IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 6, 2021, accepted February 18, 2021, date of publication February 23, 2021, date of current version March 4, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3061609

Empirical Evaluation of Attacks Against IEEE 802.11 Enterprise Networks: The AWID3 Dataset

EFSTRATIOS CHATZOGLOU[1], GEORGIOS KAMBOURAKIS[2], AND CONSTANTINOS KOLIAS[3]
[1]Department of Information and Communication Systems Engineering, University of the Aegean, 83200 Samos, Greece
[2]European Commission, Joint Research Centre (JRC), 21027 Ispra, Italy
[3]Department of Computer Science, University of Idaho, Idaho Falls, ID 83402, USA

Corresponding author: Georgios Kambourakis (gkamb@aegean.gr; georgios.kampourakis@ec.europa.eu)

ABSTRACT This work serves two key objectives. First, it markedly supplements and extends the well-known AWID corpus by capturing and studying traces of a wide variety of attacks hurled in the IEEE 802.1X Extensible Authentication Protocol (EAP) environment. Second, given that all the 802.11-oriented attacks have been carried out when the defenses introduced by Protected Management Frames (PMF) were operative, it offers the first to our knowledge full-fledged empirical study regarding the robustness of the IEEE 802.11w amendment, which is mandatory for WPA3 certified devices. Under both the aforementioned settings, the dataset, and study at hand are novel and are anticipated to be of significant aid towards designing and evaluating intrusion detection systems. Moreover, in an effort to deliver a well-rounded dataset of greater lifespan, and under the prism of an attacker escalating their assault from the wireless MAC layer to higher ones, we have additionally included several assaults that are common to IEEE 802.3 networks. Since the corpus is publicly offered in the form of raw cleartext pcap files, future research can straightforwardly exploit any subset of features, depending on the particular application scenario.

INDEX TERMS IEEE 802.11, PMF, 802.11w, WPA2, WPA3, wireless security, attacks, dataset.

I. INTRODUCTION

With the proliferation of smart portable devices like smartphones, tablets, and IoT devices, WiFi (IEEE 802.11) has been established as the dominant technology for connecting digital devices in Wireless Local Area Networks (WLAN). Actually, this trend came simply to reinforce the already strong penetration of WiFi in home, office, enterprise, connected vehicles, and even mission-critical settings. Naturally, the security of WiFi-based networks, as well as the 802.11 protocol itself, lies in the epicenter of systematic academic and industry-laden research. Yet, despite more than 20 years of continuous amendments and remediating measures, vulnerabilities discovered even for the newest versions of the protocol attest that the security of this wireless technology is a non-trivial subject that still continues to be an open and challenging problem.

External protection mechanisms should therefore be considered as imperative components of 802.11 wireless networks for defending against known or unknown exploits. Being one of such mechanisms, the conventional intrusion detection processes tend to focus their inspection onto higher-layers. Thus, corresponding systems disregard attacks that are initiated from lower-layers, including the attacks that are native to the 802.11, which reside at the link layer of the OSI protocol stack. Towards contributing to the development of robust detection protection mechanisms, our previous efforts resulted in the contribution of AWID, a corpus containing normal and malicious traffic. Spurred by the wide adoption of that dataset as a benchmark tool for intrusion detection in wireless networks, this work presents its extension with several new features. The main differences between the newest version of the AWID dataset are summarized as follows:

1) The data are offered in pcap format, along with their Pairwise Master Key (PMK) and TLS keys. This format provides the flexibility for custom-tailored feature extraction depending on the specific needs of researchers.
2) Newly discovered 802.11-specific attacks are added, including the Krack and Kr00k ones.

The associate editor coordinating the review of this manuscript and approving it for publication was Tai-hoon Kim.

3) The main focus is on enterprise versions of the protocol, which typically offer stronger security mechanisms, including the use of Protected Management Frames (PMF) introduced with the 802.11w amendment, and support for alternative network architectures.
4) Multilayer attack scenarios are embraced; in particular, a set of attacks included in the dataset gets initiated at the link layer by taking advantage of 802.11 vulnerabilities but unfolds across multiple protocols that operate in multiple layers.
5) Detailed documentation of each scenario along with empirical observations about the effect of attacks in real-life equipment. For the latter point, the concentration is on PMF robustness.

The remainder of the paper is structured as follows. The next section categorizes and portrays the attacks included in the dataset. Section III details on the testbed and the data collection procedure, while section IV evaluates empirically specific attacks included in the dataset. Key observations regarding the resilience of PMF are given in section V. The next to last section addresses the related work, while the last one concludes and provides pointers to future work.

II. DESCRIPTION OF ATTACKS

This section categorizes the attacks implemented during the creation of the dataset[1] and succinctly describes each one of them. Details on each attack are given later in section III-B.

As already mentioned, differently to the original AWID dataset [1], and in addition to the inclusion of some modern attacks, the present corpus includes a number of assaults that abuse vulnerabilities of higher-layer protocols. Nevertheless, each attack in the dataset is initiated from 802.11, and the main focus is the attack escalation factor, and in a parallel dimention, the delivery of a more well-rounded dataset of increased lifespan. Simply put, the following scenerio is an exemplar of such escalation dynamics: We assumed that first, the aggressor had successfully carried out, say, a classic Evil_Twin attack against the 802.11 network and managed to steal the credentials of a given STA. This allows them to gain access to the wired network and potentially escalate their attacks to other protocols, e.g., perform a SQL injection.

As with AWID, physical (PHY) layer attacks are considered out of scope of this work. AWID [1] classified the various attacks in three categories. Given that IEEE 802.1X Extensible Authentication Protocol (EAP) does not suffer from significant (and publicly known) key cracking and key retrieving attacks as WEP [2] does, this work classifies the various attacks in four categories, namely (a) 802.11 specific, (b) attacks against the local area network, (c) attacks initiated from the local area network but aimed against an external target, and (d) multi-layer attacks abusing vulnerabilities of multiple protocols.

[1]The dataset, coined *"AWID3"*, is publicly available on the AWID website at http://icsdweb.aegean.gr/awid/.

A. 802.11 SPECIFIC ATTACKS

This category embraces attacks that are solely exercised in the MAC layer of 802.11. We consider two kinds of assaults; Denial of Service (DoS) and Key reinstallation. The first typically aim at disrupting the connection between the basic building blocks of an 802.11 network, namely the Station (STA) and Access Point (AP), by either assaulting specific devices or stressing the resources of the network and connected devices. Most of these attacks are well-known and are included in the dataset for the sake of completeness. On the other hand, a key reinstallation assault has the purpose of reinstalling an already in use pairwise or group key in the device. As a rule, most attacks in this category exploit unprotected management frames of 802.11.

- *Deauthentication*: It is one of the most widespread and straightforward attacks in 802.11 as it can be easily mounted even by a script kiddie, and its results are immediate. That is, in the absence of IEEE 802.11w [3], if an STA receives a deauthentication frame (0×000c) stemming from the AP, it must disconnect. Therefore, in such an attack incident, the source MAC address of the deauthentication frame is normally spoofed. When the attack ends, the STA will immediately try to reconnect to the preferred AP, typically to that being closer and has the strongest signal. Therefore, this attack is often used as a stepping stone for launching more advanced ones, including Evil_Twin. Even more, the aggressor can send unprotected spoofed deauthentication frames, targeting the broadcast MAC address of a specific AP. This would disconnect all STAs from the network. On the bright side, as indicated by empirical observations discussed more extensively in section V, most manufacturers discard the latter type of frame; thus this attack variant will probably fail in newer devices.
- *Disassociation*: A disassociation frame is indented to disassociate the STA from the AP or vice-versa. This can occur for several reasons, including roaming, i.e., when the STA is about to be associated with another AP. This attack is quite similar to the deauthentication one and can exploit the broadcast MAC address of the AP as well; the main difference is that the attacker transmits spoofed unprotected frames of a different type, namely the disassociation frames (0×000a) instead. Moreover, the DoS effects of receiving this type of frames are, in theory, expected to be short lived in comparison to deauthentication. A mix of both deauthentication and dissassociation attacks, namely the *Amok* as described by the MDK tool, has been included as well.
- *(Re)Association*: The association process between the STA and AP takes place after successful open authentication. It involves the exchange of a pair of association request/response frames (0×0000/0×0001), where the response stemming from the AP designates a success or failure, and if successful, a unique Association Identifier (AID). Regarding reassociation request/response frames (0×0002/0×0003), they are used when an STA

is already associated with an AP and wishes to associate with another in the same Extended Service Set (ESS). Generally, in the presence of PMF, it can be argued that the (re)association request flooding attack works similarly to the deauthentication/disassociation one.

- *Rogue AP:* The attacker mounts an Evil_Twin, which is one of the cardinal assaults against 802.11. That is, they deploy a rogue AP in WPA2-PSK authentication mode with PMF disabled. The phony AP operates on the same channel and masquerades itself under the same MAC address and SSID as the target one. If that AP emits a stronger signal than the legitimate, an STA may be lured into connecting to the rogue AP.
- *Krack:* This exploit was introduced by the authors in [4]. It takes advantage of the fact that retransmissions of either the first or the third message of the 4-way handshake should occur if the AP does not receive message two or four, respectively. Based on this, the attacker acting as a man-in-the-middle (MitM) [5], blocks message four from reaching the AP, thus making the AP willingly re-transmit the same message toward the STA. This, however, would induce a reinstallation of the same Pairwise Transient Key (PTK) / Group Transient Key (GTK) / Integrity Group Temporal Key (IGTK) at the STA side, meaning that as the STA sends its next data frame, the data WPA2-confidentiality protocol will reuse the nonces transmitted during the original 4-way handshake. On top of that, the authors pinpointed that devices running wpa_supplicant v2.4 and v2.5 and unpatched versions of at least Android 6.0 and Android Wear 2.0 reinstalled an all-zero key. Windows and iOS do not accept retransmissions of message 3, so they were unaffected. This vulnerability has also been documented in CVE-2017-13077 to CVE-2017-13081.
- *Kr00k:* This vulnerability was exposed by ESET researchers [6], [7] and formally defined as CVE-2019-15126. It is related to Krack, applies to WPA2 as well, but the attacker is not required to be authenticated and associated to the wireless network. The vulnerability is rooted in the fact that as soon as an STA is disassociated, the TK is cleared in memory, i.e., set to all-zero. While this is normal, the researchers observed that all data frames left in the Wireless Network Controller's (WNIC) egress queue might be then transmitted while encapsulated with this static all-zero key. The attacker may take advantage of this situation by either causing a disassociation or by just passively eavesdropping on the wireless medium for data frames transmitted after legitimate disassociation events. In the first case, by recurrently inflicting disassociations, which in turn automatically leads to reassociations, the attacker can capture more data frames of this kind. Kr00k affects unpatched devices with chips by Broadcom and Cypress. The same vulnerability was also exposed for Qualcomm and MediaTek Wi-Fi chips, but in this case, the buffered frames were unencrypted even though the encryption flag was set to 1 [8]. This vulnerability has been documented in CVE-2020-3702.

B. ATTACKS AGAINST LOCAL NODES

The present category includes well-known attacks composed of a limited number of steps. They mainly take effect at a higher-layer, say, the application layer, but are initiated by a wireless malicious or compromised node and are targeted against benign nodes in the local network

- *SSH brute force*: A legacy SSH brute force attack exercised against a RADIUS server.
- *Botnet*: The attacker uses a ready to deploy a piece of the malware and devises a way to infect with it a number of STAs. If the victim executes such a file, it will connect back to a command and control (C2) server owned by the botherder and become a bot. The attacking behaviors do not include attempts of the bots against external targets.
- *Malware*: This attack also uses a file containing malicious code. In this case, however, the code aims at exploiting a certain vulnerability of the OS or the app that is targeting to, with the ultimate purpose of enabling another assault, including ransomware or installing a backdoor.

C. ATTACKS AGAINST EXTERNAL NODES

This category similarly encompasses customary attacks comprised by a limited number of steps and initiated by malicious or compromised local clients. However, in this case, the assault target lies outside the intranet.

- *SQL injection*: This attack is realized after having the aggressor insert a malformed SQL query instead of the expected input in a web form of the target. The vulnerability lies in the fact that the web application does not validate the format and structure of user inputs. Thus, the malicious inputs carry interpretable SQL code to the backend database, where they get executed with the aim to extract extraneous information or corrupt the database. SQL injection attempts can be identified by examining the data portion of HTTP POST request messages or the querystring in HTTP GET request messages.
- *SSDP amplification*: This reflection/amplification volumetric Distributed DoS (DDoS) attack utilizes the Simple Service Discovery Protocol (SSDP) that comprises the basis of Universal Plug and Play (UPnP). It potentially entangles any device that responds to UPnP requests, typically those carrying the ''generic'' ST query types *ssdp:rootdevice* or *sspd:all*. Put simply, the attacker recruits the STAs of the WLAN to send an upsurge of SSDP packets to each SSDP-enabled device, designating as source IP address that of the victim. Then, every device responds to the victim with a much larger packet, eventually resulting in DoS.

D. MULTI-LAYER ATTACKS

This class refers to multi-step attacks that manipulate mechanisms from at least two alternative layers. This highlights

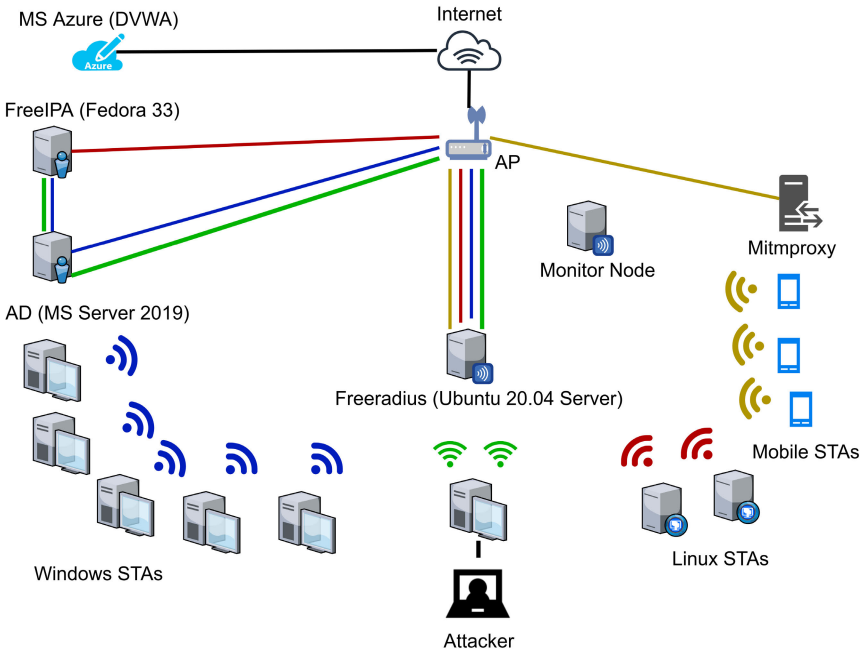


FIGURE 1. High-level view of the testbed network topology. The mitmproxy run as a service on the monitor node. The network traffic flow for disparate categories of machines is shown in different color.

on the fact that corporate users cannot mindlessly trust the medium that connects them to the internet. In this setting, the following two scenarios are considered.

- *Evil_Twin*: A variant of the Rogue AP attack. In this case, the rogue AP is deployed in open authentication mode operating on the same channel and SSID as the target one. The equipment of an unaware user may automatically connect to the benign posing AP. After that, the user is redirected to a specific web page that looks identical to a legitimate Wi-Fi's captive portal webpage. This would enable the aggressor to stealthily steal the user's login credentials. This attack scenario mandates DNS spoofing as well.
- *Website_spoofing*: The assailant clones the front webpage of a popular website, say, Instagram. Then, they perform ARP and DNS spoofing to redirect the users to their fake webpage instead of the original one.

III. DATASET

This section details the dataset created in regard to the used methodology, its structure, and contents. As already pointed out, the current dataset attempts to complement the original AWID regarding enterprise networks and update it with the inclusion of new and multi-layered attacks. To this direction, the main protocols used in this dataset were IEEE 802.1X EAP [9], 802.11w [3], and IEEE 802.11ac-2013, also known as Wi-Fi 5.

A. TESTBED

For the purposes of data gathering, and as depicted in figure 1, we created a physical lab that realistically emulates a typical enterprise infrastructure. In total, we utilized 16 different physical devices and VMs. From them, ten were used as client STAs, while one more laptop STA was operated by the mobile attacker. Two of the client STAs were running Ubuntu 20.04 desktop, five more had MS Windows 10 Pro or Enterprise, and the rest three were Samsung S20 FE on Android v10, Samsung Note 4 on Android 6.0.1, and iPhone 6s on iOS v14.2. All STAs received their last OS update on the 4th of Dec. 2020 and were locked from installing future updates. Another three devices were acting as an MS Windows Server 2019, a Fedora 33 Server, and a Ubuntu 20.04 Server. The rest of the three devices were the AP, the monitor node, and a dockerized version of the Damn Vulnerable Web Application (DVWA) residing in MS Azure. The role and hardware configuration of all the above-mentioned machines are summarized in table 1.

Specifically, to enable 802.1X EAP authentication methods, a RADIUS server running FreeRADIUS [10] in v.3.0.20 was installed on an Ubuntu 20.04 Server VM, and new certificates for each client STA and the RADIUS server have been created. As summarized in table 2, regarding 802.1X authentication, different methods were used, namely EAP-TLS, EAP-TTLS/PAP, EAP-PAP, EAP-TTLS/MSCHAPv2, and EAP-PEAP/MSCHAPv2. If present, the "/" separates between the server and client side authentication method, respectively. It is to be noted that these authentication methods were the same for each STA, for the whole duration of the dataset recording process.

The AP was an ASUS RT-AC68U in v3.0.0.4.384.81049[2] operating at 5GHz. PMF was enabled in the AP as

[2]Newest firmware versions up to v3.0.0.4.386.40558 caused increased CPU usage in the AP, even with a single STA connected to it. This strange behavior was also confirmed by relevant forums [11], [12].

TABLE 1. Devices used in the creation of the dataset. [1,2,3] Runs on a Ryzen 1700X, Ryzen 2700, Intel 4770K physical machine, respectively. 4 Refers to the MAC and IP address of mitmproxy. The letters "V" or "W" in the 7th column designate a Vendor- or Windows-based driver, respectively.

Node	Type	Brand	OS	Ver.	WNIC	Driver ver.	CPU/RAM	IP Address	MAC Address
AP	Wireless Router	ASUS RT-AC68U	Linux	2.6.36.4brcmarm	Broadcom BCM4708A0	N/A	Dual-core 800 MHz, 256 MB	192.168.2.1	0C:9D:92:54:FE:30 / 0C:9D:92:54:FE:34
Client STA 1	VM[1]	Custom	Windows 10 Pro	20H2	TP Link AC 1300 T4U v3 (USB)	1030.38.712.2019-W	Dual-core CPU, 8 GB DDR4	192.168.2.73	50:3E:AA:E4:01:93
Client STA 2	VM[2]	Custom	Windows 10 Pro	20H2	Linksys WUSB6100M (USB)	11.1.0.268 (2018)-V	Dual-core CPU, 16 GB DDR4	192.168.2.125	24:F5:A2:EA:86:C3
Client STA 3	VM[1]	Custom	Windows 10 Enterprise	20H2	TP Link AC 1300 T4U v3 (USB)	1030.22.202.2018-V	Dual-core CPU, 8 GB DDR4	192.168.2.190	50:3E:AA:E3:1F:BE
Client STA 4	VM[2]	Custom	Windows 10 Pro	20H2	Alfa AWUS 1900 (USB)	1030.38.712.2019-W	Dual-core CPU, 16 GB DDR4	192.168.2.184	00:C0:CA:A8:29:56
Client STA 5	Desktop	Custom	Ubuntu 20.04 LTS	5.4.0-56-generic	Gigabyte GC-WBAX200 (Intel AX200) (PCI-e)	46.3	Ryzen 2700 3.2GHz, 16 GB DDR4	192.168.2.254	A4:B1:C1:91:4C:72
Client STA 6	Desktop	Custom	Ubuntu 20.04 LTS	5.4.0-56-generic	Qualcomm Atheros QCA 6174A (PCI-e)	12.0.0.722(2018)	Ryzen 1700x 3.4GHz, 16 GB DDR4	192.168.2.41	94:E9:79:82:C5:77
Client STA 7	VM[2]	Custom	Windows 10 Enterprise	20H2	Alfa AWUS 1900 (USB)	1030.38.712.2019-V	Dual-core CPU, 16 GB DDR4	192.168.2.42	00:C0:CA:A8:26:3E
Client STA 8	Smartphone	Samsung S20 FE	Android 10	N/A	N/A	N/A	Exynos 990, 8 GB	192.168.2.160	B4:CE:40:C8:33:81
Client STA 9	Smartphone	iPhone 6s	IOS 14.2	N/A	N/A	N/A	Apple A9, 2 GB	192.168.2.19	88:66:A5:55:A2:D4
Client STA 10	Smartphone	Samsung Note 4	Android 6.0.1	N/A	N/A	N/A	Exynos 5433, 3 GB	192.168.2.247	A4:08:EA:2A:9A:01
RADIUS Svr	VM[3]	Custom	Ubuntu Server 20.04 LTS	5.4.0-56-generic	Realtek RTL 8111	N/A	Single-core CPU, 6 GB DDR3	192.168.2.120	00:0C:29:1D:70:F5
FreeIPA Svr (Domain Controller) / DNS / Samba Shared Network)	VM[3]	Custom	Fedora Server 33	5.9.11-200.fc33.x86_64	Realtek RTL 8111	N/A	Single-core CPU, 6 GB DDR3	192.168.2.130	00:0C:29:CF:08:AA
Active Directory (AD) / Domain Controller / DNS	VM[3]	Custom	Windows Server 2019	1809 (Build 17763.1613)	Realtek RTL 8111	N/A	Single-core CPU, 4 GB DDR3	192.168.2.239	00:0C:29:EC:72:02
Attacker	Laptop	Dell 5567	Ubuntu 18.04 LTS	5.4.0-54-generic	2x Alfa AWUS036ACH (USB) / Intel AX200 (PCI-e)	5.2.20 - 46.3	Intel i7 7550u, 16 GB DDR4	192.168.2.248	04:ED:33:E0:24:82
Monitor Node / mitmproxy	Server	Custom	Kali Linux 2020.4	5.9.0-kali-amd64	QC-WB1733D-1 (Intel AC 9260) (PCI-e)	34.6	Intel i7 4770K, 16 GB DDR3	192.168.2.21[1]	74:D0:2B:7C:5A:5E[1]
MS Azure (DVWA)	Cloud Web service	MS Azure	Linux (Docker)	N/A	N/A	N/A	Single-Core CPU, 2 GB RAM	20.50.64.3	N/A

"capable"[3], but as all the STAs supported 802.11w, PMF was always active. Basically, the main reason[4] we chose the "capable" mode was that most of the available WNIC on MS Windows did not connect to the AP if PMF was set to "required" along with WPA2-802.1X. This issue applied to MS Windows 10 v20H2 released in Oct. 2020.

TABLE 2. Authentication method per client STA in the testbed.

Device	Authentication method
Ubuntu 20.04 QCA6174A	EAP–TLS
Ubuntu 20.04 AX200	EAP–TTLS/PAP
MS Windows 10 Pro Alfa AWUS 1900	EAP-PAP
MS Windows 10 Enterprise TP-Link	EAP–TTLS/MSCHAPv2
MS Windows 10 Pro Linksys	EAP–TTLS/MSCHAPv2
MS Windows 10 Pro TP-Link	EAP–PEAP/MSCHAPv2
MS Windows 10 Enterprise Alfa AWUS 1900	EAP–PEAP/MSCHAPv2
Samsung Note 4	EAP–PEAP/MSCHAPv2
iPhone 6s	EAP–TLS
Samsung S20 FE	EAP–TLS

Due to the plethora of nearby Wi-Fi networks, we chose a specific channel, namely 36 at 5.180GHz with a channel width of 20MHz, for the communication between the AP and the client STAs. By doing so, we avoided (almost completely) capturing any extraneous frames stemming from other networks in the vicinity during the attack recording phase. Any remaining instances were manually filtered out. In this respect, and because all the participating devices in the testbed were reset to default values, we did not anonymize the dataset in any way.

To simulate the operation of a sample, but obviously not the volume of its workload network, we connected the Windows client STAs to the MS Windows Server and the Linux STAs to the Fedora server through FreeIPA [13]. Both these servers were operating as domain controllers and DNS servers. Additionally, we set up a two-way cross-forest trust between the FreeIPA and the Active Directory (AD). A Samba shared network was created and integrated with FreeIPA. Therefore, Linux and Windows-based client STAs must authenticate before connecting to the shared network. FreeRADIUS did not join a domain, thus requiring different client authentication credentials.

For capturing the wireless traffic, the monitor node employed Wireshark v3.2.7 running on a Kali Linux v2020.4. This node did not connect in any way to the AP of the testbed but operated only in monitoring mode. To decrypt the application layer traffic, we kept the TLS keys from each desktop STA. This, however, was infeasible for mobile ones. For them, we utilized *mitmproxy*[5] [14]. Except for breaking the TLS session into two parts, and thus knowing the corresponding TLS keys, the proxy was operating passively, receiving all traffic from those STAs and forwarding it to the AP and vice versa.

Each STA has been strategically placed in a certain area to simulate a realistic environment on the one hand and aid the monitor node to eavesdrop on the traffic on the other. As

[3]Wireshark versions prior to 3.4.0 could not decapsulate any frame if this flag was set to "required". This was due to the different message authentication code scheme used in the 4-way handshake employed; HMAC-SHA1, AES128-CMAC for "capable" and "required", correspondingly.

[4]Actually, there were two more reasons related to the Krack attack. First, for this attack to be successful, we had to use an unpatched supplicant, namely wpa_supplicant 2.6 on Samsung Note 4. But this version did not connect to AP under a PMF "required" / 802.1X combination. Second, to implement a 5GHz AP, we needed a 802.11w certified USB wireless adapter able also to disable hardware encryption, which however was no readily available. PCI-e wireless adapters have more stable drivers and implemented correctly the 802.11w, but are locked by vendors to not operate as 5GHz APs.

[5]Several companies do implement such a tactic for the sake of, say, scanning for malware and other unwanted content (deep packet inspection) and enforcing guidelines of acceptable use.

described in table 1, the attacker possessed three WNICs. The two USB ones were employed alongside[6] for realizing all of the 802.11 specific attacks except Krack and Rogue AP. For the rest of the attacks, the PCI WNIC was used. Also, as detailed in section III-B, the attacker did not connect to the WLAN, except for the cases of specific higher-layer attacks. For these scenarios, it is assumed they have managed to steal the login credentials of a legitimate user.

For desktop client STA devices, we employed different PCI-e WNICs, using Next Generation Form Factor (NGFF) into mini PCI-e, and then from mini PCI-e to PCI-e connectors. The main reason for this choice was that during the period the experiments took place, there was no readily available USB wireless adapter that supported 802.11w for the Linux OS.

For the dataset recording phase, and for the sake of enabling the STAs to produce diverse kinds of normal network traffic, we created a tool which automatically chooses a randomly scripted data traffic scenario. According to the selected scenario, the STA visited a given webpage and proceeded with some operations on it. To achieve this functionality, we used custom-made Python scripts in conjunction with the Selenium library in v3.141.59 [15]. Also, by using Node JS [16] along with the *Puppeteer* [17] library, we added infinite scrolling to the visited webpages. A last configuration pertained to the clients is that they needed a different browser to run the aforementioned functionality. For this objective, we used Geckodriver v0.28.0 [18] and Chromedriver v87.0.4280.88 [19]. The nine unique network traffic scenarios chosen randomly by the STAs are summarized in the below list. The random choice was made using a *Bash* or *Batch* script on Ubuntu or MS Windows, respectively.

1) Watch one from four in total predefined YouTube videos. Each STA, had different predefined videos.
2) Download NodeJS or Python 3.9 while visiting specific webpages.
3) Visit certain webpages along with the use of the Spotify music service.
4) Live streaming over Twitch.
5) Send/receive emails using ProtonMail.
6) Visit specific webpages. This scenario was included twice in the list of the available scenarios, meaning that the available traffic scenarios to choose from are ten in total.
7) Place a Skype call along with email communication.
8) Upload or download files using Dropbox Business and/or Samba.
9) Upload files using Nextcloud and Samba.

As detailed in subsection III-B, each traffic scenario was executed for a random duration of time, but always not surpassing a predefined upper limit. Precisely, as a first step, the Bash or Batch script chooses a random number from 1 to 10 pertaining to the available scenarios. This number determines the number of scenarios this STA will execute. Then, the maximum execution time of the script in sec is divided by this random number. The result determined the execution time of each of the randomly chosen scenarios. Say that the random number is 3. If divided by, say, 300, the script will appoint three random network traffic scenarios and run each of them for 100 sec. It was possible for the STA to execute the same scenario multiple times. The second part of the script is with regard to the additional tools we used. Namely, when the script chooses a random scenario, a Python script executes to realize this scenario. Lastly, regarding the attacker, we used mainly command-line options, i.e., timeouts, to automate the attacking phase.

[6]This was done for enabling the monitor node to gather as many frames as possible during the attack phase, and also to assault the maximum possible number of client STAs in the available time window.

B. DATA COLLECTION

With respect to the data collection process the following points are of particular interest.

- Each attack comprises a separate pcap file, but as explained in the next subsection, each one of these files may contain different variations of the main attack, implemented with the help of different tools. In accordance with section II and as summarized in table 3, thirteen attacks were implemented in total.
- As explained in section III-A, the automation scripts used added several variations of the normal network traffic per client STA.
- The total number of the recorded frames, the total size of each pcap file, along with the numerous network features that these files contain, can easily lead to a Big Data proportion dataset. This would make the use of the dataset cumbersome and resource-intensive. With this in mind, as seen from the 6th column of table 3, we kept the size of most pcap files under ≈2.5M frames, which corresponds to a total duration of ≈10 min. For each pcap file, the first ≈3 min are consumed to connect each STA to the AP and trigger the selected traffic scenario, while in most cases, the next ≈4 min are dedicated to normal traffic.
- As mentioned in section III-A, the decryption of pcap files was necessary for handling the captured frames and extracting the different features to be used, say, by machine learning classifiers at a subsequent phase. Given that the main difference between personal and enterprise mode is that each PMK is different in the second case, it is unfeasible to enter the ''pre-shared'' key in the Wireshark and expect to read every pcap file. So, all PMKs for each STA which made a connection to the AP must be preserved. To do so, the Radsniff [20] tool was used at the RADIUS server side. Precisely, the tool sniffs the Ethernet traffic, namely the EAP over Radius packets, arriving or leaving the server, and therefore the PMK sent to the AP within the *MS-MPPE-Recv-Key* attribute of the *Radius-access-accept* message.

Based on the above key points, the steps that realized the recording of each attack into a separate pcap file are as

TABLE 3. Matching attacks and pcap files of the dataset. The ⋙ symbol denotes the device the attack is directed to. The penultimate column designates the start and end frame number in the respective pcap, along with the attack phase they refer to. The 1st and 2nd number in the "Duration" column refer to the attack and total time duration, respectively, in minutes. For Att3, 4, and 12, the Aireplay-ng was modified to send disassociation frames instead of deauthentication ones. For Att19, the IP address range was 192.168.30.0/24. Due to an issue with the employed Wireshark version, the full decryption of pcap files for Att13, Att21 and Att14, Att18 was only possible in Linux and MS Windows versions, respectively.

File Name	Alias	Attacks	Attack Tools	⋙	Total Frames	Attack Frames	%	Size (GB)	Frame #	Duration
Deauth	Att1	Deauth. Flooding	Aireplay-ng [32], Netattack [33]	STA and AP	1,626,472	38,945	2.39	0.66	p1: 1,088,022 / 1,499,198 p2: 1,476,986 / 1,626,254	3/10
	Att2	Deauth. Flooding Broadcast								
Disass	Att3	Disass. Flooding			2,013,719	75,131	3.73	0.79	p1: 1,404,237 / 1,878,567 p2: 1,878,598 / 2,013,346	3/10
	Att4	Disass. Flooding Broadcast								
	Att5	Amok	MDK4 [34]							
(Re)Assoc	Att6	Assoc. Request Flooding	Scapy [35]	STA	1,843,939	5,503	0.29	0.91	p1: 1,145,178 / 1,597,897 p2: 1,599,579 / 1,833,964	3/10
	Att7	Reassoc. Request Flooding								
	Att8	Beacon Flooding								
Rogue_AP	Att9	Rogue AP	Hostapd [28]		1,973,193	1310	0.07	0.83	1,198,551 / 1,973,111	3/10
Krack	Att10	Channel MitM attack	Scapy, Hostapd, Dnsmasq [29]		1,438,493	49,990	3.47	0.93	1,273,943 / 1,438,423	3/10.5
	Att11	Key Reinstallation								
Kr00k	Att12	TK Reinstallation	Aireplay-ng	STA and AP	2,900,458	191,803	6.61	1.23	p1: 1,555,898 / 2,303,411 p2: 2,310,583 / 2,875,846	3/10
SSH	Att13	SSH Brute Force	Nmap [36]	Radius	2,440,588	11,840	0.48	1.05	1,356,015 / 2,440,390	3/10.5
Botnet	Att14	Botnet	Ares [22]		3,326,083	52,661	1.71	1.58	1,135,097 / 3,325,480	6.3/13
Malware	Att15	WannaCry Plus	-	STA	2,312,782	133,019	5.75	1.22	1,021,326 / 2,310,931	3.5/10.5
	Att16	TeslaCrypt								
SQL_Injection	Att17	SQL Injection	-	Webpage/user	2,598,365	2,702	0.10	1.36	1,484,773 / 2,589,043	3/10
SSDP	Att18	SSDP Amplification	Python script [27], Saddam-new [37]	Server/Webpage	8,141,523	5,535,358	67.98	2.32	p1: 1,198,154 / 1,375,104 p2: 1,387,933 / 8,122,583	3.5/10.5
Evil_Twin	Att19	Captive Portal	Apache2 [30], Hostapd, Dnsmasq	STA	3,778,728	102,059	2.70	1.87	p1: 1,420,038 / 2,707,281 p2: 2,836,921 / 3,778,728	8/15
	Att20	Eaphammer	Eaphammer [31]							
Website_spoofing	Att21	ARP & DNS Poisoning	Apache2, Bettercap [38]		2,668,583	501,772	18.80	0.98	16,410 / 2,668,583	6.5/10.3

follows: (a) Enable the recording process of PMKs through Radsniff, (b) activate the monitor node, (c) establish a new connection per client STA with the AP, (d) confirm the sound capture of each STA's 4-way handshake in the monitor node, (e) start the script that randomizes the network traffic per client STA, (f) initiate the attack, (g) end of the attack phase, (h) save the recorded file in pcap format.

During the recording phase, and as explained in the following, we used scripts to automatize the 802.11-oriented attacks, namely those included in section II. For the sake of convenience, each attack and client STA has been given an alias presented in tables 3 and 4, respectively. For Att1 to Att8 as well as Att12, the attacker possessed two Alfa AWUS036ACH WNICs. In the following, we will refer to these WNICs as *WNIC1* or *WNIC2*. For the same attacks, we used a round-robin scheme configured in the corresponding attack script, meaning that both WNICs targeted the same STA and then moved to another, always in the same order, namely iPhone, Andr-S20, Ubu-QCA, Ubu-AX, WinEnter-TP-Link, WinPro-TP-Link, WinPro-Linksys, WinEnter-Alfa, WinPro-Alfa, and Andr-Note4. Att9 exploited only WNIC1, while Att10 and Att11 utilized both WNIC1 and the PCI WNIC. For Att13 to Att18 and Att21, it is assumed that the aggressor has in their possession the login credentials (PAP) of a given STA, namely WinPro-Alfa. Also, for Att14 to Att16, they possess the login credentials of Ubu-AX to the Samba network.

Specifics per kind of attack and the associated pcap file are given in the following. Also, table 3 summarizes all the important information for each pcap file, including the attack tools used, the breakup in normal and attack frames, the total size in GB, the total and attack duration in min, and the frame numbers designating the start/end of an attack phase.

Deauthentication: It exploited both single targeted and broadcasted types of frames. Two phases, one per type of attack, namely Att1 or Att2, were used. In the first, both WNICs assaulted each STA for 12 sec, while in the second, both WNICs targeted the broadcast MAC address for 60 sec in total.

Disassociation: It also employed mainly unicast and a limited number of broadcast disassociation attack frames. Specifically, by utilizing both WNICs, Att3 was executed first, lasting 120 sec in total. A subsequent phase employed Att5 on WNIC1 for 60 sec, while at the same time, WNIC2 unleashed Att4 in 2 phases, each one for a total of 30 sec, using either Aireplay-ng or Netattack. As expected, and can be observed from the dataset, this attack also yielded a small number of Kr00k zero-TK frames.

(Re)association: It contains three attacks in a single pcap file. Two phases were performed. In the first, Att6 and Att7 were simultaneously executed through different WNICs, both targeting each STA for 12 sec. In the second, WNIC1 triggered Att6 against all STAs for 60 sec, i.e., 6 sec per STA, while at the same time WNIC2 unleashed Att8 with the purpose of further stressing the STAs.

Rogue AP: For Att9, a rogue AP was created using the same SSID, MAC address, and channel with the legitimate one. Recall from section II-A, that the rogue AP operated in WPA2-PSK with PMF disabled. We observed that more than half of the STAs were lured into repeatedly trying to connect to the rogue one. Nevertheless, as expected, these STA received a deauthentication frame due to Pairwise Master Key (PMK) mismatch; actually, a surge of deauthentication

frames were dispatched from the rogue AP. At the same time, because the APs operated under the same MAC address, this situation ignited the Security Association (SA) Query mechanism between the STA and the legitimate AP, ultimately leading to a disconnection in most of the cases.

A parallel observation pertained to the beacon frames of the rogue AP, and affected all the Windows-based STAs (which initially did not attempted to connect to the rogue AP as described previously), but the Linksys one (which did attempted to connect to the rogue AP). Surprisingly, upon receiving these beacons, these STAs immediately deleted its SA with the legitimate AP, dispatched a probe frame, and then initiated an authentication process towards one of the APs. This on the other hand made the legitimate AP to start a futile SA Query procedure with the affected STA, which ultimately led to dissasociating the STA with a reason code 9 ''STA requesting (re)association is not authenticated with responding STA (0×0009)''. After further analyzing this bizarre situation[7], we concluded that for yielding the aforementioned results, the beacons stemming from the rogue AP must at least include the same MAC address, SSID, and channel as that of the legitimate AP; otherwise these STAs will have the same behavior as the rest of the STAs, namely will try to connect to the rogue AP. Put simply, even a single ''empty'' beacon frame (it only contained the valid SSID tag parameter of the BSS) can cause this effect. This flaw may be exploited to create another variant of beacon flooding that would render any affected STA unable to connect as long as the attack is ongoing. Lastly, this effect applies only if the legitimate AP operates on 5GHz, with at least WPA2-PSK or WPA2-802.1X, and independently of the PMF mode used.

Krack: This attack was implemented in two steps against all STAs. Att10 established a MitM position, while Att11 launched key reinstallation attacks for a total of 180 sec. Note that for being able to record more effectively this attack, we chose a 2.4 GHz AP setup. The attacker implemented the assault on channel 2, while the legitimate AP operated on channel 13. Note that the only unpatched, and thus vulnerable STA to this attack was Andr-Note4.

Kr00k: Att12 unleashed timed disassociation assaults with the reason code 7 ''Class 3 frame received from non associated station''. Actually, it was observed that this was the only case this attack was prolific, achieving a zero-TK reinstallation. We transmitted this type of frames toward the victim STAs in circles of 9 sec each. That is, both WNICs attacked every STAs simultaneously; WNIC1 using the normal round robin order, while WNIC2 in the reverse order. The attack indeed yielded zero-TK encapsulated frames for at least one vulnerable device, namely the AP. Such frames can be extracted from the pcap using the *r00kie-kr00kie* Python script [21]. Lastly, an important observation was that after the AP had deleted an SA, any buffered SA Query frame for this SA were transmitted unencrypted. As it is detailed in section V, this may leave room for exploiting this type of frames against an STA.

SSH brute force: Att13 attempted unsuccessfully for 180 sec to brute-force the login credentials to the Radius server. This naturally involves the exchange of numerous Key exchange (''kex'') messages. That is, from a total of ≈3,800 SSH messages, about 20% of them are of type ''kex''. Note that SSH messages of type ''USERAUTH'' are encrypted in the pcap file because the service request phase is done over the protection of the SSH tunnel utilizing ephemeral keys.

Botnet: It assumes that the attacker gained access to the Samba Network, placed a couple of malicious files in a shared directory, and waited for a random victim STA (user) to execute them. These files, one for MS Windows and the other for Linux OS, were created with the help of Ares tool [22]. Note that due to the cross-trust forest between the AD and FreeIPA, the attacker also needed a Security Identifier (SID) to connect using the login credentials of any Windows user. In the implemented scenario, four STAs successfully executed these files and were turned to bots. Afterwards, the attacker exploited the infected STAs to execute different commands, like capturing a screenshot of a user's desktop or monitoring their activity. These pieces of stolen data where then sent to the attacker.

Malware: Regarding Att15 and Att16, the attacker placed two pieces of malware, namely WannaCry Plus [23] and TeslaCrypt [24] to the shared Samba network. The malware files were downloaded by 6 STAs (users) of the network, but were never executed for obvious reasons. In the dataset the malware files are named as ''Win32.Wannacry.exe'', ''node.exe'' and ''51B4EF5DC9D26B7A26E214CEE90598631E2EAA67'', ''finances'', respectively, and were obtained beforehand from the *theZoo* repository [25] in GitHub.

SQL injection: Att17 was performed on a Damn Vulnerable Web Application (DVWA) docker container [26] running in MS *Azure*. The webpage's ULR was https://dwva-awid2.azurewebsites.net/. The assailant exercised manually this attack for a total of 180 sec; no automatic tool was used for being able to preserve the TLS keys and decrypt the traffic. For instance, the assailant exploited commands like *%' or '0' = '0' union select user, password from dvwa.users #, %' or '0'='0' union select user, password from dvwa.users #, %' or '0' = '0*.

SSDP Amplification: Att18 was exercised against the same DVWA webpage residing on MS Azure. First, for ≈30 sec, the attacker scanned the intranet for exploitable devices using a Python script [27]; the IP destination address of such a *M-SEARCH* packet was 239.255.255.250:1900. The IP of the AP was returned, and next the same device has been identidied to reply with around 10 frames per each M-SEARCH of type ''ssdp:all''. In a subsequent phase lasting 210 sec, the assailant unleashed the attack against the DVWA webpage with the aid of the *Saddam-new* tool. That is, the tool dispatches towards the AP's IP address a surge of

[7]A short video demonstrating this behavior is available on the AWID website at http://icsdweb.aegean.gr/awid/. Also, this issue has been reported by the authors to Microsoft.

spoofed *M-SEARCH* packets with the ST header field being "ssdp:all".

Evil_Twin: This pcap file contains two different implementations of the Evil_Twin assault. The first realizes a captive portal attack. For this, we relied on Hostapd [28] (it turns a WNIC into an AP), Dnsmasq [29] (it provides DNS caching and a Dynamic Host Configuration Protocol server for LAN networks) and Apache2 server [30] (it creates an HTTP server that can operate with Hostapd and Dnsmasq and hosts the fake captive portal webpage). Via an Evil_Twin assault, we redirected every connected client to the rogue AP STA to visit the fake webpage, namely https://captiveportal.local, which afforded a self-signed certificate. Note that after the user entered their credentials to the fake webpage, they were directed to another webpage, which automatically downloaded a piece of malware, named "agent.exe" to the STA. The rogue AP operated on the same channel, SSID, and almost the same MAC address[8] (the last digit was "5" instead of "4") with the legitimate one. It is to be noted that although Windows STAs were able to establish a connection to the rogue AP, they were unable to access the Internet, and therefore be redirected to the captive portal webpage. This is because they allocated a specific DNS IP address, the one the AD Domain Controller had, and thus it expected for the same IP address in DNS responses. Therefore, this attack was effective only against the mobile STAs. On the other hand, Att20 exploited the Eaphammer tool [31]. Under certain assumptions, this tool can perform an Evil_Twin against WPA2-802.1X networks and it is one of the few that can operate well in 5 GHz. Using the same AP configuration as with Att19, we targeted all STAs by potentially exploiting any EAP downgrade misconfiguration. Then, we restarted Eaphammer and launched a "hostile portal attack" as provided by the tool. Lastly, both these attack variants could not be implemented successfully for STAs that were connected using a TLS-based method, in our case, EAP-TTLS and EAP-TLS. This was because the rogue AP (acting simultaneously as a RADIUS server) did not possess a valid certificate, i.e., one issued by a CA that the STA could trust.

Website_spoofing: After deploying Apache2, the attacker hosted a fake Instagram webpage having a self-signed certificate. By exercising ARP and DNS poisoning targeting the real Instagram page, the victim STAs were redirected to the fake one, thus allowing the attacker to steal their credentials and decrypt them using the TLS keys obtained from the server. In the dataset, DNS and ARP poisoning are done in the ≈4th min, while the victims attempted to access the real Instagram page 4 min after that. Specifically, the victims were the 5 Windows STA, the AP, and the AD domain controller. Overall, as can be observed from the corresponding dataset file, WinPro-Alfa, WinEnter-Alfa, and WinPro-Linksys fell victims to this attack.

[8]Due to the beacon issue mentioned previously in Att9 and the use of mitmproxy, we avoided using the same MAC address in both the APs.

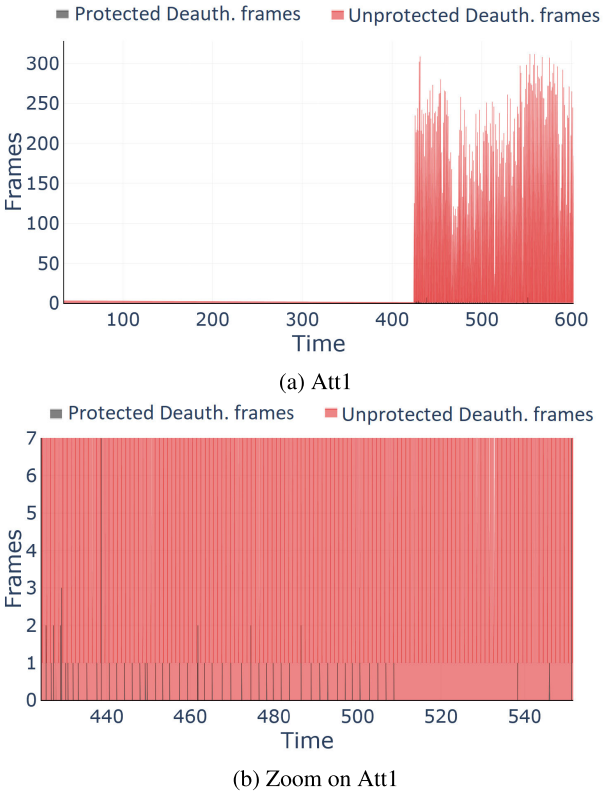


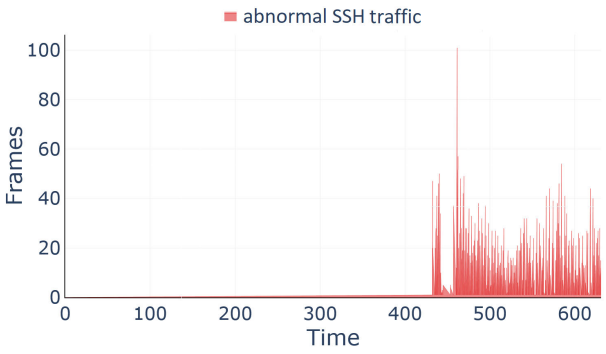
FIGURE 2. Deauthentication (Att1) signature.

IV. SIGNATURE OF ATTACKS

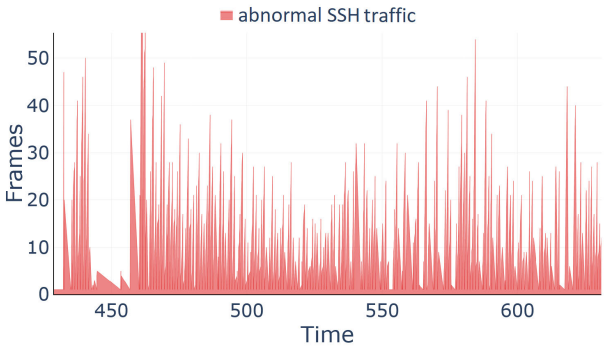
This section offers symptomatic signatures of selected attacks based on cherry-picked features. Precisely, we chose 6 assaults, that is, one per category of section II plus KracK, plus a multi-stage one synthesizing 3 different pcap files. Contrariwise to the original AWID, we opt out of evaluating the dataset by means of machine learning, given that this should be done per category of attacks, and thus would overburden the already lengthy manuscript. Naturally, future researches can exploit virtually any set of features that stem from the pcap files along with machine learning classifiers, as the dataset is offered in a cleartext form.

Flooding attacks in 802.11 demonstrate more or less the same structure; the observer discerns a sudden increase in the unprotected management frames per second. Recall that in the context of the current dataset, PMF was active, so the AP communicated using only protected deauthentication and disassociation management frames after it had established a Security Association (SA) with the corresponding STA. Based on the Deauth.pcap file of the dataset containing Att1 and Att2, figure 2 depicts the number of spoofed deauthentication frames vis-à-vis the legitimate (protected) ones. As it can be easily observed from the bottom subfigure containing this type of frames when the attack was unfolding, i.e., for 180 sec, only a tiny number of protected (black-colored) frames is present within the gush of attack ones.

Regarding Att13, figure 3 focuses on traffic containing multiple SSH requests and responses stemming from the

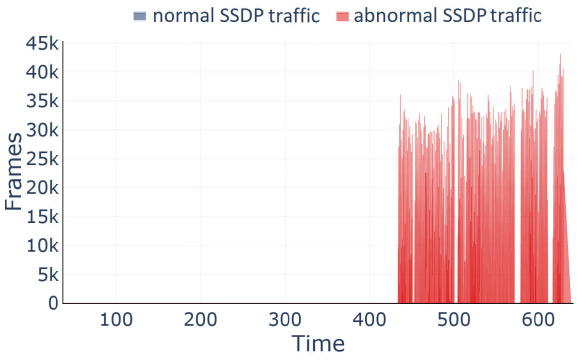


(a) Att13

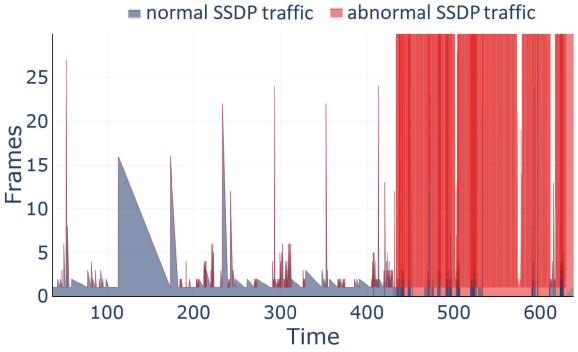


(b) Zoom on Att13

FIGURE 3. SSH brute force (Att13) signature.



(a) Att18



(b) Zoom on Att18

FIGURE 4. SSDP amplification attack (Att18) signature.

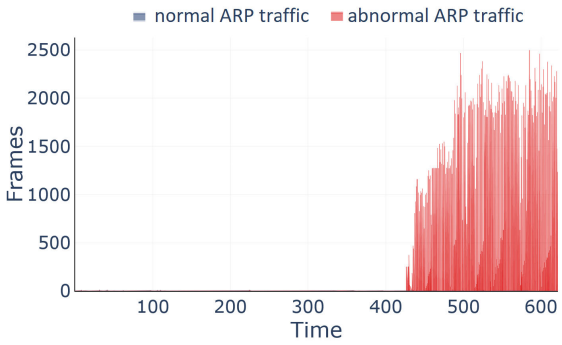
same IP source address. This sudden surge of SSH traffic may straightforwardly designate that a SSH brute force is unfolding.

Concerning Att18, figure 4 portrays a SSDP amplification attack in its blooming. That is, the aggressor sends multiple SSDP packets towards the AP acting as amplifier, having their source IP address spoofed to that of the victim, i.e., the Azure VM hosting DVWA. The generated traffic volume (SSDP requests) towards the AP reached ≈35,000 packets per sec.

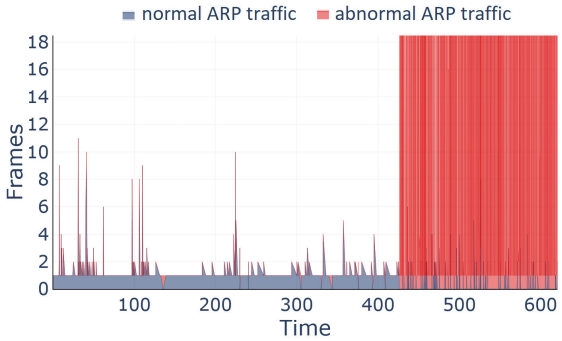
As a fourth case, we examined the Website_spoofing.pcap, namely Att21. We chose to focus only on the ARP poisoning part, because it should be easily detected by an Intrusion Detection System (IDS). Specifically, in figure 5 one can clearly observe the abnormal ARP traffic originated from the same single MAC address.

The next signature has been generated based on the Krack.pcap, i.e., Att11. Figure 6 demonstrates the recurrent transmission of a plethora of messages 1 or 3 of a 4-way handshake, namely EAPOL-Key frames, for about 180 sec. Note that these frames pertain to both of the wireless channels (2 and 13) as recorded by the monitor node.

A last signature attempts to visualize a multi-stage assault by combining 3 pcap files, namely "Deauth", "Evil_Twin", and "Website_spoofing". As observed from figure 7, after the attacker deploys a rogue AP, they mount Att1 and Att2 to disconnect the STAs from the legitimate AP in high hopes that they subsequently connect to the rogue one. This plot,



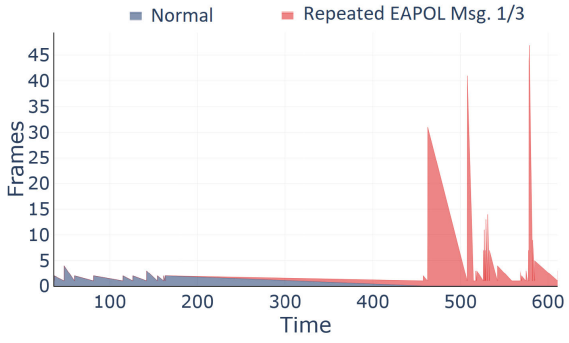
(a) Att21



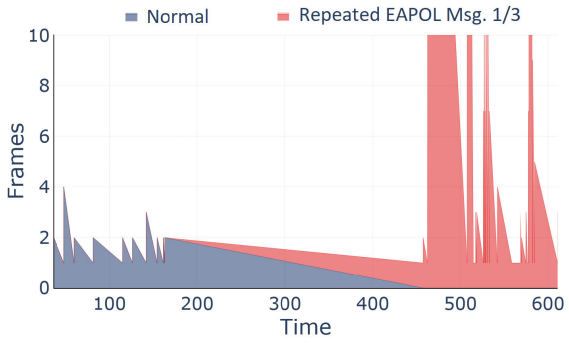
(b) Zoom on Att21

FIGURE 5. Website_spoofing (Att21) signature.

i.e., through the captive portal, may allow the attacker to steal the login credentials of some STAs. Now, having access

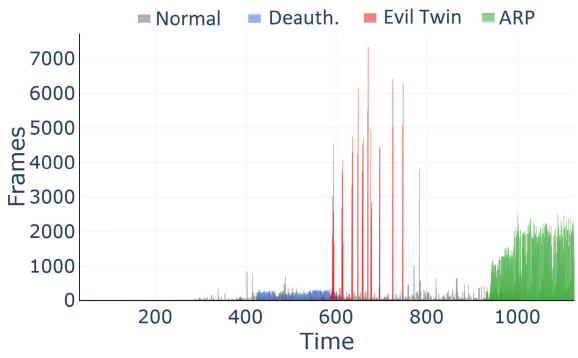


(a) Att11

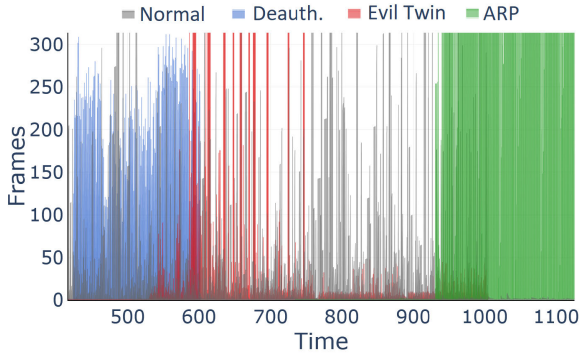


(b) Zoom on Att11

FIGURE 6. Krack (Att11) signature.



(a) Multi-stage attack



(b) Zoom on Multi-stage attack

FIGURE 7. Multi-stage attack (Att1, Att2, Att19 to Att21) signature.

to the network, the assailant launches Att21. For depicting the Evil_Twin, we relied on data frames (0×0028), namely, normal traffic produced by the legitimate AP vs. traffic generated by the rogue AP, represented by grey and red colors, respectively. Note that the traffic produced by the legitimate AP is originally encrypted, while that of the rogue one is unencrypted. For the rest of the attacks, we exploited the same features as previously. Note that both the normal Deauthentication and ARP traffic are also included in the synthesized pcap, but it is barely shown even in the zoomed subfigure. So, no data labels are given for these types of traffic.

V. KEY OBSERVATIONS REGARDING PMF

This section details on the results of 802.11 specific attacks for which we observed a somewhat different behavior than the one expected based on the 802.11w amendment. First, we elaborate on significant remarks that pertain to the resilience of certain devices when they are under attack, and second, in subsection V-B, we investigate the root reasons behind the perceived behavior. Bear in mind that the connected client STAs to the AP communicated using PMF. This could be verified easily because all STAs did received an IGTK as a part of the relevant EAPOL message in the 4-way handshake. Therefore, among others, the deauthentication, disassociation, and SA query frames (0×000d) were protected. Recall that Table 4, contains the aliases of the employed STAs during the experiments, while each attack's unique number (alias) is given in the second column of table 3. Naturally, the below analysis has to be viewed in relation to the specific hardware/software configuration of the testbed devices as given in table 1.

TABLE 4. List of STA aliases used in section V.

OS	WNIC	Alias
Ubuntu 20.04 LTS	Intel AX200	Ubu-AX
Ubuntu 20.04 LTS	Qualcomm Atheros QCA 6174A	Ubu-QCA
MS Windows 10 Pro	Alfa AWUS 1900	WinPro-Alfa
MS Windows 10 Pro	Linksys WUSB6100M	WinPro-Linksys
MS Windows 10 Pro	TP Link AC 1300 T4U	WinPro-TP-Link
MS Windows 10 Enterprise	Alfa AWUS 1900	WinEnter-Alfa
MS Windows 10 Enterprise	TP Link AC 1300 T4U	WinEnter-TP-Link
iOS 14.2	iPhone 6s	iPhone
Android 10	Samsung S20 FE	Andr-S20
Android 6.0.1	Samsung Note 4	Andr-Note4

A. DEVICE RESILIENCE TO ATTACKS

The following important remarks per kind of 802.11 attack were made, also visible in the respective pcap file.

For Att1, it was observed that almost all client STAs were disconnected after some time. Exceptions to this behavior were WinPro-Linksys and Andr-S20. Regarding Att3 all STAs but the Andr-S20 were disconnected. iPhone, Ubu-QCA, WinPro-Linksys, Ubu-AX, and Andr-S20 were disconnected after triggering Att4 and Att5. As noted in section II, several hardware vendors have implemented defenses against the broadcast of these frames. However, during our experiments and specifically for attacks Att2

and Att4, it was noticed that at least one STA suffered a disconnection.

The joined effect of Att6 and Att7 was to disconnect Ubu-QCA, WinEnter-TP-Link, WinPro-Alfa, and WinEnter-Alfa from the AP. The same result was discerned for Ubu-QCA, WinPro-Alfa, and WinEnter-Alfa after exercising Att6 and Att8 simultaneously. On the other hand, Att9 achieved the disconnection of Ubu-QCA, WinPro-Alfa, WinEnter-Alfa, WinPro-TP-Link, and WinEnter-TP-Link; the first due to PMF, while the rest because of the beacon strange effect as detailed in subsection III-B.

B. DISCUSSION

Following the analysis done in the previous subsection, the current one attempts to dig into the root causes that may lead to disconnections even in the presence of PMF. To do so, we implemented 6 auxiliary - not included in the dataset - attack scenarios, in which we scrutinized every aspect we considered important. Specifically, with the aid of a modified version of Aireplay-ng, we generally capitalized on Att3, which stresses the resilience of 802.11w, and peered into any deviation from the expected behavior.

In all these attack scenarios there was only one STA connected to the AP at any moment, and the attacker was motionless. All APs used were running with the default configuration settings. For all but the last scenario, the targets were all STAs contained in table 4. The attacker was equipped with a 2xCPU/16 GB DDR4 RAM laptop running Ubuntu 18.04 desktop, and relied in most cases on two Alfa AWUS036ACH WNICs.

Scenario I: It aimed to scrutinize the different behavior that may occur between the 802.11w ''capable'' and ''required'' mode (flag) of the ASUS RT-AC68U AP. In addition, it examined possible deviations between the Personal (2.4GHz) and Enterprise (5GHz) modes according to our setup. For the latter, every STA used the respective EAP authentication method as summarized in table 2. The assailant used 4 attack instances simultaneously exploiting one WNIC. The results are summarized in table 5. As seen in the table, each STA was either idle, i.e., connected to the AP without transmitting or receiving any data, or busy; we experimented with only two combinations, namely 2.4/idle and 5/busy, which we think are enough to demonstrate the matter. As expected, during the tests, for all the STAs, we perceived that the respective attacks did not have an immediate effect, i.e., a network disconnection, as compared to the situation where the 802.11w was disabled. On the downside, with reference to table 5, all the STAs disconnected after some time. Obviously, some of them showed greater sturdiness to these DoS attacks. For instance, two of the most resilient choices seem to be the WinPro-TP-Link and WinPro-Alfa. Additionally, from the table, it is extrapolated that in the majority of the cases it is harder to disconnect an STA from the AP if the 802.11w is set to the 2.4 GHz, ''required'' combination. We assume that this behavior is due to the lower speed rates of the 802.11n protocol and the idle condition of the STA. Another important factor was the idle or busy situation of each STA. Simply put, the greater the overload, the harder for the SA Query mechanism to cope. Lastly, regarding the mobile STAs, they demonstrated similar times regardless the setting, presumably due to the low CPU power that each wireless adapter had.

TABLE 5. Results on 802.11w. All numbers are in seconds until a disconnection takes place. Cap.: Capable, Req.: Required, N/A: The STA was unable to authenticate to the AP under the "Required" mode due to the last MS Windows update mentioned in section III.

Idle			Busy	
Personal (2.4 GHz)			Enterprise (5 GHz)	
Devices	Cap.	Req.	Cap.	Req.
Andr-S20	3	3	3	3
Andr-Note4	6	6	6	6
iPhone	9	8	10	8
WinPro-Linksys	20	20	49	N/A
WinPro-TP-Link	105	164	29	N/A
WinEnter-TP-Link	64	145	18	N/A
WinPro-Alfa	105	109	38	N/A
WinEnter-Alfa	87	127	24	N/A
Ubu-AX	5	8	17	12
Ubu-QCA	9	21	28	10

Scenario II: Instead of a dedicated hardware AP, Hostapd v2.9 [28] was installed on an Intel 4770K/32 GB DDR3 RAM machine running Kali Linux 2020.4 along with Dnsmasq to provide DHCP and DNS services. The WNIC of this machine was a Gigabyte Wb1733D-I having an Intel 9260 chipset. The AP operated in Personal Mode and PMF was configured as ''required''. The connection to the Internet was provided through USB tethering using a Xiaomi Redmi Note 8 Pro smartphone over 4G. All tested STAs were idle. Under the ''required'' setting, the iPhone device was unable to connect, so for this device ''capable'' was used. Following their bombarding with 8 attack instances, 4 per WNIC, simultaneously, Ubu-AX and Ubu-QCA demonstrated great resilience, being disconnected after 188 and 181 sec, respectively, while iPhone disconnected after 40 sec. On the other hand, WinPro-TP-Link and WinEnter-TP-Link were both disconnected after 8 and 38 sec, respectively using 4 attack instances, i.e., one WNIC. Another important observation is that, as expected, the number of the already connected STAs to the AP at a given time affects negatively the endurance of any STA in such attacks. For instance, after connecting another (busy) STA along with the Ubu-AX to the AP, the latter disconnected after 37 sec, i.e., about five times faster.

Scenario III: The purpose of this scenario was to observe any contingent deviation in the behavior of the same STA (namely, Ubu-AX) regarding the usage of PMF when WPA3 (SAE/Personal in 2.4 GHz) was used. So, Hostapd had the same hardware configuration as in the previous scenario, except it operated in WPA3. We received similar results, that is, Ubu-AX disconnected after 198 sec after attacking it with two WNICs, namely 8 attack terminals simultaneously.

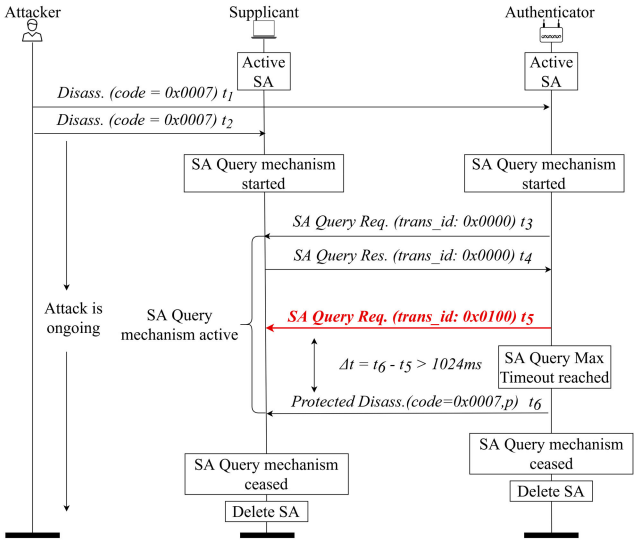
Scenario IV: It employed an Andr-S20 hotspot 2.4 GHz connection as the AP. Note that this mobile device is currently among the few that support 802.11w when

acting as AP. The 802.11w was configured as "capable". The attacker exploited both WNICS, meaning 8 attack terminals in total. iPhone and Ubu-AX remained connected for the total duration of the attack, i.e., 300 sec. The rest STAs, namely WinPro-Linksys, WinPro-Alfa, WinEnter-Alfa, WinPro-TP-Link, WinEnter-TP-Link, Ubu-QCA, and Andr-Note4, disconnected after 20, 11, 12, 20, 21, 5, and 75 sec, respectively. To further scrutinize on the stamina of iPhone and Ubu-AX, we employed the regular version of Aircrack-ng and Att1 using 8 attack terminals per WNIC for 360 sec in total. Both STAs were again able to withstand the attack, but as expected, they experienced severe delays when receiving traffic. Also, the Andr-S20 were severely overheated.

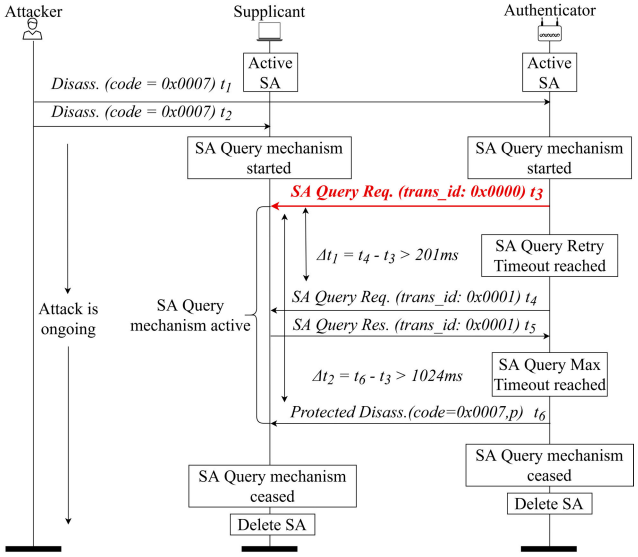
After executing the above-mentioned scenarios, it was observed that the (Re)Association Response frame (0×0001) with reason "Status code: Association request rejected temporarily; try again later (0×001e)", was not used by any AP, either Hostapd or Andr-S20, but the ASUS RT-AC68U one. In fact, the sole time this frame was transmitted by the ASUS AP was during a Reassociation request flooding exploiting random MAC addresses, and also in Att6 and Att7, which is in any case in accordance to the standard. The Time Units (TUs) value contained in this frame was 20.1 sec, which is very large vis-à-vis the 802.11w amendment [3] and the currently active 802.11-2016 standard [39]. Regarding the AP (re)association procedures the interested reader can refer to subsections 11.3.2 and 11.3.5 of [3] and [39], respectively.

Also, with reference to the specification [39], there are two other parameters, namely the SA Query maximum and retry timeouts, that were observed to have an immense impact in the communication between the AP and the STA. The first one, refers to the maximum time the specific device must wait for a SA Query response, having a default value of 1 sec. To assist the reader, the top subfigure of figure 8 depicts a typical example of the use of the SA Query maximum timeout. The second, pertains to the time the device must bide before sending another SA Query request, having a default value of 0.2 sec. Therefore, the device may send out multiple subsequent SA Query requests before the 1 sec time window of the previous one(s) expires. Nevertheless, after sending, say, 3 SA Queries, we noticed that the AP disconnected the STA after a duration corresponding to the maximum timeout of first submitted query i.e., after 1 sec, and not the timer corresponding to the last query, i.e., 1.6 sec assuming a jumping-off point the dispatch of the first query. A characteristic example of this situation is illustrated in the bottom subfigure of the same figure, where the STA is disconnected due to the late arrival of SA Query response with trans_id = 0.

With the exception of high-end, expensive APs, which cater for detailed customization in their user interface, the above-mentioned default values are either hardcoded or any reconfiguration requires recompiling the firmware. Hence, in most cases, to tamper with these values one needs to rely



(a) SA Query maximum timeout disconnection occurred due to unanswered SA Query with trans_id = 100.



(b) SA Query maximum timeout disconnection occurred due to the unanswered SA Query with trans_id = 0.

FIGURE 8. Exemplification of disconnections due to SA Query maximum and retry timeouts.

on a software AP. Specifically for Hostapd, these values are contained in its configuration file [40] and presented in list 1. For ASUS RT-AC68U, the relevant configuration files can be found in Github [41], [42], while the relevant C code is shown in List 2.

Scenario V: Having the above in mind, we further scrutinized the behavior of the 802.11w using Hostapd as the AP, by tinkering with the default values of SA Query maximum and retry timeouts. Different values, including 50, 100, 500, 3,000, 7,000, 10,000 have been tested. Nevertheless, no change in the default behavior of this AP regarding the SA Query frames was observed, and no Association response frame, namely "Timeout Interval Type: Association Comeback time (TUs)" was sent.

```
# Association SA Query maximum timeout
# (in TU = 1.024 ms; for MFP)
# (maximum time to wait for a SA Query
# response)
# dot11AssociationSAQueryMaximumTimeout,
# 1...4294967295
assoc_sa_query_max_timeout=1000

# Association SA Query retry timeout
# (in TU = 1.024 ms; for MFP)
# (time between two subsequent SA Query
# requests)
# dot11AssociationSAQueryRetryTimeout,
# 1...4294967295
assoc_sa_query_retry_timeout=201
```

LISTING 1. Hostapd retry and maximum timeout default values.

```
#ifdef CONFIG_IEEE80211W
  enum mfp_options ieee80211w;
/*dot11AssociationSAQueryMaximumTimeout*/
/*(in TUs)*/
  unsigned int assoc_sa_query_max_timeout;
/*dot11AssociationSAQueryRetryTimeout*/
/*(in TUs)*/
  int assoc_sa_query_retry_timeout;
#endif /* CONFIG_IEEE80211W */

#ifdef CONFIG_IEEE80211W
  bss->assoc_sa_query_max_timeout = 1000;
  bss->assoc_sa_query_retry_timeout = 201;
#endif /* CONFIG_IEEE80211W */
```

LISTING 2. Asus RT-AC68U retry and maximum timeout default values.

```
static const unsigned int
    sa_query_max_timeout = 1000;

static const unsigned int
    sa_query_retry_timeout = 201;
```

LISTING 3. Wpa_supplicant retry and maximum timeout default values.

Scenario VI: This final test had Hostapd v2.9 as an AP (2.4 GHz), running on an Ubuntu 18.04 Desktop machine with 8xCPU/64 GB DDR4 RAM. The wireless adapter was the Gigabyte GC-WBAX200 (Intel AX200 chipset). The 802.11w was configured as ''required''. DHCP, DNS, and Internet services are offered in the same way as in scenario II. However, in this case, we employed a software STA (Wpa_supplicant v2.9), running on an Ubuntu 20.04 desktop machine with 8xCPU/32 GB DDR4 RAM and exploiting the Qualcomm Atheros QCA 6174A as a WNIC. The STA obtained an IP address via the *dhclient*. First, the Wpa_supplicant was connected to Hostapd with the default configuration. Att1 was triggered on 10 Aireplay-ng instances (5 per wireless adapter) simultaneously, and achieved to disconnect the STA after 43 sec. Before disconnecting, the STA (a) displayed the message ''SME: SA Query timed out'', meaning that the default time of 1 sec that the Wpa_supplicant had to wait for a SA Query response had expired, and (b) sent a protected deauthentication message toward the AP with reason code 2 ''Previous authentication no longer valid''. Next, we proceeded by increasing the SA Query timeout default values in both the Wpa_supplicant and Hostapd. For the former, the relevant code is given in list 3. First off, we doubled the respective values, i.e., 2 and 0.4 sec for the SA Query maximum and retry timeout, respectively. This led in the exchange of fewer action frames between the STA and the AP, and hence disconnections became harder. In fact, the STA disconnected after 90 sec, i.e., almost twice as much as with the default values. As a next step, we quintupled the default values, namely to 5 and 1 sec, respectively. With this, the STA was disconnected after 12 min (720 sec). Nevertheless, in this case, the disconnection was not due to the SA Query maximum timeout, but because of the crashing of the Hostapd service. Even greater values for the SA Query timeouts had an identical outcome; the Hostapd paralyzed and needed to be restarted, thus inevitably disconnecting the STAs. We assume that this behavior is due to some counter overflow in the C code. Lastly, regarding Hostapd, as in scenario II, we observed identical behavior. That is, Hostapd is probably locked to a maximum CPU and RAM usage, and will not consume all the available resources on the host machine.

Given the above remarks, we ended up to the following conclusions. First, after some time, in contrast to the Hostapd, the ASUS RT-AC68U was not replying to SA queries stemming from STAs. As explained previously, this time period seems to be related to the computing resources and especially the current CPU load at the AP side. The result of this behavior was the AP to receive most of the times a Disassociation protected frame from the under-attack STA, with the reason ''STA requesting (re)association is not authenticated with responding STA (0×0009)'' or the reason ''Disassociated because sending STA is leaving (or has left) BSS (0×0008)''. Also, in a limited number of cases, the Deauthentication protected frame was transmitted by the STA, with the reason ''Previous authentication no longer valid (0×0002)''. Scarcely, the STA did not respond to SA Queries, and as a result, the AP disconnected the former after transmitting a Disassociation protected frame (0×0009).

Summarizing the above discussion, the resilience of 802.11w in DoS attacks seems to be dependent mainly on the CPU load at the AP, and especially the rather low default values of SA Query maximum and retry timeouts. Naturally, these parameters are directly proportional to extra factors, including the magnitude of the attack, the position of the aggressor and if they are stationary or not, the communication channel (2.4 or 5 GHz), the load status of each STA, and the 802.11w operation mode, namely ''capable'' or ''required''. The software driver of the WNIC is also an important factor.

For instance, Alfa and TP-Link-based devices lack of official support of 802.11w, namely the particular devices are not certified by Wi-Fi Alliance. Also, while WinPro-TP-Link, WinEnter-TP-Link on the one hand and WinPro-Alfa, WinEnter-Alfa on the other, employed the same models of WNICs, they used different drivers for each version of this OS, respectively. Namely, the WinPro-TP-Link and WinPro-Alfa worked on the uncertified MS Windows wireless drivers,

while the WinEnter-TP-Link and WinEnter-Alfa on the official TP-Link and Alfa ones, respectively. So, contrariwise to Intel AX200, which is Wi-Fi Alliance certified and uses the same drivers on both OSs, the TP-Link and Alfa-based devices were more unstable to attacks and demonstrated certain incompatibilities regarding PMF. At the same time, although iPhone is also not Wi-Fi Alliance certified, it coped very well compared to the rest of the smartphones. No less important, as shown in scenario VI, increasing the values of SA Query max and retry timeout, can mitigate the DoS effect at least to some extent. Finally, as expected, it was perceived that the behavior of 802.11w was the same regardless of the authentication method used, i.e., WPA2, WPA3, or Hotspot.

We also discerned that an association flooding attack (Att6) can attain faster results vis-à-vis a reassociation one; after some time, ≈10 sec, the AP will most probably disassociate the STA, sending to it a protected frame with reason code (0×0009) "STA requesting (re)association is not authenticated with responding STA". As already indicated, this may happen if the STA does not reply to the first SA Query send out by the AP. Also, it was observed that an association flooding can be victorious with the exploitation of only one attack terminal, and in cases where the device shows greater sturdiness, if executed in parallel with a beacon flooding. No less important, the main distinction between a (re)association and a deauthentication/disassociation flooding is that in the former case the STA will automatically reconnect to AP after a few seconds of disconnection, typically, 3 to 4 sec. In the latter case however, the STA will remain disconnected until the attack ceases.

An important remark regarding Att1 to Att12 is that when the STA disconnected itself from the AP, the latter kept buffered any remaining SA Query for that STA. When the STA tried to reconnect to the same AP, it received along with the messages of the 4-way handshake, the buffered SA Queries, if any. This leaded into a deadlock, namely the STA could not proceed with the 3rd message of the handshake, as long as it received these SA Queries. To cope with this situation, the user had to try to reconnect several times to the AP for it to clear the remaining buffered SA Query frames and be able to reconnect the STA. Therefore, as pointed out in section III, an attacker could possibly capture these unencrypted SA Query frames and keep replaying them towards the STA for blocking it to reconnect to the AP. One can say that this phenomenon shares the same roots as the Kr00k vulnerability, nevertheless the attacker's aim is fundamentally different. In fact, traces of these frames exist in at least half of the pcaps that pertain to 802.11 specific attacks. Interestingly, this behavior regarding unencrypted SA Queries persists even if the AP is updated with the latest firmware (v3.0.0.4.386.40558), which patches Kr00K as well.

Lastly, from our experiments it is derived that the unencrypted SA Query phenomenon may apply to certain STAs too. This was observed for the WinPro-Linksys, which seems to not abide with the 802.11w amendment regarding the SA Query retry and maximum timeouts. Precisely, as it can be noticed from "Disass" (Att3) and "Kr00k" (Att12) pcap files, after launching Att3 with Aireplay-ng, a deadlock situation was emerged. The STA did send SA queries, but in an uncontrolled manner; about 100 such frames per sec were dispatched towards the AP, thus preventing the latter from responding with a dissasociation protected frame when a timeout had been reached. However, at that time, the AP deleted the SA with the STA, leading to a couple of side effects. First, Aireplay-ng becomes aware of this situation and ceases the attack against the AP, concentrating only against the STA, and second the STA continued sending protected SA Query requests having the same transaction_id, but different sequence number, meaning that the replay flag was set to false. The AP on the other hand responded with unencrypted dissasociation frames. We realized that this deadlock kind of situation continues even if the attack is stopped, and eventually either (or both in some rare cases) the STA crashes after displaying a "blue screen of death" in Windows OS or it deletes the corresponding SA and sends an protected deauthentication frame towards the AP. In the latter case, if the attack is still ongoing, the STA will try to reconnect to the AP in vain. During this process however, the STA will start transmitting a lot of unprotected SA Query requests. Naturally, this strange behavior can be exploited by an attacker having the vulnerable STA and the AP to attack each other. Traces of the same behaviour for the same STA can be also observed in the "Deauth" (Att1) pcap file, however, in this case, the devices do not attack each other if the attack ceases[9].

VI. RELATED WORK

This section examines the relevant work with a particular focus on the original AWID and amendment 802.11w. Recall that contributions on the security of WPA3, including those in [43]–[45] are intentionally left out as the respective attacks have not been included in the current dataset.

As already pointed out, this work complements the original AWID dataset, which was built on WPA/WPA2 personal and released in 2015. Since then, AWID has been requested and downloaded by more than 730 universities, research labs, and companies worldwide, and to the best of our knowledge, comprises the only full-fledged 802.11-oriented corpus so far. By examining the literature, one can descry a significant number of research works that capitalized on AWID to develop and assess machine learning-driven wireless network intrusion systems [46]–[51]. Indicatively, a recent survey on research exploiting AWID is given in [52].

The present work adds a number of key features to the original corpus, namely, a WPA2 enterprise deployment under a realistic testbed, assessment of PMF resilience, the inclusion of modern attacks, including Krack and Kr00k, and the encompassing of higher-layer assaults under the prism of attack escalation. Additionally, the dataset is offered in pcap cleartext format, thus allowing the community to utilize any

[9]These issues have been reported by the authors to Wi-Fi Alliance.

possible feature residing at any layer, including the application one.

Regarding the 802.11w amendment, one can observe a rather limited number of research works devoted to its security evaluation. First, the authors in [53] reported on three potential attack scenarios against 802.11w. The first was related to Broadcast/Multicast Integrity Protocol (BIP), which provides data origin authenticity and replay protection to broadcast/multicast robust management frames. Recall that this protection is offered after the STA and AP have successfully created an Integrity Group Temporal Key Security Association (IGTKSA), meaning installed an Integrity Group Temporal Key (IGTK) transferred in the 3rd message of the 4-way handshake. By abusing this network-wide key, a malevolent insider can potentially launch a protected deauthentication/disassociation attack. The second was related to the SA Query (request/response) frame, which is meant to defend against deauthentication/disassociation attacks. The attacker first starts the exchange of such frames by sending a spoofed unprotected deauthentication one, and then blocks the legitimate SA Query response frames from reaching the AP. After some time, the AP will delete the active SA with that STA, thus obliging the latter to establish a new one with the AP. The last attack scenario exploited the Traffic Indicator Element (TIE) field added with the advent of 802.11w (and later in IEEE 802.11-2016) in the Association response frame to defend against Association request attacks. That is, when the AP receives a spoofed (re)association request frame in the name of a STA which is already connected to it, the AP must reply with a rejection notice (status code $0 \times 1e$ (30)). This response includes the Association Comeback Time (TUS). The attacker may create a spoofed Association response frame, that contains a status code (30) and a very high value of TIE, thus holding off the STA from (re)associating with the AP. For evaluating the above-mentioned attacks, the authors employed a testbed composed of Hostapd v0.7.3 as a software AP and wpa_supplicant v0.7.3 as the client STA.

The contribution in [54] offered a formal analysis of 802.11w deadlock vulnerabilities. The authors evaluated their findings on Hostapd v0.8.x and wpa_supplicant 0.8.x, after disabling the wpa_supplicant optional deadlock recovery mechanism in 802.11w. They specifically reported on three vulnerabilities along with the relevant attack cases. First, an attacker could sent an unprotected deauthentication frame after the STA receives the third message of the 4-way handshake. This would cause a protocol deadlock and eventually disconnect the STA from the AP. The second scenario assumes that the AP and STA have an active SA. Then, the attacker, impersonating the STA, sends first an unprotected authentication request followed by an association one toward the AP. This would drive the AP to delete the relevant SA and trigger a 4-way handshake with the STA. Given that the latter is unaware of this situation, the 4-way handshake will result in a timeout. After that, the AP will send an unprotected Deauthentication notification to the STA, which however will be dropped by the STA. The last deadlock was related to the Channel Switch Announcement (CSA) element, which is used by an AP in a BSS to advertise a shift to a new radio channel along with the new channel number. This element is shared in beacon, probe response, and/or action management frames. Again, this scenario supposes that the AP and the target STA have an active SA. The aggressor, impersonating the AP, transmits a beacon, instructing the victim(s) STA to shift to a different channel. Then, impersonating the STA, it sends a first association request (note that the STA is on another channel so it will not reply) followed by another one after the SA Query timeout of the first expires. Finally, the aggressor transmits a beacon frame instructing the STA to change its channel to the original one. In the meantime, the AP has deleted the relevant SA and has started a 4-way handshake with the STA, eventually leading to the same result as in the previous scenario.

Lastly, the work in [55] offers a theoretical survey about vulnerabilities and corresponding attacks which may cause DoS in 802.11w-protected networks.

Given the above analysis, it is apparent the no work so far offers a wholemeal assessment of PMF under a variety of modern equipment, both STAs and APs. From our results summarized in section V, it can be argued that while PMF makes legacy DoS attacks harder, it is not unbeatable when it comes to a persistent opponent. The most salient takeaway in this case is that due to hardware restrictions, possible software bugs, and incompatibilities vis-à-vis the standard, most off-the-self devices cannot withstand a sustained bombardment of unsolicited management frames for a prolonged period.

VII. CONCLUSION

The existing literature on 802.11 lacks a comprehensive study on attacks conducted in an IEEE 802.1X EAP environment, where also the 802.11w safeguard is enabled. Motivated by this fact, based on a full-fledged testbed, we scrutinized on more than two handfuls of attacks exercised mostly at the 802.11 MAC layer, but also on higher ones, including the application layer. Especially for the 802.11-oriented attacks, and by considering diverse setups, we meticulously studied their effect on devices with a particular focus on PMF and elaborated on probable causes. Among other interesting results, our study reveals that even with the protection of PMF, legacy DoS attacks exploiting 802.11 management frames are still quite feasible on legacy devices by using off-the-self equipment. Another goal of this work is the build up of a dataset that is offered to the community for serving either research or educational needs. In fact the dataset complements the well-known AWID, which has compiled under WPA/WPA2-personal settings. In addition, an initial firsthand evaluation of almost half of the included pcap files is offered based on selected features. This work can be extended by embracing WPA3-focused attacks along with others exercised against Independent Basic Service Set (IBSS) type of network.

REFERENCES

[1] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis, "Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 184–208, 1st Quart., 2016, doi: 10.1109/COMST.2015.2402161.

[2] *LAN MAN Standards Committee of the IEEE Computer Society IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Standard 802.11-1997, 1997. Accessed: Nov. 11, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/654749

[3] *C/LM—LAN/MAN Standards Committee 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 Amendment 4: Protected Management Frames*. Standard 802.11w-2009, 2009. [Online]. Available: https://standards.ieee.org/standard/802_11w-2009.html

[4] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in WPA2," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, M. Bhavani and I. Thuraisingham, Eds., Dallas, TX, USA, Oct./Nov. 2017, pp. 1313–1328, doi: 10.1145/3133956.3134027.

[5] M. Vanhoef and F. Piessens, "Advanced Wi-Fi attacks using commodity hardware," in *Proc. 30th Annu. Comput. Secur. Appl. Conf. (ACSAC)*, A. Hahn, K. R. B. Butler, and M. Sherr, Eds., New Orleans, LA, USA, Dec. 2014, pp. 256–265, doi: 10.1145/2664243.2664260.

[6] R. L. S. Svorencik. *Kr00k: How Kracking Amazon Echo Exposed a Billion+ Vulnerable Wifi Devices*. Accessed: Nov. 11, 2020. [Online]. Available: https://www.welivesecurity.com/2020/08/06/beyond-kr00k-even-more-wifi-chips-vulnerable-eavesdropping/

[7] ESET. *Kr00k—A Serious Vulnerability Deep Inside Wi-Fi Encryption*. Accessed: Nov. 11, 2020. [Online]. Available: https://www.eset.com/int/kr00k/

[8] ESET. *Beyond KrØØk: Even More Wi-Fi Chips Vulnerable to Eavesdropping*. Accessed: Nov. 11, 2020. [Online]. Available: https://www.welivesecurity.com/2020/08/06/beyond-kr00k-even-more-wifi-chips-vulnerable-eavesdropping/

[9] *IEEE 802.1X-2010—IEEE Standard for Local and Metropolitan Area Networks-Port-Based Network Access Control*. Accessed: Dec. 13, 2004. [Online]. Available: https://standards.ieee.org/standard/802_1X-2010.html

[10] *Freeradius*. Accessed: Nov. 11, 2020. [Online]. Available: https://freeradius.org/

[11] *Snbforums—Discussion Related to CPU Isses of Asus Ac68u Firmware Updates*. Accessed: Nov. 18, 2020. [Online]. Available: https://www.snbforums.com/threads/asus-rt-ac68u-firmware-version-3-0-0-4-386-40558-05-nov-2020.67462/

[12] *Snbforums—Recent Discussion Regarding the Issues of the Lastest Firmware of Asus Ac68u*. Accessed: Nov. 18, 2020. [Online]. Available: https://www.snbforums.com/threads/asus-rt-ac68u-firmware-version-3-0-0-4-385-20253.62620/

[13] *Freeipa*. Accessed: Nov. 11, 2020. [Online]. Available: https://www.freeipa.org/page/Main_Page

[14] *Mitmproxy—A Free and Open Source Interactive*. Accessed: Nov. 11, 2020. [Online]. Available: https://mitmproxy.org/

[15] *Selenium*. Accessed: Nov. 11, 2020. [Online]. Available: https://www.selenium.dev/

[16] *Node JS*. Accessed: Nov. 11, 2020. [Online]. Available: https://nodejs.org/en/

[17] *Puppeteer—Headless Chrome Nodejs Api*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/puppeteer/puppeteer

[18] *Geckodriver*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/mozilla/geckodriver/releases

[19] *Chromedriver*. Accessed: Nov. 11, 2020. [Online]. Available: https://chromedriver.chromium.org/

[20] *Freeradius Man Pages—Radsniff*. Accessed: Nov. 11, 2020. [Online]. Available: https://freeradius.org/radiusd/man/radsniff.html

[21] *R00kie-Kr00kie POC*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/hexway/r00kie-kr00kie

[22] *Ares—Python Botnet and Backdoor*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/sweetsoftware/Ares

[23] D. M. Blog. *Third Time's the Charm? Analysing Wannacry Samples*. Accessed: Nov. 11, 2020. [Online]. Available: https://dissectingmalwa.re/third-times-the-charm-analysing-wannacry-samples.html

[24] Kaspersky. *Teslacrypt Ransomware Attacks*. Accessed: Nov. 11, 2020. [Online]. Available: https://www.kaspersky.com/resource-center/threats/teslacrypt

[25] *Thezoo—A Repository of Live Malwares*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/ytisf/theZoo

[26] *Damn Vulnerable Web Application Docker Container*. Accessed: Nov. 11, 2020. [Online]. Available: https://hub.docker.com/r/vulnerables/web-dvwa/

[27] *SSDP Amplification Blog post—Scanning Python Script*. Accessed: Nov. 11, 2020. [Online]. Available: https://blog.cloudflare.com/ssdp-100gbps/

[28] J. Malinen. *hostapd and Wpa_Supplicant*. Accessed: Nov. 11, 2020. [Online]. Available: https://w1.fi/

[29] *Dnsmasq*. Accessed: Nov. 11, 2020. [Online]. Available: https://www.openhub.net/p/dnsmasq

[30] *Apache2—The Apache*. Accessed: Nov. 11, 2020. [Online]. Available: https://httpd.apache.org/

[31] *Eaphammer*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/s0lst1c3/eaphammer

[32] *Aircrack-ng—Wifi Security Auditing Tools Suite*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/aircrack-ng/aircrack-ng

[33] *Netattack—A Simple python Script to Scan and Attack Wireless Networks*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/chrizator/netattack

[34] *Mdk4*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/aircrack-ng/mdk4

[35] Scapy. *python-Based Interactive Packet Manipulation Program & Library*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/secdev/scapy

[36] *Nmap—The Network Mapper*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/nmap/nmap

[37] *Saddam 2nd Implementation*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/S4kur4/Saddam-new

[38] *Bettercap—The Swiss Army Knife of 802.11*. Accessed: Nov. 11, 2020. [Online]. Available: https://www.bettercap.org/

[39] *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*, Standard 802.11-2016, Accessed: Sep. 30, 2020. [Online]. Available: https://standards.ieee.org/standard/802_11-2016.html

[40] *Hostapd Configuration File*. Accessed: Nov. 11, 2020. [Online]. Available: https://w1.fi/cgit/hostap/plain/hostapd/hostapd.conf

[41] *Asuswrt ap Config*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/RMerl/asuswrt-merlin/blob/9f14d213d07fa36da459424a699bfe85f15b2286/release/src/router/wpa_supplicant-0.7.3/src/ap/ap_config.c

[42] *Asuswrt ap Config*. Accessed: Nov. 11, 2020. [Online]. Available: https://github.com/RMerl/asuswrt-merlin/blob/9f14d213d07fa36da459424a699bfe85f15b2286/release/src/router/wp_supplicant-0.7.3/src/ap/ap_config.h

[43] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the dragonfly handshake of WPA3 and EAP-PWD," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 517–533.

[44] K. Lounis and M. Zulkernine, "Bad-token: Denial of service attacks on WPA3," in *Proc. 12th Int. Conf. Secur. Inf. Netw. (SIN)*, New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–8, doi: 10.1145/3357613.3357629.

[45] K. Lounis and M. Zulkernine, "WPA3 connection deprivation attacks," in *Proc. 14th Int. Conf.* (Lecture Notes in Computer Science), vol. 12026, S. Kallel, F. Cuppens, N. Cuppens-Boulahia, and A. H. Kacem, Eds. Hammamet, Tunisia: Springer, Oct. 2019, pp. 164–176, doi: 10.1007/978-3-030-41568-6_11.

[46] S. J. Lee, P. D. Yoo, A. T. Asyhari, Y. Jhi, L. Chermak, C. Y. Yeun, and K. Taha, "IMPACT: Impersonation attack detection via edge computing using deep autoencoder and feature abstraction," *IEEE Access*, vol. 8, pp. 65520–65529, 2020, doi: 10.1109/ACCESS.2020.2985089.

[47] S. M. Kasongo and Y. Sun, "A deep learning method with wrapper based feature extraction for wireless intrusion detection system," *Comput. Secur.*, vol. 92, May 2020, Art. no. 101752, doi: 10.1016/j.cose.2020.101752.

[48] Y. Zhou, G. Cheng, S. Jiang, and M. Dai, "Building an efficient intrusion detection system based on feature selection and ensemble classifier," *Comput. Netw.*, vol. 174, Jun. 2020, Art. no. 107247, doi: 10.1016/j.comnet.2020.107247.

[49] J. W. Mikhail, J. M. Fossaceca, and R. Iammartino, "A semi-boosted nested model with sensitivity-based weighted binarization for multi-domain network intrusion detection," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 3, pp. 28:1–28:27, 2019, doi: 10.1145/3313778.

[50] M. E. Aminanto, R. Choi, H. C. Tanuwidjaja, P. D. Yoo, and K. Kim, "Deep abstraction and weighted feature selection for Wi-Fi impersonation detection," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 3, pp. 621–636, Mar. 2018, doi: 10.1109/TIFS.2017.2762828.

[51] C. Kolias, V. Kolias, and G. Kambourakis, "TermID: A distributed swarm intelligence-based approach for wireless intrusion detection," *Int. J. Inf. Secur.*, vol. 16, no. 4, pp. 401–416, Aug. 2017, doi: 10.1007/s10207-016-0335-z.

[52] A. A. Reyes, F. D. Vaca, G. A. Castro Aguayo, Q. Niyaz, and V. Devabhaktuni, "A machine learning based two-stage Wi-Fi network intrusion detection system," *Electronics*, vol. 9, no. 10, p. 1689, Oct. 2020. https://www.mdpi.com/2079-9292/9/10/1689

[53] M. S. Ahmad and S. Tadakamadla, "Short paper: Security evaluation of IEEE 802.11w specification," in *Proc. 4th ACM Conf. Wireless Netw. Secur.*, D. Gollmann, D. Westhoff, G. Tsudik, and N. Asokan, Eds., Hamburg, Germany, Jun. 2011, pp. 53–58, doi: 10.1145/1998412.1998424.

[54] M. Eian and S. F. Mjølsnes, "A formal analysis of IEEE 802.11w deadlock vulnerabilities," in *Proc. IEEE INFOCOM*, A. G. Greenberg and K. Sohraby, Eds., Orlando, FL, USA, Mar. 2012, pp. 918–926, doi: 10.1109/INFCOM.2012.6195841.

[55] B. Bertka, "802.11 w security: Dos attacks and vulnerability controls," Univ. Brit. Columbia, Vancouver, BC, Canada, Tech. Rep., 2012, pp. 1–11. [Online]. Available: http://blogs.ubc.ca/computersecurity/files/2012/04/BBertka_bbertka_571B_final.pdf

EFSTRATIOS CHATZOGLOU received the M.Sc. degree in security of information and communication systems from the University of Aegean, Samos, Greece. He was worked as a Web Developer in an Integrated Health Information System web application with the Hellenic Army General Staff, Greece. He is currently a Penetration Tester with the Hellenic National Defence General Staff, Greece. His research interests include wireless and cellular networks security, the IoT networks security, Android application security, Web application security, and machine learning.

GEORGIOS KAMBOURAKIS is currently a Full Professor with the Department of Information and Communication Systems Engineering, University of the Aegean, Greece. He has served as the Head of the Department, from September 2019 to October 2019. He was the Director of the Information Security Laboratory, from September 2014 to December 2018. He is currently on unpaid leave from the University of the Aegean, while he is working with the European Commission, European Joint Research Centre (JRC), Ispra, Italy. His research interests include mobile and wireless networks security and privacy, VoIP security, the IoT security and privacy, DNS security, and security education. He has more than 145 refereed publications in the aforementioned areas. More info at: http://www.icsd.aegean.gr/gkamb

CONSTANTINOS KOLIAS received the Ph.D. degree from the University of the Aegean, in 2014. In 2018, he joined the Department of Computer Science, University of Idaho. Before that, he was a Research Assistant Professor with the Department of Computer Science (CS), George Mason University. He is also active in the design of intelligent IDS with a special interest in privacy preserving distributed IDS. His main research interests include security and privacy for the IoT and critical infrastructures. Other areas of interest include mobile and wireless communications security, and privacy enchasing techniques for the Internet. More info at: https://www.uidaho.edu/engr/departments/cs/our-people/faculty/constantinos-kolias