FragAttacks: Forging Frames in Protected Wi-Fi Networks

Mathy Vanhoef
New York University Abu Dhabi
mathy.vanhoef@nyu.edu

In this white paper, we summarize three design flaws that we discovered the 802.11 standard that underpins Wi-Fi. One design flaw is in the frame aggregation functionality, and another two are in the frame fragmentation functionality. These design flaws enable an adversary to forge encrypted frames, which in turn enables exfiltration of sensitive data. We also discovered common implementation flaws related to aggregation and fragmentation, which further worsen the impact of our attacks. Our results affect all protected Wi-Fi networks, meaning the discovered flaws have been part of Wi-Fi since its release in 1997. In our experiments, all devices were vulnerable to one or more of our attacks, confirming that all Wi-Fi devices are likely affected.

1 INTRODUCTION

Lately major improvements have been made to the security of Wi-Fi. This includes the discovery and prevention of key reinstallation in WPA2 [37, 38, 12] and the standardization of WPA3 which [41]. Although the initial release of WPA3 had some issues [40, 39] these have been addressed in the latest release of the 802.11 standard [20]. Additionally, extra defenses have been standardized, such as operating channel validation and beacon protection, which further increases the security of Wi-Fi networks [36, 35].

Despite the recent advances in Wi-Fi security, we found design issues that went unnoticed for more than two decades. Our results affect all protected Wi-Fi networks, including old

This white paper is a shortened version of the full USENIX Security paper [34] as background for the Black Hat US '21 presentation "FragAttacks: Breaking Wi-Fi through Fragmentation and Aggregation".

networks using Wired Equivalent Privacy (WEP), up to and including the latest Wi-Fi Protected Access 3 (WPA3). Since even WEP is affected, this implies the root cause of several design flaws has been part of Wi-Fi since its release in 1997. Equally worrisome is that every single device we tested was vulnerable to at least one of our attacks.

The most trivial design flaw is in 802.11's frame aggregation functionality: by flipping an unauthenticated flag in the header of a frame, the encrypted payload will be parsed as containing one or more aggregated frames instead of a normal network packet. We abuse this to inject arbitrary frames, and then intercept a victim's traffic by making it use a malicious DNS server. Practically all devices are vulnerable to this attack.

Another two design flaws are in 802.11's frame fragmentation feature which splits large frames into smaller fragments. First, although all fragments of a frame are always encrypted under the same key, receivers are not required to check that this is indeed the case. We show that an adversary can abuse this to forge frames and exfiltrate data by mixing fragments encrypted under different keys. Second, a receiver is not required to remove (incomplete) fragments from memory when connecting to a different network. We abuse this to inject malicious fragments into the fragment cache, i. e., memory, of the victim and thereby inject arbitrary packets.

We also discovered widespread implementation vulnerabilities related to frame aggregation and fragmentation. These vulnerabilities can either be exploited on their own or make it significantly easier to abuse the discovered design issues.

We believe that the discovered design flaws went unnoticed for so long for two reasons. First, some of the functionality that we abuse is generally not considered as part of the core cryptographic functionality of Wi-Fi and therefore has received no rigorous analysis. Second, patched drivers or firmware are needed to confirm the fragmentation-based vulnerabilities in practice. When using normal drivers, certain fields of injected frames may be overwritten without the programmer realizing this. This causes attacks to fail, and as a result researchers may mistakenly conclude that devices were secure.

2 BACKGROUND

This section introduces the 802.11 standard [19] and gives a high-level description of the design flaws that we discovered.

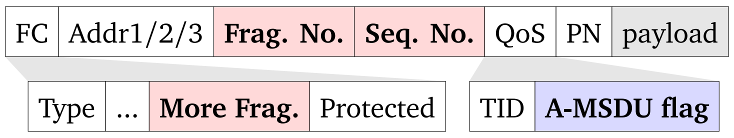

Figure 1: Layout of an encrypted 802.11 frame. Our aggregation attack abuses the field in blue, and our fragmentation attacks the fields in red. Only the payload field is encrypted.

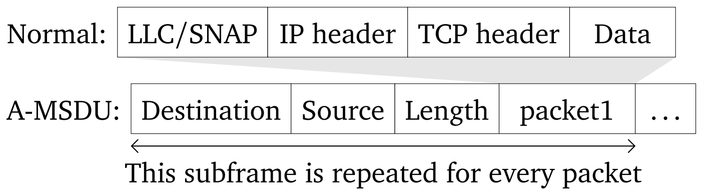

Figure 2: Payload of a normal frame with an TCP/IP header (top), and the payload of an A-MSDU frame meaning the payload contains one or more aggregated packets (bottom).

2.1 Frame layout and packet aggregation

Figure 1 shows the layout of a Wi-Fi frame. First, the Frame Control (FC) field contains several flags and defines the type of a frame, e. g., whether it is a data or management frame. This is followed by three MAC addresses defining the receiver, sender, and the destination or source of the frame. The Quality of Service (QoS) field defines the priority (TID) of the frame. The payload field of a normal frame contains the transported packet, which starts with an LLC/SNAP header—also called an rfc1042 header [29]—that defines the type of the packet, e. g., whether it is an IP or ARP packet (see Figure 2).

When the packet is small it is more efficient to aggregate multiple packets into one larger frame. The 802.11n amendment defines two aggregation methods [22], and we focus on Aggregate MAC Service Data Units (A-MSDUs). The layout of an A-MSDU frame is similar to a normal frame as shown in Figure 1, except that the A-MSDU flag in the QoS field is set, and that the payload field contains one or more A-MSDU subframes as shown in Figure 2. Each subframe starts with the equivalent of an 802.3 header: the destination and source MAC address of the packet, followed by the length of the packet.

When a receiver sees that the A-MSDU flag is set in the QoS field, it will extract all A-MSDU subframes and convert them into Ethernet frames with the destination and source addresses specified in the subframe. The problem is that, although the QoS field is authenticated, by default the A-MSDU flag is masked to zero, meaning this flag is not actually authenticated. As a result, an adversary can intercept a normal frame, set the A-MSDU flag, and the receiver will now incorrectly interpret the payload as containing A-MSDU subframes. In Section 3 we show how to abuse this to inject arbitrary frames.

2.2 Frame fragmentation

In noisy environments it can be more efficient to split large frames into smaller fragments. The layout of a fragment, also called a MAC Protocol Data Unit (MPDU), is identical to a normal frame and illustrated in Figure 1. Because of their similarity, we use the term *frame* to refer to both a normal frame and an MPDU, while the term *fragment* will be used to explicitly refer to an MPDU. When a frame is split into multiple fragments, each one is assigned an incremental 4-bit fragment number (Frag. No. in Figure 1). To allow a receiver to determine when all fragments have been received, every fragment except the last has the more fragments flag set in its frame control field. Finally, all fragments of a specific frame have the same 12-bit sequence number (Seq. No. in Figure 1). We use the notation $\text{Frag}_x(s)$ to denote a fragment with fragment number x and sequence number s. For instance, $\text{Frag}_1(9)$ denotes a 2nd fragment with sequence number 9.

2.3 Authentication and encryption

In both protected Wi-Fi networks, the client will use the 4-way handshake to negotiate a pairwise session key with the AP. This session key is used to encrypt data frames. At any point in time, the AP can start a new 4-way handshake to renew the session key.

When the (AES-)CCMP or GCMP data-confidentiality protocol is used, large frames are first split into fragments, and all fragments are then encrypted in the same way as normal frames: the payload field is authenticated and encrypted, and selected metadata is also authenticated. This metadata encompasses, among other things, all MAC addresses in the header, the fragment number, and the more fragments flags. The sequence number is not authenticated. Every encrypted frame also has a strictly increasing Packet Number (PN), commonly called a nonce, which is used to prevent replay attacks, and is implicitly authenticated by the data-confidentiality protocol. We use the notation $\text{Enc}_k^n\{f\}$ to denote the encryption of frame f using key k and packet number n.

A receiver first checks if the PN is increasing and otherwise drops the fragment (or frame). Then it decrypts the fragment and stores it until all fragments are received [19, Fig. 5-1]. On reception of the last fragment, all decrypted fragments are reassembled into the original frame. To prevent an adversary from forging a frame by combining fragments of different frames, a receiver must drop all fragments if their PNs are not consecutive.

2.4 Attack techniques and scenarios

Several attacks rely on a multi-channel machine-in-the-middle (MitM) position and one attack targets hotspot-type networks. We therefore introduce these concepts first:

Multi-Channel MitM In a multi-channel MitM, the adversary clones the real AP on a different channel, forces the client into connecting to the rogue AP on the cloned channel, and forwards frames between the client and the real AP. The adversary can then modify frames before forwarding them or not forward them at all.

Hotspot security In modern hotspot-type networks such as eduroam and Hotspot 2.0 networks each user owns unique authentication keys and as a result their encryption keys also stay secret. To prevent users from attacking each other, hotspots commonly use downstream group-addressed forwarding and client isolation. With the former feature, each client is given a random group key [3, §5.2], preventing attacks that abuse the otherwise shared group key [2]. The latter feature, client isolation, prevents users from communicating with each other, which most notably blocks ARP-based MitM attacks.

3 ABUSING FRAME AGGREGATION (CVE-2020-24588)

In this section, we present a design flaw in 802.11's frame aggregation functionality that allows an adversary to inject arbitrary packets.

3.1 Threat model

The attack works against all current data-confidentiality protocols of Wi-Fi, namely WEP, TKIP, CCMP, and GCMP, meaning all protected Wi-Fi networks are affected. The adversary must be within radio range of the victim such that a multi-channel MitM can be obtained, and the victim must support the reception of A-MSDU frames, which is a mandatory part of 802.11n [22]. Additionally, the adversary must be able to send IPv4 packets to the victim with some control over the payload and with a predictable IP identification (ID) field. In this white paper we focus on attacks against clients. If the IP address of the client is known we can directly send IPv4 packets to the victim. Otherwise, the adversary has to trick the victim into connecting to a server under the adversary's control, allowing the adversary to inject IPv4 packets over this connection.

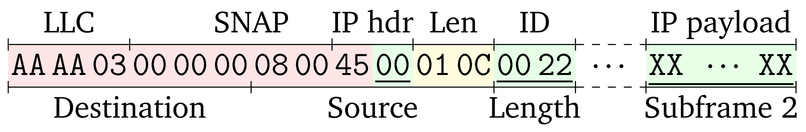

Figure 3: Parsing an 802.11 payload containing an IPv4 packet (top) as an A-MSDU frame (bottom). Green underlined bytes can be controlled by the adversary, yellow ones can be partially controlled, and red bytes have a fixed value.

3.2 Injecting frames by spoofing A-MSDUs

By default the A-MSDU flag, which informs a receiver how to parse the encrypted payload of a frame, is not authenticated (recall Section 2.1). Only when the sender and receiver support Signaling and Payload Protected (SPP) A-MSDUs is the A-MSDU flag authenticated [22, §11.17]. However, none of the devices we tested support this feature, meaning in practice the A-MSDU flag is never authenticated. This is problematic because nearly all devices we tested do support receiving A-MSDUs, meaning they can be tricked into processing normal frames as A-MSDUs, and vice versa.

We can exploit this design flaw by manipulating a normal 802.11 frame such that, when it is processed as an A-MSDU frame, one of the subframes will correspond to a packet that we want to inject. This requires the frame's payload to contain a specially crafted packet, for instance, the IPv4 packet illustrated in Figure 3. Notice that this IPv4 packet is prepended with an 8-byte LLC/SNAP header when encapsulated in an 802.11 frame (recall Section 2.1). When targeting a client in our threat model, the adversary can control the IP ID field and the payload that follows the IPv4 header. When these bytes are interpreted as A-MSDU subframes, the length field of the first subframe corresponds to the IP ID field (see Figure 3). This means the attacker can set the 2-byte IP ID field to, e. g., 34, meaning the next A-MSDU subframe starts after the TCP or UDP header of the injected frame. This leaves space to include a valid TCP or UDP header in the malicious IPv4 packet, increasing the chance that the packet is correctly routed to the victim. Finally, we remark that the IP ID field is not changed by NAT devices or other middleboxes [25], and therefore such devices will not interfere with our attack.

To change the IPv4 packet into an A-MSDU frame, the adversary establishes a multi-channel MitM between the client and AP (recall Section 2.4). The encrypted 802.11 frame containing the IPv4 packet is detected based on its length and QoS priority. The adversary sets the A-MSDU flag in the unauthenticated QoS field, causing the client to treat the frame's payload as A-MSDU subframes. The first subframe will have an unknown sender and destination MAC address and will be ignored. The second subframe will contain the packet that the adversary wants to inject, and the client will parse and process this injected packet.

3.3 Practical impact

The impact of injecting arbitrary packets depends on the services running on the victim, whether it is regularly updated, and so on. As a general example, we abused this to trick the victim into using a malicious DNS server.

Against dual-stack IPv4/6 clients, we can inject ICMPv6 router advertisements to trick the victim into using a DNS server under our control. More precisely, we abuse IPv6 stateless address autoconfiguration by injecting an ICMPv6 router advertisement that includes a malicious DNS server [23]. Against Linux, Windows 10, Android 8.1, iOS 13.4.1, and macOS 10.15.4, we confirmed that this successfully poisoned the DNS server(s) used by the OS. Once the victim is using the malicious DNS server, the adversary can redirect all traffic to their malicious server, effectively intercepting all IP-based traffic of the client. Note that the malicious DNS server will be hosted on an IPv6 address, but can still respond to DNS requests with IPv4 addresses if needed.

3.4 Experiments

All major operating systems are vulnerable to our attack, including Windows, Linux, Android, macOS, and iOS. See Section 6.1 for a detailed overview of the devices we tested. All APs we tested were also vulnerable, including home routers and professional APs. The only exception is NetBSD and OpenBSD: they do not support the reception of A-MSDUs and therefore are unaffected by the attack.

4 MIXED KEY ATTACK (CVE-2020-24587)

In this section, we focus on the first fragmentation-based design flaw, namely how the 802.11 standard allows an attacker to forge frames by mixing fragments that are encrypted under different keys.

4.1 Threat model

We first focus on the mixed key attack, which works against WEP, CCMP, and GCMP. The attack requires that one or more devices in the network send fragmented frames. Although not all devices do this by default, it is recommended to use fragmentation in noisy environments. The network must also periodically refresh the session key of connected devices, and we must be able to trick the victim into sending a packet to an attacker-controlled server. To trick the victim into sending a packet to a server under our control, we trick the victim into downloading an image from our server. This can for instance be accomplished by sending an email to the victim with an embedded image.

4.2 Exfiltrating sensitive data

The adversary's goal is to forge a packet by mixing fragments of frames that were encrypted under different keys. These fragments must have consecutive packet numbers since the receiver will otherwise discard the fragments. Although many implementations do not check whether fragments use consecutive packet numbers (see Section 6.2), our attack does assume the victim checks this, and thereby illustrates that even implementations that fully comply with the standard are vulnerable.

Mixing fragments Figure 4 illustrates our attack, where we exploit a vulnerable AP to exfiltrate data sent by the client. The attack starts with the generation of a packet towards the adversary's server (stage ①). This attacker-destined packet can, for instance, be generated by social engineering the victim into loading an innocent resource on the adversary's server. By hosting this resource on a long URL, the resulting packet will be large enough such that it is split in two fragments before transmission. These two encrypted fragments are represented by $\text{Enc}_k^n\{\text{Frag}_0(s)\}$ and $\text{Enc}_k^{n+1}\{\text{Frag}_1(s)\}$. The attacker then relies on a multi-channel MitM position to intercept all fragmented frames, and detects the attacker-destined packet based on its unique length. Note that the adversary must first collect all fragments of a frame before it can determine the length of the full frame. Once the attacker-destined packet is detected, the adversary only forwards the first fragment to the AP. The AP will then decrypt this fragment and will store the decrypted fragment in its memory.

Between stages ① and ② of the attack, the adversary forwards all frames between the client and the AP. To prevent these frames from interfering with the attack, the sequence number s is never used when forwarding a frame to the AP. This assures that the first

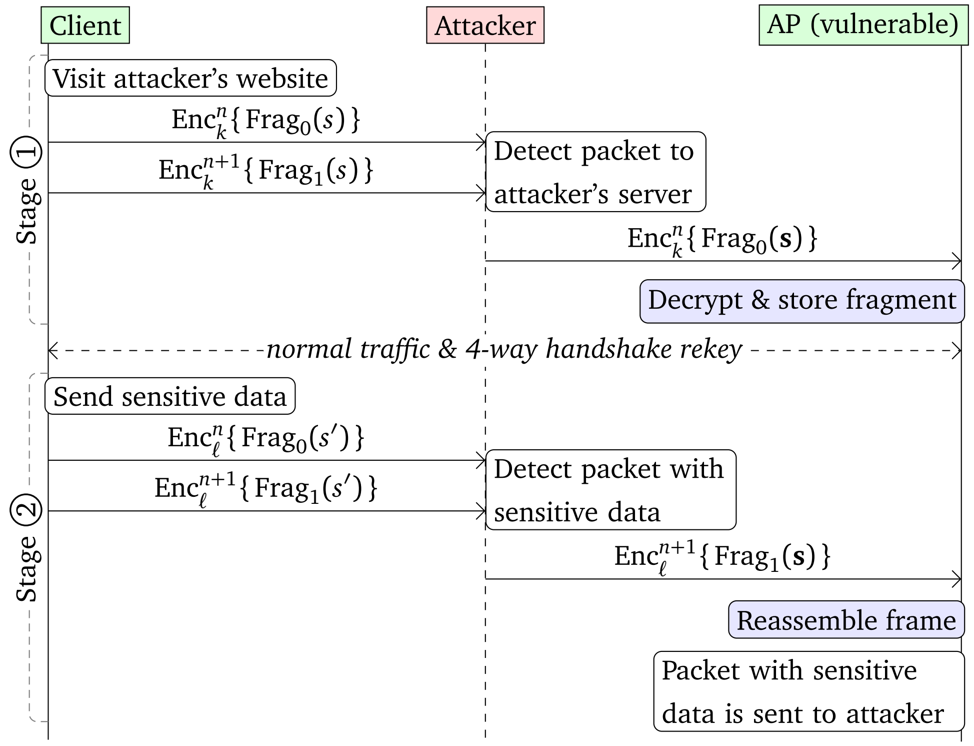
Client
Attacker
AP (vulnerable)
Stage ①
Visit attacker's website
$\text{Enc}_k^n\{\text{Frag}_0(s)\}$
$\text{Enc}_k^{n+1}\{\text{Frag}_1(s)\}$
Detect packet to attacker's server
$\text{Enc}_k^n\{\text{Frag}_0(\mathbf{s})\}$
Decrypt & store fragment
normal traffic & 4-way handshake rekey
Stage ②
Send sensitive data
$\text{Enc}_\ell^n\{\text{Frag}_0(s')\}$
$\text{Enc}_\ell^{n+1}\{\text{Frag}_1(s')\}$
Detect packet with sensitive data
$\text{Enc}_\ell^{n+1}\{\text{Frag}_1(\mathbf{s})\}$
Reassemble frame
Packet with sensitive data is sent to attacker

Figure 4: Mixed key attack against fragmentation. The first fragment is the start of an IP packet to the attacker's server, which is appended in stage ② with user data. The reassembled packet is sent to the attacker's server, exfiltrating the user data.

fragment of the attacker-destined packet is not removed from the AP's memory. Any other forwarded fragments also will not interfere with the attack, since the standard requires that a device must support the concurrent reception of at least 3 fragmented frames [19, §10.6]. Before stage ② starts, the client and AP must update, i. e., rekey, the pairwise session key from k to ℓ using the 4-way handshake. Note that the adversary can predict when rekeys occur because they happen at regular intervals, and rekeys can be detected because they cause the packet numbers of the data-confidentiality protocol to restart from zero.

Stage ② of the attack starts when the client sends a fragment containing sensitive information. This second fragment must have a packet number equal to $n+1$, and otherwise the attacker has to wait until another 4-way handshake is executed so packet numbers start from zero again. The adversary assigns sequence number s to the second fragment, which is possible because this field is not authenticated, and forwards the resulting fragment $\text{Enc}_\ell^{n+1}\{\text{Frag}_1(s)\}$ to the AP (stage ② in Figure 4). Upon reception of the second fragment, the AP combines both decrypted fragments to reassemble the packet. This packet is now a combination of the attacker-destined packet and a packet that contains

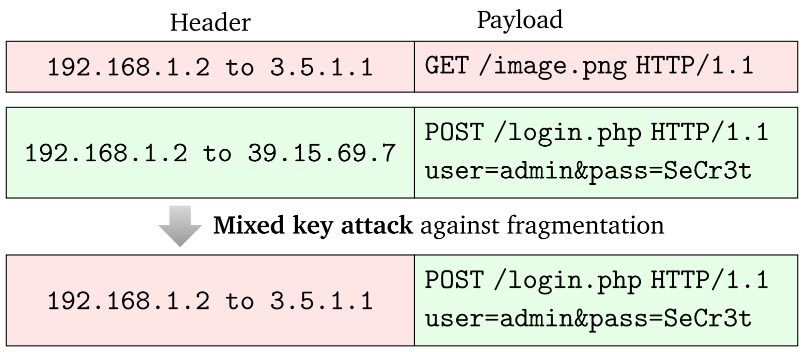

Figure 5: Mixing fragments of two packets to exfiltrate user data to the attacker's server. The red packet is sent to the attacker's server, and the green packet contains user data.

sensitive user data. Against devices that only accept fragments with consecutive packet numbers, the second fragment must have packet number $n + 1$.

Packet construction We mix a fragment of an attacker-destined IP packet with a fragment of a packet containing user data. This process is illustrated in Figure 5. The IP checksum of the forged packet is correct since it is only calculated over the IP header. The TCP checksum will be incorrect, but this has no impact on the attack: intermediate hops will still forward the packet to its final destination since they only verify the IP checksum. And since the attacker controls the final destination, they can simply ignore the incorrect TCP checksum. Finally, the attacker-destined packet must not be larger than the targeted packet with sensitive user data. Otherwise, the IP length field will be larger than the actual payload of the reassembled packet, causing the AP to drop the packet. If the IP length field is smaller than the payload, only the trailing data is discarded.

4.3 Experiments

To perform our attack, we have to inject frames with specific fragment and sequence numbers. However, wireless network cards may overwrite these fields. Additionally, network cards may reorder frames with a different QoS priority, which can also interfere with our attack. To overcome these problems, we patched the driver of Intel cards, and we patched the driver and firmware of Atheros cards.

In our tests, all major operating systems are vulnerable, including Windows, Linux, Android, macOS, and iOS. Section 6.1 contains an overview of all tested devices. A low number of devices are unaffected because they require that all fragments are received immediately after one another, and any frames sent in-between interfere with their reassembly, preventing a default mixed key attack.

5 FRAGMENT CACHE ATTACK (CVE-2020-24586)

In this section, we present a design flaw that enables an adversary to inject fragments into the memory, i. e., fragmentation cache, of victims.

5.1 Threat model

Our attacks work against WEP, CCMP, and GCMP. Similar to the mixed key attack, a device in the network must be sending fragmented frames for the attack to be possible. In this white paper we only describe attacks against APs. In particular, our attack will exploit vulnerable APs in hotspot-type networks such as eduroam, and Hotspot 2.0 networks where users can, for example, authenticate using their mobile SIM card [3]. In these networks, users may distrust each other, and they will use individual authentication and encryption keys. Our attack also works when these networks use downstream group-addressed forwarding and client isolation (recall Section 2.4).

5.2 Exfiltrating client data

We begin by attacking a vulnerable AP and exfiltrating data sent by a client. In stage ① of this attack, we spoof the MAC address of the targeted client and connect to the network using valid credentials (see Figure 6). This allows us to inject fragments into the AP's memory that are saved under the victim's MAC address. Note that the attacker possesses valid credentials since we target hotspot-type networks.

Stage ② of the attack starts when the real client sends an Auth frame in order to connect to the network. At that point, the adversary sends the encrypted fragment $\text{Enc}_k^n\{\text{Frag}_0(s)\}$ to the AP, which contains the start of an attacker-destined IP packet. The AP decrypts this fragment and stores it in its fragment cache under the victim's MAC address. After this, the attacker disconnects from the network by sending a Deauth frame, and subsequently establishes a multi-channel MitM between the client and AP. The 802.11 standard does not state that the AP must remove fragments when a client disconnects or reconnects, meaning the injected fragment stays in the fragment cache of the AP.

Between stages ② and ③ of the attack, the adversary lets the client connect normally. Additionally, the adversary never sends frames to the AP with sequence number s. This assures that the fragment that we inject in the second stage of the attack stays in the AP's fragment cache.

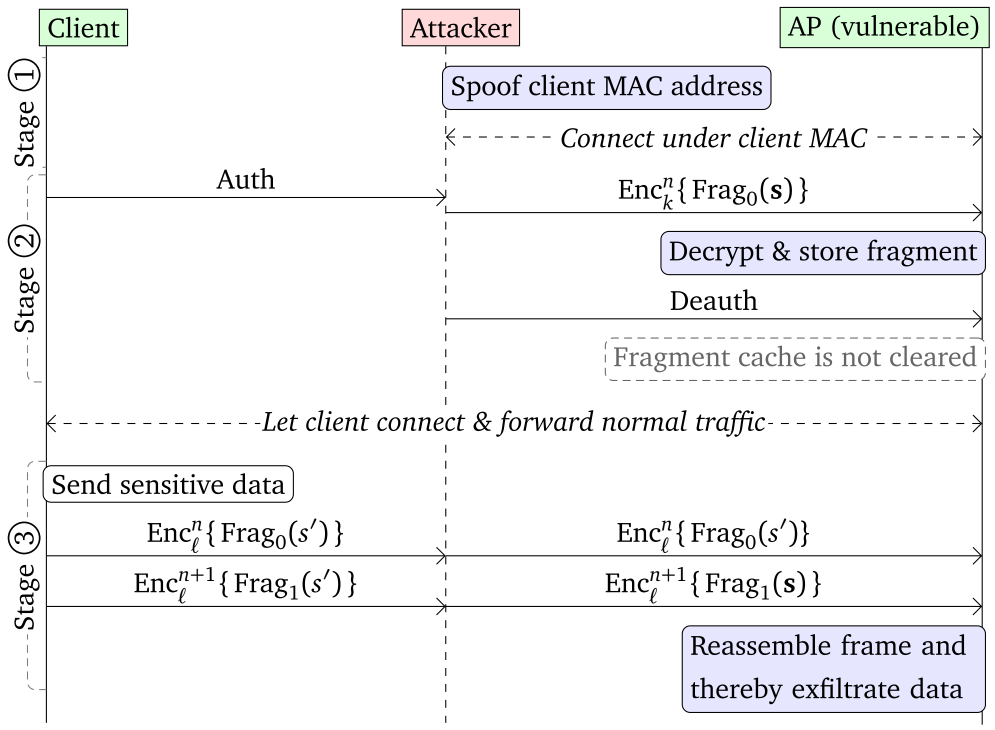
Client
Attacker
AP (vulnerable)
Stage ①
Spoof client MAC address
Connect under client MAC
Stage ②
Auth
$\text{Enc}_k^n\{\text{Frag}_0(\mathbf{s})\}$
Decrypt & store fragment
Deauth
Fragment cache is not cleared
Let client connect & forward normal traffic
Stage ③
Send sensitive data
$\text{Enc}_\ell^n\{\text{Frag}_0(s')\}$
$\text{Enc}_\ell^n\{\text{Frag}_0(s')\}$
$\text{Enc}_\ell^{n+1}\{\text{Frag}_1(s')\}$
$\text{Enc}_\ell^{n+1}\{\text{Frag}_1(\mathbf{s})\}$
Reassemble frame and thereby exfiltrate data

Figure 6: Fragment cache attack against a vulnerable AP with as goal to exfiltrate (decrypt) client data. The adversary injects a fragment with an attacker-destined IP packet, which is appended with a fragment containing sensitive data.

In stage ③ of the attack, the adversary waits until a second fragment with packet number $n + 1$ is sent. The adversary forwards this fragment to the AP with sequence number s. This causes the AP to combine it with the injected fragment since they have the same sequence number and MAC addresses. Because the AP does not store under which credentials these fragments were received, it does not realize both fragments were in fact sent by different users. The reassembled frame will contain an IP packet with as destination the adversary, and with as payload the user data (similar to Figure 5). This exfiltrates the user data to the adversary. If the frame with packet number $n + 1$ is not a second fragment, the attack can be restarted by forcibly disconnecting the client from the AP.

5.3 Packet injection

An attacker can also inject packets by poisoning the fragment cache. Against an AP this attack is similar to the data exfiltration attack of Section 5.2, except that the injected fragment Frag_0 in stage ② contains the packet to be injected. When reassembling the frame upon reception of the second fragment, unknown content will be appended to the injected frame. However, the network layer above 802.11 will discard this unknown content as padding data. The receiver knows where this padding data starts because network packets, such as IP or ARP packets, contain length fields that define the size of the packet. As a result, the adversary can inject packets under another client's identity, which is otherwise not possible in our hotspot-type networks.

5.4 Experiments

Similar to the mixed key attack, patched drivers are needed to perform the attack in practice. Windows and Linux are vulnerable with more than half of all tested network cards. The Android and iOS devices we tested were not vulnerable. Our three professional APs were not affected, but all our four home routers unfortunately were. See Section 6.1 for an overview of all tested devices.

6 EXPERIMENTS AND IMPLEMENTATION FLAWS

In this section, we elaborate on the experimental setup used to confirm the design flaws, and we present common implementation flaws related to aggregation and fragmentation.

6.1 Experimental setup

To confirm the design flaws in practice we tested smartphones, laptops, internet-of-things devices, home routers, and professional APs (see Table 1). We also tested Windows 10 and Linux 5.5 as clients using 16 wireless network cards on a Latitude 7490 and MSI GE60 (Table 2). Then we tested FreeBSD 12.1 and NetBSD 7.0 using several network cards (Table 3), and OpenBSD 6.4 using a small number of supported network cards (Section 6.7). In total this means we tested 75 devices, i. e., network card and OS combinations. In these experiments *all* devices were affected by one or more attacks. While performing experiments, we also analyzed the code of leaked and open source network stacks and found several implementation flaws related to aggregation and fragmentation.

Table 1: Devices tested using their default built-in wireless network card and operating system. The first three attacks are the design flaws discussed in Section 3, 4, and 5, respectively. The last four attacks correspond to implementation flaws discussed in Section 6.2, 6.3, 6.4, and 6.5, respectively.

	Attacks						
Device	A-MSDU	Mixed key	Cache att.	Non-con.	Plain. frag.	Bcast. frag.	Fake eapol
Huawei Y6 prime	●	●	○	●	ϟ	○	●
Nexus 5X	●	●	○	●	○	○	◒
Samsung i9305	●	●	○	●	○	◒	◒
iPhone XR	●	●	○	●	○	◒	○
iPad Pro 2	●	●	○	●	○	◒	○
MacBook Pro 2013	●	●	●	●	○	◒	○
MacBook Pro 2017	●	●	●	●	○	◒	○
Dell Latitude 7490	●	●	○	○	◐	○	○
MSI GE60	●	●	○	○	◐	○	○
Kankun smart plug	●	●	●	○	○	○	○
Xiaomi Mi Camera	●	●	●	●	ϟ	●	●
NanoPi R1	●	●	●	●	○	○	●
Canon PRO-100S	●	●	●	●	ϟ	○	○
Asus RT-N10	●	●	●	●	○	○	◒
Linksys WAG320N	●	●	●	●	○	○	◒
Asus RT-AC51U	●	●	●	●	ϟ	○	○
D-Link DIR-853	●	●	●	●	ϟ	○	●
Aruba AP-305 / 7008	●	○	○	●	○	○	○
LANCOM LN-1700	●	●	○	●	○	○	●
Cisco Catalyst 9130	●	○	○	●	○	○	○

○ Not affected	◒ Vulnerable during handshake
● Vulnerable	🐞 Resulted in crash
◐ (◑) Only first (or last) fragment must be encrypted	
◎ Accepts all fragmented frames	ϟ Accepts plaintext

Table 2: Test results against Windows (W) and Linux (L) using various network cards. The AWUS051NH and Ralink Wi-Pi did not support fragmentation on Linux. The TFWM was not supported by Windows. See Table 1 for the legend.

	Attacks													
	A-MSDU		*Mixed key*		*Cache att.*		*Non-con.*		*Plain. frag.*		*Bcast. frag.*		*Fake eapol*	
Network card	W	L	W	L	W	L	W	L	W	L	W	L	W	L
Intel 3160	●	●	○	●	○	●	○	○	◐	◐	○	○	○	○
Intel 8265	●	●	○	●	○	●	○	○	◐	◐	○	○	○	○
Intel AX200	●	●	●	●	○	○	●	●	◐	◐	○	○	○	○
AWUS036H	○	●	●	●	●	●	●	○	ϟ	◐	○	○	○	○
AWUS036NHA	●	●	○	●	○	●	●	○	◐	◐	○	○	○	○
AWUS036ACH	●	●	●	●	●	○	●	●	◎	ϟ	○	○	🐞	●
AWUS036ACM	●	●	●	●	●	●	●	○	◎	◐	●	○	●	○
AWUS051NH v2	●	●	●	○	●	○	●	○	◎	○	●	○	●	○
ZyXel NWD6505	●	●	●	●	●	●	●	○	◎	◐	●	○	●	○
TL-WN725N v1	●	●	●	●	●	●	●	○	◎	ϟ	○	○	○	○
WNDA3200	●	●	○	●	○	●	●	○	◐	◐	○	○	○	○
WN111v2	●	●	●	●	○	●	●	○	○	◐	○	○	○	○
Ralink Wi-Pi	●	●	●	○	●	○	●	○	◎	○	●	○	●	○
Sitecom WL-172	○	●	●	●	●	●	●	○	◎	◐	○	○	○	○
ZyXel M-202	●	●	○	●	○	●	●	○	◐	◐	○	○	○	○
TWFM-B003D	–	●	–	●	–	○	–	●	–	○	–	◒	–	◒

Table 3: Test results against FreeBSD (F) and NetBSD (N). Network cards at the top were tested in client mode, and the ones at the bottom in AP mode. The AWUS0351NH is not supported by NetBSD, and the TL-WN722N not by FreeBSD. See Table 1 on page 14 for the legend.

	Attacks													
	A-MSDU		*Mixed key*		*Cache att.*		*Non-con.*		*Plain. frag.*		*Bcast. frag.*		*Fake eapol*	
Network card	F	N	F	N	F	N	F	N	F	N	F	N	F	N
Intel 3160	●	○	●	○	○	○	●	●	◑	◑	○	○	●	○
Sitecom WL-172	●	○	●	○	○	○	●	●	●	◑	○	○	●	○
AWUS036H	●	○	●	○	○	○	●	●	◑	◑	○	○	●	○
AWUS051NH v2	●	–	●	–	○	–	●	–	◐	–	○	–	●	–
TL-WN725N v1	●	○	●	○	○	○	●	●	●	◑	○	○	●	○
Belkin F5D8053	●	○	●	○	○	○	●	●	◐	◑	○	○	●	○
TL-WN722N	–	○	–	○	–	○	–	●	–	◑	–	○	–	○
Sitecom WL-172	●	○	●	○	●	○	●	●	◎	ϟ	●	●	●	○
TL-WN725N v1	●	○	●	○	●	○	●	●	◎	◑	●	●	●	○
Belkin F5D8053	●	○	●	○	●	○	●	●	●	ϟ	●	●	●	○
TL-WN722N	–	○	–	○	–	○	–	●	–	ϟ	–	○	–	○

6.2 Non-consecutive packet numbers (CVE-2020-26146)

A common implementation flaw is that devices do not check whether all fragments of a frame have consecutive packet numbers, i. e., whether the received fragments indeed belong to the same frame. In our tests, all devices were affected except Windows 10 when using an Intel 3160 or 8265 card, and Linux when the kernel itself reassembles fragments (this is generally the case with SoftMAC 802.11 drivers). This means out of 68 tested devices that support fragmentation, 52 were vulnerable. See the "Non-con" column in Table 1, 2, and 3 for an overview of affected devices. Similar to the mixed key attack of Section 4, an adversary can abuse this vulnerability by mixing fragments of different packets in order to exfiltrate user data.

6.3 Mixed plaintext and encrypted fragments

Another implementation flaw is that devices reassemble mixed encrypted and plaintext fragments, instead of only accepting encrypted ones (CVE-2020-26147). This allows an attacker to replace certain encrypted fragments with plaintext ones. In our tests, 21 devices only require that the first fragment is encrypted (icon ◐), 9 that the last fragment is encrypted (icon ◑), and 3 that only one fragment is encrypted (icon ●). Moreover, 11 devices even accept plaintext frames (CVE-2020-26140), and another 9 accept fragmented but not unfragmented plaintext frames (CVE-2020-26143). We represent these last two implementation vulnerabilities using the icons ⚡ and ◎, respectively. All combined, 53 out of 68 devices that support fragmentation are affected by at least one of these implementation vulnerabilities (see column "Plain. frag" in Table 1, 2, and 3).

If the first fragment can be a plaintext one, an attacker can include a malicious packet in this fragment, which will be processed by the victim once it received all fragments. This is similar to the cache attack of Section 5.3.

In case the first fragment must be encrypted, we can combine this vulnerability with either the A-MSDU or fragment cache attack to inject arbitrary frames. When combined with the cache attack, the attacker first poisons the fragment cache of an AP or client with an encrypted fragment containing (part of) the packet to be injected. After the victim connects to the target network, the adversary injects the second fragment as plaintext, and the victim will reassemble the frame and process the injected packet.

6.4 Broadcast plaintext fragments (CVE-2020-26145)

Although broadcast frames should never be fragmented, several devices process broadcasted fragments as normal unfragmented frames. Moreover, some devices accept second (or subsequent) broadcast fragments even when sent unencrypted in a protected Wi-Fi network. An attacker can abuse this to inject packets by encapsulating them in a second fragmented plaintext broadcast frame, i. e., in a Frag_1 frame with a broadcast receiver address. Even unicast network packets, such as IPv4 or ARP packets, can be encapsulated in broadcast 802.11 frames and hence be injected in this manner.

Affected devices are listed under the column "Bcast. frag." in Table 1, 2, and 3. Notable affected devices are those of Apple and APs on NetBSD and FreeBSD. Some devices are only vulnerable during the execution of the 4-way handshake, but this does not limit attacks: a victim can be forcibly disconnected, e. g., deauthenticated or jammed, such that the victim will reconnect and execute a new 4-way handshake.

6.5 Cloaking A-MSDUs as handshake frames (CVE-2020-26144)

Devices accept plaintext 4-way handshake frames, i. e., plaintext data frames with an EAPOL LLC/SNAP header, when connecting to a network. If implemented wrongly, this can be abused to inject plaintext A-MSDUs. In particular, an adversary can construct a

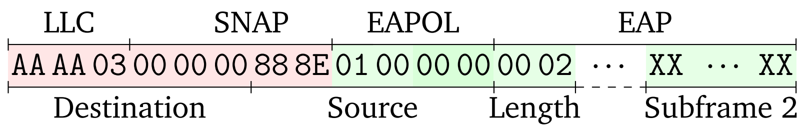

Figure 7: A-MSDU payload (bottom) whose first 8 bytes are also an EAPOL LLC/SNAP header (top). Red bytes must have the given value, and green ones can have any value.

plaintext A-MSDU whose first 8 bytes can also be interpreted as a valid EAPOL LLC/SNAP header (see Figure 7). Although this causes the destination and source address of the first subframe to be invalid, meaning the receiver drops this subframe, other subframes are still processed. Hence, an attacker can inject arbitrary packets against devices that accept plaintext A-MSDUs whose first 8 bytes equal an EAPOL LLC/SNAP header. Against FreeBSD and several devices shown in Table 1 and 2, this allows an adversary to inject plaintext A-MSDU frames.

6.6 EAPOL forwarding & fragmentation (CVE-2020-26139)

As highlighted in the previous section, devices must accept plaintext 4-way handshake frames when a client is connecting to a network. We found that some devices also *forward* plaintext handshake frames if they are destined to other clients in the network, even when the sender has not yet authenticated. Affected devices are FreeBSD and NetBSD APs, and certain home routers such as our Asus RT-N10 and Linksys WAG320N. An adversary can abuse this to perform the A-MSDU attack from Section 3 using only a multi-channel MitM position. In particular, the adversary first associates with the target network. Then, instead of starting the 4-way handshake, the adversary will send a handshake, i. e., EAPOL, frame to the AP with as final destination a client that is connected to the network. A vulnerable AP accepts and forwards this EAPOL frame to its destination, in this case the targeted client. Moreover, the AP will encrypt this frame towards the client. The adversary can then uses its MitM position to set the A-MSDU flag in the encrypted EAPOL frame. An adversary can then inject arbitrary packets by constructing an EAPOL frame as illustrated in Figure 7 and placing the packet to be injected in the second A-MSDU subframe.

6.7 Treating fragments as full frames

Certain implementations, such as OpenBSD and the ESP-12F, do not support fragmented frames. However, they are still vulnerable to attacks because they treat all frames as non-fragmented ones (CVE-2020-26142). An adversary can abuse this to inject arbitrary network packets by controlling the content that is included in one of the fragments.

7 RELATED WORK & DISCUSSION

In this section we cover related work, give an overview of all our countermeasures, discuss results, and explore future work.

7.1 Related work

Aggregation Robyns et al. presented packet-in-packet attacks that exploit aggregated MPDUs where (encrypted) frames are aggregated close to the physical layer [30]. In this aggregation method, encryption happens before aggregation, and their attacks enabled the remote injection of frames in open (but not protected) Wi-Fi networks. Similarly, other packet-in-packet attacks against different protocols are also only feasible in open networks [14, 9]. We study aggregation at a higher network layer, where encryption takes places after aggregation. Our resulting attacks apply to protected Wi-Fi networks and allow an adversary, that is within radio range of victims, to inject packets. In other work, A-MSDUs were abused to more easily trigger key reinstallations [38], but no attention was paid to the unauthenticated A-MSDU flag.

Fragmentation Previous work abused fragmentation to more efficiently exploit known flaws in WEP [7], but did not uncover flaws in (de)fragmentation features itself. Schepers et al. found that OpenBSD incorrectly handled fragmented TKIP frames [31], allowing Denial-of-Service (DoS) attacks and packet injection, but this was an implementation vulnerability and not a design flaw in the standard.

Implementation flaws in IPv4 and IPv6 (de)fragmentation have been abused for DoS attacks, firewall evasion, etc [24, 4]. It was also abused to launch off-path DNS cache poisoning attacks by bypassing its plaintext challenge-response protocol [17]. This was possible because the first fragment of a response contains the unpredictable challenge values, and an adversary can replace the second fragment with malicious data. In contrast, our attacks work against encrypted protocols. Nowadays, IP fragmentation is considered fragile [8]. Against 6LoWPAN, fragmentation was abused to launch a DoS attack by preventing (correct) packet reassembly [18].

Formal models Cremers et al. formally modeled WPA2 and demonstrated the correctness of key reinstallation defenses. Their model did not include aggregation and fragmentation functionality, and therefore missed the attacks that we discovered [12]. Other work on formally verifying and modeling WPA2 only focuses on the 4-way handshake [16, 32].

Wi-Fi security Major advancements have been made to the security of Wi-Fi. This includes the discovery and prevention of key reinstallations in WPA2 [37, 38], the release of WPA3 [41], and extra defenses such as operating channel validation and beacon protection [36, 35]. Although shortcomings in WPA3 were identified [27, 40], these have been addressed in an update to the standard [15]. Finally, a recent update to WPA3 improves the security of enterprise networks, as these were often insecurely configured [10, 5].

Other work studied Wi-Fi provisioning schemes [26], inferred and analyzed state machines [33], and studied potential electromagnetic side-channel leaks in 802.11 radios [11].

7.2 Countermeasures for the design flaws

Spoofing aggregated frames The aggregation attack of Section 3 can be prevented by updating the standard to assure the A-MSDU flag is always authenticated, i. e., assuring only SPP A-MSDUs are used. This can be accomplished by setting and adhering to the "SPP A-MSDU required" flag in the RSN element when connecting to another station or network. In theory, this assures all stations either: (1) never accept/send A-MSDUs; or (2) always authenticate the A-MSDU flag in sent and received frames [19, Table 11-12].

If dropping non-SPP A-MSDUs is not feasible, attacks can be mitigated by dropping the full A-MSDU frame if *any* of the subframe's MAC addresses do not belong to connected stations. In particular, A-MSDUs must be dropped if their first 6 bytes equal the start of an LLC/SNAP header, i. e., if the destination address of the first subframe is `AA:AA:03:00:00:00`. Although this prevents our main attack, other novel aggregation-based attacks may remain possible.

Mixed key attack Mixed key attacks of Section 4 can be prevented by not reassembling fragments that were decrypted using different keys, which is backwards-compatible because this does not occur in normal circumstances. The standard and all implementations should be updated to include this check. An efficient way to implement this is to assign an incremental key identifier to decrypted fragments, increase this identifier whenever a new key is installed, and verifying that all fragments were decrypted using the same key identifier.

Cache attack The fragment cache attack of Section 5 can be prevented by updating clients to clear the fragment cache whenever (re)connecting or (re)associating with a network. Similarly, an AP should clear all fragments received by a specific client when this client reconnects, reassociates, or disconnects from the network. These changes are backwards-compatible since legitimate devices do not rely on this vulnerable behavior. The 802.11 standard and all existing implementations should be updated to perform these actions.

8 CONCLUSION

We discovered widespread design and implementation flaws related to frame aggregation and fragmentation. Interestingly, our aggregation attack could have been avoided if devices had implemented optional security improvements earlier. This highlights the importance of deploying security improvements before practical attacks are known. The two fragmentation-based design flaws were, at a high level, caused by not adequately separating different security contexts. From this we learn that properly separating security contexts is an important principle to take into account when designing protocols.

In practice, our implementation-specific vulnerabilities are the most devastating. Several enable the trivial injection of frames, which we abused to trick a victim into using a malicious DNS server to then intercept most of the victim's traffic.

ACKNOWLEDGMENTS

We thank LANCOM, Aruba, and Cisco for their test devices, and thank Cisco for help with the disclosure. This work was supported by the Center for Cyber Security at New York University Abu Dhabi (NYUAD). The author holds a Postdoctoral fellowship from the Research Foundation Flanders (FWO).

REFERENCES

[1] `https://github.com/vanhoefm/fragattacks`

[2] Md Sohail Ahmad. Wpa too! In *DEF CON*, 2010.

[3] Wi-Fi Alliance. *Hotspot 2.0 Specification Ver. 3.1*, 2019.

[4] Antonios Atlasis. Attacking IPv6 implementation using fragmentation. In *Black Hat EU Briefings*, 2012.

[5] Alberto Bartoli, Eric Medvet, Andrea De Lorenzo, and Fabiano Tarlao. (in)secure configuration practices of WPA2 enterprise supplicants. In *WiSec*, 2018.

[6] Johannes Berg. mac80211: check PN correctly for GCMP-encrypted fragmented MPDUs. Linux commit `9acc54beb474`, 2016.

[7] Andrea Bittau, Mark Handley, and Joshua Lackey. The final nail in WEP's coffin. In *IEEE S&P*, 2006.

[8] Ron Bonica, Fred Baker, Geoff Huston, Bob Hinden, Ole Trøan, and Fernando Gont. IP fragmentation considered fragile. RFC 8900, 2020.

[9] Sergey Bratus, Travis Goodspeed, Ange Albertini, and Debanjum S Solanky. Fillory of PHY: Toward a periodic table of signal corruption exploits and polyglots in digital radio. In *USENIX WOOT*, 2016.

[10] Sebastian Brenza, Andre Pawlowski, and Christina Pöpper. A practical investigation of identity theft vulnerabilities in eduroam. In *WiSec*, 2015.

[11] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In *CSS*, 2018.

[12] Cas Cremers, Benjamin Kiesl, and Niklas Medinger. A formal analysis of IEEE 802.11's WPA2: Countering the kracks caused by cracking the counters. In *USENIX Security*, 2020.

[13] Julien Freudiger. How talkative is your mobile device? an experimental study of Wi-Fi probe requests. In *WiSec*, 2015.

[14] Travis Goodspeed, Sergey Bratus, Ricky Melgares, Rebecca Shapiro, and Ryan Speers. Packets in packets: Orson welles' in-band signaling attacks for modern radios. In *USENIX WOOT*, 2011.

[15] Dan Harkins, Jouni Malinen, and Mike Montemurro. Finding PWE in constant time. Retrieved 14 June 2020 from `https://mentor.ieee.org/802.11/dcn/19/11-19-1173-18-000m-pwe-in-constant-time.docx`, 2019.

[16] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *CCS*, 2005.

[17] Amir Herzberg and Haya Shulman. Fragmentation considered poisonous, or: one-domain-to-rule-them-all.org. In *IEEE CNS*, 2013.

[18] René Hummen, Jens Hiller, Hanno Wirtz, Martin Henze, Hossein Shafagh, and Klaus Wehrle. 6LoWPAN fragmentation attacks and mitigation mechanisms. In *WiSec*, 2013.

[19] IEEE Std 802.11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Spec*, 2016.

[20] IEEE Std 802.11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Spec*, 2020.

[21] IEEE Std 802.11ac. *Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz*, 2013.

[22] IEEE Std 802.11n. *Amendment 5: Enhancements for Higher Throughput*, 2009.

[23] Jaehoon Paul Jeong, Soohong Daniel Park, Luc Beloeil, and Syam Madanapalli. IPv6 Router Advertisement Options for DNS Configuration. RFC 8106, 2017.

[24] Malachi Kenney. Ping of death. Retrieved 14 June 2020 from `https://insecure.org/sploits/ping-o-death.html`, 1996.

[25] Amit Klein and Benny Pinkas. From IP ID to device ID and KASLR bypass. In *USENIX Security*, 2019.

[26] Changyu Li, Quanpu Cai, Juanru Li, Hui Liu, Yuanyuan Zhang, Dawu Gu, and Yu Yu. Passwords in the air: Harvesting Wi-Fi credentials from SmartCfg provisioning. In *WiSec*, 2018.

[27] Karim Lounis and Mohammad Zulkernine. Bad-token: denial of service attacks on WPA3. In *SIN*, 2019.

[28] Jouni Malinen and Mark Rison. GCMP decapsulation. Retrieved 18 May 2020 from `https://mentor.ieee.org/802.11/dcn/15/11-15-1132-02-000m-gcmp-decapsulation.docx`, 2015.

[29] J. Postel and J. Reynolds. Standard for the transmission of IP datagrams over IEEE 802 networks. RFC 1042, 1988.

[30] Pieter Robyns, Peter Quax, and Wim Lamotte. Injection attacks on 802.11n MAC frame aggregation. In *WiSec*, 2015.

[31] Domien Schepers, Aanjhan Ranganathan, and Mathy Vanhoef. Practical side-channel attacks against WPA-TKIP. In *ASIA CCS*, 2019.

[32] Rajiv Ranjan Singh, José Moreira, Tom Chothia, and Mark Ryan. Modelling of 802.11 4-way handshake attacks and analysis of security properties. In *STM*, 2020.

[33] Christopher McMahon Stone, Tom Chothia, and Joeri de Ruiter. Extending automated protocol state learning for the 802.11 4-way handshake. In *ESORICS*, 2018.

[34] Mathy Vanhoef. Fragment and forge: Breaking Wi-Fi through frame aggregation and fragmentation. In *Proceedings of the 30th USENIX Security Symposium*. USENIX Association, August 2021.

[35] Mathy Vanhoef, Prasant Adhikari, and Christina Pöpper. Protecting Wi-Fi beacons from outsider forgeries. In *WiSec*, 2020.

[36] Mathy Vanhoef, Nehru Bhandaru, Thomas Derham, Ido Ouzieli, and Frank Piessens. Operating channel validation: Preventing multi-channel man-in-the-middle attacks against protected Wi-Fi networks. In *WiSec*, 2018.

[37] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In *CCS*, 2017.

[38] Mathy Vanhoef and Frank Piessens. Release the kraken: new KRACKs in the 802.11 standard. In *CCS*, 2018.

[39] Mathy Vanhoef and Eyal Ronen. Dragonblood: Attacking the Dragonfly handshake of WPA3. In *Black Hat US Briefings*, 2019.

[40] Mathy Vanhoef and Eyal Ronen. Dragonblood: Analyzing the Dragonfly handshake of WPA3 and EAP-pwd. In *IEEE S&P*, 2020.

[41] Wi-Fi Alliance. WPA3 specification version 2.0. Retrieved 24 May 2020 from `https://www.wi-fi.org/file/wpa3-specification`, 2019.