Practical Verification of WPA-TKIP Vulnerabilities

Mathy Vanhoef
iMinds-DistriNet
KU Leuven
Mathy.Vanhoef@cs.kuleuven.be

Frank Piessens
iMinds-DistriNet
KU Leuven
Frank.Piessens@cs.kuleuven.be

ABSTRACT

We describe three attacks on the Wi-Fi Protected Access Temporal Key Integrity Protocol (WPA-TKIP). The first attack is a Denial of Service attack that can be executed by injecting only two frames every minute. The second attack demonstrates how fragmentation of 802.11 frames can be used to inject an arbitrary amount of packets, and we show that this can be used to perform a portscan on any client. The third attack enables an attacker to reset the internal state of the Michael algorithm. We show that this can be used to efficiently decrypt arbitrary packets sent towards a client. We also report on implementation vulnerabilities discovered in some wireless devices. Finally we demonstrate that our attacks can be executed in realistic environments.

Categories and Subject Descriptors

E.3 [**Data Encryption**]: Code breaking; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms

Security, Experimentation, Verification

Keywords

802.11; WPA; TKIP; DoS; fragmentation; decryption; driver vulnerabilities

1. INTRODUCTION

Modern wireless networks are based on the IEEE 802.11 set of standards. These networks have gained popularity over the years and are nowadays widely used in different scenarios, ranging from personal use to high-profile commercial use. Because of the nature of wireless transmission special care must be taken to preserve the privacy and security of wireless networks. The original IEEE 802.11 standard supported a basic security algorithm called Wired Equivalent Privacy (WEP). Unfortunately WEP suffers from major design flaws and is considered completely broken [8, 19, 4].

An improvement of WEP is the Temporal Key Integrity Protocol (TKIP). Created as an intermediate protocol to the more secure CCMP, it was designed to run on existing WEP hardware [14, §11.4.1]. This affected many of the design decisions [14, 6]. Most notably it still uses WEP encapsulation and relies on a weak Message Integrity Check (MIC) algorithm called Michael [6]. Because the Michael algorithm provides inadequate security [20, 13, 22] countermeasures were added. TKIP and its countermeasures are explained in detail in Sect. 2.

Surprisingly, TKIP is still supported by a large number of networks. In Section 6.1 we report on an experiment where we collected information about wireless network usage in two Belgian municipalities and found that 71% of encrypted networks support TKIP. Furthermore, 19% of networks using encryption only allow TKIP.

In this paper we present a novel Denial of Service (DoS) attack on TKIP. Moreover, we take two ideas suggested in a paper by Beck [1] and significantly improve on them. In contrast with the paper of Beck, our improvements are also implemented and tested in practice. The first idea applies the known fragmentation attack on WEP [4] to TKIP. This allows an attacker to send an arbitrary amount of packets to a client. As a proof of concept we implemented a port scanner. The second idea is to construct a prefix that resets the internal state of the Michael algorithm. We will show that this enables us to efficiently decrypt arbitrary packets sent towards a client. We also report on several vulnerabilities found in the implementation of some wireless adapters and drivers. All attacks are designed against WiFi networks operating in infrastructure mode, and are tested when authentication is done using a passphrase and when using a personal username and password.

We hope that the publication of these novel attacks will motivate people to disable TKIP.

The remainder of this paper is organised as follows. Section 2 describes the details of the TKIP protocol. In Section 3 we explain the Denial of Service (DoS) attack. Section 4 discusses our fragmentation and portscan attack. In Section 5 the Michael state reset and decryption attack is explained. Section 6 investigates whether TKIP is still supported in practice and discusses experimental evaluation of our attacks. Finally, we summarise related work in Sect. 7 and conclude in Sect. 8.

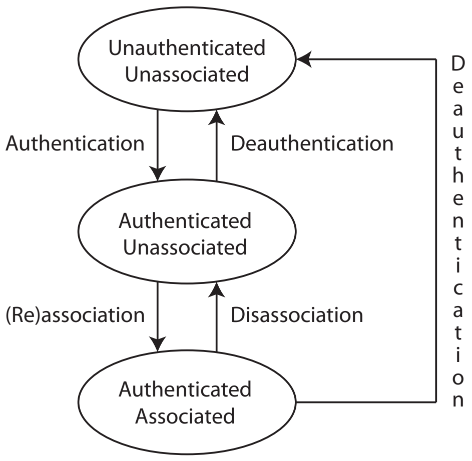

Figure 1: States a client can be in when connecting to a wireless network.

2. TEMPORAL KEY INTEGRITY PROTOCOL (TKIP)

This section describes the relevant parts of the IEEE 802.11 standard with a focus on the TKIP specification [14, §11.4.2]. We will also explain one of the first attacks on TKIP, called the Beck and Tews attack.

2.1 Connecting

A client connects to a wireless network by first authenticating and then associating with the Access Point. A state diagram of this process is shown in Fig. 1. There are two authentication methods. The first one is called Shared Key authentication and was based on WEP. Unfortunately this method is inherently insecure. Nowadays only the second method is used, called Open System authentication. As the name implies it imposes no real authentication. It is essentially just a formality, and if TKIP or CCMP is used actual authentication will happen at a later stage. Once authenticated, the client sends an association request to the AP. This request includes the secure authentication and encryption protocol it wants to use. If the AP supports the requested protocols the association is successful, and the AP informs the client that the association has completed.

Once authenticated and associated, a 4-way handshake is performed when using TKIP. The handshake negotiates the keys used by TKIP and is defined using IEEE 802.1X EAPOL-Key frames. This results in a 512-bit pairwise transient key (PTK) that is shared between the AP and client. From the PTK a 128-bit temporal encryption key (TK) is derived, as well as two 64-bit Message Integrity Check (MIC) keys: one for AP to client communication and one for client to AP communication. These keys are renewed after a user defined interval, commonly called the rekeying timeout. Most APs by default use a timeout of 1 hour. After a key has been negotiated, the client and AP can send encrypted data frames to each other.

The client or AP can end the connection at any time by sending a disassociation or deauthentication message. The older versions of the 802.11 standard left these messages unprotected. This means an attacker can forge them and forcibly close the connection between an AP and client.

6	6	1	3	*var*
Destination	Source	Priority	Reserved	Payload

Figure 2: Input data given to the Michael algorithm. Destination and source represent MAC addresses. If the QoS extension is not used, priority is set to zero. Numbers denote the size of the field in bytes, where *var* defines a variable-length field.

	8	*var*	8	4
802.11 header	TSC	Data	MIC	ICV
⟵ unencrypted ⟶		⟵ encrypted ⟶		

Figure 3: Simplified format of an unfragmented TKIP frame.

Continuously injecting such forged deauthentication packets causes a DoS attack [18]. This attack is well known and is called the deauthentication attack. It can be prevented by enabling protected management frames, a feature introduced in the IEEE 802.11w amendment [14, §4.5.4.9].

2.2 Sender

When sending a TKIP frame first the MIC value of the MAC Service Data Unit (MSDU) is calculated. The purpose of the MIC is to protect both the integrity and authenticity of the message. Recall that the MSDU is essentially the complete data packet that needs to be transmitted, which is fragmented into smaller MAC Protocol Data Units (MPDU) fragments if it is too big to be sent at once. The MIC is calculated over the MSDU by the Michael algorithm. Michael is a keyed hash function taking two inputs: the input data to calculate the MIC of and a secret key. The input data given to the Michael algorithm is shown in Fig. 2. For calculating the MIC a different secret key is used for AP to client communication than is used for client to AP communication. Both MIC keys are derived from the PTK. The calculated MIC value is 8 bytes long. However, the Michael algorithm is not sufficiently secure. In particular it is possible to efficiently retrieve the MIC key given the data and calculated MIC value [20]. The designers realized this and included countermeasures in an attempt to mitigate potential attacks. These countermeasures are essential to our attacks and are discussed in detail in Sect. 2.3.

The MSDU concatenated with the MIC is fragmented into MAC Protocol Data Units (MPDUs) if necessary. At most 16 fragments (MPDUs) are supported. Each MPDU then undergoes WEP encapsulation. This is done so TKIP can be implemented on old WEP hardware. WEP encapsulation appends an Integrity Check Value (ICV) to the MPDU, which is simply a 32 bit CRC computed over the given data. Then it encrypts the packet using the RC4 stream cipher. The key used for encryption is called the WEP seed and is calculated by a mixing function that combines the temporal key (TK), transmitter MAC address, and the TKIP Sequence Counter (TSC). Figure 3 illustrates the final layout of an unfragmented TKIP frame.

Finally the resulting link-layer frame is constructed by adding the appropriate 802.11 headers. This includes the TKIP Sequence Counter (TSC), which is a replay counter that increases every time a MPDU frame is sent successfully.

In Fig. 3 the TSC is given a size of 8 bytes, but in reality the TSC is only 6 bytes long. The remaining bits of the field are used for other purposes or are reserved. To prevent replay attacks the receiver drops frames that are not received in order. That is, the counter must always be increasing though gaps are allowed.

2.3 Receiver

When receiving a TKIP frame the client or AP first checks if the TSC is in order. If not, the frame is silently dropped. It then proceeds by checking if the ICV is correct. If not, the frame is also silently dropped. Once all MPDU's are received they are reassembled into the original MSDU and its MIC value is verified. If it is correct the frame is accepted and the receiver updates its TSC replay counter, otherwise the TKIP countermeasures kick in. These countermeasures were added to detect active attacks against the weak Michael algorithm [14, §11.4.2.4.1]. One class of such active attacks involves injecting multiple forged packets in the hope at least one of them has a valid MIC. In case such a packet is found the attacker learns the MIC key for the particular communication direction, as the MIC key can be derived when given the plaintext data and calculated MIC value.

The countermeasures are as follows [14, §11.4.2.4]:

- When a client receives a MSDU with an invalid MIC value it will send a MIC failure report to the AP.
- An AP receiving an invalid MIC value does not broadcast a MIC failure report but only logs the failure.
- If the AP detects two MIC failures within one minute all TKIP clients connected to the AP will be deauthenticated and the AP will not receive or transmit any TKIP-encrypted data frames for one minute. Once this minute is passed clients can reassociate with the AP and negotiate a new PTK.

2.4 Quality of Service Extension

Quality of Service (QoS) enhancements for wireless traffic were first defined in the IEEE 802.11e amendment. Most modern APs support this amendment [17]. It defines 8 different channels, each having their own QoS needs. A channel is defined by its Traffic Identifier (TID) and is internally represented by 4 bits, making some devices actually support 16 different channels. For the implementation of our attacks we assume only 2 QoS channels exist, though devices supporting more channels are also susceptible to our attacks. Tests showed that our assumption holds in practice. Additionally we note that by default most traffic is sent over the first QoS channel.

Essential for us is that each QoS channel has a separate TSC [14, §11.4.2.6]. This means that we can capture a packet transmitted on one QoS channel and replay it on another QoS channel having a lower TSC value. This enables an attacker to pass the TSC check, though he still has to pass the ICV and MIC checks in order to forge a message.

2.5 Beck and Tews Attack

One of the first known attacks on TKIP was discovered by Beck and Tews [20]. It was a variation of the chopchop attack on WEP [10] and works by decrypting a packet one byte at the time. Because this attack is used as the basis for the fragmentation and Michael reset attack it will be explained in detail.

First we will explain the chopchop attack when applied to a TKIP packet. It begins by taking an encrypted packet and removing the last byte. Let C denote the obtained shortened encrypted packet. With high probability the ICV of C is invalid. However, it can be corrected if one knows the plaintext value of the removed byte. Correcting the ICV of the unencrypted shortened message can be represented by $M' = M \oplus D$, where M is the unencrypted shortened message, D is the correction being applied, $\oplus$ denotes the XOR operator, and M' is the unencrypted shortened message with a valid CRC. It has been proven that D only depends on the plaintext value of the removed byte [10]. Interestingly we can apply this modification directly on C. Letting K denote the keystream used to encrypt the packet we get that $C = M \oplus K$. We can now make the following derivation:

$$\begin{aligned} C' = M' \oplus K &= (M \oplus D) \oplus K && (1) \\ &= (M \oplus K) \oplus D && (2) \\ &= C \oplus D && (3) \end{aligned}$$

The second equation follows from the associativity of the XOR operator. We see that C' is the encrypted shortened packet with a valid ICV, and that it can be obtained by directly applying the modification to the encrypted shortened packet C.

This technique can be used to decrypt TKIP packets sent towards the client as follows. An attacker tries all 2^8 possible values of the removed byte. For each guess the modification D is applied and the resulting packet is injected with a different priority y. Assuming that QoS channel y has a lower TSC it will pass the TSC check, and the client will decrypt the packet. Then the ICV is verified. If the guess of the attacker was wrong the ICV is invalid and packet will silently be dropped (without generating a MIC failure). On the other hand, if the guess was correct, the ICV will be valid. However, with high probability the MIC value of the shortened packet will be wrong. As a result the client sends a MIC failure report. Thus a correct guess can be detected by listening for the corresponding MIC failure. Note that an AP isn't vulnerable to the attack because it never sends MIC failure reports. To avoid triggering the TKIP countermeasures at most one byte can be decrypted each minute.

Because we must wait one minute after decrypting a byte, it is infeasible to decrypt all bytes using this method. Instead the Beck and Tews attack targets an ARP reply packet and decrypts only the ICV and the MIC value. This takes on average 12 to 15 minutes. The remaining content of the packet is guessed. A particular guess can be verified by checking if the calculated ICV of the predicted packet equals the decrypted ICV. If they match, the guess is very likely correct. Once the ARP reply has been decrypted we can use the inverse Michael algorithm to calculate the MIC key used for AP to client communication [20]. Combined with the keystream of the decrypted ARP reply, an attacker can now forge 3 to 7 packets having a length smaller or equal to the ARP reply. The precise number of packets that can be forged depends on the number of supported QoS channels.

3. DENIAL OF SERVICE

This section describes a novel attack we discovered. When closely inspecting the QoS extension to TKIP we notice that the keystream is independent of the priority (i.e., QoS channel) used to transmit the frame. On the other hand the calculated MIC value does depend on the priority of the MSDU. Say that we capture a packet sent with priority x and replay it with a different priority y. Assuming QoS channel y has a lower TSC it will pass the TSC check, and the receiver will use the correct keystream to decrypt the packet. As a result it will also pass the ICV check. However, the changed priority will cause the receiver to expect a different MIC value. Hence a MIC failure occurs. Replaying this packet a second time will trigger a second MIC failure. After these two MIC failures the AP will shut down all TKIP traffic for 1 minute. Repeating this process every minute will prevent any TKIP protected communication, effectively causing a DoS. If the network does not use QoS we can forge the QoS header when replaying the packet. As discovered by Morii and Todo, most clients will not check whether the network is actually using QoS and simply accept the packet [17]. Therefore the only requirement is that one or more clients *support* the QoS extension, which is true for most modern wireless adapters [17].

To implement the attack we had to patch the compat-wireless drivers of Linux. The original driver modified the QoS header when in monitor mode, which interfered with our attack. Monitor mode is a feature supported by some wireless adapters and drivers allowing one to capture all wireless traffic. It also enables injection of arbitrary 802.11 frames. Our tool monitors the traffic of a network and shows whether it supports TKIP or QoS. Additionally it shows a list of connected clients and basic statistics such as the cipher suite it is using, number of MIC failures, number of association failures, etc. When a vulnerable TKIP packet is captured its priority is changed and the packet is replayed. If the network isn't using QoS our tool will forge the QoS header. We show in Sect. 6 that our implementation was found to be very reliable.

Our attack appears to be one of the more effective DoS attacks that can be launched against a networking using TKIP. The simplicity of the attack not only makes it easy to implement and debug, it also assures it is applicable in many real world environments. In Section 7 a thorough comparison is given to currently known DoS attacks. Disabling the TKIP countermeasures prevents the attack. However, most APs do not provide this option, and with good reason. As mentioned in Sect. 2.3 the countermeasures were included to detect active attacks against the weak Michael algorithm [6].

Another option to prevent the attack is to make the keystream dependent on the priority. This entails a change in the protocol, meaning all devices implementing TKIP would have to be updated. Though preventing the attack, such a modification does not appear feasible in practice.

4. MORE AND BIGGER PACKETS

The Beck and Tews attack allows forging 3 to 7 packets of at most 28 bytes [20]. Our goal is twofold: we want to inject both more and bigger packets. In this section we assume the Beck and Tews attack has already been executed, meaning we have obtained the MIC key for AP to client communication. Our technique is similar to the fragmentation attack on WEP [4] and an improvement of a suggested attack by Beck [1]. To the best of our knowledge, this is the first time such an attack on TKIP has been implemented and proved to be possible.

4.1 Exploiting Fragmentation

Since we know the MIC key, sending additional packets only requires obtaining new keystreams. Recall that each TSC corresponds to a different keystream. Because RC4 is used, XORing an encrypted packet with its corresponding unencrypted packet unveils the keystream used during encryption. Hence, finding new keystreams reduces to predicting the content of encrypted packets.

All 802.11 packets start with a LLC and SNAP header of 8 bytes. If we know whether it is an ARP, IP, or EAPOL packet, this header can be predicted. Fortunately we can identify ARP and EAPOL packets based on their length, and can consider everything else to be IP. The first byte of an IP packet consists of the version and the header length, which is generally equal to 0x45. The second byte containing the Differentiated Services Field can be predicting based on the priority of the 802.11 frame [18]. Finally the next 2 bytes, representing the length of the IP packet, can be derived from the length of the 802.11 frame. As a result we can predict the first 4 bytes of an IP packet. Predicting the first 14 bytes of ARP packets is also possible. These bytes consist of the hardware type and size, protocol type and size, request type, and finally the MAC address of the sender. All these fields can be predicted in an IPv4 network. Because EAPOL packets are rarely transmitted we simply ignore them. Our predictions can then be XORed with the encrypted packets, revealing the first 12 bytes or more of the keystream.

These short keystreams are combined using fragmentation at the 802.11 layer. As mentioned in Sect. 2.2 the 802.11 protocol allows a frame to be fragmented into at most 16 MPDUs. Each MPDU must have a unique TSC value and is encrypted with the keystream corresponding to that TSC. All fragments must include an ICV, though the MIC value is calculated over the complete MSDU and can be spread over multiple MPDUs. Using fragmentation to combine the keystreams we can inject packets with 112 bytes of payload. Assuming there is enough traffic on the network, we can inject an arbitrary amount of packets. Compared to the Beck and Tews attack this is a significant improvement.

4.2 Implementation: Performing a Portscan

Implementing the fragmentation attack required patching the compat-wireless drivers of Linux. The original driver failed to correctly inject MPDUs, which caused the attack to fail. Using our patched driver we successfully tested the fragmentation attack against several devices, and found that our heuristics to predict the first bytes of packets were very accurate (see Sect. 6). To reduce packet loss we also detect whether the client acknowledged receiving the MPDU, otherwise the MPDU is retransmitted. As a proof of concept we implemented a port scanner. This requires injecting a large amount of packets and is thus ideal to test our fragmentation attack.

The port scanner is given a file containing the ports to scan and works by injecting TCP SYN request to each port. The SYN packets do not contain any options, assuring there is enough keystream to inject them. The encrypted TCP SYN-ACK reply is detected by its length. In practice most packets are larger than a SYN-ACK packet, meaning it has a distinctive short length. After scanning a port a TCP RST packet is always sent, even if no SYN-ACK was detected. This is done to prevent the client from retransmitting a potentially undetected SYN-ACK packet. Note that if the replies of the client can be sent to an IP under our control, we essentially have bidirectional communication and can connect to open ports.

The attack can be mitigated by preventing the Beck and Tews attack. This means disabling the client from sending MIC failure reports, or using a short rekeying time of 2 minutes or less [20].

5. DECRYPTING ARBITRARY PACKETS

In this section we describe a state reset attack on the Michael algorithm and we show how it can be used to decrypt arbitrary packets sent towards the client.

5.1 The Michael Algorithm

The state of the Michael algorithm is defined by two 32-bit words (L, R) called left and right. To calculate the MIC value of an 802.11 packet the state is first initialised to the MIC key. Then the data shown in Fig. 2 is processed, which is padded so that its length is a multiple of 4 bytes. Finally the calculation is finalised and resulting MIC value is outputted. All data is processed in 32-bit words by a block function.

The block function $B(L, R)$ is an unkeyed 4-round Feistel-type construction, taking as input a Michael state and returning a new state [14, §11.4.2.3.3]. When processing an input word M the next state is given by $B(L \oplus M, R)$. We will let L' stand for $L \oplus M$. The block function can be inverted [22], and its inverse is denoted by B^{-1}. Note that we can predictably influence L at the start of the block function using M. For convenience the notation $B((L, R), M)$ is used to represent $B(L \oplus M, R)$, denoting the new internal state after processing M.

5.2 Michael State Reset

If the Michael state ever returns to the initial state, all data processed so far has no influence on the MIC value. This idea can be used to construct a prefix packet which resets the state, allowing us to append a packet whose MIC value is calculated only over the appended data. As suggested by Beck [1] this could be done by appending two *magic words* to the prefix. These so-called magic words are ordinary 32-bits data words, but chosen in such a way so they reset the internal state of the Michael algorithm. Using them an attacker can append any encrypted packet to the prefix without invalidating the MIC value of the complete packet. Unfortunately Beck didn't provide a thorough theoretical analysis. We generalise the problem to finding a list of magic words that will transform a start state (L_s, R_s) to an end state (L_e, R_e).

We could try to use just one magic word M_1 to reset the state. In that case $B(L_s \oplus M_1, R_s)$ must return (L_e, R_e). Assuming that processing a random word M using the block function results in a random state, a guess for M_1 can be modelled as a Bernoulli trial with a success probability of 2^{-64}. Since we can try at most 2^{32} values for M, finding a solution reduces to having the first success after 2^{32} trails. This follows a geometric distribution. We get

$$\Pr[X \leq 2^{32}] = 1 - \left(1 - 2^{-64}\right)^{2^{32}} \approx 2,328 \cdot 10^{-10} \tag{4}$$

where X follows a geometric distribution with a success probability of 2^{-64}. Such a low chance of finding a solution is unusable.

A better option is to use two magic words, denoted by M_1 and M_2. This gives us one intermediate state (L_i, R_i) to work with. We begin by calculating $B^{-1}(L_e, R_e) = (L'_i, R_i)$. Note that we cannot calculate L_i because M_2 is still unknown. Nevertheless, this teaches us the required value for R_i. Then we brute force the first magic word M_1. For each possible value we apply the block function. If we obtain the required value for R_i we have found a valid intermediate state (L_i, R_i), since using the guessed value for M_1 and setting M_2 to $L_i \oplus L'_i$ results in a solution:

$$\begin{aligned} B(B((L_s, R_s), M_1), M_2) &= B(L_i \oplus M_2, R_i) && (5)\\ &= B(L'_i, R_i) && (6)\\ &= (L_e, R_e) && (7) \end{aligned}$$

The first equation is trivial. The second equation follows from our choice of M_2. Finally, the third equation follows from the calculation $B^{-1}(L_e, R_e) = (L'_i, R_i)$.

A solution is found if the guess for M_1 results in the required value for R_i, which has a probability of 2^{-32}. Similar to the previous case, the probability of finding a solution can be modelled by a geometric distribution:

$$\Pr[Y \leq 2^{32}] = 1 - \left(1 - 2^{-32}\right)^{2^{32}} = 0,6321\ldots \tag{8}$$

where Y follows a geometric distribution with a success probability of 2^{-32}. This implies that roughly 27% of the time 2^{32} calculations are performed yet no solution is found. As an experiment we ran 50000 runs where random Michael states had to be connecting using two magic words. In 31518 runs a solution was found, resulting in a success probability of 63,036%. This closely matches our analyses. On an 3,10 GHz Intel Core i5-2400 it took on average 11,14 seconds to find a solution, with a standard deviation of 6,33 seconds.

Another strategy is to use three magic words M_1, M_2, and M_3. This gives us two intermediate states (L_{i1}, R_{i1}) and (L_{i2}, R_{i2}) to work with. Again we start by calculating $B^{-1}(L_e, R_e) = (L'_{i2}, R_{i2})$. Next we take 2^{16} random values for the first magic word and compute the list of resulting intermediate states (L_{i1}, R_{i1}). We then try to brute-force the second magic word by applying the reverse Michael algorithm to the state (L'_{i2}, R_{i2}). If the resulting value for R_{i1} is in the list of earlier calculated states we have found a solution. Let L_{i1} be the value accompanying R_{i1}, then the magic words M_1, $M_2 = L_{i1} \oplus L'_{i1}$, and $M_3 = L_{i2} \oplus L'_{i2}$ provide a solution:

$$\begin{aligned} &B(B(B((L_s, R_s), M_1), M_2), M_3) \\ &\qquad = B(B(L_{i1} \oplus M_2, R_{i1}), M_3) && (9)\\ &\qquad = B(L_{i2} \oplus M_3, R_{i2}) && (10)\\ &\qquad = (L_e, R_e) && (11) \end{aligned}$$

The first equation is trivial. The second and third follow from our choice of M_2 and M_3, and the usage of the reverse block function to calculate L'_{i1} and L'_{i2}.

The probability that the result of a guess for M_2 is in the list is 2^{-16}. The probability of finding a solution can again be modelled as a geometric distribution. We get

$$\Pr[Z \leq 2^{32}] = 1 - \left(1 - 2^{-16}\right)^{2^{32}} \approx 1 - 7,2 \cdot 10^{-28463} \tag{12}$$

where Z follows a geometric distribution with a success probability of $p = 2^{-16}$. Practically this shows that a solution will always be found. The average number of required guesses for the second word is $E[Z] = 1/p = 2^{16}$. As an experiment we ran 50000 runs where random states had to be connecting using three magic words. In all runs a solution was found. It took on average $65814 = 2^{16.017...}$ guesses for M_2 until a solution was found, with a standard deviation of 66407 guesses. On our 3,10 GHz Intel Core i5-2400 this corresponded to an average running time of 2,96 milliseconds. Though requiring more magic words, these results are significantly better than the previous two cases.

5.3 Decryption Attack

Our goal is to decrypt arbitrary packets sent towards the client. We will accomplish this by appending the targeted packet to a specially crafted prefix. The prefix will simulate the behaviour of a ping request, making the client echo back the appended data. We construct the prefix such that the reply is send to an IP under our control, meaning we will receive the plaintext content of the targeted packet, effectively decrypting it. To assure that the MIC value of the constructed packet is correct we will apply the Michael state reset attack to the ping-like prefix. This allows us to append the targeted packet without invalidating the MIC value. The resulting frame is sent to the client using the fragmentation attack.

Contrary to the suggestion by Beck [1] we cannot use an ICMP ping request as the prefix. This is because it includes a checksum calculated over the header and the data section. But since we do not know the plaintext data of the full packet, we cannot calculate a correct checksum. Instead we will construct a UDP prefix, where specifying a checksum is optional. Sending a UDP packet to a closed port results in an ICMP destination unreachable reply containing the first 8 bytes of the UDP packet. However, on Windows, Linux, and Android the ICMP unreachable reply contains a full copy of the original UDP packet. So when targeting these operating systems the client will reply with the complete content of our constructed packet. In particular this includes the plaintext content of the targeted packet. Even large packets can be quickly decrypted using this method.

Again we created a proof of concept tool in Linux. It listens for packets sent towards the client. Once a vulnerable packet has been captured, the UDP prefix is constructed and the Michael state reset attack is applied. The resulting UDP prefix is transmitted using the fragmentation attack, followed by the targeted packet (which is marked as the final fragment of the MSDU). In practice this means the final fragment will usually be bigger than all previous fragments. Though this is not allowed by the 802.11 specification, nearly all devices will accept the packet (see Sect. 6). The ICMP unreachable reply sent by the client will include the prefix, magic bytes, and the unencrypted data of the targeted packet. Among other things, this allows an attacker to decrypt a TCP packet, learn the sequence number, and hijack the TCP stream to inject arbitrary data [12]. As a consequence, malicious data could be injected when the client opens a website. Again the attack can be mitigated by preventing the Beck and Tews attack.

6. EXPERIMENTS

In this section we begin by investigating whether TKIP is still supported in practice. Then we evaluate how much devices adhere to the relevant aspects of the 802.11 standard, we report on implementation vulnerabilities discovered in some wireless devices, and we discuss how our findings impact the attacks. Finally our attacks are tested in realistic settings.

Table 1: Number of WiFi networks supporting the given encryption schemes for several regions. Note that one network can support multiple schemes.

Region	Open	WEP	TKIP	CCMP	#Networks
Leuven	381	618	3307	3143	5023
Heverlee	121	288	1212	1149	1886

6.1 Networks Supporting TKIP

Some new routers (for instance the Belkin N300 router) do not support TKIP anymore, in accordance with the security roadmap of the WiFi Alliance. The WiFi Alliance tests products and hands out certifications if they conform to certain standards. Their new roadmap specifies that, as of 2011, new APs are no longer allowed to support a TKIP only option [7]. Even mixed mode, which simultaneously allows TKIP and CCMP in the same network, is no longer a requirement. Finally, in 2014 TKIP is disallowed completely. Based on this one would think that TKIP is no longer widely supported. Surprisingly, we found the opposite to be true.

To investigate whether TKIP is still supported in practice we surveyed wireless networks in two Belgian municipalities (Leuven and Heverlee). Detecting networks was done using passive scanning, consisting of monitoring wireless traffic for beacon frames. These frames contain all information necessary to connect to a wireless network. In particular it includes the name of the network, the MAC address of the AP, and the encryption schemes supported by the network. During an initial tests we found that active scanning detected few additional networks (less than 6%), so in an attempt to prolong battery life only passive scanning was used.

Several trips, of around an hour long, were made on foot while scanning for networks. The raw capture was written to file and later analysed for beacon frames using a custom tool. This approach allowed us to make improvements to our tool after collecting the raw captures. We uniquely identified networks by their name. This is necessary because one network can be advertised by multiple APs. However we also encountered several APs advertising a network named after a vendor or product. These are default network names used by a particular device and do not represent the same network. Therefore we treated each of these APs as a unique network. Similarly there were several APs advertising a network with an empty name. These APs were also treated as an unique network.

In total we detected 6803 unique networks. The number of networks supporting a particular encryption scheme is shown in Table 1. Note that a handful of networks were present both in Leuven and Heverlee. We found that 93% of the networks used encryption, and that 66% supported TKIP. When considering only encrypted networks, 71% of them supported TKIP. Additionally, 19% of networks using encryption only allow TKIP. We believe the reason so many networks still support TKIP is because most routers, when configured to use WPA2, by default use mixed mode (allowing both TKIP and CCMP). We even observed that WEP is still used by 14% of encrypted networks.

Table 2: Results of various tests on different wireless adapters. For laptop and USB adapters, L and W denote it only works on Linux or Windows, respectively. Yes means it works on both, no means it works on neither. Open Source router firmware was tested on the Asus RT-N10.

		DoS	Fragmentation				Replay	Unenc.
			diff. size	eff. frag.	skip TSC	any MIC		
Laptop and USB:	Intel 4965AG	yes	yes	no	L	no	no	no
	Belkin F7D1102AZ	yes	yes	yes	yes	W	yes	L
	Belkin F5D7053	yes	L	L	L	L	yes	yes
	Alfa AWUS036h	yes	yes	yes	yes	W	W	W
	Ralink WA-U150BB	yes	yes	yes	yes	L	yes	W
Mobile Devices:	iPod MC086LL	yes	yes	no	yes	no	no	no
	iPad MC980NF	yes	yes	no	yes	no	no	no
Access Points:	Linksys WAG320N	yes	yes	yes	yes	no	no	no
	WRT54G 4.21.5	yes	yes	yes	yes	no	no	no
	Scarlet VDSL Box	yes	yes	no	no	yes	no	yes
	Cisco Aironet 1130 AG	yes	yes	yes	yes	no	no	no
	Asus RT-N10 1.0.2.4	yes	yes	yes	yes	no	no	no
	Tomato 1.28	yes	yes	yes	yes	no	yes	no
	DD-WRT v24-sp2	yes	yes	yes	yes	no	no	no

6.2 Adherence to the 802.11 Specification

The wireless adapters in Table 2 were tested for implementation details impacting our attacks. While doing this we encountered several vulnerabilities present in some wireless devices, and decided to test for their presence on all devices. Wireless routers and mobile devices were tested using their default configurations, with open source wireless router firmware was tested on the Asus RT-N10. Laptops and USB wireless adapters were tested on Linux using the compat-wireless 3.6.2-1-snp drivers, and on Windows using the default installed drivers.

For the DoS attack to work a client must send a MIC failure report when an invalid MIC has been detected, and the AP must shut down the network after two MIC failures. We found that all wireless adapters, in all configurations, implement this properly. As a result our DoS attack is applicable to all tested devices (see Table 2 column DoS). Additionally we tested the DoS on a network not using QoS, making our tool forge the QoS header. Again all adapters were vulnerable to the attack, confirming results by Todo et al. [21].

Fragmentation support has been tested for several properties. We found that the Belkin F5D7053 adapter on Windows incorrectly implemented fragmentation. Sending a fragment to this adapter always resulted in a MIC failure, though it worked fine under Linux. All other devices supported fragmentation.

An important property of fragmentation is whether the last fragment is allowed to be bigger than the previous fragments. Strictly speaking this is not allowed in the 802.11 standard [14, §9.5] yet we rely on this during the decryption attack. We found that all devices supporting fragmentation permitted this behaviour (see Table 2 column diff. size).

As mentioned in Sect. 2.3 the receiver should update its replay counter after reassembling the MSDU. This leaves open the possibility to send each fragment of an MSDU using the same TSC, keystream, and priority. If allowed, an attacker is then able to inject 16 fragments using only one keystream. In Table 2 column eff. frag. we see that this technique is indeed possible on several devices.

Finally we tested if fragments can be sent without using a sequential replay counter, i.e., whether we can skip TSC values. The 802.11 standard permits this behaviour for TKIP [14, §11.4.2.6], though some devices only accepted sequential TSC (see Table 2 column skip TSC).

Surprisingly we also discovered critical implementation flaws in some devices, particularly in wireless USB adapters. Several do not drop packets with an already used TSC, allowing an attacker to replay packets (see Table 2 column Replay). Another flaw is that certain devices do not verify the MIC value of a fragmented TKIP packet (see Table 2 column any MIC). This removes the requirement of first having to execute the Beck and Tews attack to obtain the MIC key. After all, if the MIC value is not verified we do not have to calculate it, meaning there is no reason to know the MIC key. On Windows the Ralink WA-U150BB drops fragmented packets with an incorrect MIC, but does not transmit a MIC failure. As mentioned in Sect. 3, dropping the packet but not sending a MIC failure can open the door for novel attacks on the MIC key.

More worrisome, we also encountered wireless adapters that accepted unencrypted packets while connected to an encrypted network (see Table 2 column Unenc.). In particular the vulnerability is present in the Windows drivers for the AWUS036h, a device popular for its wide reception and high transmission power. Another device susceptible to the attack was the Scarlet VDSL Box. This is a router that is handed out by the Belgium ISP Scarlet when a customer buys a VDSL internet connection. The reason this case is interesting is because the vulnerability has a higher impact when present on APs. An attacker could abuse the flaw to easily inject arbitrary traffic into the network, but also to send packets to the internet using the public IP of the victim. It might be an interesting future research direction to test for additional implementation flaws on a wider array of devices.

6.3 Verifying Our Attacks

According to the 802.11 specification, the TKIP countermeasures must disable only TKIP traffic for one minute [14, §11.4.2.4]. However, most of the APs we tested disabled all

Table 3: Impact of our DoS attack on several Access Points. The term *Both* denotes that TKIP and CCMP traffic was disabled. Open source router firmware was tested on the Asus RT-N10.

Access Point	Protocols Disabled	
	WPA1	WPA2
Linksys WAG320N	Both	Both
WRT54G 4.21.5	TKIP	Both
Scarlet VDSL Box	Both	TKIP
Cisco Aironet 1130 AG	Both	Both
Asus RT-N10 1.0.2.4	Both	TKIP
Tomato 1.28	Both	Both
DD-WRT v24-sp2	Both	Both

wireless traffic, including CCMP protected traffic in mixed mode. When targeting these devices one single TKIP client allows an attacker to take down the complete wireless network. More precisely, the behaviour of an AP depends on whether WPA1 or WPA2 is being used (see Table 3). Practically the difference between WPA1 and WPA2 is mainly between supported encryption schemes: WPA1 mandates TKIP support and optionally allows CCMP, while the reverse is true for WPA2. For example, the WRT54G only supports TKIP when using WPA1, while the Belkin N300 only supports WPA2 with CCMP.

For the fragmentation, portscan, and decryption attack to work we need to be able to accurately predict the first 12 bytes of encrypted packets. To test our prediction algorithm we monitored TKIP traffic generated by visiting several websites for 20 minutes. In total 36643 data packets were captured, of whom 36642 were predicted correctly. The single miss was an EAPOL packet, which we purposely ignored. Further, we weren't able to capture all packets as an attacker, meaning the keystreams corresponding to certain TSCs remained unknown. In total 5680 packets were not captured, consisting of 13% of all traffic.

The fragmentation and portscan attack has been tested against Windows, Linux, iOS, and Android. When connected to a networking using TKIP, all were found to be vulnerable. Additionally we tested the attacks under two authentication mechanisms: the first being a shared passphrase and the second being a personal login and password verified using PEAP-MSCHAP v2. In both situations the attacks were successful. These tests clearly demonstrate the reliability of the attacks.

For the decryption attack we found that, when connected to a network using TKIP, Windows, Linux, and Android were vulnerable. Mac OS X and iOS only include the first 8 bytes of the UDP packet in the ICMP unreachable reply, meaning the targeted packet is never included. Nevertheless, we did receive ICMP unreachable replies from all operating systems, meaning the Michael reset attack worked in all cases. Installing a firewall which blocks the ICMP unreachable replies prevents the decryption attack, though the Michael reset attack itself will remain possible.

7. RELATED WORK

Several DoS attacks exist on wireless networks. Arguably the most well-known attack consists of forging deauthentication frames to either the client or AP [2]. This is possible because the deauthentication message is not protected using any keying material. Continuously sending them will result in a DoS attack. An advantage of our attack is that it requires replaying only two frames each minute to disconnect *every* client that is using TKIP. Hence our attack is stealthier and requires less power to execute. Furthermore, nowadays the deauthentication attack can be prevented by enabling protected management frames [14, §4.5.4.9]. Compared to the DoS attack of Glass and Muthukkumarasamy [9] our attack is easier to execute, since their attack requires a man-in-the-middle position. Another advantage is that our attack can be used to easily verify whether the client sends MIC failure reports, and thus to see if it is possible to perform the Beck and Tews attack [17]. The Beck and Tews attack could also be changed to a DoS attack. Once the first MIC failure has been detected, the corresponding packet could be injected a second time. However, on average 128 packets have to be injected before a MIC failure is triggered. This makes the attack easier to detect. It also isn't as easy to implement compared to our DoS attack, making it more difficult to execute in practice.

Könings et al. found DoS vulnerabilities at the physical and MAC layer of 802.11, some halting traffic for one minute with minimal packet injection [15]. Contrary to our DoS, their attacks do not simultaneously disconnect all clients connected to a network. Bicakci and Tavli give a survey on DoS attacks at the physical and MAC layer. The attacks they discuss require injecting a large amount of frames [3].

One of the first attacks on TKIP was found by Beck and Tews and decrypts an ARP reply packet. As a result the MIC key for AP to client communication can be obtained [20] and a few small packets can be injected. It only works if the QoS extension is enabled and takes 12–15 minutes to execute. Halvorson et al. improved the attack, allowing larger packets to be injected [11]. However their technique does not allow injection of more packets, and takes longer to execute compared to our fragmentation attack. Morii and Todo came up with a modification removing the requirement that the AP must have enabled QoS [17]. They found that even if QoS was not used, clients will still accept and process a QoS frame. This allows an attacker to forge the QoS header if not present. In another paper Todo et al. managed to reduce the execution time of the Beck and Tews attack to 8–9 minutes [21].

The Michael algorithm was designed by Ferguson [6]. It was quickly revealed to be invertible by Wool [22]. Based on this Wool suggested a related message attack on TKIP. Huang et al. showed that Michael is not collision free and suggested a packet forgery attack [13].

Beck suggested the use of two magic words to reset the Michael state [1]. Unfortunately, no thorough theoretical analysis was provided, and no implementation in a practical setting was given. Based on the fragmentation attack on WEP by Bittau et al. [4], Beck also suggested using the fragmentation attack on TKIP, though no implementation was made to verify his ideas.

Wireless drivers have been tested before for vulnerabilities [5]. Most of the techniques focus on fuzzing with the aim of finding code injection attacks. However, only a limited amount of research has been done to find logical implementation flaws. In particular we found no previous work testing for replay attacks, seeing if it is possible to inject plaintext packets into an encrypted network, or testing whether the MIC of fragmented TKIP packets is verified.

Moen et al. have analysed the key scheduling algorithm in TKIP [16]. They found that, given less than 10 RC4 packet keys, it is possible to recover the temporal key (TK) with a time complexity of $\mathcal{O}(2^{105})$ compared to a brute force attack with complexity $\mathcal{O}(2^{128})$.

8. CONCLUSION

TKIP was designed as an intermediary solution to mitigate the existing flaws of WEP. We found that TKIP is still supported by a large number of networks. Further, we showed that TKIP fails to provide sufficient security, by describing and implementing several new attacks. In addition, during our experiments, we identified several critical implementation vulnerabilities in several wireless devices. We conjecture that a more substantial test of these implementations will reveal more vulnerabilities.

Specifying a short rekeying interval prevents the fragmentation and decryption attack, unfortunately this does not prevent our DoS attack. To secure a wireless network it is strongly advised to only support the more secure CCMP.

9. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers and the shepherd for their valuable feedback.

This research is partially funded by the Research Fund KU Leuven, and by the EU FP7 project NESSoS. With the financial support from the Prevention of and Fight against Crime Programme of the European Union (B-CCENTRE). Mathy Vanhoef holds a Ph. D. fellowship of the Research Foundation - Flanders (FWO).

10. REFERENCES

[1] M. Beck. Enhanced TKIP michael attacks. Retrieved 4 Februari, 2013, from `http://download.aircrack-ng.org/wiki-files/doc/enhanced_tkip_michael.pdf`.

[2] J. Bellardo and S. Savage. 802.11 denial-of-service attacks: real vulnerabilities and practical solutions. In *Proceedigns of the USENIX Security Symposium*, 2003.

[3] K. Bicakci and B. Tavli. Denial-of-service attacks and countermeasures in IEEE 802.11 wireless networks, 2009.

[4] A. Bittau, M. Handley, and J. Lackey. The final nail in WEP's coffin. In *IEEE Symposium on Security and Privacy*, pages 386–400, 2006.

[5] L. Butti and J. Tinnés. Discovering and exploiting 802.11 wireless driver vulnerabilities. *Journal in Computer Virology*, 4(1):25–37, 2008.

[6] N. Ferguson. Michael: an improved MIC for 802.11 WEP. *IEEE doc. 802.11-2/020r0*, Jan. 2002.

[7] G. Fleishman. Say goodbye to WEP and TKIP. Retrieved 26 November, 2012, from `http://bit.ly/cSFSvj`, 2010.

[8] S. R. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In *Selected Areas in Cryptography*, pages 1–24, 2001.

[9] S. M. Glass and V. Muthukkumarasamy. A study of the TKIP cryptographic dos attack. In *15th International Conference on Networks*. IEEE, 2007.

[10] M. Guennoun, A. Lbekkouri, A. Benamrane, M. Ben-Tahir, and K. El-Khatib. Wireless networks security: Proof of chopchop attack. In *WOWMOM*, pages 1–4, 2008.

[11] F. M. Halvorsen, O. Haugen, M. Eian, and S. F. Mjølsnes. An improved attack on TKIP. In *14th Nordic Conference on Secure IT Systems*, NordSec '09, 2009.

[12] B. Harris and R. Hunt. Review: TCP/IP security threats and attack methods. *Computer Communications*, 22(10):885–897, 1999.

[13] J. Huang, J. Seberry, W. Susilo, and M. W. Bunder. Security analysis of michael: The IEEE 802.11i message integrity code. In *EUC Workshops*, pages 423–432, 2005.

[14] IEEE Std 802.11-2012 (Rev. of IEEE Std 802.11-2007). *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2012.

[15] B. Könings, F. Schaub, F. Kargl, and S. Dietzel. Channel switch and quiet attack: New DoS attacks exploiting the 802.11 standard. In *LCN*, 2009.

[16] V. Moen, H. Raddum, and K. J. Hole. Weaknesses in the temporal key hash of WPA. *Mobile Computing and Communications Review*, 8(2):76–83, 2004.

[17] M. Morii and Y. Todo. Cryptanalysis for RC4 and breaking WEP/WPA-TKIP. *IEICE Transactions*, 94-D(11), 2011.

[18] S. Park, K. Kim, D. Kim, S. Choi, and S. Hong. Collaborative QoS architecture between DiffServ and 802.11e wireless LAN. In *Vehicular Technology Conference*, 2003.

[19] A. Stubblefield, J. Ioannidis, and A. D. Rubin. A key recovery attack on the 802.11b wired equivalent privacy protocol (wep). *ACM Trans. Inf. Syst. Secur.*, 7(2), 2004.

[20] E. Tews and M. Beck. Practical attacks against WEP and WPA. In *Proceedings of the second ACM conference on Wireless network security*, WiSec '09, 2009.

[21] Y. Todo, Y. Ozawa, T. Ohigashi, and M. Morii. Falsification attacks against WPA-TKIP in a realistic environment. *IEICE Transactions*, 95-D(2), 2012.

[22] A. Wool. A note on the fragility of the Michael message integrity code. *IEEE Transactions on Wireless Communications*, 3(5):1459–1462, 2004.