Breaking WPA-TKIP Using Side-Channel Attacks*

DOMIEN SCHEPERS, Northeastern University
AANJHAN RANGANATHAN, Northeastern University
MATHY VANHOEF, New York University Abu Dhabi

We show through wardriving experiments that WPA-TKIP is still widely deployed in a large percentage of today's wireless networks. Specifically, we measure the usage of cipher suites in protected Wi-Fi networks in several distinct geographic areas and find that 44.81% of protected networks still support the old WPA-TKIP cipher. Motivated by these results, we systematically analyze the security of WPA-TKIP implementations.

We thoroughly investigate the security of devices that deploy WPA-TKIP by inspecting all the components of their Wi-Fi stack, thereby discovering new side-channel vulnerabilities. Particularly, we expose vulnerabilities in the power management and fragmentation mechanism, leverage hardware decryption modules, and present two fatal vulnerabilities due to incorrect handling of hardware-decrypted frames. We validate our findings on major operating systems such as Linux and OpenBSD. Furthermore, we investigate specific implementations of Mediatek and Broadcom devices and find them to be vulnerable to our side-channel attacks as well.

Our novel side-channel attacks bypass existing countermeasures and recover the Michael message authentication key in as little as 1 to 4 minutes. Using this key, an adversary can then decrypt and inject network traffic. In contrast, previous attacks needed 7 to 8 minutes. These results stress the urgent need to stop using WPA-TKIP.

1 INTRODUCTION

Wireless networks have significantly evolved over the recent years due to increased demands and requirements of new applications, and so too have their security protocols. For instance, recently the Wi-Fi Alliance released Wi-Fi Protected Access 3 (WPA3) [35], and created a certification for Opportunistic Wireless Encryption [12]. WPA3 is expected to replace the current WPA2 security protocol with better security guarantees (e.g., longer keys), and offers an optional updated protocol to easily configure Wi-Fi on the new generation of smart things and systems without traditional input devices. Previous iterations of wireless security protocols, such as Wired Equivalent Privacy (WEP) and version one of WPA, have shown to be vulnerable to numerous attacks. For example, the first attack on WEP was demonstrated in 2001 [9], causing WEP to be officially deprecated by the IEEE in 2004 [3]. Several attacks on the Temporal Key Integrity Protocol (TKIP), an interim solution introduced following the WEP attacks, have also been demonstrated. In 2009, Tews and Beck published the first practical attacks against WPA-TKIP [26]. Several improvements have been made to the original attack making it more efficient and practical [18, 27, 29]. Numerous countermeasures were introduced to limit the effect of potential attacks. For example, Tews and Beck propose using a short rekeying interval, and disabling the transmission of Michael MIC failure reports [26].

*This report is primarily based on our academic paper [23] and some novel (survey) results.

Authors' addresses: Domien Schepers, Northeastern University, schepers.d@husky.neu.edu; Aanjhan Ranganathan, Northeastern University, aanjhan@northeastern.edu; Mathy Vanhoef, New York University Abu Dhabi, mathy.vanhoef@nyu.edu.

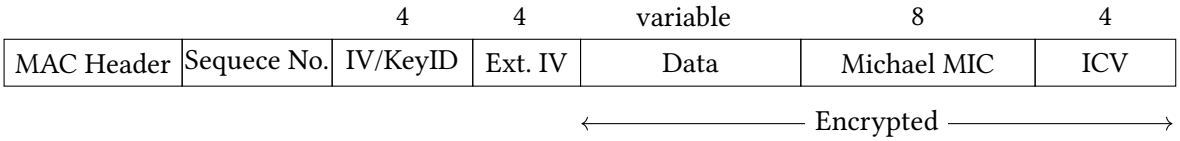

Fig. 1. Simplified construction of an expanded TKIP Media Access Control Protocol Data Unit (MPDU).

Given these well-known vulnerabilities, and the potential implications of an attack, one would assume that support for such legacy protocols are limited, and in majority of the cases no longer deployed in modern Wi-Fi networks. However, our wardriving efforts in several geographic areas revealed that, although WEP deployments have significantly reduced over the years, nearly half of all encrypted Wi-Fi networks still support and use WPA-TKIP. Specifically, our experiments across three different countries show that TKIP is still supported by 44.81 percent of encrypted networks. It is important to note that TKIP was even partly deprecated by the Wi-Fi Alliance in 2015 [34]. Supporting insecure legacy protocols increases the attack surface, and it is our belief that through this paper we will help accelerate the process of completely deprecating support for WPA-TKIP.

In this paper, we introduce novel side-channel attacks that abuse power management features, fragmentation of data frames, and the hardware decryption features of a wireless network card. Unlike previously known attacks, the presented attacks do not rely on specific features such as the Michael MIC failure reports and are therefore unaffected by existing countermeasures. Furthermore, our attacks enable an adversary to attack the Access Point (AP), instead of only the client, and are significantly faster to execute. As a result, an adversary can now also decrypt data sent by the AP, and can inject frames towards any client. Given that TKIP is still supported by a large number of wireless networks, where even modern clients must use it to decrypt group-addressed frames, our research has a significant impact on the users of these networks.

2 BACKGROUND

In this section we describe relevant parts of the IEEE 802.11i standard [3], focus on the specification of TKIP, and explain the original Beck and Tews attack against TKIP [26].

2.1 Temporal Key Integrity Protocol

The Temporal Key Integrity Protocol (TKIP) was designed to fix the weaknesses of WEP, while still being able to run efficiently on old hardware. In order to provide confidentiality, TKIP uses the Rivest Cipher 4 (RC4) stream cipher, which is no longer secure [9]. TKIP fixes the weaknesses of the WEP IV, and added replay protection, by using a TKIP Sequence Counter (TSC) [1]. The TSC is 6-byte long, and is increased monotonically for each Media Access Control Protocol Data Unit (MPDU). If a received frame has a lower than expected sequence number, it has to be dropped in order to prevent replay attacks. The TSC is transmitted in the IV and Extended IV fields, where the extended IV bit is always set when TKIP is used. Note that the TSC is different from the sequence number field at the beginning of the frame. More precisely, the sequence number field is used to detect retransmissions, and is present even in plaintext frames. TKIP also added a new integrity check named Michael Message Integrity Code

(MIC) [8], providing better security against the alterations of data packets, due to weaknesses in the ICV. Figure 1 shows the simplified construction of a TKIP MPDU.

When constructing TKIP frames, large MAC Service Data Unit (MSDU) can be fragmented in at most 16 MAC Protocol Data Units (MPDU) fragments. To encrypt a frame, TKIP first generates a per-packet key using a two-phase key mixing process [1]. The first phase uses the transmitter MAC address, Temporal Key (TK), and the 32 most significant bits of the TSC, to generate a so called TKIP-mixed Transmit Address and Key (TTAK). The second phase uses the TTAK, TK and remaining 16 least significant bits of the TSC to generate the WEP seed. Generating the WEP seed in two phases is computationally more efficient as the first phase has to be computed only once every 65536 (2^{16}) packets. It is even possible to calculate the seeds in advance for the reason that the TSC increases monotonically per MPDU. Next, the Michael algorithm generates an 8-byte keyed MIC that is calculated over the entire MSDU, as it takes as input the Michael key, source address, destination address, priority and plaintext MSDU data. Since the Michael MIC can only be verified when the entire MSDU has been received by the recipient, it will be carried in the last MPDU fragment. The resulting MPDUs are encapsulated through WEP, using the newly generated IV and RC4 key. WEP encapsulation adds a 4-byte ICV, which is therefore calculated for each individual MPDU fragment. Finally, a Frame Check Sequence (FCS) is calculated as a 32-bit CRC over the entire frame, including the MAC header, and appended to the MPDU.

2.2 TKIP Countermeasures

The Michael MIC provides stronger security guarantees than a Cyclic Redundancy Check (CRC), but still offers weak defenses against message forgeries [14]. However, it is the best that could be achieved on existing hardware. In 2004, it was shown that the Michael algorithm is invertible [36]. Given that the Michael algorithm provides inadequate security, several countermeasures were added in order to mitigate attacks [1]. The defined countermeasures are such that, when a client receives an invalid MIC, it will send a Michael MIC Failure Report. If an access point receives two such reports within one minute, all clients using TKIP are temporarily blocked for one minute. After the one-minute interval the clients can reassociate and negotiate fresh cryptographic keys. An adversary can easily exploit these countermeasures to perform a Denial of Service (DoS) against all users [29].

2.3 Quality of Service

The IEEE 802.11e amendment defines Quality of Service (QoS) enhancements for wireless LAN networks [2]. The Wi-Fi Alliance provides an interoperability certification named Wi-Fi Multimedia (WMM), also known as Wireless Multimedia Extensions (WME), which is a subset of the IEEE 802.11e amendment. The specification provides eight different communication channels, representing different QoS needs. The QoS enhancements have an important impact on wireless networks supporting TKIP, as the Michael MIC covers the channel number (i.e., the priority of the frame). Additionally, under the QoS enhancements each channel has its own TKIP Sequence Counter (TSC) to verify received frames. As a result, an adversary can replay a packet on a QoS channel with a lower TSC, as this approach will bypass the TSC verification [17].

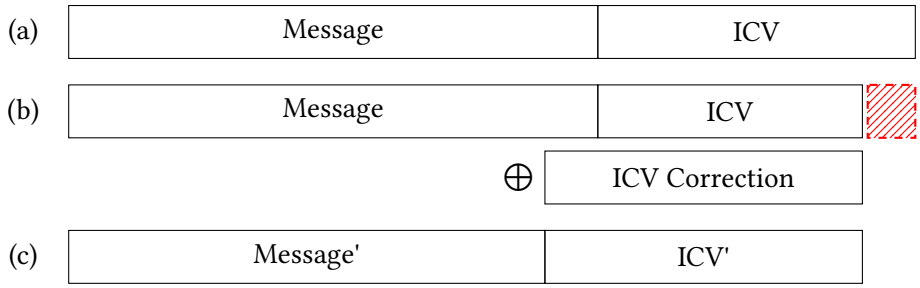

Fig. 2. Illustration of the ChopChop Attack.

2.4 Beck and Tews Attack

In 2009, Tews and Beck presented the first practical attack against wireless networks that use TKIP [26]. It is based on the ChopChop attack against WEP, which enables an adversary to decrypt packets without knowing the encryption key. The ChopChop attack was first described by KoreK on the NetStumbler forum in September 2004 [15], and Guennoun et al. have theoretically proven the attack [11]. The attack is possible by exploiting the lack of replay protection and usage of a weak ICV in the WEP protocol. The ChopChop attack works by repeatedly chopping off the last byte of an encrypted message, recovering its value, and repeating this process for the next byte until the full plaintext is recovered. Chopping off a byte will result in a corrupted message since the ICV, calculated as a 32-bit CRC, is no longer valid. However, due to the linearity of a CRC, if one knows the plaintext value of the chopped-off byte, a valid ICV can be reconstructed [11]. Using an oracle, the adversary can learn if the modified message resulted in a valid ICV, thereby learning the plaintext value of a byte. In the original ChopChop attack against WEP, the oracle consisted of having the Access Point (AP) relay a message within the network. When the adversary injects the chopped packet, the AP will verify the ICV. If an erroneous guess was made for the plaintext value of the chopped-off byte the packet will be silently dropped, if guessed correctly it will be relayed by the AP, thereby revealing the correct guess. Alternatively, an adversary can send the message to the AP coming from an unauthenticated client [26]. If the ICV was constructed correctly, the AP will generate and sent an error message, otherwise the packet is silently discarded. Thus, by guessing all 256 (2^8) possible values, or 128 on average, one can recover the plaintext value of a byte. Figure 2 provides an illustration of the ChopChop attack. A WEP-encrypted message (a) consists of a Message and 4-byte ICV. The ChopChop attack will repeatedly chop off the last byte, and perform an ICV Correction based on the guessed plaintext value, as seen in (b). This results in a newly constructed message (c) that, if guessed correctly, will contain a valid ICV.

Tews and Beck used a variant on this technique to decrypt packets directed towards a client in a wireless network using TKIP. Packets have to be injected into the network using a different QoS channel, having a lower TSC. This allows the adversary to bypass the TSC verification in TKIP, and have the client verify the ICV. If the guessed value for the chopped-off byte was incorrect, the frame will be silently dropped, since it has an invalid ICV. If it was correct, the frame will be processed, and is likely to result in a Michael MIC failure. Upon detecting a Michael MIC failure, the client will sent a Michael MIC Failure Report. The adversary can listen for such reports, as an oracle, therefore learning if the guessed

value was correct. Due to the TKIP countermeasures, an adversary can decrypt at most one byte per minute. The attack can only target traffic towards the client, since the AP does not send Michael MIC Failure Reports. Tews and Beck showed how this attack can be used to decrypt an ARP reply message in an average of 12 to 15 minutes [26]. They do so by decrypting the last 12-bytes, consisting of the 8-byte Michael MIC and 4-byte ICV. When the plaintext Michael MIC and ICV are recovered, it is possible to guess the remaining bytes since most of the bytes in an ARP reply are known. The guesses can be verified by calculating the ICV and matching it with the decrypted result. Using the inverse Michael algorithm [36], the Michael MIC key for AP-to-client traffic can be recovered. In combination with the recovered keystream from the ARP reply, an adversary is able to forge a new message for every available QoS channel [26]. The length of the forged messages can be equal to at most the length of the recovered keystream.

Countermeasures against the Beck and Tews attack consist of using a short rekeying interval, and disabling the transmission of the failure reports [26]. Alternatively, a random time-delay could be added to the transmission of the failure reports. In particular, for Linux systems, if the `CONFIG_DELAYED_MIC_ERROR_REPORT` build option in HostAPs supplicant is set, the client delays the transmission of the Michael MIC Failure Report by a random amount of time between 0 and 60 seconds [16].

Over the years, several improvements have been made to the original attack of Beck and Tews [18, 27, 29], allowing for an improved execution time of 7 to 8 minutes [27]. Like the original Beck and Tews attack, all of these improvements are based on the transmission of Michael MIC Failure Reports. In this paper we will show how, even when the transmission of failure reports is disabled, we still succeed in devising attacks against WPA-TKIP.

3 MOTIVATION

To better understand the usage of WPA-TKIP in today's wireless networks, we conducted an extensive survey. This survey was conducted in December 2018 in four distinct neighborhoods, across three different countries: (1) Back Bay and Fenway in Boston of the United States; (2) Hasselt in Belgium; and (3) Leipzig in Germany. We used Kismet to passively collect beacon frames by channel hopping over channels 1, 6 and 11. These three channels are the only non-overlapping channels in the 2.4 GHz band. The experiment was conducted on a Raspberry Pi 3 using a TP-Link TL-WN722N Wi-Fi Dongle. The collected beacon frames contain the Service Set Identifier (SSID), the AP's MAC address (i.e. the BBSID), and the Robust Secure Network (RSN) information of a wireless network. The RSN element contains information such as the supported pairwise cipher suites, and the group cipher suite. The group cipher defines the cipher that is used to encrypt multicast and broadcast traffic. In a mixed-mode network, where the AP supports both TKIP and CCMP for the pairwise ciphers, the oldest cipher is chosen as the group cipher. This assures that all clients in the network support the group cipher, and hence can receive encrypted multicast and broadcast frames. All this information in the RSN element was used to build a list of unique networks and the cipher suites they support. The uniqueness of a wireless network is based on its BSSID.

We discovered a total of 29550 unique networks, of which 80.97 percent (23926 out of 29550) supported encryption. Since we want to know how people configure protected Wi-Fi networks, we exclude open

Table 1. Usage of TKIP in protected Wi-Fi networks.

Year	Country	Region	TKIP	CCMP	Total
2018	U.S.A.	Back Bay	4958 (44.82%)	6101	11059
2018	U.S.A.	Fenway	1818 (38.03%)	2962	4780
2018	Belgium	Hasselt	3221 (58.24%)	2310	5531
2018	Germany	Leipzig	725 (28.36%)	1831	2556
			10722 (44.81%)	13204	23926

networks in the remainder of our analysis. That is, our analysis only considers protected Wi-Fi networks. Out of the 23926 protected networks, we discovered 15839 in Boston, 5531 in Hasselt, and 2556 in Leipzig. Surprisingly, a total of 44.81 percent (10722 out of 23926 protected networks) support TKIP as their group cipher suite. More detailed statistics of the usage of WPA-TKIP in each region are provided in Table 1. In 2019, we performed a new survey in the United States and the United Arab Emirates, showing similar results. For comparison, in April 2016, an independent study of Vanhoef showed that 57.52 percent of encrypted wireless networks supported WPA-TKIP [28].

In spite of the work of researchers, vendors disabling TKIP, and the Wi-Fi Alliance who is discouraging the usage of WPA-TKIP, we still find that nearly half of all protected Wi-Fi network uses WPA-TKIP, which is a deeply concerning amount. Another concern is that, even though researchers discovered several weaknesses in TKIP, the IEEE has not yet deprecated TKIP. Although in July 2017 members of the IEEE unanimously agreed that TKIP should either be removed from the standard, or that TKIP should at least be deprecated, in the end no action was taken [20]. Instead, in May 2018, members of the IEEE voted to keep WEP and TKIP in the 802.11 standard [21]. Moreover, they also voted that TKIP should not yet be marked as deprecated. This decision was made because there are still known implementations of TKIP on the market [24], and because the countermeasures discussed in Section 2.2 and 2.4 mitigate most weaknesses. As a result, we expect that without extra action, TKIP will keep being used in practice.

Motivated by this ever-lasting support for the insecure TKIP protocol, our goal is to devise new attacks against it. In particular, we show how we can bypass all existing countermeasures, and show that our attacks are significantly faster than existing ones. We hope this paper will help accelerate the process of completely deprecating support for WPA-TKIP.

4 FINDING NEW SIDE-CHANNELS

In this section we systematically analyze the security of TKIP implementations. Unfortunately, even when all countermeasures against known attacks are enabled, we found that devices are still vulnerable to new attacks. In particular, we inspected the full Wi-Fi stack of TKIP implementations, and discovered various new side-channels.

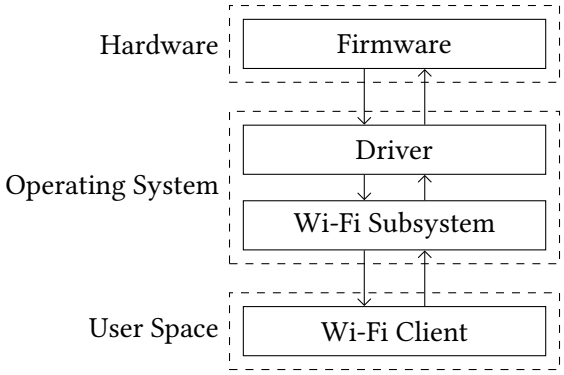

Fig. 3. Main components of a Wi-Fi stack.

4.1 Security Analysis Methodology

To assess the security of devices that implement TKIP, we systematically analyze the Wi-Fi stack of several operating systems. Note that previous works on TKIP only focused on the entity that generates MIC failures, instead of inspecting the full Wi-Fi stack, meaning they were unable to detect all side-channel vulnerabilities. However, in this paper we analyze all the major components of the Wi-Fi stack. As illustrated in Figure 3, the main components encompasses the firmware running on the network card, the driver and Wi-Fi subsystem in the operating systems, and the Wi-Fi management code that runs in user space (e.g. Linux's hostapd). Investigating all these components enabled us to discover cross-layer attacks that abuse properties of various components of the Wi-Fi stack. More precisely, we analyzed all components of the Wi-Fi stack in the following manner:

(1) We analyzed the countermeasures that were implemented to mitigate existing TKIP attacks.
(2) We searched for logical bugs, such as forgetting to verify the integrity of (fragmented) frames.
(3) We looked for new side-channel vulnerabilities that reveal whether the ICV of a TKIP frame is correct.

We specifically investigated open-source operating systems such as Linux and OpenBSD. Additionally, we inspected the source code of MediaTek and Broadcom routers. Analyzing these implementations results in a representative overview of the state of TKIP security, since open source code is heavily used in both clients and access points. Another advantage of focusing on open-source implementations, is that the discovered attacks are easier to debug and reproduce, and that implementations can be more thoroughly analyzed since the code is public. All combined, investigating the full Wi-Fi stack of these implementations led to the discovery of various new (cross-layer) side-channel attacks.

4.2 Deployed Countermeasures

Our code inspection of client-side TKIP implementations revealed that Ubuntu implements the delayed reporting of MIC failures. This means the client delays the transmission of MIC failures by a random amount of time between 0 and 60 seconds. Similarly, Debian, Gentoo, Alpine Linux, and several other Linux distributions, also delay the transmission of MIC failures. This countermeasure prevents all

Listing 1. Sequence of IEEE 802.11 Rx handlers in Linux kernel version 3.11.

```
void ieee80211_rx_handlers(struct ieee80211_rx_data *rx,struct sk_buff_head *frames)
    // Surrounding code is skipped for brevity
    CALL_RXH(ieee80211_rx_h_decrypt)
    CALL_RXH(ieee80211_rx_h_check_more_data)
    CALL_RXH(ieee80211_rx_h_uapsd_and_pspoll)
    CALL_RXH(ieee80211_rx_h_sta_process)
    CALL_RXH(ieee80211_rx_h_defragment)
    CALL_RXH(ieee80211_rx_h_michael_mic_verify)
}
```

variants of the Beck and Tews attack. On OpenBSD, the client sends two MIC failure reports back-to-back when it encounters two MIC failures within a minute. Additionally, after sending both failure reports, it disconnects from the network. In case there is only one MIC failure within a minute, no failure reports are transmitted. This countermeasure prevents all variants of the Beck and Tews attack. The iNet wireless daemon (iwd) on Linux does not send MIC failure reports at all. Although this prevents all variants of the Beck and Tews attack, this approach is not recommended since it might enable novel (brute-force) attacks against the Michael algorithm. On the AP side, Linux's hostapd daemon by default refreshes the group key every 10 minutes if TKIP is used. However, by default it does not periodically refresh the pairwise session key. Finally, the Mediatek and Broadcom routers that we inspected typically refresh the group key every 60 minutes. All combined, we can observe that most implementations have adopted countermeasures that mitigate, or even completely prevent, all variants of the Beck and Tews attacks. Unfortunately, our new side-channel attacks are able to bypass these countermeasures.

4.3 Linux Power Management Oracle

Against Linux, we discovered that power management features of Wi-Fi can be abused to determine whether the ICV of an injected TKIP frame is correct. In particular, we can abuse the advanced power management features that were added to the standard in 2005 by amendment 802.11e [2]. This amendment defines features that allow a device to enter a long-term sleep mode. Devices can then either wake-up at scheduled intervals to receive buffered data from the AP (called S-APSD), or they can wake up at random moments in time to request buffered data from the AP (called U-APSD). With U-APSD, the client requests buffered data from the AP by sending a PS-Poll frame towards it. The PS-Poll frame is a control frame and is transmitted unauthenticated and unencrypted. When the AP receives the PS-Poll frame, it will reply with the buffered data, or with a Null-frame if there is no buffered data available. We found that U-APSD, with its PS-Poll frame, can be used as an oracle against Linux devices. To understand why the oracle works, we have to look at the Linux kernel code. In Linux kernel versions 3.11 and lower, there is a logical flaw in the way received frames are processed. Frames will be decrypted first, before their header flags, such as the power save mode, are checked. As a result, a client is marked by the AP as being in power save mode only when decryption has succeeded. Listing 1 shows the sequence of 802.11 receive (Rx) handlers in Linux kernel version 3.11. On the receipt of a Wi-Fi frame, the Linux kernel will first attempt to perform WEP decryption in the `h_decrypt` handler. This process will verify if the ICV is correct, and if not, the MSDU will be dropped silently. However, if correct, the

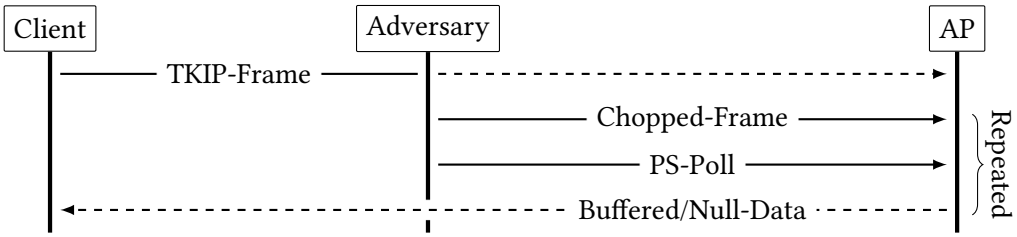

Fig. 4. Abusing power management features of Linux's Wi-Fi stack to determine whether the ICV of the TKIP frame is correct. When the AP replies using a Null-Data frame, the ICV is correct. If the AP does not reply, the ICV is wrong.

next handlers will be called. One of these handlers inspects the power save flags in the MAC header, to check if the client is entering power saving mode. Therefore, if the AP has marked the client as being in power saving mode, we know that the ICV of the TKIP frame was correct. Using the PS-Poll message, the adversary can test if a client has entered power saving mode. That is, if the client entered power saving mode, the AP replies to the PS-Poll frame with the buffered data, or with a Null-frame. However, if the client was not marked as being in power saving mode, the AP will ignore the PS-Poll frame. Thus, if the PS-Poll message results in a reply from the AP, we know the ICV was correct.

To summarize, our side-channel attack works as follows. First, the adversary captures a TKIP-encapsulated message from the client to the Access Point (AP). The frame will be used to perform a ChopChop attack, and may be received by the AP, a MITM position is not required. In order to use the power management oracle, the adversary has to modify the frame as follows:

- Set the Power Management bit in the MAC header. This will notify the AP that the client is entering power saving mode.
- Change the Quality of Service (QoS) Traffic Identifier (TID). This allows us to use a channel with a lower TKIP Sequence Counter (TSC), passing the TSC verification. If no QoS header is present, one can be inserted.
- Set the fragment number to one. This indicates the MPDU is fragmented, and that the Michael MIC is present in the last fragment. Therefore, the AP will not verify the Michael MIC for this frame, effectively bypassing TKIP countermeasures.

With these changes to the TKIP frame, the adversary can perform a ChopChop attack to recover the plaintext message. In order to verify if a chopped-off byte is guessed correctly, a PS-Poll message is sent to the AP. The PS-Poll message is spoofed as having the source address of the target client. If the guessed byte was correct, the AP will reply with buffered data or a null-data frame. If the guessed byte was wrong, the AP silently drops the chopped TKIP frame, and hence will not send any response to the PS-Poll frame. Repeatedly chopping off bytes, and guessing their values using the above technique, allows an adversary to decrypt the frame. Figure 4 shows a simplified illustration of the attack method. We evaluated the attack against a Linux machine running hostapd. Similar to the original Beck and Tews attack, our goal is to decrypt an ARP request. After using the power management oracle to decrypt its content, we can recover the Michael MIC key, which is possible because the Michael algorithm is invertible [36]. However, unlike Tews and Beck, it is not necessary to wait one minute after each successfully

decrypted byte, since our attack method does not trigger TKIP countermeasures (i.e. Michael MIC Failure Reports). As a result, our attack is significantly faster to execute. Comparing to Tews and Beck, we can save 11 minutes of waiting for the 1-minute TKIP time-out countermeasure to end, reducing the execution time of our attack to 1 to 4 minutes.

From Linux kernel version 3.12-rc1 onward, the order of the Receive (Rx) handlers was changed, and the decryption handler was called after processing the power save flags in MAC headers. As a result, a client will be marked as asleep if the power management flag was set in the MAC header, regardless of successful decryption. Therefore the oracle will no longer work. We remark that the kernel version in which our oracle is no longer present, Linux 3.12, was released on November 2, 2013. From March 2015 onward, usage of TKIP is being partly discouraged by the Wi-Fi Alliance [6]. Therefore, we argue that devices which have TKIP enabled by default, are likely to be running an older and therefore vulnerable kernel version. Regardless, we will describe another side-channel technique in Section 5.2 that also works against newer Linux kernel version.

4.4 MediaTek Fragmentation Oracle

Against MediaTek, we discovered an oracle that leverages the way in which fragmented frames are cached and reassembled. Recall that large MSDUs are fragmented into MPDUs, and each MPDU is given a fragment number which is increased monotonically. In all but the last frame, the "More Fragments" flag is set in the frame's MAC header. Upon receipt of a fragmented frame, it will be cached by the receiver for later re-assembling. Upon receipt of the last fragment, the receiver re-assembles and processes the message. Interestingly, if the AP receives a fragmented TKIP frame with a correct ICV, this fragment will overwrite the previous fragment in the cache, or may even cause the fragment cache to be cleared. We will show how this behavior can be abused as a side-channel. Analyzing the side-channel revealed an additional vulnerability, where the AP accepts plaintext frames from an authenticated client, independent of wether the client uses WPA-TKIP or AES-CCMP. The vulnerability was verified by injecting a plaintext ARP request, resulting in the receipt of an encrypted ARP reply. Since the AP's replies are encrypted, its contents are unknown to an adversary. However, this is sufficient for an adversary to easily inject plaintext fragmented frames to verify the effects of the fragmentation oracle. As an advantage, this means that the adversary does not need to fragment an additionally captured TKIP frame during an attack.

As an example, we first describe how an adversary can inject a fragmented plaintext ARP request to the AP. ARP is used for discovering the link layer address associated to a network layer address. The protocol header contains, among other things, a link and network layer addresses for the source and destination. The field for the link layer source address is filled by the address of the target client. During transmission, this ARP request may be fragmented into two smaller Wi-Fi frames. Given that an AP processes (plaintext) data frames only from authenticated clients, the source address of the targeted client has to be spoofed. When the AP receives and re-assembles the fragments, it will send a reply to the spoofed hardware source address contained in the ARP request, even if this address is non-existing (i.e. spoofed to a random address). If the network uses encryption, such as TKIP, the AP replies with an encrypted message, even if the ARP request was received in plaintext. Since this works for plaintext

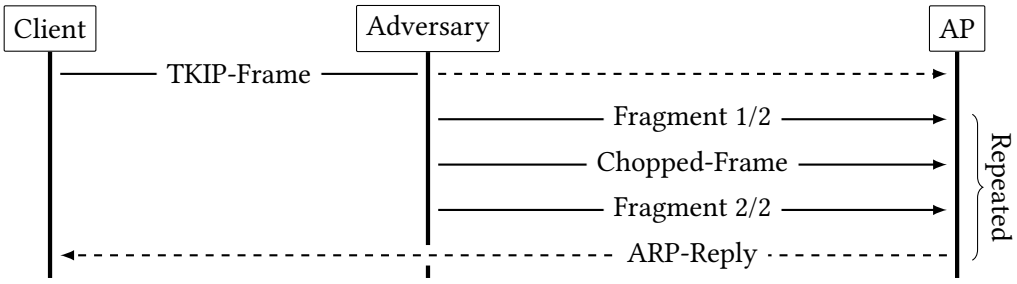

Fig. 5. Abusing the defragmentation code of MediaTek's Wi-Fi stack to determine whether the ICV of the TKIP frame is correct. If the AP does not reply, the ICV is correct. When the AP replies using an ARP reply, the ICV is wrong.

ARP requests, the adversary does not need prior knowledge about the TKIP key(stream). Our attack now works as follows. First, the adversary captures a TKIP-encapsulated message from the client to the AP, which may be used to perform a ChopChop attack. In order to use the fragmentation oracle, the adversary has to modify the captured frame. Similar to the power management oracle, the fragmentation oracle bypasses TSC verification and TKIP countermeasures by adjusting the QoS channel and fragment number:

- Change the Quality of Service (QoS) Traffic Identifier (TID). This allows us to use a channel with a lower TKIP Sequence Counter (TSC), passing the TSC verification. If no QoS header is present, one can be inserted.
- Set the fragment number to one. This indicates the MPDU is fragmented, and that the Michael MIC is present in the last fragment. Therefore, the AP will not verify the Michael MIC for this frame, effectively bypassing TKIP countermeasures.

With these changes, the adversary can perform a ChopChop attack to recover the plaintext message. To verify if a chopped-off byte is guessed correctly, the adversary sends the first fragment of a plaintext ARP request, then the modified fragmented TKIP frame, and finally sends the last fragment of the ARP request. If the guessed byte was correct, the fragmented TKIP frame causes the AP to remove the first ARP fragment from its fragmentation cache. Upon receipt of the second ARP fragment, the AP will be unable to reconstruct the plaintext ARP request, and therefore will not send an ARP reply. If the guessed byte was wrong, the AP silently drops the chopped frame. Upon receipt of the second fragment, the AP re-assembles the ARP fragments and sends an encrypted ARP reply to the random hardware source address provided in the plaintext ARP request. If an adversary crafts every ARP request with a unique hardware source address, it can easily be identified to which fragments a reply is sent. By sniffing the network for ARP replies to these randomly generated addresses, the correct guess can be detected, allowing for the plaintext message to be recovered. Figure 5 shows a simplified illustration of the attack method. The adversary captures a TKIP-frame, which may be received by the AP, that is, a man-in-the-middle position is not required. The chopped-off and fragmented ARP frames are then repeatedly sent to the AP, potentially triggering ARP replies from the AP, therefore successfully recovering the plaintext message using the fragmentation oracle.

We evaluated the attack against an ASUS RT-AC51U running firmware version 3.0.0.4. A proof-of-concept attack has been implemented, verifying that keystream and the client-to-AP Michael MIC key can be recovered. The attack has not been optimized in regards of its execution time. However, recall that the fragmentation oracle does not trigger TKIP countermeasures, and will therefore be faster to execute than existing attacks. It has been shown that MediaTek devices may be vulnerable to a downgrade attack from AES-CCMP to WPA-TKIP [32]. As a result, clients using AES-CCMP may be downgraded to WPA-TKIP and fall victim to our attack. Therefore it is important that one should disable WPA-TKIP, even if it is used in combination with CCMP.

Finally, while inspecting the plaintext ARP injection vulnerability, we found that any plaintext data packet can be injected towards the AP. As a proof-of-concept, we created a portscanner that injects plaintext TCP-SYN requests, resulting in the receipt of an encrypted RESET or SYN-ACK message. These replies have different lengths, because the SYN-ACK message includes an option field in its TCP header. As a result, we can determine if a port is open or closed.

5 ABUSING HARDWARE DECRYPTION

In this section we present new side-channels attacks that are introduced due to the usage of hardware decryption. Additionally, we present two fatal vulnerabilities in OpenBSD, whose root cause is the incorrect handling of hardware-decrypted frames.

5.1 The Impact of Hardware Decryption

During the second phase of our security analysis, we inspected the interactions between all layers (i.e. components) of the Wi-Fi stack. While doing this we discovered that the hardware decryption support of Wi-Fi chips can introduce additional side-channel vulnerabilities. Moreover, we observed that OpenBSD incorrectly handled fragmented frames that underwent hardware decryption, resulting in a trivial frame forging attack. Our hardware-based decryption side-channels enable an adversary to determine whether the ICV of a TKIP frame is correct. Similar to our previous side-channels, this can then be used to guess and verify the value of a chopped off byte. The main idea behind the side-channels is that certain drivers drop an incoming frame if hardware decryption failed. Note that with TKIP, a (hardware) decryption failure occurs if the ICV is incorrect.[1] In other words, frames are only forwarded to the Wi-Fi subsystem of the operating system if the ICV of the TKIP frame is correct. Any detectable side-effect of forwarding a frame to the Wi-Fi subsystem can therefore be abused to determine if the ICV of a TKIP frame is correct. In the remainder of his section, we discuss and evaluate various of these side-effects, most of which are implementation-dependent.

5.2 Linux Power Management

One side-effect of forwarding the frame to the Wi-Fi subsystem of the Linux operating system, is that the power management flags of the frame are inspected. Recall that our power-management side-channel of Section 4.3 abused this side-effect to determine if the ICV of a TKIP frame is correct. However, this attack only worked against Linux kernel 3.11 and lower, because newer kernels decrypted the frame

[1] TKIP uses an authenticate-then-encrypt construction, meaning a frame's authenticity can only be checked after decryption. When the ICV is correct but the Michael MIC is wrong, there was no decryption failure, but an authentication failure.

Listing 2. Processing frames on Linux kernel 3.12 and above when hardware decryption is used.

```
void process_received_frame() {
  // 1. Driver-specific: drop frame if hardware decryption failed
  if (hardware_decrypt_failed) return;
  // 2. Retransmitted (plaintext) frames are dropped
  CALL_RXH(ieee80211_rx_h_check_dup);
  // 3. The power management flags are inspected
  CALL_RXH(ieee80211_rx_h_sta_process)
  // 4. Optional software decryption, and TSC replay check
  CALL_RXH(ieee80211_rx_h_decrypt)
  // 5. Defragment fragmented frames
  CALL_RXH(ieee80211_rx_h_defragment);
  // 6. Check authenticity and report possible MIC failures
  CALL_RXH(ieee80211_rx_h_michael_mic_verify)
}
```

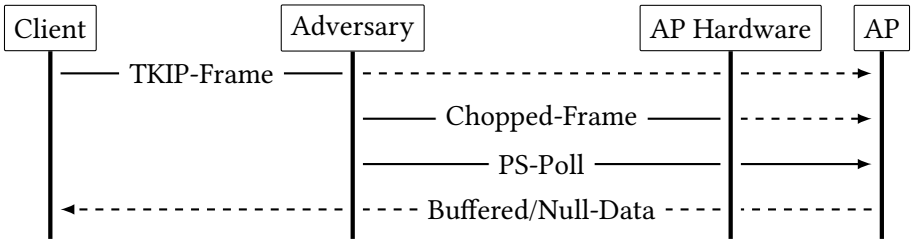

Fig. 6. Abusing power management features of Linux's Wi-Fi stack, in combination with hardware decryption, to determine whether the ICV of the TKIP frame is correct. When the AP replies using a Null-Data frame, the ICV is correct. If the AP does not reply, the ICV is wrong.

in software after, instead of before, inspecting the power management flags of received frames. This different order of operations in newer kernels effectively mitigated our attack. However, with hardware decryption, the Wi-Fi chip again decrypts frames before the operating system receives and inspects the power management flags. More concretely, Listing 2 shows the actions that a received frame undergoes on Linux 3.12 and above, assuming that hardware decryption is enabled. From Listing 2 we learn that, if the driver drops frames for which hardware decryption failed, the power management flags are inspected only if hardware decryption succeeds. That is, if hardware decryption is used in combination with Linux kernel 3.12 or above, and the driver drops frames when hardware decryption fails, frames with an invalid TKIP ICV do not have their management flags inspected. This enables us to perform the same side-channel attack as in Section 4.3 against new Linux kernels. Summarized, we chop the last byte from a packet, guess its value, correct the ICV, and detect a correct guess by seeing if the AP replies to a PS-Poll packet. Figure 6 shows a simplified illustration of the attack method.

However, not all drivers drop the frame if hardware decryption failed. For example, we inspected the source code of 17 widely-used Linux drivers, and found that 10 of them do not drop frames if hardware decryption failed. Instead, these drivers forward the original encrypted frame to the Wi-Fi subsystem

of the operating system. This is done so the kernel can try to decrypt the frame itself, which can be useful if the hardware used an outdated or incorrect key. However, with other Wi-Fi chips the original frame is lost after decryption, because the chip decrypts the frame in-place. As a result, the driver has no other option than to drop the frame when hardware decryption failed. Finally, we remark that the 802.11 standard does not explicitly state whether an implementation should first decrypt a frame, or first inspect the power management flags. In other words, this behavior is implementation-dependent. We evaluated the attack against an Intel AC 8265 that was running Arch Linux with kernel version 4.20. This confirmed the Intel driver dropped frames for which hardware decryption failed. Our experiments also confirmed that under this condition the power-management side-channel attack remains possible against new Linux kernels. Inspecting the source code of the Intel driver revealed that the driver is forced to drop the frame because the Wi-Fi chip decrypts the frame in-place, meaning the original frame is lost if decryption failed. On average, it took around 4 to 6 minutes to decrypt the Michael MIC and ICV, and to subsequently recover the MIC key. The main reason this attack is slower than the original power-management attack of Section 4.3, is because with the Intel AC 8265 card, the AP generated a high amount of retransmissions. This meant that, when guessing the value of the chopped off byte, we could only verify 10 guesses every second in a reliable manner.

5.3 Linux Retransmission Detection

As shown in step 2 of Listing 2, the Wi-Fi subsystem of Linux also filters retransmitted (plaintext) frames in case the Wi-Fi chip did not do so already. This is accomplished by storing the sequence number of the previously received frame in the variable `last_seq_ctrl`, and comparing it to the sequence number of incoming frames. In case these sequence numbers are equal, a retransmission is detected, and the incoming frame is dropped. More importantly, this means that every frame processed by the kernel's Wi-Fi subsystem influences the state of the kernel. In particular, each frame causes the variable `last_seq_ctrl` to be updated. When combined with hardware decryption, and assuming the driver drops frames for which decryption failed, this means only TKIP frames with a correct ICV cause `last_seq_ctrl` to be modified. Therefore, if we can detect whether `last_seq_ctrl` has a specific (modified) value, we obtain a new side-channel vulnerability that can determine whether the ICV of a TKIP frame is correct. To test whether `last_seq_ctrl` equals x, we inject a Security Association (SA) query request with a sequence number equal to x, and with the retry bit set in the MAC header. If `last_seq_ctrl` indeed equals x, the frame is treated as a retransmitted one, and therefore is silently dropped. However, if `last_seq_ctrl` does not equal x, the client or AP will reply using a SA query response. Note that the SA query handshake is part of the Management Frame Protection (MFP) standard, which has been supported by Linux since kernel 2.30, and hence we can safely assume it is supported by all clients and access points. Additionally, Linux replies to SA query requests even if management frame protection is not used in the network.

One complication is that on Linux, the user space daemon that manages AP functionality and replies to SA queries, e.g. hostapd, may also be filtering retransmitted frames. This might cause complications when the kernel thinks a frame is new, but the user space daemon still treats it as a retransmitted frame. To avoid this situation from arising, we first let the adversary inject a SA query request with a fresh sequence counter different from x. This assures that, when later on in the attack the kernel's

last_seq_ctrl does *not* equal x, the user space daemon will indeed reply to the SA query request. Otherwise, even if last_seq_ctrl does not equal x, hostapd may still treat it as a retransmission and hence ignore it. All combined, to detect whether the ICV of a TKIP frame is correct, an adversary must perform the following actions:

(1) Inject a SA query response with a fresh sequence number different than x. This causes the user space daemon to update its last received sequence number of SA query frames.
(2) Inject a probe response with a sequence number of x. This sets the kernel's last_seq_ctrl variable to x, without affecting the user space daemon.
(3) Inject the TKIP frame using a sequence number different than x. Note that this sequence number is unauthenticated, and is different from the TKIP Sequence Counter (TSC) that is used for detecting malicious replays (recall Fig. 1). Hence an adversary can modify the sequence number of a TKIP frame without affecting its decryption.
(4) Inject a SA query request with a sequence number of x.

If the adversary receives a SA query response after injecting these frames, the injected TKIP frame had a correct ICV. This is because the SA query request is only accepted if the kernel's last_seq_ctrl variable differs from x. And this is only the case when the TKIP frame has a valid ICV, and therefore overwrote the last_seq_ctrl variable.

A second obstacle we encountered, is that Linux tracks the last received sequence number for each Quality of Service Traffic Identifier (QoS TID) separately. In other words, for a frame to be treated as a retransmission, it must use the same sequence counter as the previously received frame for the given QoS TID. However, the SA query frame that we use to detect if last_seq_ctrl was modified, is a management frame, and therefore does not support separate QoS TIDs. Instead, a SA query frame only supports a single default QoS TID. To remedy this problem, we inject the TKIP frame without a QoS header. This causes both the TKIP frame and the SA query frame to use the same default QoS TID. Unfortunately, stripping the QoS header from the TKIP frame introduces another problem. Namely, the adversary can no longer abuse the QoS TID to bypass the TKIP replay check (recall that each QoS TID has its own receive replay counter). Fortunately, we discovered that practically all Wi-Fi chips that support hardware decryption, do not perform replay detection in hardware. Instead, the Wi-Fi chips only performs decryption and authentication, but replay detection is left to the driver or operating system. For example, in Listing 2, replays are detected by the Wi-Fi subsystem of Linux in the h_decrypt handler. In other words, an adversary can simply inject TKIP frames using an old replay counter, and Linux will only realize this is a replayed frame in the h_decrypt handler. As a result, retransmissions are detected in the h_check_dup handler before TKIP replay detection. This means a replayed TKIP frame with a correct ICV will influence the last_seq_ctrl variable, but a TKIP frame with an incorrect ICV will not affect last_seq_ctrl.

We also remark that the authenticity of a TKIP frame is verified in the h_michael_mic_verify handler, which is executed only if the frame passed the TKIP replay detection in h_decrypt. This implies that usage of an old replay counter also avoids the generation of MIC failures, meaning we no longer have to rely on fragmented frames to suppress MIC failures.

We evaluated the attack in practice using an Intel AC 8265 as the victim, which was running Arch Linux with kernel 4.20. To inject frames, one must use a Wi-Fi card that allows us to control the sequence number of injected frames. In our experiments, we used the AWUS051NH v2 dongle. Note that other Wi-Fi cards may overwrite the sequence number of injected frames, causing the attack to fail. For the AP we used a TL-WN722N. During our experiments, we observed that our side-channel sometimes incorrectly reports a TKIP ICV as being correct. To avoid such false positives, we only treat a frame as having a correct ICV if the side-channel reports a positive result twice. Using this setup, we successfully performed the attack and decrypted the ICV and Michael MIC of a sniffed TKIP frame, to subsequently recover the MIC key. When testing 10 guesses for the chopped byte every second, the attack took on average 2 to 4 minutes. Sending more than 10 guesses every second makes the side-channel unreliable, because too many frames have to be injected in a small time-frame.

Against Linux kernel 3.6.11.8 and below, our side-channel attack even works against broadcast and multicast frames (i.e. group-addressed frames that are encrypted using the group key). This means that even a client which uses CCMP as the pairwise ciphers, can still be attacked if the network uses TKIP as the group cipher. Since nearly half of all protected Wi-Fi networks still use TKIP as the group cipher, this has a significant practical impact. On newer Linux kernels however, the sequence number field in group-addressed frames is not inspected [7]. As a result, these implementations can only be attacked if they use TKIP as the pairwise cipher, in which case all Linux versions are vulnerable. Finally, another advantage of this side-channel is that it works against both clients and APs.

5.4 OpenBSD BlockAck Side-Channel

On OpenBSD, several Wi-Fi drivers utilize the hardware decryption capabilities of Wi-Fi chips as well. To investigate whether this introduces side-channels, we also audited the OpenBSD source code for detectable side-effects that indicate whether a frame was forwarded to its Wi-Fi subsystem. Recall that if a driver only forwards frames when hardware decryption succeeded, any noticeable side-effect of forwarding the frame results in an exploitable side-channel. Surprisingly, we quickly discovered a technique to determine if a frame was forwarded to the Wi-Fi subsystem of the operating system. This technique is based on the Block Acknowledgement (BA) feature of 802.11e [2], which allows a station (i.e. a client or AP) to acknowledge multiple QoS data frames at once. To use this feature, a BA agreement must first be established. Once this is done, the transmitter can request the receiver to acknowledge multiple QoS data frames at once. This request is made by setting the Block Ack flag in each frame's QoS header. However, if no BA agreement was negotiated, and the Block Ack flag is set in a frame's QoS header, OpenBSD will transmit an action frame instructing the sender to first establish a BA agreement. Importantly, the QoS flags of data frames are not authenticated. Therefore, an adversary can modify the QoS header without affecting the decryption of the frame. As a result, an adversary can use the following steps to determine if the ICV of a TKIP frame is correct:

(1) If not already present, include a QoS header in the TKIP frame, and set a QoS TID having a low replay counter.
(2) Set the Block Ack flag in the QoS header, and mark the frame as being fragmented. Send the resulting frame to the victim.

(3) If the victim replies with an action frame, the ICV of the frame was correct. If there is no reply, the ICV was wrong.

Note that the frame will be processed because we are injecting the frame using a QoS TID that has a lower replay counter than the original or default QoS TID of the frame. In case the TKIP ICV is correct, hardware decryption will succeed, and the driver will forward the frame to the Wi-Fi subsystem. As a result, OpenBSD will inspect the QoS header, and will notice that the Block Ack flag is set, without there being a BA agreement. This causes OpenBSD to transmit an action frame with an error status code. On the other hand, if the TKIP ICV is wrong, and the driver drops frames for which hardware decryption failed, OpenBSD will silently ignore the frame. Finally, MIC failures during hardware decryption are avoided by marking the frame as being fragmented. This causes the hardware to skip Michael authentication. Additionally, the OpenBSD kernel will also not try to check the Michael MIC, because the frame is almost immediately dropped due to the invalid Block Ack flag.

We tested the attack against OpenBSD 6.4, where the victim used an AWUS051NH v2 Wi-Fi dongle. We chose this dongle because it is often recommended for its high reception [4], and because it supports hardware decryption on OpenBSD. For the attacker we used a TL-WN722N Wi-Fi dongle with Arch Linux as the operating system. To inject frames with a modified QoS Ack Policy field, we had to patch the Linux kernel so it would not overwrite the QoS header when injecting frames. With this modification, we were able to successfully perform the side-channel attack against OpenBSD. On average, it took about 2 to 4 minutes to decrypt the ICV and Michael MIC, and subsequently recover the MIC key. Our side-channel works against both unicast Wi-Fi frames (which are encrypted with the session key) and against group-addressed Wi-Fi frames. Similar to the retransmission side-channel against Linux, this means the attack even works against modern clients that support CCMP. After all, if the network uses TKIP as the group cipher, all clients must use TKIP to decrypt group-addressed frames.

5.5 Broadcom Code Inspection

During our analysis, we also inspected the source code of Broadcom-based routers. In particular, we investigated leaked source code of the RT-AC86U, which internally uses a Broadcom Wi-Fi chip. This code inspection indicated that the driver drops frames for which hardware decryption failed. Since the router uses Linux, we conjecture that it is affected by both the power management side-channel of Section 5.2, and the retransmission detection side-channel of Section 5.3.

5.6 OpenBSD Fragment and Replay Attack

While inspecting the source code of OpenBSD, we noticed that it did not contain code to handle fragmented Wi-Fi frames. Although this seems harmless, not handling fragmented frames introduces a surprising vulnerability when combined with hardware decryption. When hardware decryption is enabled, the Wi-Fi chip will decrypt fragmented TKIP frames. However, because the frame is fragmented, the authenticity of the frame cannot yet be checked. This is because the Michael MIC is not calculated over individual fragments, but only over the full frame. Therefore, the Wi-Fi chip forwards decrypted fragments to the operating systems. It is then the responsibility of the operating system to reassemble all fragments, and subsequently authenticate the reassembled frame. Instead of reassembling fragmented

frames, OpenBSD treats individual fragments as full frames, and will not verify its authenticity. An adversary can abuse this to inject arbitrary packets as follows:

(1) Derive some known keystream, by predicting the content of encrypted packets (e.g. based on unique lengths), and xoring the captured ciphertext with the predicted plaintext.
(2) Append a CRC to the packet to be injected, and encrypt it using the derived keystream.
(3) Mark the frame as being fragmented, and inject it towards the OpenBSD victim.

When an OpenBSD victim receives the injected frame, the Wi-Fi chip will decrypt the fragment, and verify the ICV. Since the ICV is correct, it will be stripped from the fragment, and the remaining content of the frame is forwarded to the OpenBSD operating system. The Wi-Fi subsystem of OpenBSD will then treat the fragment as a full frame. Moreover, it will assume the hardware already verified the authenticity of the frame, while in reality only the ICV was verified. As a result, the adversary successfully injected the packet.

Rather worryingly, we also discovered a second vulnerability in how OpenBSD handles hardware-decrypted frames. In particular, we found that OpenBSD does not perform replay detection when using hardware encryption. The reason this happens, is because the Wi-Fi chips only decrypts the frame, but does not check for replays. Instead, it remains the responsibility of the operating system to check for replays. Unfortunately, OpenBSD does not perform replay checks when the frame got decrypted by hardware. Against both TKIP and CCMP, this enables an adversary to trivially replay frames. Moreover, by combining this with the TKIP fragmentation bug explained in the previous paragraph, an adversary can use a single keystream to inject an infinite amount of packets.

We tested our attacks against OpenBSD 6.4 using an Intel AC 8265 card, and an AWUS051NH dongle. When the victim was using the AWUS051NH, we could replay both TKIP and CCMP frames. When the victim was using the Intel card, we could only replay CCMP frames. Replaying TKIP frames against the Intel card was not possible because it only uses hardware decryption for CCMP. Against the AWUS051NH we also confirmed that capturing a single TKIP keystream can be used to inject an infinite number of packets. Finally, we observed that when using software decryption, the Wi-Fi subsystem of OpenBSD will attempt to verify the Michael MIC of individual TKIP fragments. This will result in MIC failures, since individual TKIP frames do not have a valid Michael MIC. As a result, if a legitimate client sends fragmented TKIP frames to OpenBSD, this will cause a self-inflicted denial-of-service.

6 DISCUSSION

The side-channel attacks discussed in this paper are significant improvements over previously known attacks [18, 26, 27, 29]. Our attacks (i) do not rely on Michael MIC Failure Reports, (ii) bypass countermeasures, and (iii) attack APs in addition to the clients. Attacking the AP has as an advantage that keystream can be more easily generated. That is, after recovering a client-to-AP Michael MIC key, an adversary can inject traffic into the network and trigger, for example, an encrypted ping reply from another target client. Furthermore, several of our attacks target multicast and broadcast traffic, meaning even modern AES-CCMP clients can be attacked in a network supporting TKIP. In addition to being able to decrypt network traffic, an adversary is also able to inject forged messages. As a result of the above, the impact on networks supporting WPA-TKIP increases significantly.

Injecting Network Traffic. The presented attacks allow an adversary to recover a keystream and the Michael MIC key. With this information, an adversary can inject network traffic [26, 29]. A message is forged by calculating the Michael MIC using the recovered key, calculating the ICV as a CRC-32, and encrypting the result using the recovered keystream. The length of the forged message can be at most the length of the recovered keystream. The forged message can now be injected onto a QoS channel having a lower TSC value. In practice, most traffic is transmitted on channel 0, leaving 7 additional channels for messages to be injected on [26]. From this point forward, an adversary can recover additional keystream much faster since the Michael MIC key has been recovered, and only the last 4-bytes containing the ICV need to be decrypted.

Decrypting Network Traffic. The Michael MIC key can also be used to decrypt frames [29]. This is accomplished by sending a ping-style packet towards the victim, and using the Michael reset attack to append an encrypted TKIP frame to the ping request [29]. We spoof the source IP of this ping request, so the victim will send the reply to a server under our control. The client will now echo the content of the ping request, which equals the TKIP frame, meaning we effectively decrypted the frame.

Fortunately, our side-channels have limited impact on CTR mode with CBC-MAC Protocol (CCMP). CCMP was introduced as a more robust security protocol replacing WPA-TKIP. CCMP uses the counter mode in the AES block cipher for data confidentiality, and CBC-MAC for the generation of a MIC providing data integrity and data origin authentication. In relation to our side-channel attacks, there are three notable differences between CCMP and TKIP. Recall from background Section 2 that large MSDUs are fragmented into smaller MPDUs. First, in the CCMP protocol, a MIC value is calculated for every MPDU, unlike TKIP, which attaches its Michael MIC only to the last fragment or MPDU. Second, unlike a TKIP MPDU, the CCMP MPDU has no ICV. Third, unlike the Michael MIC in TKIP, the MIC attached to a CCMP MPDU is not known to be invertible. As a result of these differences, each decrypted CCMP MPDU is immediately verified for data integrity and data origin authentication, without the need to wait for all fragments to arrive. In contrast, decrypted TKIP fragments are processed before being reassembled, and influence power save management and the fragmentation cache, without being authenticated using its MIC. To summarize, our side-channel attacks work against TKIP because operations are performed on decrypted but unauthenticated data.

7 RELATED WORK

The first practical attack against WPA-TKIP was found in 2009 by Tews and Beck [26], and abused MIC failure reports as an oracle to decrypt frames. Over the years, several improvements have been made to it, lowering restrictions on the attack and making it faster. Ohigashi and Morii proposed an improvement where IEEE 802.11e QoS features are no longer required [18]. Todo et al. proposed two improved methods, QoS forgery and reverse chopping, reducing the time it takes an adversary to recover keystream and the Michael MIC key to 7-8 minutes [27]. Vanhoef and Piessens have demonstrated a fragmentation attack and introduced the Michael reset attack, allowing an adversary to inject an arbitrary amount of packets [29]. In contrast to these attacks, our side-channel attacks do not rely on the transmission of Michael MIC failure reports and are therefore (i) able to attack the AP in addition to the clients, (ii) significantly faster to execute, and (iii) resistant against the proposed countermeasures related to sending Michael MIC failure reports. Apart from abusing MIC failure reports, other works

identified weaknesses in how TKIP uses the RC4 encryption algorithm [5, 19]. Although certainly a security issue, the attacks are not as practical as the side-channel attacks we presented in this paper.

In 2005, a correctness proof for the 4-way and group key handshake was presented in [13]. Unfortunately, the 4-way handshake was still vulnerable to key reinstallation attacks (KRACKs) [30, 31]. In addition, researchers identified vulnerabilities in technology surrounding Wi-Fi, such as Wi-Fi Protected Setup (WPS) [22, 33]. In [10], message forging attacks against AES-CCMP were described, and several WPA2 implementations have been analyzed in works such as [25, 32] revealing how clients may be downgraded from AES-CCMP to WPA-TKIP. Downgrading clients to WPA-TKIP makes them vulnerable to the attacks devised against it. These works show that designing and implementing secure Wi-Fi protocols is a hard task, and requires care in order to prevent (side-channel) attacks.

8 CONCLUSION

We systematically analyzed the security of several WPA-TKIP implementations, and were able to devise side-channel attacks against all of them. Our side-channel attacks bypass all existing countermeasures, and allow an adversary to decrypt and inject network traffic in a matter of minutes, making them much faster than previously known attacks. This is problematic, given that nearly half of all encrypted networks still provide support for WPA-TKIP. These results show that implementing WPA-TKIP without side-channel vulnerabilities is a hard task to perform, and we conjecture that all implementation are vulnerable to (possibly novel) side-channel vulnerabilities. We hope our results will help accelerate the process of completely deprecating support for WPA-TKIP, and encourage everyone to use more secure solutions such as WPA3.

REFERENCES

[1] IEEE Std 802.11. 2012. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Spec.*

[2] IEEE Std 802.11e. 2005. *Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements.*

[3] IEEE Std 802.11i. 2004. *Amendment 6: Medium Access Control (MAC) Security Enhancements.*

[4] Aircrack-ng. 2019. FAQ: What is the best wireless card to buy? Retrieved 19 January 2019 from https://www.aircrack-ng.org/doku.php?id=faq.

[5] Nadhem AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. 2013. *On the Security of RC4 in TLS and WPA.* Technical Report.

[6] Wi-Fi Alliance. 2015. Technical Note: Removal of TKIP from Wi-Fi Devices.

[7] Johannes Berg. 2013. [138/251] mac80211: fix duplicate retransmission detection. Retrieved 19 January 2019 from https://lore.kernel.org/patchwork/patch/405688/.

[8] Niels Ferguson. 2002. Michael: an improved MIC for 802.11 WEP. *IEEE doc* 802, 2 (2002).

[9] Scott Fluhrer, Itsik Mantin, and Adi Shamir. 2001. Weaknesses in the key scheduling algorithm of RC4. In *SAC*. Springer, 1–24.

[10] Pierre-Alain Fouque, Gwenaëlle Martinet, Frédéric Valette, and Sébastien Zimmer. 2008. On the Security of the CCM Encryption Mode and of a Slight Variant. In *International Conference on Applied Cryptography and Network Security*. Springer.

[11] Mouhcine Guennoun, Aboubakr Lbekkouri, Amine Benamrane, Mohamed Ben-Tahir, and Khalil El-Khatib. 2008. Wireless networks security: Proof of chopchop attack. In *WoWMoM*. IEEE.

[12] Dan Harkins and Warren Kumari. 2017. Opportunistic Wireless Encryption. RFC 8110. https://doi.org/10.17487/RFC8110

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

[14] Jianyong Huang, Jennifer Seberry, Willy Susilo, and Martin Bunder. 2005. Security analysis of michael: the IEEE 802.11 i message integrity code. In *IEEE EUC*.
[15] KoreK. 2004. chopchop (Experimental WEP attacks). Retrieved 21 January 2019 from http://www.netstumbler.org/unix-linux/chopchop-experimental-wep-attacks-t12489.html.
[16] Jouni Malinen. 2008. ChangeLog for wpa_supplicant v0.6.6. Retrieved 19 January 2019 from https://w1.fi/cgit/hostap/plain/wpa_supplicant/ChangeLog.
[17] Masakatu Morii and Yosuke Todo. 2011. Cryptanalysis for rc4 and breaking wep/wpa-tkip. *IEICE Trans. on Inf. and Systems* 94, 11 (2011), 2087–2094.
[18] Toshihiro Ohigashi and Masakatu Morii. 2009. A practical message falsification attack on WPA. *Proc. JWIS* (2009).
[19] Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. 2014. Big Bias Hunting in Amazonia: Large-Scale Computation and Exploitation of RC4 Biases (Invited Paper). In *ASIACRYPT*. 398–419.
[20] Jon Rosdahl. 2017. Minutes REVmd – July 2017 Berlin. Retrieved January 21, 2019 from https://mentor.ieee.org/802.11/dcn/17/11-17-0857-01-000m-minutes-revmd-july-2017-berlin.docx
[21] Jon Rosdahl, Mark Hamilton, and Michael Montemurro. 2018. Minutes REVmd – May 2018 – Warsaw. Retrieved January 10, 2019 from https://mentor.ieee.org/802.11/dcn/18/11-18-0616-00-000m-minutes-revmd-may-2018-warsaw.docx
[22] Amirali Sanatinia, Sashank Narain, and Guevara Noubir. 2013. Wireless spreading of WiFi APs infections using WPS flaws: An epidemiological and experimental study. In *IEEE CNS*. 430–437.
[23] Domien Schepers, Aanjhan Ranganathan, and Mathy Vanhoef. 2019. Practical Side-Channel Attacks against WPA-TKIP. In *Proceedings of the 2019 ACM on Asia Conference on Computer and Communications Security*. ACM.
[24] Graham Smith. 2018. Resolution for WEP/TKIP removal CIDs. Retrieved January 21, 2019 from https://mentor.ieee.org/802.11/dcn/18/11-18-0652-01-000m-resolution-for-wep-tkip-removal-cids.docx
[25] Chris McMahon Stone, Tom Chothia, and Joeri de Ruiter. 2018. Extending Automated Protocol State Learning for the 802.11 4-Way Handshake. In *European Symposium on Research in Computer Security*. Springer, 325–345.
[26] Erik Tews and Martin Beck. 2009. Practical attacks against WEP and WPA. In *Proceedings of the second ACM conference on Wireless network security*. ACM.
[27] Yosuke Todo, Yuki Ozawa, Toshihiro Ohigashi, and Masakatu Morii. 2012. Falsification attacks against WPA-TKIP in a realistic environment. *IEICE TRANSACTIONS on Information and Systems* 95, 2 (2012), 588–595.
[28] Mathy Vanhoef. 2016. *A Security Analysis of the WPA-TKIP and TLS Security Protocols*. Ph.D. Dissertation. KU Leuven.
[29] Mathy Vanhoef and Frank Piessens. 2013. Practical verification of WPA-TKIP vulnerabilities. In *AsiaCCS*. ACM, 427–436.
[30] Mathy Vanhoef and Frank Piessens. 2017. Key reinstallation attacks: Forcing nonce reuse in WPA2. In *CCS*. ACM, 1313–1328.
[31] Mathy Vanhoef and Frank Piessens. 2018. Release the Kraken: New KRACKs in the 802.11 Standard. In *CCS*. ACM, 299–314.
[32] Mathy Vanhoef, Domien Schepers, and Frank Piessens. 2017. Discovering logical vulnerabilities in the Wi-Fi handshake using model-based testing. In *ACSAC*.
[33] Stefan Viehböck. 2011. Wi-Fi protected setup pin brute force vulnerability. *CERT Vulnerability Note VU* 723755 (2011).
[34] Wi-Fi Alliance. 2015. Technical Note: Removal of TKIP from Wi-Fi Devices. Retrieved 8 January 2019 from https://www.wi-fi.org/downloads-public/Wi-Fi_Alliance_Technical_Note_TKIP_v1.0.pdf/17196.
[35] Wi-Fi Alliance. 2018. WPA3 Specification Version 1.0. Retrieved 8 January 2019 from https://www.wi-fi.org/file/wpa3-specification-v10.
[36] Avishai Wool. 2004. A note on the fragility of the "Michael" message integrity code. *IEEE Transactions on Wireless Communications* 3, 5 (2004), 1459–1462.