer will transmit a last batch of 128 frames, every other one containing different random values in all of its fields. Next, the fuzzer will iterate for the same frame subtype. This process will continue until some kind of connection disruption between the target AP and the associated (target) STA is detected. As detailed in Section 4.3, to identify and log such connection disruption events for subsequent examination, a separate module has been implemented.

It is important to note that in the random mode and for Beacon and response (probe, association, reassociation) frames, the user can choose through the CLI to fuzz either the AP or the STA. Put another way, in the former case (AP); the respective frames carry as a destination and source MAC address that of the target AP and STA, respectively, while in the latter case, the MAC addresses are vice versa. When fuzzing an AP with such unsolicited frames, it is expected that the frames will be just ignored; this, however, is not to be taken for granted, as discussed in Section 6. On the other hand, when fuzzing an STA through such frames, namely, the attacker impersonates the AP, it may inflict quality of service issues to the STA, making the AP deauthenticate the STA; this behavior was also observed in the context of our experiments given in Section 6.

4.2. Control Frames

Apart from the management frames, as presented in Table 2, WPAxFuzz is also capable of fuzzing 802.11 control frames; at the time of writing, this functionality was still under development and testing. Due to the simplicity of the control frames structure, i.e., lack of payload in almost half of the cases and rather small size in bytes, each cycle of the fuzzing process dispatches three frame batches for the selected frame subtype, with the rest of the functionality being fundamentally identical to that of the management frames fuzzing detailed in Section 4.1.

- *Frame Control Flags*: First, the fuzzer transmits 128 frames with the 1-byte frame control flags receiving values randomly generated with *Blab*. Recall from Section 4.1 that every two consecutive frames are identical.
- *Payload overflow*: In the second stage, a payload produced by *Blab* is added to certain frame subtypes. The payload depends on the particular mode. Namely, for the *Standard* mode, *Blab* will generate a payload only to frame subtypes that do accommodate a payload based on the 802.11 standard. Regarding the *Random* mode, a payload is added irrespective of whether this frame subtype carries a payload or not with reference to the standard. As for the payload size in bytes in the *Standard* mode, it is prefixed and abides by the standard for each frame subtype, while in the *Random* mode, it is determined randomly by *Blab*.
- *Mixed*: The last stage combines the previous two into a single process, populating the frame control flags and payload fields simultaneously.

As shown in the third line of Table 2a, a special case of control frames is concerned with the so-called Control Frame Extension. The latter comprises a subset of control frames, where the subtype field of the Frame Control Field is set to 0x0110. These frames are detailed in Table 2b. Precisely, the distinction among the frames of Table 2b is performed by means of Frame Control Flags; each byte flag indicates a different control frame in the particular subset. Values from 0x02 to 0x0a are used by the standard for the frames in Table 2b, while the rest of the values are reserved. As a result, the fuzzing process for these frames is distinct; the Frame Control flags are immutable, and the fuzzing process happens only for the payload.

4.3. Monitoring Module

The monitoring module comprises two processes running in parallel. The first tracks down any deauthentication or disassociation frame communicated between the target AP and the corresponding STA. Upon such a frame being perceived, the fuzzing process pauses, and it is up to the user to manually reconnect the STA and resume the fuzzing if needed; the fuzzing cycle will recommence from the last preceding point, thus excluding the whole batch of frames, say, an SSID, which caused the disconnection. Note that the same category (SSID in the previous example) will also be excluded in the subsequent loops of the fuzzing process. The second monitoring feature addresses the case in which there is no disconnection event, but the STA transits to a "zombie" state; the connection seems alive, but the AP and the STA do not actually communicate; in this case, the STA does not respond to *ICMP echo requests*. If such a state is perceived, the fuzzing process pauses and prompts the user to reconnect the STA and resume; the "reconnect" warning will pop up and persist until the user manually restores the connection or terminates the fuzzer. It was observed, however, that in some rare cases, the STA does respond to *ping*, i.e., it is still connected to the AP, but it cannot access any Internet service. It is important to note that both the aforementioned monitoring processes (threads) can either execute on the same WNIC or separate ones.

Table 2. Control Frames based on the 802.11-2020 standard.

(a)

Control Frames Considered by the Fuzzer	**List of Fields**				
	MAC Header			**Payload**	**FCS**
	Frame Control Field	**RA**	**TA**		
Beamforming Report Poll	✓	✓	✓	Feedback Segment Retransmission Bitmap	✓
VHT/HE NDP Announcement	✓	✓	✓	Sounding Dialog Token, STA Info List	✓
Control Frame Extension	Collection of control frames in Table 2b				
Control Wrapper	✓	✓	✗	Carried Frame Control, HT Control, Carried Frame	✓
Block Ack Request (BAR)	✓	✓	✓	BAR Control, BAR Information	✓
Block Ack	✓	✓	✓	BA Control, BA Information	✓
Power Save-Poll (PS-Poll)	✓	✓	✓	✗	✓
Request to Send (RTS)	✓	✓	✓	✗	✓
Clear to Send (CTS)	✓	✓	✗	✗	✓
ACK	✓	✓	✗	✗	✓
Contention Free-End (CF-End)	✓	✓	✓	✗	✓
CF-End & CF-ACK	✓	✓	✓	✗	✓

(b)

Control Frame Extension	**List of fields**				
	MAC Header			**Payload**	**FCS**
	Frame Control Field	**RA**	**TA**		
Poll	✓	✓	✓	Response Offset	✓
Service period request (SPR)	✓	✓	✓	Dynamic Allocation Info, BF Control	✓
Grant	✓	✓	✗	Dynamic Allocation Info, BF Control	✓
DMG CTS	✓	✓	✓	✗	✓
DMG DTS	✓	✓	✓	NAV-SA, NAV-DA	✓
Sector sweep (SSW)	✓	✓	✓	SSW, SSW Feedback	✓
Sector sweep feedback (SSW-Feedback)	✓	✓	✓	SSW Feedback, BRP Request, Beamformed Link Maintenance	✓
Sector sweep Ack (SSW-Ack)	✓	✓	✓	SSW Feedback, BRP Request, Beamformed Link Maintenance	✓
Grant Ack	✓	✓	✓	Reserved, BF Control	✓

(a) Structure of each control frame subtype. The Frame Control Field includes the protocol version (indicates the 802.11 protocol and is typically equal to 00), a type equal to 1 (indicating a control frame), a subtype (designating the type of the control frame), and the Frame Control Flags. A check-mark designates that this control frame subtype contains the corresponding field, while an x-mark stands for the opposite. (b) Control frame subtypes in the control Frame Extension group. The Frame Control Field type is set to 0x01, the subtype is 0x0110, and the Frame Control Flags designate a different Control frame, say, the flag byte 0x04 indicates that the transmitted frame is a "Grant".

Figure 1 depicts a high-level view of the fuzzing process. The master process is responsible for the fuzzing operation, while two parallel threads are monitoring for deauthentication or disassociation frames and STA aliveness through *ping* requests, respectively. The latter threads are synchronized and can pause the execution of the main process. Further, the monitoring threads do not share a critical area in memory, so mutual exclusion is unneeded. Altogether, the monitoring threads are in an endless loop, while only the fuzzing process pauses and resumes depending on the case.

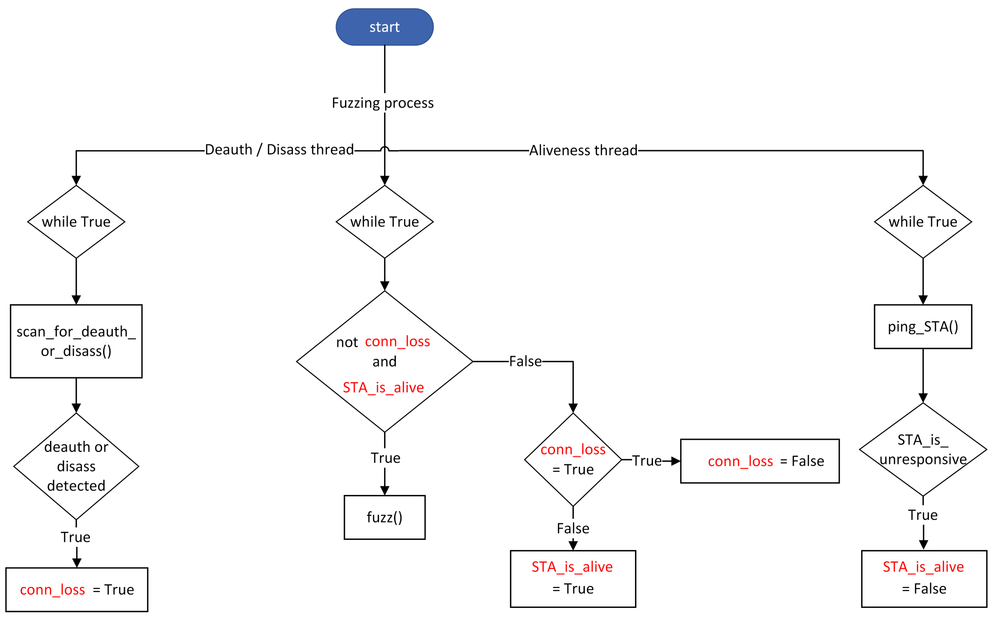

Figure 1. A bird's eye view of the fuzzing process. Note that the main process and the two threads run in parallel in an endless loop until the user aborts.

4.4. SAE

Based on the methodology given in Sections 4.1 and 4.2 and partly on the attacks detailed in [2], we created a specific fuzzing module exclusively destined for WPA3-SAE. For executing this module, three more parameters need to be entered in the configuration file of WPAxFuzz: the alternative radio band the AP operates, along with the corresponding MAC and Wi-Fi passphrase. Briefly, this module focuses on the so-called SAE Commit and SAE Confirm Authentication frames, which are exchanged during the SAE handshake. According to the standard [1] and with reference to the last line of Table 1, both these frames carry the *Authentication algorithm* (3), the *Authentication Sequence* (1 for Commit and 2 for Confirm), and a *Status code*, namely, a value between 0 and 65535, with 0 standing for "Successful". Note that in accordance with [1], *Status code* values between 1 and 129 (except 4, 8, 9, 20, 21, 26, 29, 36, 48, 66, 69-71, 90-91, 116, 124, and 127) designate a different failure cause, while the rest are reserved by the protocol.

In more detail, the current module, selected through WPAxFuzz's CLI, optionally capitalizes on the burst frame sending mode of [2]; namely, it sprays multiple frames, i.e., 128, at once toward the target AP. It comprises three different circles: (i) transmit SAE (Authentication) frames to the radio channel the target STA operates, (ii) transmit SAE frames to a different radio channel than that of the target STA(s), and (iii) either of the previous, but with the burst mode enabled. Further, each fuzzing cycle is executed over seven diverse variants based on the stateless approach of the WPA3-SAE authentication procedure as follows.

1. An empty SAE auth frame.
2. A valid (well-formed) SAE-Commit frame followed by (1).
3. A valid SAE-Commit frame, followed by a SAE-Confirm frame with the so-called *Send-Confirm* field set to 0. Recall that the *Send-Confirm* field carries the counter of the already sent Confirm frames, hence acting as an anti-replay counter.
4. As with (3), but the value of the *Send-Confirm* field is set to 2. This specific value (2) was chosen because, as explained in [2], using a value between 2 and 65,534 for this field, "the AP disconnected the target STA after 20 sec on average".
5. A valid SAE-Commit frame.
6. A valid SAE-Confirm frame with the *Send-Confirm* field equal to 0.
7. As with (6), but the *Send-Confirm* field's value is set to 2.

As with the Management frames module, the present one uses the same monitoring logic and is split into two different types of fuzzing procedures, namely, *Standard* and *Extensive*. The former is rooted in the methodology of the work in [2]; that is, it assigns specific values to the SAE frame fields in an effort to forcibly disconnect a STA or paralyze the AP. For instance, the *Authentication algorithm* field is fuzzed using specific, cherry-picked values, including 0, 1, 2, and 200, and not random ones generated by *Blab* or otherwise. On the other hand, the *Extensive* mode concentrates on grindingly testing every valid SAE field combination; that is, every possible value in the range of 0 to 65535, making it far more time-consuming vis-à-vis the *Standard* mode.

It is noteworthy to mention that while the management frames module of the fuzzer also examines Authentication frames, there exists a significant difference compared to the current one. That is, the two modules implement a diverse fuzzing logic: the management frames mode uses *Blab* from the construction of the seeds, while the SAE mode uses legitimate values within the range defined by the 802.11-2020 standard. With reference to Section 6, this is the main reason why the former mode managed to discover new misconfigurations in the Authentication frames; the latter mode only confirmed still unpatched issues originally identified in [2].

5. Attack Module

No less important, the tool offers an attack module serving a dual purpose; first, for quickly verifying the outcomes of the fuzzing process toward any device, and second, for aiding in brewing up exploits. That is, after the fuzzing process is over, this module can be used to automatically collect all the problematic frames from the log files and transmit them through the attacker's WNIC toward the target (victim) device. Note that at the time of writing, this module excluded SAE.

In further detail, the attack module includes two separate functionalities. The first gathers all the frames that caused a connection disruption during the fuzzing process from the corresponding log files and transmits them one after the other; this way, only the truly problematic frames (true positives) are kept. In the case where a frame is found positive, e.g., it inflicts DoS to the target device, an exploit is automatically crafted at runtime, as presented in Listing 1. This facilitates the researcher to easily identify potential vulnerabilities in a particular device. The second functionality, which at the time of writing is in beta version, transmits all the frames that were sent until the moment a connection disruption at the STA side was sensed. This can be used to trace for frame sequences (at least two frames of the same subtype) that jointly interrupt the connection between the AP and the STA. The maximum length of this process is up to one cycle, with reference to Table 1.

Listing 1. Outline of the generated exploits.

```
from scapy.all import Dot11, RadioTap, sendp
dot11 = Dot11(type=0, subtype={SUBTYPE}, addr1={DESTINATION_MAC}, addr2={SOURCE_MAC
    }, addr3={AP_MAC})
MAC_header = RadioTap()/dot11
payload = {SEED}
frame = MAC_header / payload
print('\n- - - - - - - - - - - - - - - - -')
print('Testing the exploit')
print('- - - - - - - - - - - - - - - - - ')
while True:
   sendp(frame, count=1, iface={ATT_INTERFACE}, verbose=0)
```

6. Results

WPAxFuzz was used against seven off-the-shelf APs by different renowned vendors. Due to time constraints regarding the reporting and patching of the identified issues by the vendors, only the management frames were fuzzed. Additionally, we rechecked vulnerabilities initially reported in [2] through the fuzzer's SAE module. Specifically, the goal of the testing process was to identify issues that lead to DoS; that is, the target STA is either disconnected from the AP or abruptly transits to a no-Internet access state; in either case, the STA needs to be manually reconnected to the AP. In all the fuzzing tests, only one STA was associated with the tested AP. Namely, the target STA was a laptop machine on Windows 10 Pro, equipped with an Intel AX200 WNIC with driver version 22.140.0.3. The fuzzer exploited an Alfa AWUS036AC USB Wi-Fi adapter connected to a Kali Linux machine. The results regarding the management frames are recapitulated in Table 3. As seen in the table, the APs are categorized by chipset vendor, namely, Broadcom, Qualcomm, Intel, and MediaTek. Each AP received its firmware update on 1st of April, 2022. All the APs supported both WPA2 and WPA3 except TP-Link TL-WA1201, which was tested only on WPA2.

Table 3. Results for management frames (random mode). Each frame subtype was tested for 1 h. The star exhibitor denotes issues found when targeting the STA, namely, the destination MAC address is that of the STA. Unstable issues are marked with a dagger symbol. Vendor "Vendor5" remains undisclosed due to policy reasons related to bug bounty rewards. However, for the sake of verifiability, further information about this vendor and the respective AP model can be given after contacting the authors.

(a) Issues Found in WPA3-enabled APs.

Frame Subtype	**WPA3**						
	Broadcom			**Qualcomm**		**MediaTek**	**Intel**
	ASUS	**TP-Link**	**Xiaomi**	**Linksys**	**Vendor5**	**D-Link**	**Netgear**
	RT-AX88U 3.0.0.4.386.48631	**AX10v1 V1_220401**	**Mi AX1800 3.0.34**	**MR7350 1.1.7.209317**	**Undisclosed -**	**DIR-X1860 v1.10WWB09_Beta**	**RAX40 1.0.4.102**
Beacon	✗	✗	✓	✓	✓*	✓†	✗
Probe Req	✗	✗	✗	✗	✗	✗	✗
Probe Res	✗	✗	✗	✗	✓*	✗	✗
Assoc Req	✗	✗	✗	✗	✓	✗	✗
Assoc Res	✗	✗	✗	✗	✗	✗	✗
Reassoc Req	✗	✗	✗	✗	✓	✗	✗
Reassoc Res	✗	✗	✗	✗	✗	✗	✗
Auth	✓†	✗	✓	✓	✓	✓†	✗

(b) Issues found in WPA2-enabled APs.

Frame Subtype	**WPA2**				
	Broadcom			**Qualcomm**	**MediaTek**
	ASUS	**TP-Link**		**Vendor5**	**D-Link**
	RT-AX88U 3.0.0.4.386.48631	**AX10v1 V1_220401**	**TL-WA1201 1.0.2_20220103**	**Undisclosed -**	**DIR-X1860 v1.10WWB09_Beta**
Beacon	✗	✗	✓*	✓*	✗
Probe Req	✗	✗	✓	✗	✗
Probe Res	✗	✗	✓*	✓*	✗
Assoc Req	✗	✗	✓	✓	✓
Assoc Res	✗	✗	✗	✗	✗
Reassoc Req	✗	✗	✓	✓	✓
Reassoc Res	✗	✗	✗	✗	✗
Auth	✓†	✗	✓	✓	✓

Regarding the management frames, with reference to Section 4.1, all the tests were performed in the *Random* mode. Specifically, each management frame in Table 1 was fuzzed for 1 hour, regardless of the identified issues and the exploited fields in each frame subtype. We only consider if the specific frame subtype yielded an issue or not. This means that a check-mark in Table 3 may correspond to more than one issue identified for the same frame subtype, but for a diverse (frame) field. Overall, for the examined frame type, 27 issues were found; that is, 12 and 15 for WPA3 and WPA2, respectively, suggesting that such issues are not scarce and apply to both certifications to almost the same degree. Recall that each one of the identified flaws leads to DoS, requiring user intervention to reinstate the connection. Therefore, all of them are positioned at least on the scale between medium and high severity in terms of the Common Vulnerability Scoring System (CVSS).

An additional important consideration is that, as shown in Table 3, a foursome of the discovered issues was characterized as unstable. This means that while the fuzzer did detect a connection disruption between the STA and the AP, the corresponding exploit generated by the attack tool was ineffective. It is assumed that such issues are either incidental or most probably emerge following the transmission of a certain (sequential) pattern of frames carrying specific seeds, which apparently is complicated to perceive and analyze. As already pointed out in Section 5, to investigate such situations, the attack module is currently enriched with functionality that transmits all the frames that were sent until the moment a connection disruption is observed.

With respect to SAE frames, we evaluated the fuzzer against all seven APs of Table 3. The results revealed that Xiaomi and Linksys APs are still susceptible to all the corresponding assaults described in § 4 and 6 of [2], while the Vendor5's AP is vulnerable to the so-called "Doppleganger", "Back to the future", "Bad sequence number", and "Radio confusion revisited" attacks; for more details on these attacks, the reader is referred to [2]. This outcome suggests that even after the release of a Common Vulnerabilities and Exposures (CVE) ID, in this case, CVE-2021-37910, CVE-2021-40288, CVE-2021-41753, and CVE-2021-41788, the patching of vulnerabilities by chipset and AP vendors may take considerable time even for contemporary devices.

By following a Coordinated Vulnerability Disclosure (CVD) process, we are in the stage of evaluating the corresponding exploits and divulging the discovered issues to each affected vendor. Thus far, we have received affirmative feedback from two vendors, who were able to reproduce and verify our results. Upon their release, all the relevant patches and/or CVE IDs will be announced in the GitHub repository of the tool.

Regarding the results in Table 3, an important point deserving further investigation is that apart from the AP's chipset, the results of the fuzzing process may differ depending on the STA's WNIC as well. Precisely, it was observed that the fuzzing process affected STAs equipped with Intel WNICs the same way, while sometimes differences may be perceived between WNICs embedded in iOS and Android products. For instance, taking as an example Vendor5's AP in Table 3, while certain malformed Beacon and Probe request frames always caused a connection disruption for STAs equipped with an Intel WNIC chipset, the same frames were sometimes ineffective against an iPhone 11, which integrates a Broadcom chipset. Another noteworthy remark is that the number of discovered issues may differ significantly among APs incorporating a chipset from the same vendor; this is apparent in Table 3 for both WPA2 and WPA3 APs. The reader should also keep in mind that when testing the Beacon frames and a glitch is found, then possibly every associated STA to the AP will experience connectivity issues. This is because a Beacon frame is sent to the broadcast address.

As a concluding remark, for the most part, the root cause of these latent vulnerabilities is owed to business logic flaws; that is, design and implementation flaws in software applications, in our case, in the respective chipset's firmware. Among others, this may be due to either a software bug, a misconfiguration, an unwitting supposition, or a misconception regarding the standard during the software development phase. In particular, an AP cannot properly handle different types of incoming invalid or malformed frames originating from a spoofed MAC address. Under such a circumstance, the typical reaction of the AP is to abruptly disconnect the legitimate STA(s) or drive them to a no-Internet access, "zombie" state. It is worth noting that, initially, the testing methodology followed by the current work was positively assessed in [2], and the aim of the WPAxFuzz tool is to extend this methodology across all the frame types of 802.11.

7. Conclusions

Despite the seventh generation of Wi-Fi already being on the horizon, the literature lacks a full-grown fuzzing tool to test against latent misconfigurations and vulnerabilities in Wi-Fi equipment, specifically APs. The work at hand aspires to fill this noticeable gap by contributing and evaluating such a software tool, dubbed WPAxFuzz, which is offered to the community as open source. The tool is capable of fuzzing both WPA2- and WPA3-compatible products and incorporates several components that aid in the verification and further investigation of the identified issues. To this end, WPAxFuzz not only provides different fuzzing modes depending on the case but also (a) embeds a monitoring functionality for detecting on-the-fly connection or service disruptions at the STA side and (b) an attacking module aiming at affirming the discovered issues and crafting the corresponding exploits automatically. We detail WPAxFuzz's functionality and evaluate it against a significant mass of APs by renowned vendors. The derived results, some of them already successfully replicated and verified by the respective vendors, includin more than two tens of issues discovered to be potentially exploitable across different types of management frames, demonstrating the capacity of the tool. An interesting avenue for future work is to expand WPAxFuzz to also embrace the rest of 802.11 management frames, namely, deauthentication and disassociation, as well as data ones.

Author Contributions: Conceptualization, G.K. and E.C; methodology, G.K., V.K. and E.C.; software, V.K. and A.D.; validation, G.K., E.C. and C.Z.; investigation, E.C., G.K., V.K. and C.Z.; writing—original draft preparation, G.K., V.K. and E.C.; writing—review and editing, G.K., V.K., E.C. and C.Z.; project administration, G.K., E.C. and C.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: In agreement with some of the vendors, WPAxFuzz will be uploaded to a public GitHub repository after the reported vulnerabilities have been patched. The discovered exploits will also be published in the same repository.

Acknowledgments: We are thankful for the bug bounty reward regarding the identified vulnerabilities.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACK	Acknowledgment
AP	Access Point
CLI	Command-Line Interface
CVD	Coordinated Vulnerability Disclosure
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DoS	Denial of Service
FCS	Frame Check Sequence
ICMP	Internet Control Message Protocol
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
JSON	JavaScript Object Notation
MAC	Media Access Control
OS	Operating System
PMF	Protected Management Frames
RA	Receiver Address
SAE	Simultaneous Authentication of Equals
SSID	Service Set Identifier
STA	Station
TA	Transmission Address

VM	Virtual Machine
Wi-Fi	Wireless Fidelity
WNIC	Wireless Network Interface Card
WPA2	Wi-Fi Protected Access 2
WPA3	Wi-Fi Protected Access 3

References

1. IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems—Local and Metropolitan Area Networks–Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. In *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*; IEEE: New York, NY, USA, 2021; pp. 1–4379. [CrossRef]
2. Chatzoglou, E.; Kambourakis, G.; Kolias, C. How is your Wi-Fi connection today? DoS attacks on WPA3-SAE. *J. Inf. Secur. Appl.* **2022**, *64*, 103058. [CrossRef]
3. Vanhoef, M.; Ronen, E. Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–21 May 2020; pp. 517–533. [CrossRef]
4. Chatzoglou, E.; Kambourakis, G.; Kolias, C. Empirical Evaluation of Attacks Against IEEE 802.11 Enterprise Networks: The AWID3 Dataset. *IEEE Access* **2021**, *9*, 34188–34205. [CrossRef]
5. Schepers, D.; Ranganathan, A.; Vanhoef, M. On the Robustness of Wi-Fi Deauthentication Countermeasures. In Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Antonio, TX, USA, 16–19 May 2022; pp. 245–256.
6. Chatzoglou, E.; Kambourakis, G.; Kolias, C.; Smiliotopoulos, C. Pick Quality Over Quantity: Expert Feature Selection and Data Preprocessing for 802.11 Intrusion Detection Systems. *IEEE Access* **2022**, *10*, 64761–64784. [CrossRef]
7. Chatzoglou, E.; Kambourakis, G.; Kolias, C. WiF0: All Your Passphrase Are Belong to Us. *Computer* **2021**, *54*, 82–88. [CrossRef]
8. Vanhoef, M. A Time-Memory Trade-Off Attack on WPA3's SAE-PK. In Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop, Nagasaki, Japan, 30 May 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 27–37. [CrossRef]
9. Li, J.; Zhao, B.; Zhang, C. Fuzzing: A survey. *Cybersecurity* **2018**, *1*, 6. [CrossRef]
10. Manès, V.J.; Han, H.; Han, C.; Cha, S.K.; Egele, M.; Schwartz, E.J.; Woo, M. The Art, Science, and Engineering of Fuzzing: A Survey. *IEEE Trans. Softw. Eng.* **2021**, *47*, 2312–2331. [CrossRef]
11. The WPAxFuzz Tool. Available online: https://github.com/efchatz/WPAxFuzz (accessed on 7 October 2022).
12. Nikolai Tschacher. Dragonfuzz. Available online: https://github.com/NikolaiT/dragonfuzz (accessed on 31 July 2022).
13. Marais, S.; Coetzee, M.; Blauw, F. Simultaneous Deauthentication of Equals Attack. In Proceedings of the Security, Privacy, and Anonymity in Computation, Communication, and Storage: SpaCCS 2020 International Workshops, Nanjing, China, 18–20 December 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 545–556. [CrossRef]
14. Aki Helin. Blab—A Grammar-Based Data Generator. Available online: https://gitlab.com/akihe/blab (accessed on 31 July 2022).