Fragment and Forge: Breaking Wi-Fi Through Frame Aggregation and Fragmentation

Mathy Vanhoef, *New York University Abu Dhabi*

https://www.usenix.org/conference/usenixsecurity21/presentation/vanhoef

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Fragment and Forge: Breaking Wi-Fi Through Frame Aggregation and Fragmentation

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

Abstract

In this paper, we present three design flaws in 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 in various ways, 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, ranging from WEP all the way to WPA3, 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. Finally, we present a tool to test whether devices are affected by any of the vulnerabilities, and we discuss countermeasures to prevent our attacks.

1 Introduction

In the last few years, major improvements have been made to the security of Wi-Fi. Most notably this includes the discovery and prevention of key reinstallation in WPA2 [18, 57, 58], and the standardization of WPA3 which, among other things, prevents offline dictionary attacks [60]. Additionally, extra defenses have been standardized, such as operating channel validation and beacon protection, which further increases the security of Wi-Fi networks [54, 55]. These recent improvements are a welcome addition to Wi-Fi since it continues to be one of the main methods used to access the Internet. Additionally, Wi-Fi is used in home networks to prevent outsiders from accessing personal printers, security cameras, smart home devices, and so on. In enterprise networks, Wi-Fi plays an equally important role since it authenticates users, it protects access to internal services, and it secures content while being transmitted to, for instance, local file servers, smart presentation screens in meeting rooms, and so on.

Despite the recent advances in Wi-Fi security, we found design issues that went unnoticed for more than two decades. These issues were discovered by analyzing open source Wi-Fi stacks and systematically inspecting the 802.11 standard. Our results affect all protected Wi-Fi networks, including old 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 that we tested were 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 missing check 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. Most devices were affected by at least one of these attacks.

Apart from design flaws 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. The most common implementation vulnerability is that receivers do not check whether all fragments belong to the same frame, meaning an adversary can trivially forge frames by mixing the fragments of two different frames. Against certain implementations it is also possible to mix encrypted and plaintext fragments, to inject plaintext aggregated frames by masquerading them as handshake messages, and to inject plaintext fragmented (broadcast) frames. Several other implementation flaws have also been discovered, and we created a tool to test for all of them [1].

We believe that the discovered design flaws went unnoticed for so long for two main 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 or formal 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 are secure, while in reality they are vulnerable.

Because our findings affect all Wi-Fi devices, we contacted the Industry Consortium for Advancement of Security on the Internet (ICASI) to help with a multi-party coordinated disclosure. We are also collaborating with the Wi-Fi Alliance to distribute information to vendors.

To summarize, our main contributions are:

- We present a design flaw in 802.11's frame aggregation functionality that can be abused to inject arbitrary frames and demonstrate resulting attacks in practice (Section 3).
- We present a design flaw in 802.11's frame fragmentation feature where a receiver accepts fragments encrypted under different keys. We show how this can be abused to forge frames and exfiltrate data (Section 4).
- We present another design flaw where we poison the fragment cache of a receiver and abuse this to inject packets and exfiltrate data (Section 5).
- We discover widespread implementation flaws and created a tool to test for all vulnerabilities in this paper [1]. Our tool can test both clients and Access Points (APs) and covers more than 45 test cases (Section 6).

Finally, we discuss related work, all our countermeasures, and our results in Section 7, and we conclude in Section 8.

2 Background

This section introduces the 802.11 standard [31] and gives a high-level description of the design flaws that we discovered.

2.1 Frame layout and packet aggregation

Figure 1 shows the layout of an 802.11 frame and we start with explaining its general-purpose fields. 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 of the frame, which is called the Traffic Identifier (TID) in 802.11. The payload field of a normal frame contains the transported packet, which starts with an LLC/SNAP header—sometimes also called an rfc1042 header [44]—that defines the type of the packet, e. g., whether it is an IP or ARP packet (see Figure 2).

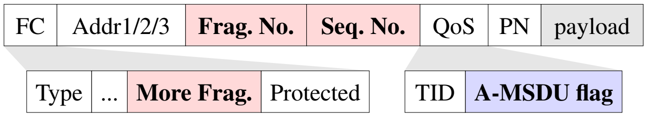

Figure 1: Layout of an encrypted 802.11 frame. Our aggregation attack abuses the field in blue, and our fragmentation attacks the fields in red. Only the payload field is encrypted.

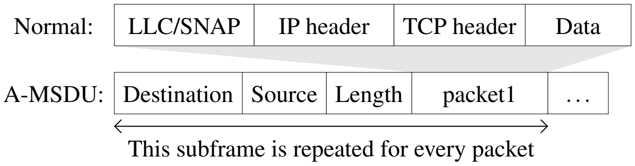

Figure 2: Contents of the payload field in a normal frame with an example TCP/IP header (top), and the contents of a frame with the A-MSDU flag set meaning it contains one or more aggregated packets (bottom).

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 [33], and we focus on Aggregate MAC Service Data Units (A-MSDUs), which all 802.11n-capable devices are required to support. 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. Note that the packet itself starts with an LLC/SNAP header, just like in a normal frame. Finally, each subframe except the last is padded so that its length is a multiple of 4.

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, so that if a fragment gets corrupted, only that fragment has to be retransmitted. 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). This means a frame can be split into at most 2^4 fragments. 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). Only unicast data frames are (de)fragmented, and such frames can be recognized by the type subfield in the frame control field and by the receiver MAC address (Addr1). In this paper, 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 2^{nd} fragment with sequence number 9.

By default, a frame is only split into fragments when it is larger than the configured fragmentation threshold. This fragmentation threshold is independent of the maximum packet size, i. e., the Maximum Transmission Unit (MTU). When a device supports dynamic fragmentation, which is part of 802.11ax, a transmitter can split a frame into fragments independent of the fragmentation threshold [30]. In particular, when a client is assigned a fixed-duration transmit opportunity, called a resource unit in 802.11ax, it can fill the last part of this duration with a fragmented frame.

2.3 Authentication and encryption

In both protected home and enterprise Wi-Fi networks, the client will eventually 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, frames larger than the fragmentation threshold 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 because its value is only known immediately before the station is able to transmit [39]. Note that encrypted frames can be recognized by the protected flag in the FC field. 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 [31, Fig. 5-1]. On reception of the last fragment, all decrypted fragments are reassembled into the original frame. Since the fragment number and more fragments flag are authenticated, an adversary cannot change the number of fragments or their relative position. Additionally, 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.

The older, but not deprecated [48], WPA-TKIP data-confidentiality protocol does not authenticate the fragment number and more fragments flag, and does not check that the PNs of the fragments are consecutive. Instead, the reassembled frame is authenticated using the Michael algorithm. When using the broken and deprecated WEP protocol, the fragment number and more fragments flag are not authenticated, and the reassembled frame is not separately authenticated. This results in a novel attack against WEP where an attacker can mix and reorder fragments of different frames (see Section 4.5).

2.4 Attack techniques and scenarios

Although exploiting each discovered design flaw requires a different threat model, which is described at the start of every section, there are similarities between most threat models. In particular, several attacks rely on a multi-channel machine-in-the-middle (MitM) position, some also rely on a relaxed BEAST threat model, and one attack targets hotspot-type networks. We therefore introduce these concepts first:

Multi-Channel MitM Many (known) attacks require the ability to block, modify, or delay encrypted frames sent between the client and AP. To reliably do this, Vanhoef and Piessens introduced the multi-channel MitM position [56]. In this MitM technique, 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. Recently a defense against this MitM has been ratified into the draft 802.11 standard, called operating channel validation [55], but it is not yet used in practice. As a result, the multi-channel MitM position can be reliably established in practice: the only requirement is that the adversary possesses two Wi-Fi antennas and is within radio range of the client and AP.

BEAST threat model The BEAST attack against TLS introduced a novel threat model where the victim is tricked into executing malicious JavaScript code [20]. This can for example be accomplished by social engineering the victim into visiting a website under control of the adversary, and enables the adversary to make the victim send a large amount of traffic. Other attacks against TLS also relied on this threat model [3,5,12,20,22,42,46,47], and we call it the BEAST threat model. In a relaxed version of this threat model, we only require that the victim connects to a server of the adversary without requiring the execution of malicious JavaScript code.

Hotspot security Hotspots used to be synonymous with open and insecure Wi-Fi networks. However, this is no longer the case. In modern hotspot-type networks such as eduroam, and Hotspot 2.0 networks where users can, e. g., authenticate using their mobile SIM card [6], 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 [6, §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

In this section, we present a design flaw in 802.11's frame aggregation functionality that allows an adversary to inject arbitrary packets by making a victim process normal Wi-Fi frames as aggregated ones. We abuse this to perform a port scan and to trick a victim into using a malicious DNS server. This design flaw has been assigned CVE-2020-24588.

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 [33]. 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 Section 6.3, 6.5, and 6.6 we abuse implementation flaws to perform the attack under alternative assumptions. This section focuses on a general attack technique, where an adversary can send such IPv4 packets to a client or AP as follows:

Attacking clients If the IP address of the client is known, and no firewall is blocking incoming packets, we can directly send IPv4 packets to the victim. Otherwise, we assume that the adversary is able to make the victim connect to a server under the adversary's control, allowing the adversary to inject IPv4 packets over this connection. A wide-scale method to accomplish this is to register a misspelled domain name [43], or to exploit third-party advertisements in popular websites [27]. A relaxed BEAST threat model can also be used, where the victim is social engineered into visiting the attacker's website.

Attacking APs To attack APs, the IP ID field of at least one connected client must be predictable. This can be the case for older clients [37], and on some devices this field always equals zero [53]. We also rely on the BEAST threat model to make this client send IPv4 packets with a given payload.

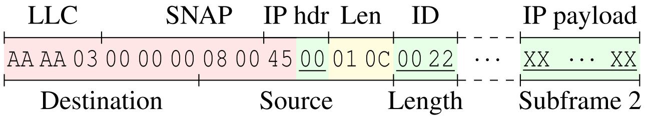

Figure 3: Parsing an 802.11 payload containing an IPv4 packet (top) as an A-MSDU frame (bottom). Green underlined bytes can be controlled by the adversary, yellow ones can be partially controlled, and red bytes have a fixed value.

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 [33, §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 [37], 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.

We can attack APs in a similar way if at least one client uses predictable IP IDs, which is the case for certain older Operating Systems (OSs) [37, 49]. By relying on the BEAST threat model, we make this client perform a POST request containing attacker-controlled binary data. This essentially causes the transmission of IP packets with a partially attacker-controlled payload. If the ID field of this IPv4 packet is correctly predicted, the second A-MSDU subframe will correspond to attacker-controlled data. As a result, the attacker can inject arbitrary packets. The AP will forward this packet to its next destination, which is the gateway or any client in the network. Finally, if a client always uses an IP ID value of zero, it is possible to use the injection technique in Appendix C.

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 performed the following two attacks against an IPv6 or IPv4 capable victim:

Portscan We performed a portscan against IPv4 and IPv6 hosts to demonstrate the injection of a large number of packets. Open ports were detected based on the length of the encrypted TCP SYN/ACK replies.

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 [34]. 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.

Against IPv4-only clients, a similar attack is possible if we can obtain the 4-byte transaction identifier that the client includes in its DHCP discover and requests. This identifier is normally unpredictable [19]. However, we found that iOS and macOS randomly generate an identifier on boot, but then increment it for each DHCP message. Similarly, Halvorsen et al. found that Mac OS X used predictable identifiers [24]. Moreover, certain IoT devices such as our Xiaomi security camera randomly generate a transaction identifier on boot and reuse this value in all DHCP messages. This means that if one transaction identifier can be leaked or brute-forced, it becomes possible to spoof DHCPv4 messages and force the client into using a malicious DNS server. Finally, our ESP-12F always uses the same identifier, even after reboots, meaning we can trivially make it use a malicious DNS server if we can inject packets towards it.

3.4 Applicability to short A-MSDUs

In Directional Multi-Gigabit (DMG) networks, defined by amendment 802.11ad, stations can also send short A-MSDUs where each subframe only consists of a length field and the transported data. Short A-MSDUs can only be encapsulated inside DMG frames because only these frames define the short A-MSDU flag in the QoS field [31, §9.2.4.5]. This flag is always authenticated in DMG networks [31, §12.5.3.3.3]. Since DMG frames should only be sent in DMG networks, the short A-MSDU flag is always authenticated, and hence cannot be manipulated by an attacker. Nevertheless, we recommend that the standard more explicitly requires that the short A-MSDU flag should only be used when it is authenticated.

An implementation risk is that the hardware supports and authenticates the short A-MSDU flag, but that the software-based network stack does not support short A-MSDUs. In that case, short A-MSDUs may be treated as normal A-MSDUs. Unfortunately, few devices currently support 802.11ad, meaning we were unable to check whether any devices were affected by such implementation-specific issues.

3.5 Spoofing A-MSDUs as normal frames

We can also trick a victim into processing A-MSDU frames as normal frames. This causes the destination MAC address of the first A-MSDU subframe to be processed as the start of an LLC/SNAP header. This means that the resulting LLC/SNAP header is only valid when the target has the (locally administered) MAC address `AA:AA:03:00:00:00`. Because of this limitation, it is unlikely that this can be abused in practice.

3.6 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.

We tested end-to-end attacks against several clients. During these tests, we used two TL-WN722N dongles for the multi-channel MitM, and we reliably obtained this MitM position by spoofing channel switch announcements [55]. We detected the injected IPv4 packet based on its length, set the A-MSDU flag before forwarding it to the victim, and successfully injected router advertisements to poison the victim's DNS server.

When testing the attack against FreeBSD and Linux 4.9 and above, we noticed that we were unable to inject packets as described in Section 3.2. Upon closer inspection we found that these operating systems strip away the first 8 bytes of an A-MSDU frame if these bytes look like a valid LLC/SNAP header, and then further process the frame. This behavior is not compliant with the 802.11 standard. When the first 8 bytes are stripped, the length field of the first A-MSDU subframe corresponds with the first two bytes of the source IP address. If the victim is not behind a firewall, we can spoof the source address of our IPv4 packets such that the injected packet will again be contained in the second A-MSDU subframe. If the victim blocks spoofed IP addresses, we can rent a server on Amazon AWS with an IP address in the subnet 3.5.0.0/16 [7]. The first A-MSDU subframe then has a length of 773 bytes, which leaves sufficient space to inject malicious packets.

3.7 Discussion

To prevent aggregation attacks, stations must either not use A-MSDUs, or always authenticate the A-MSDU flag, i. e., only use SPP A-MSDUs. We elaborate on this in Section 7.2.

We conjecture that turning normal frames into A-MSDUs can also be abused as an oracle to leak data. For instance, an AP may act differently depending on the values that are located at the A-MSDU header fields. We leave a more detailed analysis on abusing A-MSDUs to leak data as future work.

4 Mixed Key Attack against Fragmentation

In this section, we first discuss the shared root cause of the two fragmentation-based design flaws that we discovered. We then focus on the first design flaw, namely how the 802.11 standard allows an attacker to forge frames by mixing fragments that are encrypted under different keys. This design flaw has been assigned CVE-2020-24587. We show how to abuse this flaw to exfiltrate client data and, for instance, recover sensitive info sent over plaintext HTTP connections.

4.1 Fragmentation design flaws

At a high level, the discovered fragmentation flaws are caused by not properly separating different security contexts and their associated memory, receive queues, or fragment caches:

Mixed key attack A first problem is that the 802.11 standard does not require that each fragment was decrypted using the same key. Therefore, an attacker can forge frames by mixing fragments of frames that were encrypted under different keys, i. e., by mixing fragments belonging to different security contexts. This design flaw will be further discussed and abused in this section.

Fragment cache poisoning The 802.11 standard also does not state when decrypted fragments should be removed from memory, i. e., from the fragment cache. That is, decrypted fragments are not dropped when the security context changes due to a (re)connect or (re)association. An attacker can abuse this to inject fragments into a victim's fragment cache, and then combine this with legitimate fragments to inject packets or exfiltrate decrypted fragments (see Section 5).

4.2 Threat model

We first focus on the mixed key attack, which works against WEP, CCMP, and GCMP. The older TKIP protocol is only affected when the receiver forgets to verify the authenticity of reassembled frames (see Section 6.7).

The attack requires that one or more devices in the network send fragmented frames. Although not all devices do this by default, because their configured fragmentation threshold is equal to or bigger than the MTU, it is recommended to use fragmentation in noisy environments. Moreover, 802.11ax devices are expected to support dynamic fragmentation, making the usage of fragmentation more common in practice (recall Section 2.2). For instance, our Cisco Catalyst 9130 has dynamic fragmentation enabled by default, and Aruba APs also support it. With this in mind, our fragmentation-based attacks are especially relevant against new devices.

To perform the attack, 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. Although most networks by default do not periodically refresh the session key, we do remark that this assumption matches the requirement of certain key reinstallation attacks against WPA2 [57, §3.4]. To trick the victim into sending a packet to a server under our control, we can rely on the relaxed BEAST threat model. In Section 5 and 6.2, we exploit additional design and implementation flaws to perform mixed key attacks without these assumptions.

4.3 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

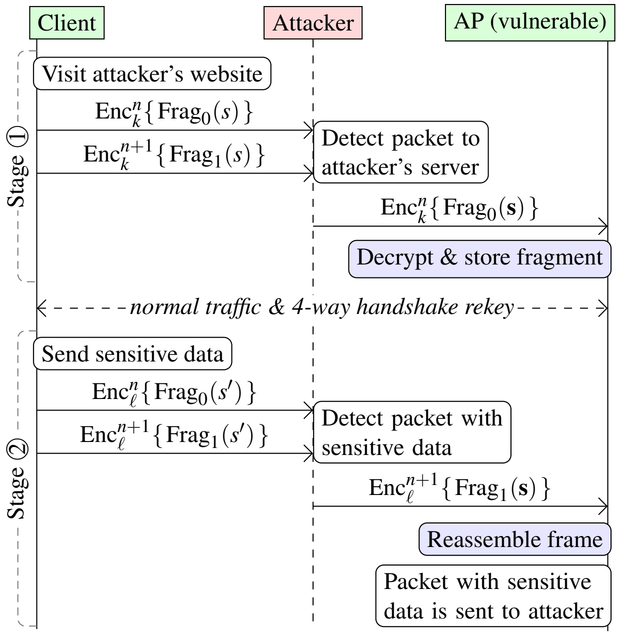
Client
Attacker
AP (vulnerable)
Stage ①
Visit attacker's website
$\mathrm{Enc}_k^n\{\mathrm{Frag}_0(s)\}$
$\mathrm{Enc}_k^{n+1}\{\mathrm{Frag}_1(s)\}$
Detect packet to attacker's server
$\mathrm{Enc}_k^n\{\mathrm{Frag}_0(\mathbf{s})\}$
Decrypt & store fragment
normal traffic & 4-way handshake rekey
Stage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

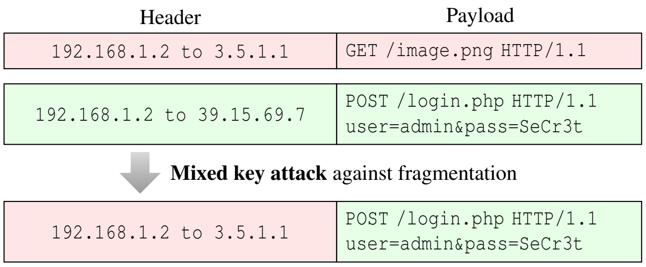

Figure 5: Mixing fragments of two packets to exfiltrate user data to an attacker-controlled server. The red packet is sent to the attacker's server, and the green packet contains user data.

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 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 [31, §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 $\mathrm{Enc}_\ell^{n+1}\{\mathrm{Frag}_1(s)\}$ 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 sensitive user data.

Against devices that only accept fragments with consecutive packet numbers, the second fragment must have packet number $n+1$. To increase the chance that this is the case, an adversary can rely on the BEAST threat model to make the client send background traffic using malicious JavaScript.

Packet construction In our attack, 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.

The data that can be exfiltrated depends on the configuration of the network and the victim. When a fragmentation threshold of 512 bytes is used, which is for example recommended by Arch Linux [8], data beyond this position is located in the second fragment meaning it can be exfiltrated. Therefore we can exfiltrate HTTP cookies, POST data, basic auth credentials, etc. Additionally, with the BEAST threat model, malicious JavaScript can let the client perform cross-origin requests with extra parameters in the URL that push the cookie towards the second fragment. This is similar to now-standard methods used in attacks against TLS [4, 20, 22, 42].

4.4 Attack variations

Vulnerable clients Against clients we can perform a similar attack to forge packets. However, the TCP or UDP checksum of the forged packet only has a 2^{-16} chance of being correct, meaning the packet will likely be dropped. Nevertheless, attacks remain possible against multimedia streaming protocols that run over UDP-Lite. This is because the checksum in UDP-Lite is only over a portion of a packet, so certain data can be changed without invalidating the checksum [35]. More general mixed key attacks against clients are possible when combined with fragment cache attacks (see Section 5).

Multiple key support The 802.11 standard contains an optional feature where the sender can pick between two keys to encrypt unicast frames [31, §12.6.21]. This is useful to facilitate the switchover to a new session key, and is made possible by including a key ID in a frame's header that identifies the used key. When this feature is used, two fragments that are encrypted under different keys can be forwarded to the target immediately after one another. This makes attacks easier in case the target removes fragments from memory after a certain timeout, or if any frames sent in-between the fragments interfere with their reassembly on the target.

4.5 Applicability to WEP and TKIP

When using WEP with dynamic rekeying, it is also possible to mix fragments that are encrypted under different keys. We empirically confirmed this against Linux, which shows that the discovered design flaw has been part of Wi-Fi since its release in 1997. Moreover, because WEP does not require that fragments have consecutive packet numbers, i. e., initialization vectors, an adversary can even mix fragments of two frames that are encrypted under the same key (see also Section 6.2).

The TKIP protocol is unaffected because, unlike CCMP and GCMP, its authenticity check covers the full frame instead of the individual fragments. However, some devices do not perform this authenticity check, meaning mixed key attacks can still be possible against TKIP in practice (see Section 6.7).

4.6 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 (see Appendix A).

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. This is, for example, the case against NetBSD, FreeBSD, and against a few network cards on Windows and Linux. We remark that this behavior is not compliant with the standard. Additionally, we found a method to still perform mixed key attacks against FreeBSD. This technique relies on non-trivial conditions and is further discussed in Appendix E. Finally, OpenBSD is not affected because it does not support fragmentation.

All four home routers that we tested were affected, though only one of our three professional APs was affected. Against our Aruba AP-305, any frame sent in-between fragments interferes with their reassembly. Our Cisco Catalyst 9130 did not support renewing the pairwise session key, meaning a default mixed key attack is not possible.

To keep experiments more reproducible, we performed end-to-end attacks against Linux using virtual Wi-Fi interfaces. In particular, we implemented and successfully performed an attack against a vulnerable AP to exfiltrate (decrypted) fragments sent by the client.

Against Linux, the attack is more tedious because it clears fragments from memory after two seconds. This can be overcome in the BEAST threat model, where malicious JavaScript on the client can trigger the transmission of an attacker-destined packet before the 4-way handshake completes. After this, traffic can be generated such that a packet with sensitive data is sent with a high enough packet number within two seconds, which can then be exfiltrated. We successfully tested this method against Linux 4.9.

4.7 Discussion

To prevent mixed key attacks, a receiver should verify that all fragments were encrypted under the same key. We elaborate on this in Section 7.2.

Our attacks assumed that the network periodically refreshes the pairwise session key. In our experience this is not done by default by most routers and APs. However, this does not limit attacks in practice because nearly all implementations accept non-consecutive packet numbers, meaning our attacks are possible without relying on rekeys (see Section 6.2).

A limitation of our attack is that it can only be used when one or more devices send fragmented frames. However, we conjecture that dynamic fragmentation can be abused to induce the transmission of fragmented frames. With dynamic fragmentation, a transmitter will send fragmented frames in order to fill allocated time slots (recall Section 2.2). To induce this type of fragmentation, we can use our MitM position to spoof the 802.11ax capabilities element of the client or AP, and advertise that they support dynamic fragmentation. An experimental analysis of this technique is left as future work.

5 Poisoning the Fragment Cache

In this section, we present a design flaw that enables an adversary to inject fragments into the memory, i. e., fragmentation cache, of victims. We show that this vulnerability allows an adversary to exfiltrate client data and inject arbitrary packets. This design flaws has been assigned CVE-2020-24586.

5.1 Threat model

Our attacks work against WEP, CCMP, and GCMP. The TKIP protocol is only affected if the authenticity of reassembled frames is not verified (see Section 6.7). Similar to the mixed key attack, a device in the network must be sending fragmented frames for the attack to be possible. In Section 6.3 we abuse implementation flaws to perform fragment cache attacks without this assumption. We also make the following assumptions depending on whether the target is a client or AP:

Vulnerable APs 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 [6]. 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).

Vulnerable clients We assume the client will connect to a protected Wi-Fi network of which the adversary also knows the password. The client does not trust this network, and will not send sensitive data when connected to this network. Such a network can be a coffee shop or conference network where the password is publicly shared. Note that in practice an adversary can listen to probe requests to obtain the networks that (old) devices are willing to connect to [21], and can use password sharing apps to obtain the password of nearby hotspots.[1]

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.

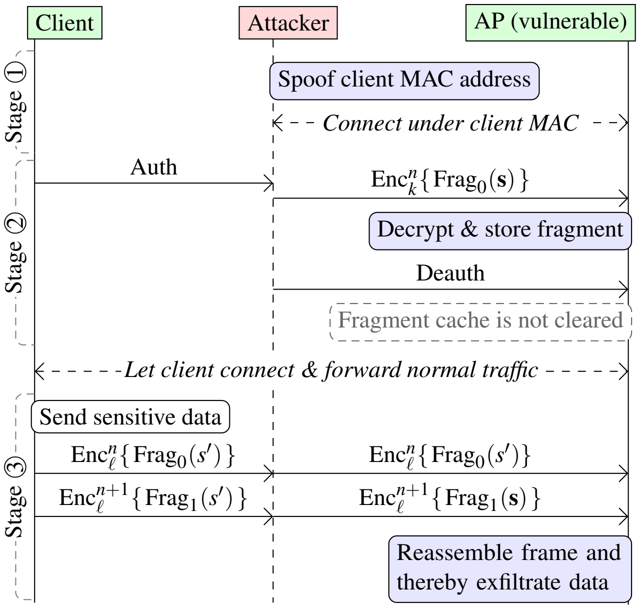
Client
Attacker
AP (vulnerable)
Stage ①
Spoof client MAC address
Connect under client MAC
Auth
$\text{Enc}_k^n\{\text{Frag}_0(\mathbf{s})\}$
Stage ②
Decrypt & store fragment
Deauth
Fragment cache is not cleared
Let client connect & forward normal traffic
Stage ③
Send sensitive data
$\text{Enc}_\ell^n\{\text{Frag}_0(s')\}$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

[1] Example apps are http://wifimap.io or http://instabridge.com

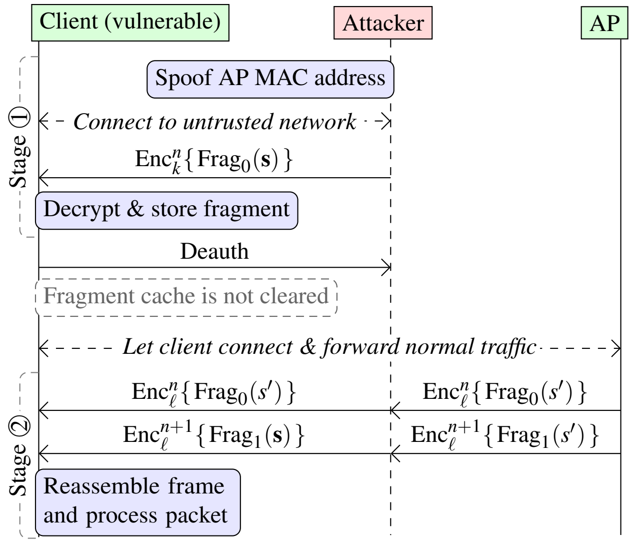
Client (vulnerable)
Attacker
AP
Stage ①
Spoof AP MAC address
Connect to untrusted network
$\text{Enc}_k^n\{\text{Frag}_0(\mathbf{s})\}$
Decrypt & store fragment
Deauth
Fragment cache is not cleared
Let client connect & forward normal traffic
Stage ②
$\text{Enc}_\ell^n\{\text{Frag}_0(s')\}$
$\text{Enc}_\ell^n\{\text{Frag}_0(s')\}$
$\text{Enc}_\ell^{n+1}\{\text{Frag}_1(\mathbf{s})\}$
$\text{Enc}_\ell^{n+1}\{\text{Frag}_1(s')\}$
Reassemble frame
and process packet

Figure 7: Fragment cache attack against a client with as goal to inject a packet. We abuse this to force the client into using our DNS server while being connected to a trusted network.

another client's identity, which is otherwise not possible in our hotspot-type networks.

To abuse fragment cache poisoning against a client, we rely on a novel threat model where the client will connect to an untrusted protected network, but will only send sensitive data when connected to a trusted network. For instance, a company laptop can be configured to only send sensitive data when connected to the company network, but the laptop is also used for casual internet surfing by, for example, connecting to a coffee shop network with a publicly shared password.

To attack a client, the adversary first spoofs the MAC address of the trusted (company) network but advertises the SSID of the untrusted (coffee shop) network (see stage ① in Figure 7). Once the client connected to this rogue network, the adversary injects fragment $\text{Frag}_0(s)$ into the victim's memory. This fragment contains the packet to be injected.

Between stages ① and ② of the attack, the client is disconnected from the untrusted network, after which it connects to the trusted (company) network. While the client is connecting, the adversary establishes a multi-channel MitM position between the client and AP. In this MitM position, the adversary forwards all frames between the client and AP, while avoiding to use sequence number s in frames towards the client. Note that the 802.11 standard does not require that the client clears its fragment cache when (re)connecting to an AP.

Stage ② of the attack starts when the AP of the trusted network sends a second fragment with packet number $n+1$. The adversary forwards this frame with sequence number s, such that the client will reassemble it with the injected fragment $\text{Frag}_0(s)$. Similar to the attack against the AP, the network layer of the client will discard the content in the second fragment as padding bytes, and will subsequently process the packet contained in Frag_0. In practice, an adversary can use this packet injection capability to trick the client into using a malicious DNS server (recall Section 3.3). This in turn enables the adversary to intercept data that the client only transmits while connected to the trusted (company) network.

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. Out of the tested BSD systems, only FreeBSD is vulnerable in AP mode and when using the injection technique of Appendix E. 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.

To keep experiments easier to reproduce, we again tested end-to-end attacks against Linux user virtual Wi-Fi interfaces. The target network used EAP-PWD, meaning users authenticate themselves using a username and password. In this setup we successfully attacked a vulnerable AP, poisoned its fragment cache with the start of an IP packet towards our server, and exfiltrated (decrypted) fragments sent by the victim.

We also successfully confirmed the attack against WEP on Linux, meaning this design flaw has been part of the 802.11 standard since its release in 1997.

Finally, we note that attacking Linux is non-trivial because it clears fragments from memory after two seconds. Nevertheless, attacks against Linux APs are practical, because there we can inject the malicious fragment right before a client is already attempting to connect to the AP, which assures delay between the injected and forwarded fragment is low.

5.5 Discussion

A backwards-compatible defense is to clear the fragment cache when (re)connecting or (re)associating with a station. We elaborate on this in Section 7.2.

Our cache poisoning attack is only possible if a device in the network uses fragmentation. Similar to our mixed key attack, we conjecture that our MitM position can pretend that the client and AP support dynamic fragmentation, and thereby induce the use of fragmentation against 802.11ax devices.

In practice, if a device is vulnerable to cache attacks, it is likely also vulnerable to mixed key attacks. This is not guaranteed though, because mixed key attacks can be prevented while cache attacks remain possible (and vice versa).

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 frame 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 6.8). 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. In general, whether a device is affected depends on the OS, network card, and whether it is acting as a client or AP. 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.

We created a tool that can test if clients or APs are affected by the discovered design and implementations flaws [1]. It can test home networks and enterprise networks where authentication is done using, e. g., PEAP-MSCHAPv2 or EAP-TLS. Our tool supports over 45 test cases, and over all devices combined we performed more than a thousand tests.

6.2 Non-consecutive packet numbers

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. This flaw has been assigned CVE-2020-26146.[2] 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. The details of this attack are illustrated in Figure 9 in the Appendix.

The vulnerability affects CCMP and GCMP. TKIP is only affected if the authenticity of reassembled frames is not verified (see Section 6.7). The WEP protocol is vulnerable by design, meaning this can be considered a fourth novel design flaw. Interestingly, when GCMP was introduced in 2013, its specification did not require that GCMP-encrypted fragments must have a consecutive PNs [32, §11.4.3]. The 802.11 group noticed this mistake in 2015 and updated the standard to require this check for GCMP as well [41]. Due to this temporary design flaw, Linux 4.0 to 4.4 was vulnerable when using GCMP even when the kernel reassembled fragments [11].

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	●	●	○	●	⚡	○	●
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	●	●	●	●	⚡	●	●
NanoPi R1	●	●	●	●	○	○	●
Canon PRO-100S	●	●	●	●	⚡	○	○
Asus RT-N10	●	●	●	●	○	○	◒
Linksys WAG320N	●	●	●	●	○	○	◒
Asus RT-AC51U	●	●	●	●	⚡	○	○
D-Link DIR-853	●	●	●	●	⚡	○	●
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	⚡ Accepts plaintext

6.3 Mixed plaintext and encrypted fragments

Another common implementation flaw we countered 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

[2] For each implementation flaw we list a reference CVE identifier, however, vendors may use different CVEs because an implementation flaw normally receives a unique CVE for each affected codebase. For further details see [1].

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	–	●	–	●	–	○	–	●	–	○	–	◒	–	◒

least one of these implementation vulnerabilities (see column "Plain. frag" in Table 1, 2, and 3).

The defragmentation code in Linux tries to enforce that all fragments are encrypted by checking whether they have consecutive PNs. Unfortunately, this check is implemented insecurely: after decrypting a frame, its PN is stored a session variable, and the PN of the previous fragment is compared to this session variable. As a result, when a (second) plaintext fragment is received, it checks whether the PN in this session variable is consecutive to the previous fragment, and does not realize this PN is unrelated to the received plaintext fragment. This means the PN check can be bypassed by first forwarding a valid encrypted fragment towards Linux using a consecutive PN but under a different sequence number, and then injecting a plaintext fragment under the correct sequence number (see Figure 10 in the appendix for details).

Practical impact 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 A-MSDU attack, an attacker uses its multi-channel MitM position to set the A-MSDU flag of an encrypted first fragment. After this, the attacker injects a plaintext fragment, upon which the victim reassembles both fragments and processes the resulting A-MSDU. The idea is now that the second subframe will correspond to the payload of the plaintext fragment and contains a packet that the attacker wants to inject. An obstacle is that the first encrypted fragment, which the adversary cannot control, must result in a small first subframe of predictable length, such that the second subframe is contained in the injected (second) plaintext fragment. This can be assured by predicting the IP ID of packets, similar to the A-MSDU attack against clients in Section 3.2. A second limitation is that not all devices support fragmented A-MSDUs. In particular, out of 56 devices that supported A-MSDUs, 33 properly handled fragmented A-MSDUs, 9 received them as malformed frames, and the other 14 silently discarded them.

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 11 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 F5D053	●	○	●	○	○	○	●	●	◐	◑	○	○	●	○
TL-WN722N	–	○	–	○	–	○	–	●	–	◑	–	○	–	○
Sitecom WL-172	●	○	●	○	●	○	●	●	◎	ϟ	●	●	●	○
TL-WN725N v1	●	○	●	○	●	○	●	●	◎	◑	●	●	●	○
Belkin F5D053	●	○	●	○	●	○	●	●	●	ϟ	●	●	●	○
TL-WN722N	–	○	–	○	–	○	–	●	–	ϟ	–	○	–	○

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. An advantage of this combination compared to a default cache attack is that it can be performed even when no devices in the network send fragmented frames.

Applicability to WEP and TKIP We also tested WEP on Linux and found that an adversary could trivially set the more fragments flag, since it is not authenticated, and subsequently combine this first encrypted fragment with plaintext fragments. The TKIP protocol is only affected if the authenticity of reassembled frames is not verified (see Section 6.7).

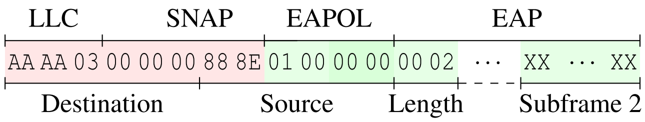

Figure 8: An A-MSDU payload (bottom) whose first 8 bytes are also a valid EAPOL LLC/SNAP header (top). Red bytes must have the given value, and green ones can have any value.

6.4 Broadcast plaintext fragments

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 (CVE-2020-26145). 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

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 (CVE-2020-26144). In particular, an adversary can construct a plaintext A-MSDU whose first 8 bytes can also be interpreted as a valid EAPOL LLC/SNAP header (see Figure 8). 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. Similar to Section 6.4, some devices are only vulnerable during the execution of the 4-way handshake. Against an AWUS036ACH on Windows 10, an A-MSDU starting with a valid EAPOL header resulted in a blue screen of death. Finally, some implementations strip away the first 8 bytes of an A-MSDU if these bytes equal a valid LLC/SNAP header (recall Section 3.6). This is not compliant with the standard and does not prevent attacks against vulnerable devices.

6.6 EAPOL forwarding & fragmentation

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 (CVE-2020-26139). 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 8 and placing the packet to be injected in the second A-MSDU subframe.

Against a NetBSD AP, an adversary can also send a large EAPOL frame, after which the AP will fragment, encrypt, and forward it to the targeted client. In other words, the adversary can abuse a NetBSD AP to generate encrypted fragments. This can be combined with the cache attack by first poisoning the fragment cache of the victim, and then generating a second encrypted fragment through the NetBSD AP that causes the victim to reassemble the fragments and process the injected packet. Another option is to abuse this against clients that accept fragmented frames as long as the last fragment is encrypted. Unfortunately, most NetBSD drivers do not support sending fragmented frames, and only send the first fragment and drop subsequent ones. Nevertheless, drivers such as `ath` do transmit all fragments, which an adversary can exploit to more easily perform fragmentation-based attacks.

6.7 Skipping the TKIP authenticity check

Fragmentation attacks should be impossible against TKIP because it verifies the authenticity of the full (reassembled) frame. However, we found several network cards on Linux and Windows that do not verify the authenticity of reassembled TKIP frames (CVE-2020-26141). On Windows, the AWUS036H, AWUS036ACH, and TL-WN725N are affected, and on Linux the NWD6505 and AWUS036ACM are affected. Against these devices our fragmentation-based attacks are possible even if the old TKIP protocol is used.

6.8 Treating fragments as full frames

Certain implementations, such as OpenBSD and the ESP-12F, do not support A-MSDUs or 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. This can be accomplished in the relaxed BEAST threat model by making the client load an attacker-controlled URL or resource, such that the resulting request or response is fragmented at the Wi-Fi layer, and one of the fragments purely consists of attacker-controlled data (which is then treated as a full frame).

In the case of OpenBSD, the more fragments flag is not included in the associated metadata when decrypting a fragment, causing decryption to fail on all but the last fragment. The last fragment does not have this flag set, meaning it is successfully decrypted and will be processed as a full frame. OpenBSD can also offload decryption to the Wi-Fi chip. In that case, all fragments are properly decrypted, but OpenBSD treats each decrypted fragment as an unfragmented frame. In both cases it is possible to inject arbitrary network packets by controlling the content that is included in the last fragment.

We confirmed the resulting attack(s) against the ESP-12F, which even accepted plaintext frames, and against OpenBSD 6.6 when it acted as a client using a Belkin F5D8053 v3 or Intel 8265. We conjecture that other devices, which also do not support fragmentation, can be attacked in similar ways.

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 [45]. 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 [15,23]. 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 [58], but no attention was paid to the unauthenticated A-MSDU flag.

Fragmentation Previous work abused fragmentation to more efficiently exploit known flaws in WEP [13], but did not uncover flaws in (de)fragmentation features itself. Schepers et al. found that OpenBSD incorrectly handled fragmented TKIP frames [50], 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 [9,36]. It was also abused to launch off-path DNS cache poisoning attacks by bypassing its plaintext challenge-response protocol [28]. 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 [14].

Against 6LoWPAN, fragmentation was abused to launch a DoS attack by preventing (correct) packet reassembly [29].

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 [18]. Other work on formally verifying and modeling WPA2 only focuses on the 4-way handshake [26,51].

Wi-Fi security Lately major advancements have been made to the security of Wi-Fi. This includes the discovery and prevention of key reinstallations in WPA2 [57,58], the release of WPA3 [60], and extra defenses such as operating channel validation and beacon protection [54,55]. Although shortcomings in WPA3 were identified [40,59], these have been addressed in an update to the standard [25]. Finally, a recent update to WPA3 improves the security of enterprise networks, as these were often insecurely configured [10,16].

Other work studied Wi-Fi provisioning schemes [38], inferred and analyzed state machines [52], and studied potential electromagnetic side-channel leaks in 802.11 radios [17].

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 [31, Table 11-12].

The RSN element also contains a flag to indicate whether the device supports SPP A-MSDUs. When a device does not set this flag, but does set the SPP required flag, this means no A-MSDUs should to sent to it. In other words, if a device does not support SPP A-MSDUs, this flag combination instructs peers to never accept or send A-MSDUs [31, Table 11-12]. This flag combination also prohibits the device itself to send or accept A-MSDUs, preventing all possible aggregation attacks.

Unfortunately, most devices ignore the SPP flags in the RSN element, and will send or accept non-SPP A-MSDUs independent of these flags. Therefore, if a device sets the SPP required flag, and a peer still sends non-SPP A-MSDUs, these will be dropped. In other words, setting the SPP required flag may degrade reliability. It also means the attack of Section 3.5 remains possible because, when the sender does not authenticate the A-MSDU flag, it masks the A-MSDU flag to zero in the authenticated metadata (recall Section 2.1). As a result, an attacker can unset the A-MSDU flag without the receiver noticing this. Nevertheless, the impact of this attack appears low, and as a defense we therefore still recommend to set and adhere to the SPP required flag in the RSN element.

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.

To mitigate (but not prevent) attacks against receivers, a transmitter can decide to never use fragmentation. However, this may reduce reliability. Note that clearing the fragment cache whenever installing a key does not prevent mixed key attacks when using multiple key support (recall Section 4.4).

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.

7.3 Overall discussion

Test considerations Several devices were not affected by our default attack(s), but only to minor variants, e. g., FreeBSD and OpenBSD. Therefore, we recommend to only consider a device secure if there are explicit checks in the code to prevent attacks and if practical tests show it is indeed not vulnerable.

To test attacks, driver and firmware patches are required to reliably inject fragmented frames. Otherwise important fields may be overwritten, causing attacks to fail. This obstacle when testing attacks may be one reason why the discovered design flaws went unnoticed for more than two decades.

Future work Crucial future work is formally modeling 802.11's aggregation and fragmentation features to evaluate, and increase confidence in, the correctness of our defenses.

It is also worthwhile to investigate how 802.11ax can be abused to induce fragmentation in practice, since this would increase the impact of our fragmentation-based attacks.

We also believe it is important to study in more detail how different flaws can be combined in practical attacks. Finally, we consider it interesting future work to analyze other (proprietary) protocols for similar fragmentation-based flaws.

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/fragattack

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

[3] Nadhem J. Al Fardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *IEEE S&P*, 2013.

[4] Martin R Albrecht and Kenneth G Paterson. Lucky microseconds: a timing attack on amazon's s2n implementation of TLS. In *Eurocrypt*, 2016.

[5] Nadhem AlFardan, Daniel Bernstein, Kenneth Paterson, Bertram Poettering, and Jacob Schuldt. On the security of RC4 in TLS and WPA. In *USENIX Security*, 2013.

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

[7] Amazon. AWS IP address ranges. Retrieved 3 June 2020 form https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html, 2020.

[8] Arch Linux Wiki. Network configuration / wireless. Retrieved 18 February 2020 from https://wiki.archlinux.org/index.php/Network_configuration/Wireless, 2020.

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

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

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

[12] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In *CCS*, 2016.

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

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

[15] 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.

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

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

[18] 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.

[19] Ralph Droms. Dynamic Host Configuration Protocol. RFC 2131, 1997.

[20] T. Duong and J. Rizzo. Here come the xor ninjas. In *Ekoparty Security Conference*, 2011.

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

[22] Christina Garman, Kenneth G. Paterson, and Thyla Van der Merwe. Attacks only get better: Password recovery attacks against RC4 in TLS. In *USENIX Security*, 2015.

[23] 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.

[24] Finn Michael Halvorsen and Olav Haugen. Cryptanalysis of ieee 802.11i TKIP. Master's thesis, 2009.

[25] 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.

[26] 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.

[27] Alex Hern. Major sites including new york times and BBC hit by ransomware malvertising. *The Guardian*, 2016.

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

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

[30] IEEE P802.11ax/D4.3. *Amendment 1: Enhancements for High Efficiency WLAN (draft)*, 2019.

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

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

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

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

[35] Lars-Erik Jonsson, Lars Åke Larzon, Gorry Fairhurst, Stephen Pink, and Mikael Degermark. The Lightweight User Datagram Protocol (UDP-Lite). RFC 3828, 2004.

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

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

[38] 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.

[39] Jie Liang. Simplifying implementation of CCMP mode. Retrieved 29 May 2020 from `mentor.ieee.org/802.11/dcn/03/11-03-0122-00-000i-simplifying-implementation-of-ccmp-mode.ppt`, 2003.

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

[41] 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.

[42] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This POODLE bites: exploiting the SSL 3.0 fallback, 2014.

[43] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You are what you include: Large-scale evaluation of remote JavaScript inclusions. In *CCS*. ACM, 2012.

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

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

[46] Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, and Yuval Yarom. The 9 lives of bleichenbacher's CAT: New cache attacks on TLS implementations. In *IEEE S&P*, 2019.

[47] Eyal Ronen, Kenneth G. Paterson, and Adi Shamir. Pseudo constant time implementations of TLS are only pseudo secure. In *CCS*, 2018.

[48] Jon Rosdahl, Mark Hamilton, and Michael Montemurro. Minutes REVmd – may 2018 – warsaw. Retrieved 14 September 2020 from `https://mentor.ieee.org/802.11/dcn/18/11-18-0616-00-000m-minutes-revmd-may-2018-warsaw.docx`, 2018.

[49] Flavia Salutari, Danilo Cicalese, and Dario J Rossi. A closer look at IP-ID behavior in the wild. In *International Conference on Passive and Active Network Measurement*. Springer, 2018.

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

[51] 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.

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

[53] Dr. Joseph D. Touch. Updated Specification of the IPv4 ID Field. RFC 6864, 2013.

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

[55] 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.

[56] Mathy Vanhoef and Frank Piessens. Advanced Wi-Fi attacks using commodity hardware. In *ACSAC*, 2014.

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

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

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

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

A Driver and firmware modifications

Our test tool relies on Linux's virtual interface support and we used it with a TL-WN722N to inject frames. To avoid the kernel function `ieee80211_tx_h_sequence` from overwriting sequence numbers when using multiple virtual interfaces, we patched it to not modify non-zero sequence numbers. To avoid the firmware from overwriting the sequence and fragment number, we patched `ath_tgt_tx_seqno_normal` to not modify `wh->i_seq` and `wh->i_seq[0]`.

To prevent injected frames with a different QoS priority from being reordered, we patched `ath9k_htc_tx_data` to set `tx_hdr.tidno` to zero independent of the frame's priority.

More details and other required workarounds, including patches for other wireless network cards, can be found on [1].

B Non-consecutive packet number attack

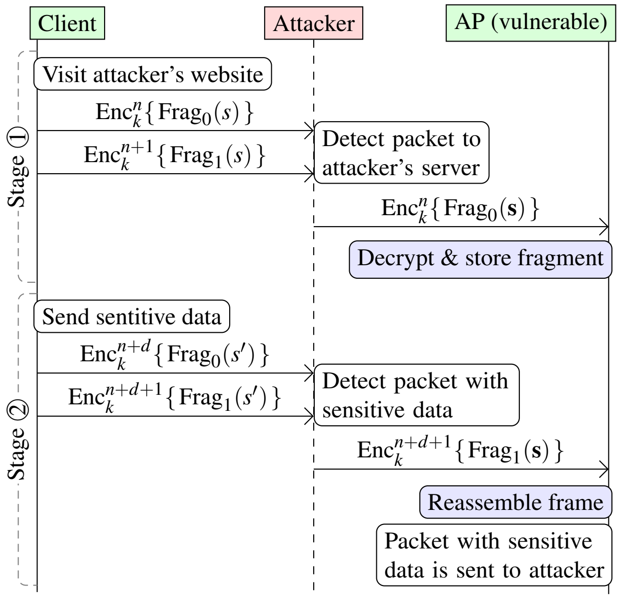
Client
Attacker
AP (vulnerable)
Stage ①
Visit attacker's website
$\text{Enc}_k^n\{\,\text{Frag}_0(s)\,\}$
$\text{Enc}_k^{n+1}\{\,\text{Frag}_1(s)\,\}$
Detect packet to attacker's server
$\text{Enc}_k^n\{\,\text{Frag}_0(\mathbf{s})\,\}$
Decrypt & store fragment
Stage ②
Send sentitive data
$\text{Enc}_k^{n+d}\{\,\text{Frag}_0(s')\,\}$
$\text{Enc}_k^{n+d+1}\{\,\text{Frag}_1(s')\,\}$
Detect packet with sensitive data
$\text{Enc}_k^{n+d+1}\{\,\text{Frag}_1(\mathbf{s})\,\}$
Reassemble frame
Packet with sensitive data is sent to attacker

Figure 9: Attacking an AP that accepts fragments with non-consecutive PNs. 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. Between stages ① and ②, the attacker never uses sequence number s in forwarded frames.

C Advanced Aggregation (A-MSDU) attacks

In our aggregation attack, we used IPv4 packets with an attacker-controlled IP identification value and payload. However, it may be infeasible to send such IPv4 packets to a victim. Instead, an adversary may wish to abuse devices that send IPv4 packets with an IP ID value of zero. This is useful when combined with the fragmentation bug of Section 6.3 to control a second plaintext fragment containing injected packets, or to attack an AP using a client in the BEAST threat model.

When setting the A-MSDU flag of an encrypted frame that transports an IPv4 packet with an ID value of zero, the last two bytes of the destination IP address become the length field of the *second* A-MSDU subframe. Depending on the IP addresses used by the network, this results in a short subframe, allowing an adversary to control the content of the third A-MSDU subframe. For instance, when using the subnet 192.168.1.0/24, the length of the second subframe is between 256 and 512 bytes, leaving enough space to inject frames by controlling the content of the third subframe. When targeting IPv4 packets sent to the internet that have an identification value of zero, we conjecture that traffic analysis can be used to detect which server a user is connecting to, and based on this the adversary can predict when the last two bytes of the IP address result in a short A-MSDU subframe.

D Plaintext fragment injection against Linux

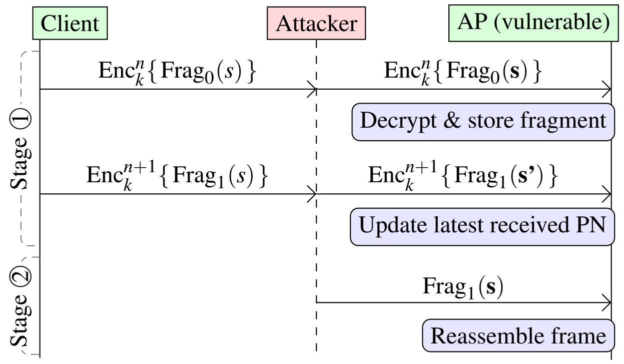
Client
Attacker
AP (vulnerable)
Stage ①
$\text{Enc}_k^n\{\,\text{Frag}_0(s)\,\}$
$\text{Enc}_k^n\{\,\text{Frag}_0(\mathbf{s})\,\}$
Decrypt & store fragment
$\text{Enc}_k^{n+1}\{\,\text{Frag}_1(s)\,\}$
$\text{Enc}_k^{n+1}\{\,\text{Frag}_1(\mathbf{s'})\,\}$
Update latest received PN
Stage ②
$\text{Frag}_1(\mathbf{s})$
Reassemble frame

Figure 10: Tricking Linux into accepting plaintext fragments. In stage ① the attacker forwards a legitimate encrypted Frag_0. The second fragment is forwarded under a different sequence number so it will not be combined with Frag_0. However, Linux does update the session variable containing the latest received PN to $n+1$. In stage ② the attacker injects a plaintext fragment with sequence number s. The PN in the session variable is now consecutive to the one of Frag_0, and since the plaintext fragment has the same sequence number s as Frag_0, the encrypted and plaintext fragment will be combined. The resulting frame will then be processed (or forwarded) by Linux.

E Fragmentation attacks against FreeBSD

The default mixed key and fragment cache attack do not work against FreeBSD, because it rejects fragments if an unrelated frame (of the same sender) is received in-between these fragments.

This can be overcome by realizing that broadcast frames do not influence the defragmentation process of FreeBSD. An adversary can use this to forward 4-way handshake frames without affecting the defragmentation process by encapsulating them inside an A-MSDU frame with a broadcast receiver address, where the destination MAC address in the A-MSDU subframe equal the unicast address of the receiver.

To use this technique in an attack, the other station has to send *plaintext* 4-way handshake messages. This is the default behavior of devices in the cache attack, but not during the rekey handshake in the mixed key attack. Nevertheless, the RT-AC51U AP does send 4-way handshake frames in plaintext during a rekey. Therefore, when a FreeBSD client is connected to such an AP, an adversary can capture the plaintext 4-way handshake messages, and encapsulate them into broadcast A-MSDU frames. This causes FreeBSD to renew the pairwise session key without affecting the defragmentation process, allowing an adversary to perform a mixed key attack against FreeBSD. We successfully tested this technique against FreeBSD 12.1 when connected to our RT-AC51U. We also performed a fragment cache attack against a FreeBSD AP when it was using an TL-WN725N as a network card.