Discovering Logical Vulnerabilities in the Wi-Fi Handshake Using Model-Based Testing

Mathy Vanhoef
imec-DistriNet, KU Leuven

Domien Schepers
imec-DistriNet, KU Leuven

Frank Piessens
imec-DistriNet, KU Leuven

{mathy.vanhoef,frank.piessens}@cs.kuleuven.be, research@domienschepers.com

ABSTRACT

We use model-based testing techniques to detect logical vulnerabilities in implementations of the Wi-Fi handshake. This reveals new fingerprinting techniques, multiple downgrade attacks, and Denial of Service (DoS) vulnerabilities.

Stations use the Wi-Fi handshake to securely connect with wireless networks. In this handshake, mutually supported capabilities are determined, and fresh pairwise keys are negotiated. As a result, a proper implementation of the Wi-Fi handshake is essential in protecting all subsequent traffic. To detect the presence of erroneous behaviour, we propose a model-based technique that generates a set of representative test cases. These tests cover all states of the Wi-Fi handshake, and explore various edge cases in each state. We then treat the implementation under test as a black box, and execute all generated tests. Determining whether a failed test introduces a security weakness is done manually. We tested 12 implementations using this approach, and discovered irregularities in all of them. Our findings include fingerprinting mechanisms, DoS attacks, and downgrade attacks where an adversary can force usage of the insecure WPA-TKIP cipher. Finally, we explain how one of our downgrade attacks highlights incorrect claims made in the 802.11 standard.

Keywords

802.11; Wi-Fi; handshake; testing; fuzzing

1. INTRODUCTION

Nowadays most Wi-Fi networks are secured using Wi-Fi Protected Access 2 (WPA2) [1]. Indeed, even public hotspots can now use WPA2 encryption thanks to the Hotspot 2.0 program [32]. Moreover, once vendors implement Opportunistic Wireless Encryption [15], open Wi-Fi networks can also use encryption (though without authentication). This follows the advice of RFC 7435, which states that encryption should be used even when authentication is not available [12]. All such encrypted networks rely on the Wi-Fi handshake to securely negotiate fresh pairwise keys. These keys are used to encrypt normal traffic. Therefore, a correct and secure implementation of the Wi-Fi handshake is essential to assure all transmitted data is properly protected.

ASIA CCS '17, April 02 - 06, 2017, Abu Dhabi, United Arab Emirates

ISBN 978-1-4503-4944-4/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3052973.3053008

The Wi-Fi handshake, which relies on a shared master key to negotiate fresh pairwise keys, has previously been formally analyzed [16, 17, 23, 30]. However, we are not aware of any works that test implementations of the (4-way) handshake. While implementations of certain enterprise authentication mechanisms have been inspected [24, 5], these mechanism negotiate master keys, and hence do not include the 4-way handshake. In other works merely the first stage of the Wi-Fi handshake is tested, which consists only of network discovery using unprotected management frames (i.e. beacons and probe responses). Additionally, these works only try to detect common programming mistakes such as buffer overflows, NULL pointer dereferences, and so on. In other words, the 4-way handshake stage, which negotiates fresh pairwise keys, has not yet been tested. In contrast, network protocols such as TLS have undergone rigorous testing for both logical vulnerabilities and common programming mistakes during various stages of the handshake [3, 11, 25]. Inspired by these works, we show how to rigorously and efficiently detect logical implementation vulnerabilities during various stages of the Wi-Fi handshake.

To systematically test implementations of the Wi-Fi handshake, we propose a model-based technique. First, we model the Wi-Fi handshake by defining the sequence of messages exchanged in a normal handshake. In this model, only the type of a message is taken into account, and dynamic parameters such as nonces and keys are abstracted. We then define a set of test generation rules that take this model (i.e., sequence of abstract messages) and generate a set of invalid or unexpected handshake executions. For example, a rule may generate an execution where a required message is skipped, or a message has an invalid integrity check. If an implementation does not reject such faulty executions, an irregularity has been detected. We then manually inspect irregularities to determine whether they pose a security risk. By defining appropriate test generation rules, we can generate a representative set of test cases, which explore various edge cases in each stage of the Wi-Fi handshake.

We apply our testing technique against 12 authenticator-side implementations of the Wi-Fi handshake. This uncovered irregularities in all implementations, and revealed multiple vulnerabilities. Irregularities can be abused to fingerprint devices. Notable vulnerabilities are downgrade attacks against Broadcom and MediaTek-based routers. Here, an adversary can force usage of WPA-TKIP instead of the

more secure AES-CCMP. Intriguingly, our downgrade attack against Broadcom also highlights incorrect claims made in the 802.11 standard. Also notable are DoS attacks against Windows 7 and OpenBSD. Additionally, we found implementation bugs that, although they do not directly lead to practical attacks, are concerning. We consider these results surprising since, compared to handshakes used in protocols like TLS, the Wi-Fi handshake is not that complex (see Section 2). Put differently, even though the Wi-Fi handshake is relatively simple, our testing technique still managed to detect a substantial number of bugs and vulnerabilities.

To summarize, our main contributions are:

- We model the Wi-Fi handshake, and define test generation rules that operate on this model, with as goal to generate a representative set of test cases. These tests explore edge cases at each stage of the handshake.
- We create a tool that can execute the generated tests against authenticator-side implementations of the handshake, and use it to test 12 concrete implementations.
- We report several vulnerabilities, as well as new fingerprinting techniques, that were found during our tests.

The remainder of this paper is organized as follows. Section 2 introduces (a model of) the Wi-Fi handshake. We describe our testing technique in Section 3, and present the results of applying it to several implementations in Section 4. Finally, we summarize related work in Section 5 and conclude in Section 6.

2. THE Wi-Fi HANDSHAKE

In this section we introduce relevant aspects of the 802.11 standard [20], and present a model of the Wi-Fi handshake.

2.1 Technical and Historical Background

The original 802.11 standard supported a rudimentary security protocol called Wired Equivalent Privacy (WEP). Unfortunately, WEP contains major design flaws and is considered completely broken [13, 26, 4]. To address these security issues, the IEEE designed both a short-term and long-term solution. Their long-term solution is called (AES-)CCMP. It uses AES in counter mode for confidentiality, and CBC-MAC for authenticity. However, because many vendors implemented the cryptographic primitives of WEP in hardware, older WEP-compatible devices would not be able to support CCMP. To address this issue, the (WPA-)TKIP protocol was designed as a short-term solution. Similar to WEP, it is based on the RC4 cipher, meaning WEP-capable hardware can support TKIP using only firmware upgrades.

Due to the slow standardization process of the IEEE, the Wi-Fi Alliance already began certifying devices based on a draft version of the 802.11i amendment. This certification program was called Wi-Fi Protected Access (WPA), and required support for TKIP, but did not mandate support for CCMP. For clarity, we refer to it as WPA1. Unfortunately, this led to the common misunderstanding that WPA1 is synonymous with TKIP. Once 802.11i was standardized, the Wi-Fi Alliance started the WPA2 certification program. This certification requires that a device supports CCMP, but does not mandate support for TKIP. This led to another common misconception, namely that WPA2 means (AES-)CCMP is used, while in reality a WPA2 network might still use (or support) TKIP.

Since WPA1 and WPA2 are both based on the 802.11i standard, they are nearly identical to each other. Nevertheless, because WPA1 was based on a draft of the standard, and WPA2 on the final version, there are some minor technical differences between them. Briefly summarized, the most important differences for us are:

- The RSNE is encoded somewhat differently in WPA1 than in WPA2 (see Section 2.2).
- In WPA1, the group key handshake is part of the Wi-Fi handshake, and transports the group key. In WPA2, this key is transported in the 4-way handshake, and no group key handshake is required (see Section 2.5.4).
- In WPA2, message 4 of the 4-way handshake must have the `Secure` flag set in the key info flags, while for WPA1 this bit must not be set (see Section 2.5.4).
- In WPA2, the descriptor type field in EAPOL-Key frames equals 2, while it should equal 254 when using WPA1 (see Section 2.5).

In all other aspects, WPA1 is technically identical to WPA2. Therefore, unless mentioned otherwise, we will treat WPA1 as identical to WPA2. Finally, we use the term Robust Security Network (RSN) to refer to 802.11i security mechanisms in general (i.e. it can refer to both TKIP and CCMP).

We remark that usage of TKIP is being discouraged by the Wi-Fi Alliance [33]. However, WPA2-certified devices are still allowed to support TKIP and CCMP simultaneously [33]. In practice this means more than half of all encrypted networks still support TKIP [29]. Depending on the software these networks use, it means they could be vulnerable to one of our downgrade attacks (see Section 4).

2.2 Stage 1: Network Discovery

Wireless stations can discover networks by passively listening for beacons, or by actively sending probe requests. When an AP receives a probe request, it will respond with a probe response. This process is shown in stage 1 of Figure 1. Both beacons and probe responses contain the name and capabilities of the wireless network. Among other things, this includes the Robust Security Network information Element (RSNE). This element contains the supported pairwise cipher suites of the network, the group cipher suite being used, and the security capabilities of the AP. In our model, the cipher suite can be either TKIP or CCMP. The bit-wise encoding of the RSNE differs between WPA1 and WPA2. Nevertheless, in both WPA versions, exactly the same information is being represented.

Once the client has found a network to connect to, the actual handshake can start. During the handshake the client is generally called the supplicant, and the access point is called the authenticator. In this paper we treat the terms authenticator and AP as synonyms.

2.3 Stage 2: Authentication and Association

In the second stage, the supplicant starts by (mutually) authenticating with the authenticator. Once authenticated, the supplicant continues by associating with the network. This processes is illustrated in stage 2 of Figure 1.

The 802.11 standard defines four authentication methods: Open System authentication, Shared Key authentication, Fast BSS Transition (FT), and Simultaneous Authentication

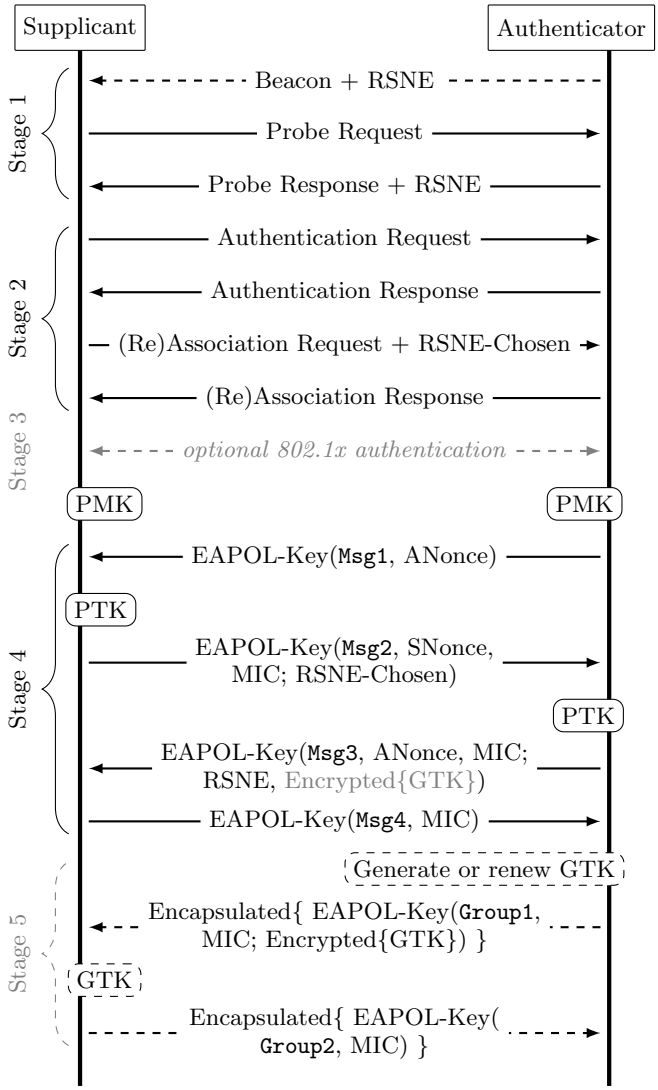

Figure 1: Frames sent in the 802.11 handshake, including their most important parameters. Optional stages or parameters are shown in (dashed) gray.

of Equals (SAE) [20, §4.5.4.2]. SAE offers password-based authentication, and claims to be resistant against passive attacks, active attacks, and dictionary attacks [14]. However, few of the devices we tested support SAE, and hence we exclude it from our model of the Wi-Fi handshake. Fast BSS Transition relies on keys derived during a previous connection with the network. Additionally, it embeds other stages of the handshake in the authentication and reassociation frames. This reduces the roaming time when a supplicant moves from one AP to another. However, because it is rarely supported on commodity routers, we also exclude it from our model. Shared Key authentication is based on WEP, and deprecated due to inherent security weaknesses [20, §11.2.1]. Therefore, it is commonly disabled, and we can also exclude it from our model. In other words, during the first authentication stage, we only have to consider Open System authentication. It allows any supplicant to successfully authenticate. On a network using RSN security, the actual authentication will be performed at stage 4 during the 4-way handshake.

Once authenticated, the supplicant continues by associating to the network. This is done by sending an association request to the AP (see Figure 1). It informs the AP of the features that the supplicant supports (e.g. the supported bit rates). More importantly, at this stage the supplicant also chooses the pairwise and group cipher suites it wants to use. This choice is encoded in an RSNE element. If the supplicant encodes the RSNE using the conventions of WPA1, the WPA1 variant of the handshake will be executed. Otherwise, the WPA2 variant will be executed. Because the AP also uses an RSNE element to advertise its *supported* list of cipher suites, we will use the term RSNE-Chosen when the RSNE encodes the *chosen* cipher suites. The AP replies with an association response, informing the supplicant whether the association was successful or not.

2.4 Stage 3: 802.1x Authentication

The third stage is optional, and consist of 802.1x authentication to a back-end Authentication Server (e.g. a RADIUS server). During this stage the authenticator acts as a relay between the supplicant and the Authentication Server. In practice 802.1x is commonly used in enterprise networks, where users can login using a username and password. The end result of this stage is that the supplicant and authenticator share a secret Pairwise Master Key (PMK). Because (parts of) this stage have already been investigated in other works [2, 18, 8, 24, 5], or are based on TLS [6, 3, 11, 25], we do not include this stage of the handshake in our model. Instead, we assume the supplicant and authenticator already share a secret PMK. It can be a cached PMK from an earlier 802.1x session, or one derived from a pre-shared key.

2.5 Stage 4: The 4-way Handshake

The fourth stage consists of a 4-way handshake. It provides mutual authentication based on the PMK, detects possible downgrade attacks, and negotiates a fresh session key called the Pairwise Transient Key (PTK). Downgrade attacks are detected by cryptographically verifying the RSNEs received during the network discovery and association stage. The PTK is derived from the Authenticator Nonce (ANonce), Supplicant Nonce (SNonce), and the MAC addresses of both the supplicant and authenticator. The 4-way handshake may also transport the current Group Temporal Key (GTK) to the supplicant.

The 4-way handshake is defined using EAPOL-Key frames. Figure 2 illustrates the structure of these frames. We will briefly discuss the most important fields. First, the descriptor type determines the remaining structure of an EAPOL-Key frame. Although WPA1 uses the value 254 for this field, and WPA2 uses the value 2, both define an identical remainder structure of the EPAOL-Key frame.

Following this 5-byte header, is the key information field. It consists of a 3-bits key descriptor version subfield, and eight (one-bit) flags called the key info flags. The key descriptor field defines the cipher suite that is used to protect the frame. This is either AES with HMAC-SHA1, or RC4 with HMAC-MD5. The choice between these cipher suites, and hence the value of the key descriptor field, depends on the pairwise RSN cipher being negotiated. More precisely, if CCMP is chosen, then AES with SHA1 is used. Otherwise RC4 with MD5 is used. The replay counter field is used

Protocol Version 1 byte	Packet Type 1 byte	Body Length 2 bytes
Descriptor Type – 1 byte		
Key Information 2 bytes		Key Length 2 bytes
Key Replay Counter – 8 bytes		
Key Nonce – 32 bytes		
EAPOL Key IV – 16 bytes		
Key RSC – 8 bytes		
Reserved – 8 bytes		
Key MIC – variable		
Key Data Length 2 bytes		Key Data variable

Figure 2: Layout of EAPOL-Key frames [9, §11.6.2].

to detect replayed frames. When the supplicant replies to an EAPOL-Key frame of the authenticator, it must use the same replay counter as the one in the previously received EAPOL-Key frame. The authenticator always increments the replay counter after transmitting a frame.

Finally, the integrity of the frame is protected using a Message Integrity Check (MIC), and the key data field is encrypted if it contains sensitive data. Recall that the used encryption algorithm is specified in the key descriptor field.

When using WPA2, the receiver of an EAPOL-Key frame can distinguish among the different messages of the handshake by inspecting the key info flags. To refer to the n-th message of the 4-way handshake, we simply use the term *message n*. Additionally, we say that the 4-way handshake negotiates TKIP or CCMP encryption, when the chosen pairwise cipher is TKIP or CCMP, respectively.

In Figure 1, which illustrates our model of the Wi-Fi handshake, we use the following notation:

EAPOL-Key(`MsgType`, Nonce, MIC; Key Data)

It represents a frame with key info flags that identify a message of type `MsgType`, and with the given nonce (if present). When the MIC parameter is present, the frame is protected using a message integrity check. Finally, all parameters after the semicolon are stored in the key data field (see Figure 2). The notation Encrypted{GTK} is used to stress that, if the GTK is included, it must be encrypted using the PTK.

2.5.1 Message 1

The first message of the 4-way handshake stage is sent by the authenticator, and contains the randomly generated ANonce of the authenticator. The key info flags that must be set in this message are `Pairwise` and `Ack`, which are represented by the label `Msg1`. Note that this message is not protected by a MIC, and hence can be forged by an attacker.

When the supplicant receives this message and learns the ANonce, it posses all information to calculate the PTK.

2.5.2 Message 2

This message contains the random SNonce of the supplicant, and is protected using a MIC. The key info flags that must be set are `Pairwise` and `MIC`, which are represented by the label `Msg2`. The Key Data field contains the authenticated RSNE-Chosen element, containing the chosen cipher suites by the supplicant that were previously sent in the (re)association request.

When the authenticator receives this message, it can calculate the PTK, verify the MIC, and compare the (authenticated) RSNE-Chosen in this message with the one that was previously received in the association request. If these RSNEs differ, the handshake is aborted.

2.5.3 Message 3

Message 3 is send by the authenticator. It again contains the ANonce [16, §5.3], and is secured by a MIC. The required key info flags are represented by `Msg3`, and are `Pairwise`, `MIC`, and `Secure`. The Key Data field includes the RSNE, which contains the supported cipher suites of the AP. Additionally, if WPA2 is used, it also includes the encrypted GTK. If WPA1 is used, the GTK is transferred to the supplicant using a group key handshake (see Section 2.6).

When the supplicant receives this message, it determines whether the (authenticated) RSNE in this message is identical to the one previously received in beacons and probe requests. If they differ, a downgrade attack has been detected, and the handshake is aborted.

2.5.4 Message 4

The supplicant sends message 4 to the authenticator, to confirm that the handshake has been successfully completed. This last message is also protected by a MIC. When WPA2 is used, the required key info flags are `Pairwise`, `MIC`, and `Secure`. However, for WPA1, the required key flags does not include `Secure`. We use `Msg4` to represent the required key info flags for both WPA versions. Note that message 2 and message 4 have the same required key info flags if WPA1 is used. The only way to differentiate message 2 and 4 in this situation is to see whether there is data present in the key data field. Once the authenticator received message 4, normal (encrypted) data frames can be transmitted.

Although the 802.11 standard claims that this message serves no cryptographic purpose [20, §11.6.6.8], we will show in Section 4.3 that this message is essential in preventing downgrade attacks.

2.6 Stage 5: Group Key Handshake

The last stage is required when WPA1 is used to transport the group key to the supplicant. This key protects broadcast and multicast traffic. In both WPA1 and WPA2, this procedure is also used to periodically renew the group key.

In Figure 1 we use the notation Encapsulated$\{\cdot\}$ to denote that the complete EAPOL-Key frame may also be protected using link-layer encryption (e.g., using TKIP or CCMP).

3. TESTING TECHNIQUE

In this section we introduce our testing technique. First we describe our general approach, and then we present various test generation rules which are designed to trigger erroneous behaviour in the implementation under test. Finally, we explain how the generated tests are executed in practice.

3.1 General Approach

Our testing technique is designed to test all stages of the Wi-Fi handshake in a black-box manner. This is achieved by taking our model of the Wi-Fi handshake, and applying test

generation rules on it to generate a set of test cases. A test case is essentially a sequence of messages to be transmitted to the authenticator, the expected replies, and a method to determine whether this exchange resulted in a successful connection or not. Ideally, the set of generated test cases covers all (possible erroneously handled) edge cases at each stage of the handshake. However, in practice we can only execute a finite number of test cases, meaning we must define representative and well-chosen test generation rules.

To construct a concrete test case, we start from our handshake model defined in Section 2. Recall that our model of the handshake is essentially a sequence of abstract messages exchanged in a normal handshake. In particular, this can be a successful handshake using WPA1 or WPA2, negotiating either TKIP or CCMP as the pairwise cipher suite. Hence, we have four fundamental instantiations of a normal handshake execution. These normal executions of the handshake can already be treated as (trivial) tests which must always be accepted by the Implementation Under Test (IUT). More interesting tests are created by modifying these four fundamental executions. To do this, we take a fundamental execution, and modify it according to a given test generation rule. These rules modify the execution of the handshake, with as goal to determine how the IUT reacts to various (correct and incorrect) modifications of the handshake.

Our test generation rules are inspired by an analysis of the specification, rudimentary code inspections of some implementations, and already known (and patched) implementation vulnerabilities. This step, namely defining an appropriate set of test generation rules, is arguably the most important step in our technique. After all, these rules determine the types of flaws and vulnerabilities that can be discovered.

When a generated test case has failed, manual inspection is required to determine the precise type of bug present in the implementation. If the implementation is open source, this can be done by inspecting the source code. Otherwise, additional black-box tests are performed to study the flaw in more detail. Once the bug is properly understood, we can investigate whether it can be abused by an adversary. Any possible attacks are then separately implemented and tested to confirm their validity.

3.2 Test Generation Rules

We define two categories of test generation rules. Rules in the first category manipulate messages as a whole, while rules in second category only change parameters (i.e., fields) of messages. The test generation rules also define whether a specific modification should result in a failed or successful handshake. For example, skipping a message should result in a failed connection, whereas retransmitting a message should still result in a successful connection.

Our fist category of rules can be regarded as generalizations of the Skip and Repeat heuristic rules that Beurdouche et al. used to generate so-called deviant traces [3]. We define two test generation rules in this category:

1. **Dropped messages:** Generate test cases where each message, together with its expected responses (if any), is removed.

2. **Injected messages:** Generate test cases where each message allowed in a handshake, together with its expected responses (if any), is inserted before every message that is normally transmitted. Note that this rule also tests whether an implementation properly handles retransmitted messages.

The second category of rules changes (implicit) parameters of messages. These can be regarded as generalizations of the Hop rule that Beurdouche et al. used [3]. We use the following set of rules:

3. **Invalid RSNE (cipher suite):** Generate test cases where the association request and message 2 have a modified RSNE. In particular, we generate test cases such that all possible cipher suite combinations are tested. Recall from Section 2 that valid values for the pairwise cipher are TKIP, CCMP, or TKIP/CCMP. The group cipher suite is either TKIP or CCMP.

4. **Invalid EAPOL-Key descriptor type:** Generate test cases where the descriptor type in each EAPOL-Key frame is switched to an unexpected value. In particular, we switch value 2 (used by WPA2) with value 254 (used by WPA1), and visa versa. We also generate test cases where a random value other than 2 and 254 is being used.

5. **Invalid EAPOL-Key key info flags:** Generate test cases that together try all possible combinations of key info flags. We test all combinations of the eight key flags: `Pairwise`, `Install`, `Ack`, `MIC`, `Secure`, `Error`, `Requested` and `Encrypted`. This results in a total of 2^8 possible combinations for every EAPOL-Key frame in the handshake.

6. **Switched EAPOL-Key cipher suite:** Generate test cases where each EAPOL-Key message is protected using a different cipher suite. This implies changing the value of the key descriptor version subfield (see Section 2.5). That is, AES with SHA1 is replaced by RC4 with MD5, and visa versa.

7. **Invalid EAPOL-Key replay counter:** Generate test cases where the replay counters in EAPOL-Key frames are adjusted to be either lower or higher than the correct replay counter.

8. **Invalid EAPOL-Key nonce:** Generate test cases where a nonce is added to EAPOL-Key frames that should not be containing a nonce. Its value can be copied from either the supplicant or authenticator, or it can be randomly generated.

9. **Invalid EAPOL-Key MIC:** Create test cases where each EAPOL-Key frame has an invalid MIC, with the `MIC` flag either set or not set in the key info flags.

We assume that these rules test independent code in an implementation. This means combining several rules should not increase the number of discovered bugs. In other words, we will not combine rules when generating test cases. However, inspired by a DoS attack that poisons the supplicant with a forged ANonce [16], we found one useful exception to this assumption. A combination of rule 2 with rule 8 gives:

10. **Injected Nonce**: This rule injects a forged message 1 or message 2 after, and before, a valid message 1 or message 2, respectively. The injected message contains a random nonce and an invalid MIC, with the `MIC` flag either set or not set in the key info flags.

Applying all the above test generation rules on the four fundamental instantiations of the handshake, always results in a finite number of test cases. Moreover, most of the generation rules are deterministic. Only rule 8 contains nondeterminism, but it is unlikely to influence the outcome of a test case. This means that if we rerun all the test cases, we will obtain the same results. Hence our testing technique, in contrast with random fuzzing, creates repeatable results.

3.3 Executing Test Cases

We implemented a test harness that executes the generated test cases. Because this is a large, complex, and time consuming task, we only support testing authenticator-side implementations of the handshake. The test harness translates each abstract message in the test case into a concrete message that can be transmitted. For example, it fills in the correct MAC addresses, nonce values, replay counters, and so on. Optionally, it encrypts the appropriate fields of the frame, calculates a valid MIC value, etc. In order to do this, some state information needs to be maintained. For example, upon receipt of message 1, the generated PTK must be stored. In general, state information is updated after receiving and processing every message. When a message is transmitted by the test harness, the keys and replay counters are taken from the stored state.

In all test cases, our test harness first waits for a beacon. This is done to confirm that the authenticator is still up and running. If during the execution of a test case a deauthentication message is received, we can conclude a connection has failed. As a result, we can already stop executing the test case, saving us from having to explicitly determine whether a successful connection was established or not. Timeouts are used to identify if an authenticator is no longer responding while executing a test case. The timeout interval can be adjusted depending on the implementation being tested.

3.4 Validating and Resetting the Connection

After running a test case, we have to determine whether the handshake resulted in a valid connection or not. This can then be compared to the expected result. For example, some generation rules should not negatively impact a handshake execution (e.g. retransmitting messages), while other rules should result in a failed connection (e.g. dropping messages). Merely listening for deauthentication frames, which are transmitted when the authenticator aborts an ongoing handshake, is not sufficient. This is because it may be that, according to the authenticator, the handshake is not yet completed, meaning it is still waiting for certain messages.

To verify if a connection was established or not, we try to communicate with the AP using normal (encrypted) data frames. We do this using the link-layer Address Resolution Protocol (ARP), avoiding the need to request an IP address. In particular we send an ARP request to the authenticator (gateway), where we request the MAC address of the gateway. Note that the gateway's IP address is known by the test harness. If the connection was successful, the gateway will reply to the sender MAC address of the ARP request.

Finally, we have to reset the connection after running a test case. This must be done both if the connection was successful or unsuccessful. Otherwise, the authenticator will be in an unknown state, possibly affecting the execution of any subsequent test cases. Fortunately, this can easily done by sending a deauthentication frame to the authenticator.

Table 1: Implementations of the Wi-Fi handshake that were tested for logical vulnerabilities.

Impl. Name	Version	Hardware
Broadcom	5.10.56.46	RT-N10
Broadcom	5.10.85.0	WAG320N
Hostapd	2.6	TL-WN722N
OpenBSD	6.0 generic 2148	WL-172
Telenet	Ver 30.10.2016	Home gateway
MediaTek	3.0.0.9	RT-AC51U
Windows 7	build 7601	TL-WN722N
Windows 10	build 10240	TL-WN722N
Apple Airport	7.6.7	Time Capsule
Apple macOS	10.12 (Sierra)	MacBook Pro
Aerohive	Ver 1.11.2016	HiveAP 330
Aironet (Cisco)	Ver 1.11.2016	Aironet 1130 AG

4. RESULTS

In this section we present the results of applying our testing technique to 12 authenticator-side implementations of the Wi-Fi handshake. Table 1 lists all the tested implementations. Note that we tested multiple versions of some.

First, our results show that all implementations exhibit different behaviour, which can be abused to fingerprint an implementation. Second, we discovered new types of vulnerabilities that are present in several devices. Finally, we analyze implementation-specific vulnerabilities and deviations.

4.1 Fingerprinting Mechanisms

Any difference in how an implementation processes certain messages, or uses certain fields, can be used to fingerprint and identify the implementation. We discovered two fields of EAPOL-Key frames that are particularly useful for this purpose. They are the key info field, and the descriptor type field. To fingerprint devices based on these fields, we must be able to execute a (partial) Wi-Fi handshake with the device. This means that we must posses credentials allowing access to the underlying network, or that unauthenticated encryption must be supported. We do not consider these serious limitations. For example, Hotspot 2.0 and opportunistic wireless encryption both offer encryption to untrusted devices [32, 15]. Similarly, in an enterprise network such as eduroam, devices may have access to the network, but they are not considered trusted. In other words, we can safely assume that an adversary can execute a Wi-Fi handshake with a targeted device. Hence she can determine how the target processes certain EAPOL-Key frames. The resulting behavioral fingerprint can then be used to identify the implementation, enabling implementation-dependent attacks.

4.1.1 EAPOL-Key Info Flags Fingerprint

Our first observation is that most vendors require, or prohibit, different key info flags in messages received during the Wi-Fi handshake. Recall from Section 2.5 that the key info field of an EAPOL-Key frame contains eight one-bit flags. Normally these flags identify properties of the message, which in turn can be used to distinguish between the different messages in a handshake. The eight flags are: `Pairwise`, `Install`, `Ack`, `Mic`, `Secure`, `Error`, `Requested`, and `enCrypted`. We will use the capitalized letter of each flag as a shorthand to represent the flag.

The key info fingerprint is defined as the required, al-

Table 2: Needed (i.e. required), allowed, and prohibited key info flags in EAPOL-Key frames sent by the supplicant. The capital letter of each flag is used as a shorthand: `Pairwise`, `Install`, `Ack`, `Mic`, `Secure`, `Error`, `Requested`, and `enCrypted`. An underlined Flag is WPA2-specific, while a crossed out Flag is WPA1-specific.

	Message 2			Message 4			Group message 2		
	Needed	Allowed	Prohibited	Needed	Allowed	Prohibited	Needed	Allowed	Prohibited
Broadcom	PM	ISERC	A	PMS	ISERC	A	MS	IRC	PAE
Hostapd	PM	ISEC	AR	PM	ISEC	AR	MS	IERC	PA
OpenBSD	PM	ISEC	AR	PM	ISEC	AR	M	ISEC	PAR
Telenet	PM	ISC	AER	PM	ISC	AER	*message is ignored*		
MediaTek	PM	IC	ASER	PM	ISC	AER			
Windows	PM	ISER	AC	PMS	IER	AC	*Device only supports WPA2, meaning the handshake never includes group message 2.*		
Apple OSes	PM	ISEC	AR	PMS	IEC	AR			
Aerohive	PM	ISEC	AR	PM	ISEC	AR			
Aironet	PM	ISEC	AR	PM	ISEC	AR			

lowed, and prohibited key info flags for each message in the Wi-Fi handshake. Table 2 contains the resulting fingerprint of all implementations. Note that MediaTek and Telenet to do require reception of group message 2 (which is only present in a WPA1 handshake). Since this message is effectively ignored, we cannot construct a fingerprint of it. For Broadcom and hostapd, there is a small difference in the treatment of flags depending on whether WPA1 or WPA2 is used. Broadcom rightly requires the `Secure` flag in message 4 when WPA2 is used, while this flag is optional for WPA1. In contrast, hostapd prohibits usage of the `Secure` flag for message 2 if WPA1 is used, while it allows this flag for WPA2. Interestingly, hostapd instantly deauthenticates the supplicant if WPA1 is used and message 2 has the `Secure` bit set, instead of simply ignoring the message. For other prohibited flags, hostapd simply ignores the EAPOL-Key message. Also note that Aerohive and Aironet are the only two implementations that have an identical key info flags fingerprint. Nevertheless, we can still distinguish them using our next fingerprinting mechanism.

For all implementations the `Pairwise` flag must be set if, and only if, the message is part of the 4-way handshake. Similarly, during the group key handshake, the `Pairwise` flag is always prohibited. We also remark that message 2 and 4 have the same set of required and prohibited flags in certain implementations. This means the key flags are not being checked to determine whether it is message 2 or 4. Instead, these implementations likely differentiate between message 2 and 4 by checking if the message contains an RSNE element or not. In a normal handshake, message 4 never contains an RSNE (see Figure 1).

Finally, different versions of Broadcom, Windows, and Apple implementations, all have the same key info fingerprint. Hence the key info fingerprint can generally be used to distinguish different implementations, but it cannot be used to identify specific versions of an implementation.

4.1.2 EAPOL-Key Descriptor Type Fingerprint

Our second observation concerns the descriptor type field of an EAPOL-Key frame. It should contain the value 2 if WPA2 is used, and 254 if WPA1 is used. In principle this field defines the remaining layout of the EAPOL-Key frame. However, this layout is identical for both WPA1 and WPA2. Hence the values 2 and 254 are equivalent in practice. Nevertheless, not all implementations treat these values as being identical. Some require that the value matches the type of handshake being executed, some allow both values, and even others allow *any* byte value. Table 3 gives an overview of the behaviour of each implementation. The difference in how this field is treated by each implementation can be used to fingerprint an implementation.

Table 3: Allowed values for the descriptor type field. *Any* means all byte values are allowed, *valid* means it must be either 2 or 254, and *match* means it must match the type of handshake being executed.

Impl.	Message 2	Message 4	Group Msg. 2
Broadcom	valid	valid	valid
Hostapd	valid[a]	valid[a]	valid[a]
OpenBSD	match	match	match
Telenet	valid	valid	ignored[b]
MediaTek	valid	valid	
Windows 7	match	match	not applicable[c]
Windows 10	match	match	
Apple OSes	any	any	
Aerohive	valid	valid	
Aironet	match	match	

[a] This is for WPA2 handshakes. For WPA1 handshakes, hostapd only allows descriptor type value 254.
[b] For these implementations, group message 2 is not required to complete the handshake.
[c] These devices only support WPA2, meaning the handshake never includes group message 2.

4.2 TKIP Countermeasures DoS

Surprisingly, several implementations incorrectly implement the TKIP countermeasure procedure. This procedure is a defense mechanism designed to protect the weak Message Integrity Check (MIC) algorithm used by TKIP. Although better message integrity algorithms were available when TKIP was designed, they were too computationally expensive for old WEP-compatible hardware. Therefore a custom but weak algorithm called Michael is used. To mitigate active attacks against Michael, a countermeasure procedure is activated by the AP, when two frames with a wrong MIC are received within one minute. This procedure kicks all clients using TKIP off the network, clears all keys used by TKIP, and disables any associations that request TKIP as the pairwise or group cipher suite. When a supplicant receives a frame with a wrong MIC, it informs the AP of this

by sending a MIC failure report. This report is an EAPOL-Key frame with the `MIC`, `Error`, and `Request` key info flags set. Note that these reports are automatically generated using test generation rule 5 of our testing technique (see Section 3.2). The AP treats a MIC failure report similar to receiving a TKIP frame with a wrong MIC. Hence, if the AP receives two MIC failure reports within a minute, the TKIP countermeasures are activated.

We discovered several flaws in how MIC failure reports are handled by implementations. Note that an attacker can cause a client to send a failure report by capturing a TKIP packet, modifying it, and then injecting the packet [28]. Injecting a failure report in a CCMP-only network requires the proper credentials. However, due to the rise of Hotspot 2.0 and opportunistic encryption, this is not a major limitation.

4.2.1 Impossible TKIP Countermeasures

When a network does not allow TKIP, supplicants should never be sending MIC failure reports to the AP. After all, in this situation it is impossible that the supplicant received a frame with a wrong TKIP MIC. Nevertheless, Broadcom, Windows 10, and Aerohive, accept MIC failure reports even if the network is configured to only use CCMP. Their countermeasure procedure kicks all stations of the network, including those that are not using TKIP. Additionally, Windows 10 and Aerohive refuse all connections during the countermeasure procedure, even if the station only wishes to use CCMP. This means a malicious supplicant can render a CCMP network completely unusable for one minute, simply by sending two MIC failure reports. While Broadcom's implementation does accept connections that only use CCMP during the countermeasures, we found another vulnerability that can still be abused to block all network access (see Section 4.2.2). These findings are surprising since one expects that if TKIP is disabled, the AP ignores MIC failure reports, and hence would not initiate the countermeasure procedure.

We also found that three implementations already accept MIC failure reports during the 4-way handshake. In particular, instead of sending message 4, a failure report can already be sent to the AP (see Table 4). Moreover, Windows 10 and OpenBSD permanently block connections once the countermeasure procedure is started. This results in an efficient DoS attack, where a malicious supplicant can permanently take down the network. Only after restarting the Windows 10 or OpenBSD AP are connections allowed again.

Interestingly, when inspecting the source code of hostapd, we discovered that hostapd v0.7.2 and older also accepted MIC failure reports in CCMP-only networks [21].

4.2.2 Broadcom: Repeated Countermeasure DoS

Our testing technique also uncovered a method to permanently take down a Broadcom network. The problem is that Broadcom's implementation accepts MIC failure reports, and initiates a new TKIP countermeasure period, even when an existing TKIP countermeasure period is in progress. When abusing this to trigger multiple simultaneous countermeasure periods, the AP eventually dies and becomes unresponsive.

Malicious clients can abuse this to take down a network. First, clients that only use CCMP can still connect during a TKIP countermeasure period (as they should be able to). Once connected, the client sends two MIC failure reports to trigger another TKIP countermeasure period. Sending these failure reports is possible even if the network only supports CCMP (see Table 4). Repeatedly reconnecting and triggering a new countermeasure period will eventually crash the AP. In experiments, we were able to permanently take down a CCMP-only network within 10 seconds. Connectivity could only be restored by rebooting the router.

Table 4: TKIP countermeasure behaviour of tested devices. The second column indicates whether the AP accepts failure reports when the network only supports CCMP. The third column shows whether a failure report can be sent during the 4-way handshake, and in which message(s). The last column shows the duration of the countermeasure period.

Implementation	In CCMP	In 4-way HS	Downtime
Broadcom	yes	no	Permanent
Hostapd	no	Msg4	1 minute
OpenBSD	no	Msg4	Permanent
Telenet	yes	no	1 minute
MediaTek	no	no	1 minute
Windows 7	no	no	none[a]
Windows 10	yes	no	Permanent
Apple OSes	no	no	none[a]
Aerohive	yes	Msg4	1 minute
Aironet	no	no	none[a]

[a] We were unable to trigger the TKIP countermeasures on these devices, because they only supported CCMP.

4.2.3 OpenBSD: Permanent DoS

OpenBSD is vulnerable to a permanent DoS attack. This attack works against networks supporting TKIP. The problem is that the TKIP countermeasure procedure of OpenBSD permanently blocks new connections, instead of only blocking them for 1 minute. Because networks that support TKIP must also use it as their group cipher [29], all clients will use TKIP as their group cipher. And since new connections using TKIP are not allowed, this effectively prevents any client from connecting. Hence a malicious supplicant merely has to send two MIC failure reports in order to bring down a network supporting TKIP. Connectivity can only be restored by restarting OpenBSD's access point.

4.3 Downgrade Attack by Forging Message 4

In a WPA1 handshake, Broadcom cannot distinguish message 2 and message 4 of the 4-way handshake. The difference between these messages, in both WPA1 and WPA2, is that message 2 includes data in the key data field of the EAPOL-Key frame, while message 4 does not (see Figure 1). More specifically, only message 2 includes an RSNE element. However, when Broadcom is expecting message 4, it does not verify that the data field of the received frame is empty. This means that, when the supplicant retransmits message 2, it may get treated as message 4 by Broadcom. Note that if WPA2 is used, an additional difference between both messages is that message 4 contains the `Secure` flag, while message 2 does not. Broadcom does check for this WPA2-specific difference (see Table 2), meaning the message confusion vulnerability only occurs when WPA1 is used.

4.3.1 Vulnerability Analysis

We can abuse this flaw to forge message 4. In particu-

lar, we induce the supplicant into retransmitting message 2, which will then be treated as a valid message 4 by Broadcom. Surprisingly, based on the 802.11 standard, the ability to forge message 4 should not introduce practical attacks. More precisely, in an informative analysis of the 4-way handshake, the 802.11 standard states that message 4 is only required for reliability and not security [20, §11.6.6.8]:

> "While Message 4 serves no cryptographic purpose, it serves as an acknowledgment to Message 3. It is required to ensure reliability and to inform the authenticator that the supplicant has installed the PTK [..]"

Several comments on a draft version of the standard also indicate that the 802.11 working group underestimated the importance of message 4 [19]. These comments ranged from claiming that message 4 is only required in case ANonce is predictable, to even suggesting the removal of message 4.

However, we discovered that message 4 is essential in preventing downgrade attacks against the 4-way handshake. That is, if an adversary can forge message 4, she can perform a partial downgrade attack against the authenticator. This not only shows that the 802.11 standard incorrectly claims that message 4 serves no cryptographic purpose, it also results in a practical attack against Broadcom's implementation. Moreover, this also demonstrates that the proposed 3-way handshake in [31] is not secure. We conclude that a secure implementation of the handshake must verify message 4 before the handshake can be considered successful.

4.3.2 Downgrade Attack

Our attack forces the authenticator into using TKIP, even though the supplicant and authenticator support CCMP. Normally, CCMP should be used in such a situation.

The adversary starts the attack by setting up a rogue AP that will act as a man-in-the-middle between the supplicant and authenticator. She modifies all beacons and probe responses, so it appears that the authenticator (AP) only supports TKIP (see Figure 3). As a result, the supplicant will connect to the authenticator, and request TKIP as the chosen pairwise cipher suite. At this point the adversary will forward message 1 and 2 of the 4-way handshake without modification. However, it will block message 3, assuring that the supplicant never sees this message. Blocking this message is essential since it contains the real RSNE of the authenticator, which includes both TKIP and CCMP. This RSNE differs from the one that the adversary advertised in beacon and probe requests. The supplicant would abort the handshake if this difference is detected.

The adversary now exploits the fact that Broadcom cannot distinguish message 2 and 4 of the handshake. Note that message 2 or 4 cannot be forged directly since they are protected by a MIC. Instead, we induce the supplicant into retransmitting a valid message 2 with an increased replay counter. This is accomplished by forging message 1, which can be done since it is not protected by a MIC. When the supplicants receives the forged message 1, it retransmits message 2. The retransmitted message 2 is forwarded to the authenticator, which will be wrongly treated as being a (valid) message 4. The authenticator now thinks the 4-way handshake has been successful, and installs the session keys to enable transmission of normal (encrypted) traffic.

After the attack, the authenticator will encrypt frames using TKIP. For example, the first message of the group key

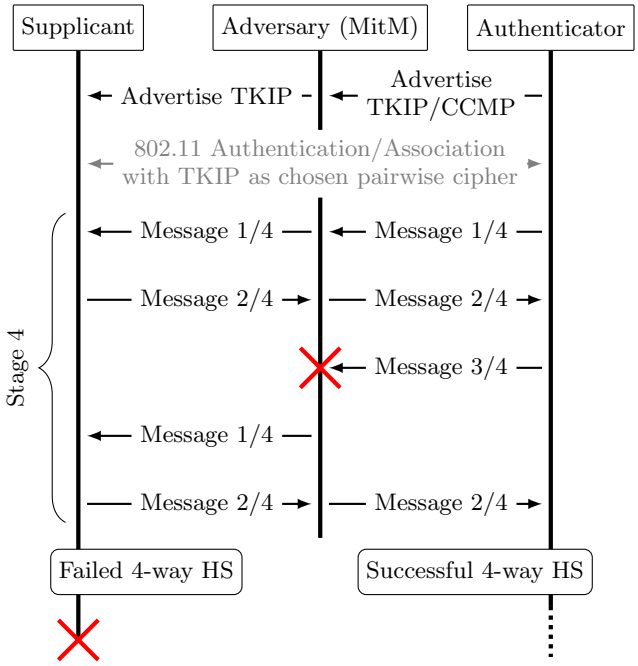

Figure 3: Downgrading a Broadcom authenticator to TKIP, when both TKIP and CCMP is enabled. The attack only works against WPA1, since Broadcom only confuses message 2 and 4 in WPA1.

handshake will be encrypted using TKIP. However, because the supplicant never received message 3, it did not complete the handshake. Hence the supplicant will not process these encrypted frames. Nevertheless, it is clearly problematic that we can force the authenticator into using TKIP, even though CCMP was supported by both devices.

4.3.3 Discussion

In order to execute the attack in practice, the adversary must obtain a MitM position between the supplicant and authenticator. A naive approach would be to set up a rogue AP with a MAC address different from the real AP. Unfortunately, using a different MAC address would interfere with the 4-way handshake, because the negotiated session keys depend on the MAC address of the authenticator and supplicant. Using the same MAC address as the real AP is also not an option, since the supplicant and authenticator would simply start communicating directly with each other.

Our solution is to use a channel-based MitM attack, where the adversary clones the AP on a different channel [29]. Cheap USB dongles can be used as jammers to force supplicants into connecting with the cloned AP [29]. With this approach we are sure supplicants will never directly communicate with the real authenticator, since they are on different channels. This means we do not have to use different MAC addresses when forwarding frames. As a result, the supplicant and authenticator will negotiate the same session keys, and will be able to successfully complete the handshake.

As mentioned, it is important that the supplicant does not receive message 3 of the handshake. Otherwise it will notice that the advertised RSNE only contains TKIP, while the real RSNE contains both TKIP and CCMP. If this difference is detected, the supplicant will abort the handshake,

meaning we cannot trigger the retransmission of message 2. Thankfully, in a channel-based MitM attack, frames can be blocked by simply not forwarding them.

4.4 Insufficient RSNE Verification

We discovered several implementations that exhibited dubious behaviour when interpreting and authenticating RSNE elements. This includes incorrectly parsing the RSNE in association requests, as well as failing to compare this RSNE with the one contained in message 2 of the 4-way handshake.

4.4.1 Authenticator Checks

Telenet and MediaTek do not compare the RSNE element received in the association request with the one contained in message 2. While this does not immediately lead to an attack, it is an indication that the underlying code might be poorly written, and therefore is concerning behaviour.

Windows 7 and 10 do not correctly interpret the RSNE element in association requests. Even though a network may only support CCMP, a client can request TKIP as either the pairwise or group cipher. Moreover, it is even allowed to request two pairwise ciphers, something that is normally impossible. In all cases Windows acts as if CCMP was requested for both the pairwise and group cipher. When the authenticated RSNE in message 2 is received, the pairwise cipher(s) must match those requested in the association request, even though a different pairwise cipher is actually being used. However, the group cipher suite is not compared to the one requested in the association request.

Aerohive also accepts association requests where two pairwise ciphers are requested, though the second requested pairwise cipher must be supported by the network. We conjecture that Aerohive ignores all requested pairwise ciphers, except the last one. Finally, when a network supports both TKIP and CCMP as the pairwise cipher, Broadcom also accepts either TKIP or CCMP as the requested group cipher in an association request, independent of the group cipher that is actually being used by the network.

4.4.2 Supplicant Checks: MediaTek Downgrade

Because Telenet and MediaTek do not verify the RSNE in message 2, we conjectured they also did not check the RSNE in message 3. Note that this is only applicable if the router supports both client and AP functionality. Here we found that only MediaTek supports client functionality, with as goal to extend to coverage area of another AP.

To test whether the supplicant-side implementation of MediaTek verifies the RSNE in message 3, we modified hostapd. Our modified version always includes CCMP in the RSNE of message 3, even if only TKIP is enabled. We then configured hostapd to broadcast a TKIP-only network. As a result, the RSNE in beacons and probe responses differs from the RSNE in message 3. The supplicant should notice this difference and abort the handshake. However, we found that MediaTek does not detect this difference, and instead successfully connects to our modified hostapd AP. Hence an adversary can perform a downgrade attack against MediaTek simply by cloning the network and only advertising support of TKIP. This is especially problematic because MediaTek's client functionality is used to extend the range of another AP. That is, during an attack the traffic of all devices connected to the MediaTek router will be forwarded over a connection that has been downgraded to TKIP.

4.5 Implementation-Specific Findings

For each device we now present unique behaviour they exhibit, summarize previously discussed issues, and report the vulnerability disclosure steps we made so far.

4.5.1 Broadcom

One peculiar observation is that Broadcom's implementation replies with a disassociation frame when the supplicant sends EAPOL-Key frames before associating with the AP. Other implementations reply with a deauthentication frame in such situations. This unusual behaviour can be used to fingerprint and identify a Broadcom implementation.

As presented in Section 4.3, Broadcom cannot distinguish message 2 and 4 of the 4-way handshake. An adversary can use this to downgrade the authenticator into using TKIP. It is also affected by our impossible TKIP countermeasure attack of Section 4.2.1. Moreover, it contains a DoS vulnerability that permanently makes the network unusable (see Section 4.2.2). We reported these issues to Broadcom roughly four months ago. However, it is not yet clear when they will be fixed. Finally, Broadcom incorrectly interprets parts of the RSNE in association requests (see Section 4.4).

4.5.2 Hostapd

Hostapd v0.7.2 and older are vulnerable to our impossible TKIP countermeasure attack of Section 4.2.1. We also found that hostapd instantly deauthenticates the supplicant if WPA1 is used and message 2 has the `Secure` bit set, instead of simply ignoring the message. This can be used to fingerprint and identify a hostapd implementation.

4.5.3 OpenBSD

As discussed in Section 4.2.3, OpenBSD contains a DoS vulnerability where sending two MIC failure reports will make the network permanently unusable. Fortunately, this attack is only possible against a network that supports TKIP. This issue has been reported and fixed. Because it was a DoS vulnerability, it was not assigned a Common Vulnerabilities and Exposures (CVE) identifier.

4.5.4 Telenet

As discussed in Section 4.4, the implementation of the handshake used in Telenet devices does not verify that the RSNE received in the association request matches the one contained in message 2. While this is not directly exploitable, it is concerning behaviour. Finally, Telenet is affected by our impossible TKIP countermeasure attack (see Section 4.2.1).

4.5.5 MediaTek

MediaTek does not verify the RSNE in both message 2 and 3 of the 4-way handshake. As shown in Section 4.4, this can be abused in a downgrade attack when the supplicant is using MediaTek's implementation of the Wi-Fi handshake.

We also found that an adversary can launch a DoS attack against MediaTek. Here the adversary injects a forged message 2 with a random nonce and an invalid MIC, right after the supplicant transmitted the real message 2. On reception of message 2, MediaTek generates a new temporary PTK due to the changed SNonce value. The problem is that this temporary PTK is saved as the real PTK before the MIC of message 2 is verified. Normally, if the MIC is not valid, the temporary PTK should not be saved.

MediaTek only recently has a vulnerability disclosure process. This delayed us from reporting these issues, and because of this it is not yet clear when patches will be available.

4.5.6 Windows

In Section 4.4 we showed that Windows 7 and Windows 10 do not correctly interpret the RSNE in an association request. Additionally, in message 2 the authenticator ignores the group cipher suite. We also showed that Windows 10 is vulnerable to our impossible TKIP countermeasure attack (see Section 4.2.1). Moreover, against Windows 10 this attack results in a permanently unusable network, and connectivity can only be restored by restarting the AP.

Against Windows 7 we also discovered an unauthenticated targeted DoS attack that permanently prevents a specific client from connecting to the network. The attack works by sending two association requests after one another, with as sending MAC address the targeted victim. After this, the victim can no longer connect to the network. We conjecture that by sending two association requests, the internal state associated to the sender MAC address somehow gets corrupted. As a result, it is no longer possible to continue the handshake, or start a new one, when using this MAC address. This allows an adversary to permanently block specific MAC addresses from connecting to the network.

These issues have been reported and are being addressed.

4.5.7 Aerohive

We also tested an Aerohive HiveAP 330, which is a professional industrial-grade access point. For example, it is used to host the wireless network of the BruCON security conference in Belgium. At these types of conferences, all attendees are given the pre-shared key of the wireless network. This means individuals posses credentials allowing access to the network, but they are the opposite of trusted actors.

As mentioned in Section 4.4, Aerohive devices exhibit dubious behaviour when interpreting and verifying received RSNE elements. Additionally, in Section 4.2.1 we found that a supplicant can trigger the TKIP countermeasures even though the network is only using CCMP. Hence a malicious individual can bring down the complete network by injecting two frames every minute. This vulnerability was reported to Aerohive, and is fixed in HiveOS version 6.5r6.

4.5.8 Cisco Aironet

Aironet is the only implementation where inserted authentication requests do not impact the remaining execution of the handshake. For example, if the supplicant sends an authentication request after associating, or during the 4-way handshake, the AP does not restart the handshake. Put differently, the supplicant can continue by sending the next message of the handshake. Other implementations reset the Wi-Fi handshake when receiving an authentication request in the middle of an ongoing handshake. Although the behaviour of Aironet poses no security risks, it can be used to fingerprint and identify an Aironet device.

4.6 Discussion and Future Work

While our testing technique discovered various issues, it has some limitations. First, we do not combine test generation rules. Second, the structure of all generated messages are valid. That is, only their content is invalid. As a result, not all code of an implementation may be tested. An interesting future research direction is to determine how much code is covered, and use this to improve test generation rules.

5. RELATED WORK

Wi-Fi drivers have been tested for vulnerabilities, but only with as goal to detect general errors such as buffer overflows and NULL pointer dereferences [22, 7]. Moreover, they only tested how implementations handle unprotected management frames. In [28] the authors test implementations of WPA-TKIP for various logical vulnerabilities, but do not test the Wi-Fi handshake. In contrast, we tested implementations of the Wi-Fi handshake for logical flaws.

Beck and Tews were one of the first to attack TKIP in practice [27]. Also notable is the DoS attack against TKIP by Vanhoef and Piessens [28]. They discovered that capturing a TKIP frame, and injecting it using a different quality-of-service priority, causes the receiver to calculate a wrong MIC value [28, §3]. When doing this twice, the TKIP countermeasures are activated, and the network becomes unusable for one minute. In this work we found that some implementations process MIC failure reports, and activate the TKIP countermeasures, even though TKIP is not enabled.

The design of the 4-way handshake has been extensively analyzed [16, 17, 23, 31, 30]. In [16, 23] DoS attacks were discovered, improvements were proposed, and based on this a correctness proof was given in [17]. Wang et al. proposed a 3-way handshake to mitigate similar DoS attacks [31], but we have shown this proposal does not defend against downgrade attacks. Finally, [30] presents a downgrade attack that forces the group key to be encrypted using RC4 when transported in the 4-way handshake. Issues in enterprise authentication mechanisms have also been inspected [24, 5], though our focus is on other stages of the Wi-Fi handshake.

Recently, implementations of the Transport Layer Security (TLS) protocol have undergone rigorous tests for both logical vulnerabilities [3, 11] and for cryptographic failures and boundary violations [25]. Our work is inspired by [3]. In this work Beurdouche et al. constructed a simplified model of the TLS handshake. Based on this model they constructed so-called deviant traces. These traces are sequences of messages that should not be accepted by a secure implementation of TLS. Deviant traces are generated using three heuristic rules. In a sense our tests cases are similar to deviant traces: they are both generated based on a model of the protocol, and if such a trace or test case is accepted by an implementation, an irregularity has been detected. Beurdouche et al. also had to manually analyze irregularities to determine whether they pose any security risks.

Techniques that reconstruct the state machine of a protocol implementation can also be used to manually discover vulnerabilities. In [11], the format of valid protocol messages must first be defined, after which the state machine is reconstructed by querying the implementation as a black-box. This was used to discover several vulnerabilities in TLS implementations. Comparetti et al. proposed a technique that does not require one to specify the format of valid network messages in advance [10]. However, their method requires access to the binary code implementing the protocol.

6. CONCLUSION

We developed a new model-based testing technique to analyze implementations of the Wi-Fi handshake. Surprisingly,

not only did this uncover implementation-specific vulnerabilities, we also discovered more general types of vulnerabilities that are present in various devices. For example, we found that several implementations accept MIC failure reports, even though the network is not using WPA-TKIP. This can be exploited as a DoS attack to take down a network that is only allowing usage of AES-CCMP. Another example is that several implementations do not properly verify the advertised cipher suites in message 2 or 3 of the handshake. For MediaTek implementations, we even demonstrated that this allows an adversary to downgrade the connection from AES-CCMP to WPA-TKIP.

We also discovered severe implementation-specific vulnerabilities. For example, triggering the WPA-TKIP countermeasures against OpenBSD or Windows 10 causes the network to become unavailable indefinitely. Connectivity can only be restored by rebooting the AP. Additionally, against Windows 7, an adversary can block specific clients from connecting to the network. To recover from this targeted DoS attack against Windows 7, the AP must be restarted.

Finally, our downgrade attack against Broadcom, which forces the authenticator into using WPA-TKIP, highlights incorrect claims made in the 802.11 standard. Though the standard claims that message 4 of the 4-way handshake serves no cryptographic purpose, we demonstrated that message 4 is essential in preventing downgrade attacks.

7. ACKNOWLEDGMENTS

This research is partially funded by the Research Fund KU Leuven and by the imec High Impact Initiative Distributed Trust. Mathy Vanhoef was supported by a Ph. D. fellowship of the Research Foundation - Flanders (FWO).

8. REFERENCES

[1] WiGLE: WiFi encryption over time. Retrieved 23 October 2016 from https://wigle.net/enc-large.html.

[2] N. Asokan, V. Niemi, and K. Nyberg. Man-in-the-middle in tunnelled authentication protocols. In *SPW*, 2003.

[3] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *IEEE SP*, 2015.

[4] A. Bittau, M. Handley, and J. Lackey. The final nail in WEP's coffin. In *IEEE SP*, 2006.

[5] S. Brenza, A. Pawlowski, and C. Pöpper. A practical investigation of identity theft vulnerabilities in eduroam. In *WiSec*, 2015.

[6] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov. Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations. In *IEEE SP*, 2014.

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

[8] A. Cassola, W. Robertson, E. Kirda, and G. Noubir. A practical, targeted, and stealthy attack against WPA enterprise authentication. In *NDSS*, 2013.

[9] I. . L. S. Committee et al. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Standard*, 2012.

[10] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol specification extraction. In *IEEE SP*, 2009.

[11] J. De Ruiter and E. Poll. Protocol state fuzzing of TLS implementations. In *USENIX Security*, 2015.

[12] V. Dukhovni. Opportunistic security: Some protection most of the time. RFC 7435, 2014.

[13] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In *SAC*, Lecture Notes in Computer Science, 2001.

[14] D. Harkins. Simultaneous authentication of equals: a secure, password-based key exchange for mesh networks. In *SENSORCOMM*, 2008.

[15] D. Harkins and W. Kumari. Opportunistic wireless encryption. RFC Draft 7, Feb. 2017.

[16] C. He and J. C. Mitchell. Analysis of the 802.11 i 4-Way handshake. In *Proceedings of the 3rd ACM workshop on Wireless security (WiSE)*, 2004.

[17] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *CCS*, 2005.

[18] H. Hwang, G. Jung, K. Sohn, and S. Park. A study on MITM (man in the middle) vulnerability in wireless network using 802.1x and EAP. In *ICISS*, 2008.

[19] IEEE 802.11 WG. Letter ballot 52 comments. Retrieved 20 October 2016 from http://www.ieee802.org/11/LetterBallots/preLB86/11-03-033r10-I-LB52-Comments.xls, June 2003.

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

[21] J. Malinen. Ignore Michael MIC failure reports if cipher is not TKIP. hostapd commit `fbc72d32c`, 2011.

[22] M. Mendonça and N. Neves. Fuzzing Wi-Fi drivers to locate security vulnerabilities. In *EDCC*, 2008.

[23] C. H. J. C. Mitchell. Security analysis and improvements for IEEE 802.11i. In *NDSS*, 2005.

[24] P. Robyns, B. Bonné, P. Quax, and W. Lamotte. Short paper: exploiting WPA2-enterprise vendor implementation weaknesses through challenge response oracles. In *WiSec*, 2014.

[25] J. Somorovsky. Systematic fuzzing and testing of TLS libraries. In *CCS*, 2016.

[26] A. Stubblefield, J. Ioannidis, and A. D. Rubin. A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP). *TISSEC*, 2004.

[27] E. Tews and M. Beck. Practical attacks against WEP and WPA. In *WiSec*, 2009.

[28] M. Vanhoef and F. Piessens. Practical verification of WPA-TKIP vulnerabilities. In *ASIA CCS*, 2013.

[29] M. Vanhoef and F. Piessens. Advanced Wi-Fi attacks using commodity hardware. In *ACSAC*, 2014.

[30] M. Vanhoef and F. Piessens. Predicting, decrypting, and abusing WPA2/802.11 group keys. In *USENIX Security*, 2016.

[31] L. Wang and B. Srinivasan. Analysis and improvements over DoS attacks against IEEE 802.11i standard. In *NSWCTC*, 2010.

[32] Wi-Fi Alliance. *Hotspot 2.0 (Release 2) Technical Specification v1.1.0*, 2010.

[33] Wi-Fi Alliance. Technical note: Removal of TKIP from Wi-Fi devices, Mar. 2015.