COMPREHENSIVE ANALYTICAL REPORT ON THE MAJOR VULNERABILITY DISCOVERED IN WPA2 WI-FI SECURITY PROTOCOL

Draft: 22:08 PM October 17, 2017

Published: 12:00 PM October 23, 2017

Updated: 12:00 PM October 23, 2017

Scan two-dimensional code to obtain the latest version of the

Contents

1 Overview

October 15, Mathy Vanhoef, postdoctoral security researcher at Leuven University in Europe, disclosed the critical vulnerability of the Wireless Network (Wi-Fi) Protection Protocol standard WPA2 [1,2], which allows attackers listen for Wi-Fi traffic between the computer and the access point within the Wi-Fi range. The vulnerability affects the protocol itself and is valid for WPA and WPA2, so all the software or hardware that supports WPA / WPA2 protocol are affected.

After the disclosure of the vulnerability, engineers from Antiy and Lianshi Networks made a quick response, and analyzed it together to form this report.

2 Vulnerability Analysis

WPA stands for Wi-Fi Protected Access, including two standards WPA and WPA2, is a protocol for protection of wireless network (Wi-Fi) [3]. WPA implements most part of the IEEE 802.11i standard, which is the transition plan for replacing the WEP before 802.11i is complete, and then will be replaced by WPA2. Since both WPA and WPA2 are based on 802.11i, they are almost identical at the technical level, with the main difference being that WPA2 requires a more secure CCMP. WPA and WPA2 use the four-way handshake defined in 802.11i, and the client (Station, STA) and access point (AP) authenticate each other with a four-way handshake, and negotiate the session key called Pairwise Transient Key PTK. PTK is generated by Pairwise Master Key (PMK), ANonce, SNonce and both MAC addresses, etc. PMK is generated by the information that both parties know like login password. And the temporary key (Temporal KEY, TK) used for subsequent normal data encryption is derived from PTK. The relationship between each key and parameter is shown in the following figure:

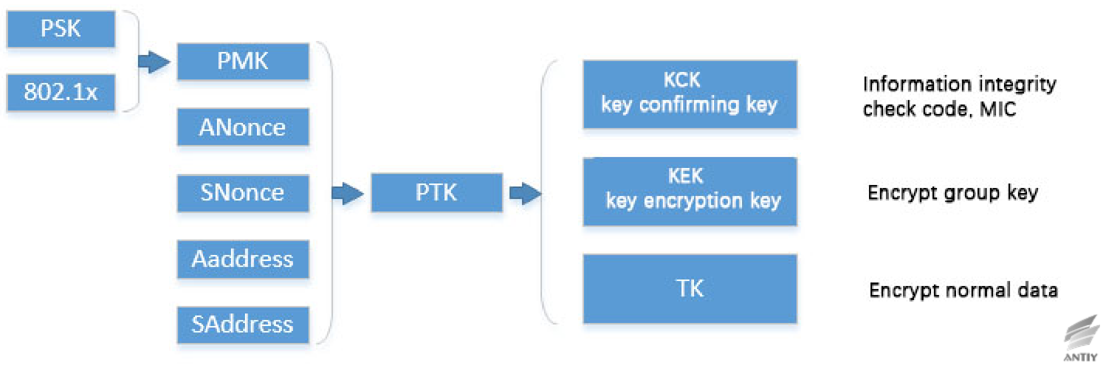


Figure 2-1 The relationship between four key handshake and parameter in four-way handshake

The process of four-way handshake can be summarized as follows:

1）AP sends its own ANonce to STA;

2）STA generates SNonce and calculates PTK, and then sends SNonce and information integrity check code MIC to AP;

3）AP receives SNonce, calculate PTK (At this time both have PTK), the group key GTK will be encrypted and sent to the STA with MIC;

4）STA receives GTK, installs PTK and GTK, sends ACK to confirm. AP installs PTK after confirming.

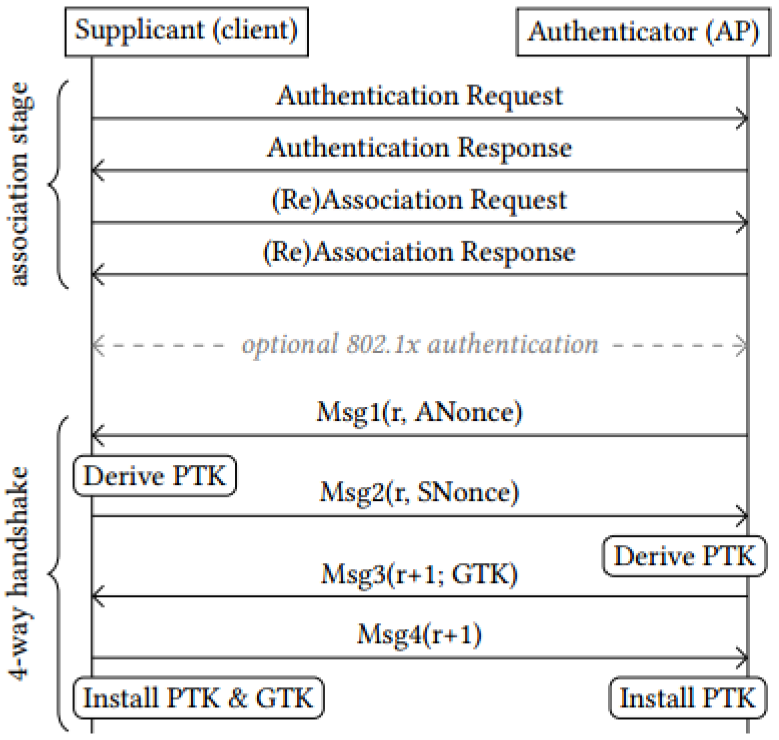


Figure 2-2 Four-way handshake process

Vanhoef named the disclosed vulnerability attack as KRACK (Key Reinstallation Attack), where attackers use the man in the middle (MITM) to attack the third phase of four-way handshake interaction verification of WPA / WPA2 protocol within the Wi-Fi range. At this point the victim has installed a key, after replaying attacks by MITM, it will force victims to use the previous keystream to encrypt data. Since WPA / WPA2 is symmetric encryption, it is possible to obtain reusable keystreams through ciphertext of simple plaintext. The attacker can decrypt Wi-Fi traffic data once the keystream is obtained.

2.1 Principles

The core use of this vulnerability is the key reinstallation, which is based on the four-way handshake process when the connection is established in the WPA / WPA2 protocol. During the four-way handshake, the AP and the client will negotiate an encryption key for encrypting the next communication data, and the client verifies the MIC after receiving the third handshake from the AP (message 3). If the encryption key is installed correctly to encrypt the normal data frame, and send a response to the AP as a confirmation. According to the protocol rules, if the AP can not receive the confirmation correctly, it will cause the data to be retransmitted and resend the message 3. The client will reinstall the same session key each time message 3 is received. An attacker can use this handshake process to send a message 3 in a violent incremental manner, thereby forcing the resetting of the number of incremental packets used by the data privacy protocol and receiving the replay counter, resulting in key reuse. An attacker could replay, decrypt and / or fake a packet in this way.

KRACK attacks can be divided into four kinds of scenarios:

1) Key reinstallation attack when the client (victim) accepts the unencrypted retransmission of message 3;

2) Key reinstallation attack when the victim only accepts the encrypted retransmission of message 3;

3) Instant key installation for handshake attack on the group key;

4) The delay key installation of the group key handshake.

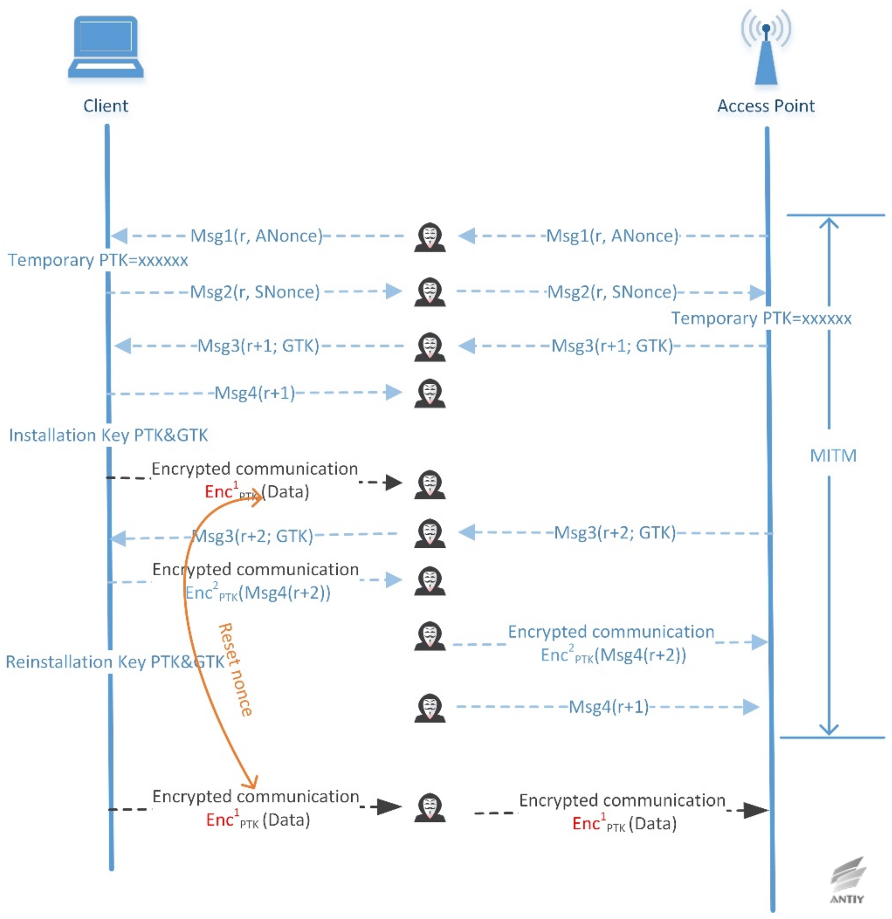


Figure 2-3 Key reinstallation attack when the client (victim) accepts the unencrypted retransmission of message 3

Figure 2-3 shows the attacker decrypts a ciphertext packet attack process. If the attacker knows the plaintext of the first transmitted ciphertext packet, the keystream used to encrypt the plaintext data can be recovered. Since the design of the four-way handshake protocol allows PTK and GTK to be reinstalled by retransmitting the message 3 and resetting the nonce value of the packet to be sent, therefore the client will encrypt the next packet with the same key stream, leading to the next ciphertext packet sent by the client decrypted.

The above decryption process is based on the premise that the attacker knows the first packet plaintext, but the attacker sometimes can not predict all the field values in the packet (such as a random field that may exist), so the attacker may need to retransmit the message 3 to collect more data for decryption. Figure 2-3 shows only the attack process to restore a ciphertext packet, but the attacker can pass multiple retransmissions and select the retransmission time (waiting for the client to send enough data before retransmission), or even force the client to re-execute the four-way handshake protocol to achieve the purpose of decrypting multiple packets through authenticating the client.

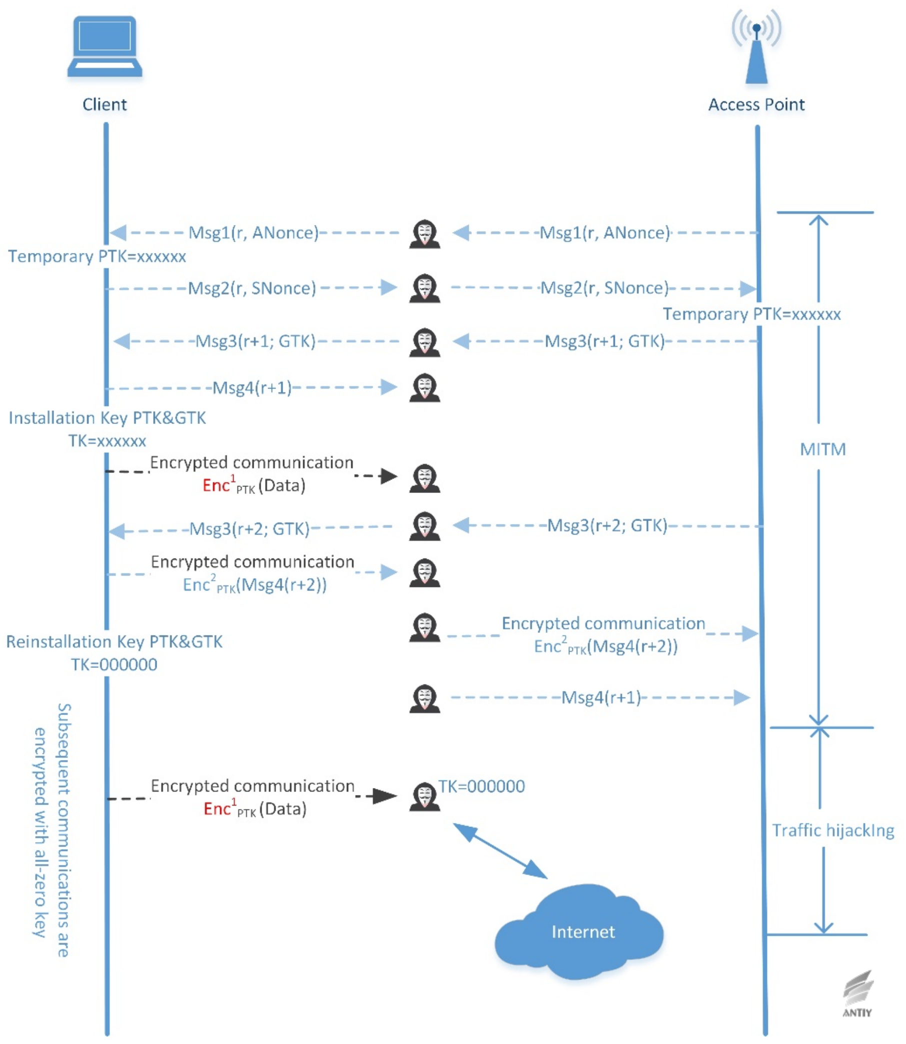


Figure 2-4 Use all-zero key to replace real key attack

when the unencrypted retransmission of message 3 is accepted

Since the error implementation of key reinstallation attack after version 2.4 and 2.5 of wpa_supplicant (Android 6 and the above) receives the retransmitted message 3, so that encryption key TK is set to all zero after that. This error greatly simplifies the ciphertext recovery attack, based on the all-zero key, the attacker can decrypt the subsequent packets sent by the client without the need of knowing the plaintext. Considering the above condition, the attacker can achieve the client traffic hijacking, monitoring and tampering all the data sent by the client after forcing it to use all-zero key by MITM.

2.2 The cipher analysis of nonce reuse in KRACK attack

The consequences of the Nonce reuse are closely related to the confidentiality protocol of the data used. The data encryption algorithms adopted by TKIP, CCMP and GCMP are stream cipher RC4, authenticated encryption algorithm AES-CCM and AES-GCM, respectively. The encryption part of AES-CCM and AES-GCM is based on the stream encryption of CTR mode. It can be considered that all these three protocols adopted stream encryption, that is, the key stream generated by the plaintext data and the algorithm get XOR bit by bit to achieve ciphertext data. The problem with stream encryption is always generating the same keystream when reusing nonce with key-fixed conditions. This feature can be used to decrypt the packet.

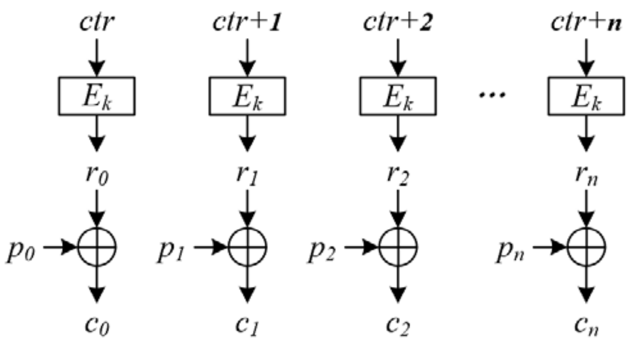


Figure 2-5 The diagram of the encryption process in CTR mode

The diagram above shows the process of encrypting plaintext message $P = p_0 \parallel p_1 \parallel \cdots \parallel p_n$ by using the key k and counter *ctr* in CTR mode, of which p_i *and* $0 \le i \le n$ represent the packet length, respectively (When using AES algorithm, it is 128 bit, and the length of the plaintext message is not necessarily the integer multiples of the packet length). It is worth noting that some of the details involved in the application of the CTR mode for encryption are not shown in the CCM and GCM modes. In the case of correct use, the value of the counter *ctr* is continuously accumulative (the value does not repeat!) and generates the key stream $r_0 \parallel r_1 \parallel \cdots \parallel r_n$ with strong pseudo random characteristics under the action of algorithm and key. Then, the correspondent ciphertext is:

$$C = c_0 \parallel c_1 \parallel \cdots \parallel c_n = p_0 \oplus r_0 \parallel p_1 \oplus r_1 \parallel \cdots \parallel p_n \oplus r_n$$

The encryption process based on the stream cipher RC4 in TKIP is similar with this, so we will not repeat here.

In the attack of KRACK, it is possible to force the victim to reuse nonce by replaying message 3, resulting in the repetition of the same counter *ctr* value in the above encryption process. In the condition of the same key and the same counter *ctr*, it will generate the same value of key stream $r_0 \parallel r_1 \parallel \cdots \parallel r_n$. Then the attacker can decrypt the packet accordingly. We use *KeyStream* to represent the key stream, P_1 and P_2 to represent the plaintext data. If the *KeyStream*, *P1*, and *P2* have the same bit length, the ciphertexts corresponding to the plaintext are:

$$C_1 = P_1 \wedge KeyStream$$
$$C_2 = P_2 \wedge KeyStream$$

of which ^ expresses the XOR operation bit by bit. The attackers can collect the ciphertext C_1 and C_2. If the attackers know that the ciphertext corresponds to the plaintext P_1, then they can restore the information of the plaintext P_2:

$$:P_2 = C_2 \wedge keystream = C_2 \wedge (P_1 \wedge C_1)$$

In practice, packets with known content can be found usually, so it can be considered that the ciphertext packets can be decrypted according to the above process when reusing nonce in the condition of fixed key. Even if the packets with known content cannot be accessed, it may be decrypted to restore the plaintext in the condition of having enough knowledge of the message type (for instance, the message belongs to English character). **It is worth noting that the nonce reuse will cause the ciphertext packets to be decrypted, but it would not lead to the leak of key TK, PTK, PMK and the login password of WiFi. Therefore, the WPA2 password system is only bypassed and has not been breached.** The security of block cipher algorithm (AES) itself has ensured the encryption key k cannot be leaked even in the condition of known input $ctr \parallel ctr+1 \parallel \cdots \parallel ctr+n$ and output $r_0 \parallel r_1 \parallel \cdots \parallel r_n$.

The three data encryption protocols of TKIP, CCMP and GCMP also provide data integrity protection except for data confidentiality. However, reusing nonce in different data encryption conditions will bring different levels of security risks in terms of data integrity protection.

When using the TKIP protocol, the attacker can further attack the Michael algorithm to access the corresponding MIC key after decrypting the complete TKIP packet (including the MIC field). This is due to the vulnerability of the Michael algorithm itself. In the conditions of given plaintext data and MIC value, the attacker can restore the MIC key. Then the attacker can forge a data frame in the data transmission direction (TKIP uses different MIC keys in different data transmission directions) with the recovered MIC key.

When using CCMP protocol, although studies have demonstrated the possibility of data forgery attack in the condition of reusing nonce, the attack is on the theoretical level and difficult to generate real forgery packet in practice, which can only perform replay attacks and packet decryption.

The security problem caused by nonce reuse is most serious when using the GCMP protocol. The nonce reuse enables the attacker to recover the authentication key (H) in the GCM mode. Since the GCMP protocol uses the same key for data protection in both directions of the data transmission, this endows the attacker the ability to forge packet in both directions of the data transmission. As a working mode of authentication encryption, the GCM mode is a combination of the CTR encryption algorithm and the GHASH authentication algorithm. The CTR algorithm section encrypts directly by adopting the key k passed to the GCM mode, while the authentication subkey H required for the GHASH operation is the 128-bit ciphertext value obtained by the AES algorithm using the key k to encrypt all 128 bits of all 0 plaintext. The French cryptologist Joux pointed out that the attacker can recover the value of the authentication subkey H (note that the input key of GCM cannot be inferred from the value of H, which is guaranteed by the security of the AES algorithm itself) when reusing nonce. After the attacker gets the value of H, the data integrity provided by GCMP performs practically no function, so the attacker can forge the packet.

Overall, the KRACK attack has a significant impact on TKIP and GCMP, and attackers can replay, decrypt, and forge packets. Though the attackers cannot forge CCMP, as long as they can get the serial number, they could hijack the TCP stream and inject the malicious data based on the characteristics of the TCP/IP protocol, which also results in serious consequences.

3 Vulnerability Impacts and Response

This vulnerability allows adversary to decrypt Wi-Fi flow data, reassemble packet, hijack TCP connections, and inject malicious content into HTTP connections, etc. KRACK attacks are common, which apply to all types of devices that connect or use the Wi-Fi network through WPA/ WPA2. It is effective to personal and corporate networks and any cryptographic kits (WPA-TKIP, AES-CCMP and GCMP), including Android, iOS, Windows, Linux, MacOS, OpenBSD, embedded and Internet of Things (IoT) devices, especially for Android 6 and more advanced versions. Researchers have indicated that it may make a "perfect attack" targeted to wpa_supplicant of Android 6 system client, the attack technology of which is easy to implement. According to mobile security statistics of Antiy, Android 6 accounts for about 41.9% of Android devices.

Due to the large-scale impact of this vulnerability, vendors have initiated responses or released patches. Responses made by some vendors are shown as follows:

Table 3-1Responses Made by Some Vendors[4]

Vendor' s Name	Official Response	Notes	Date of Last Detection	Date of Recent Update
Apple	No official response; an unofficial response released through twitter.	Apple has confirmed wpa2 vulnerability , and fixed the vulnerability in iOS, acOS, tvOS, and watchOS of beta version.	Oct. 17, 2017	Oct. 17, 2017
Arch Linux	wpa_supplicant,hostapd	N/A	Oct. 16, 2017	Oct. 16, 2017
AVM	Link	This security issue is being investigated and will be updated when needed to be released. EOM	Oct. 17, 2017	Oct. 17, 2017

		and EOS products will also be updated according to the content in the link.		
Cisco	Link	Cisco said wireless products were affected by these vulnerabilities.	Oct. 16, 2017	Oct. 16, 2017
Debian	Link	Patches are added to fix the WPA protocol vulnerability.（CVE-2017-13077，CVE-2017-13078，CVE-2017-13079，CVE-2017-13080，CVE-2017-13081，CVE-2017-13082，CVE-2017-13086 ，CVE-2017-13087，CVE-2017-13088）.	Oct. 16, 2017	Oct. 16, 2017
Fedora	Link	Fixed version, which can be installed manually.	Oct. 17, 2017	Oct. 17, 2017
FortiNet	Link	FortiAP 5.6.1 is used to fix the following vulnerabilities: CVE-2017-13077 CVE-2017-13078 CVE-2017-13079 CVE-2017-13080 CVE-2017-13081 CVE-2017-13082	Oct. 16, 2017	Oct. 16, 2017
FreeBSD Project	Link	It is based on wpa_supplicant 2.5. It is recommended to use a wired connection or install the security/ wpa_supplicant port or package until the fix patch is complete.	Oct. 16, 2017	Oct. 16, 2017
HPE Aruba	Link	N/A	Oct. 16, 2017	Oct. 16, 2017
Intel Corporation	Link	N/A	Oct. 16, 2017	Oct. 16, 2017
LineageOS	Link	N/A	Oct. 17, 2017	Oct. 17, 2017
Linux	Patch Link	wpa_supplicant 2.4 and more advanced version have been affected. wpa_supplicant v2.6 has been affected.	Oct. 16, 2017	Oct. 16, 2017
Microsoft	Patch Link	When you click the link, accept EULA, and then click the link again.	Oct. 16, 2017	Oct. 16, 2017
Mikrotik	Link	The fixed version was released last week, so if you often upgrade the device, you do not need further operations.	Oct. 16, 2017	Oct. 16, 2017
OpenBSD	Link	For OpenBSD 6.1 and 6.0, an errata patch for the wireless stack has been released. A state transition error may cause the old WPA key reinstallation. The binary update for amd64 and i386 platform can be obtained through the syspatch procedure. The source code patch can be found on the	Oct. 16, 2017	Oct. 16, 2017

		corresponding errata page. Since this will affect the kernel, it will need to be restarted after patching.		
Red Hat, Inc.	This vulnerability may impact the wpa_supplicant version provided by Red Hat Enterprise Linux 6 和 7 LINK, Link	N/A	Oct. 16, 2017	Oct. 16, 2017
Sophos AP	Link	N/A	Oct. 17, 2017	Oct. 17, 2017
SUSE/openSUSE	Link	N/A	Oct. 16, 2017	Oct. 16, 2017
TP-Link	Link	N/A	Oct. 17, 2017	Oct. 17, 2017
Ubuntu	Link	In Ubuntu 17.04，Ubuntu 16.04 LTS and Ubuntu 14.04 LTS, Wpasupplicant and hostapd can be directly updated.	Oct. 16, 2017	Oct. 16, 2017
WatchGuard	Link	Oct., 15, 2017 (Sunday): AP120,320,322,420 ： 8.3.0-657 version, only cloud model. Oct., 30, 2017 (Monday): AP300：2.0.0.9 version ， AP100,102,200 ， 1.2.9.14 version，AP120,320,322,420：8.3.0-657 version，non-cloud (GWC Model).	Oct. 16, 2017	Oct. 16, 2017
WiFi Alliance	Link	Users should refer to the Wi-Fi device vendor's website or security advice to determine whether their device has been affected and has installed available updates.	Oct. 16, 2017	Oct. 16, 2017

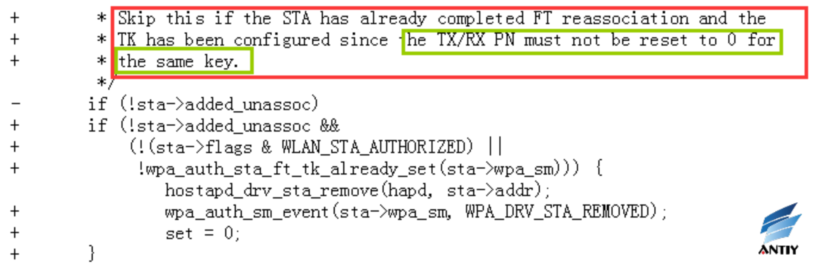
Note:

If the link in the table cannot be opened, you can get it from reference 4.

We have analyzed eight patches mentioned in the above table, the role of the principle of each patch is as follows:

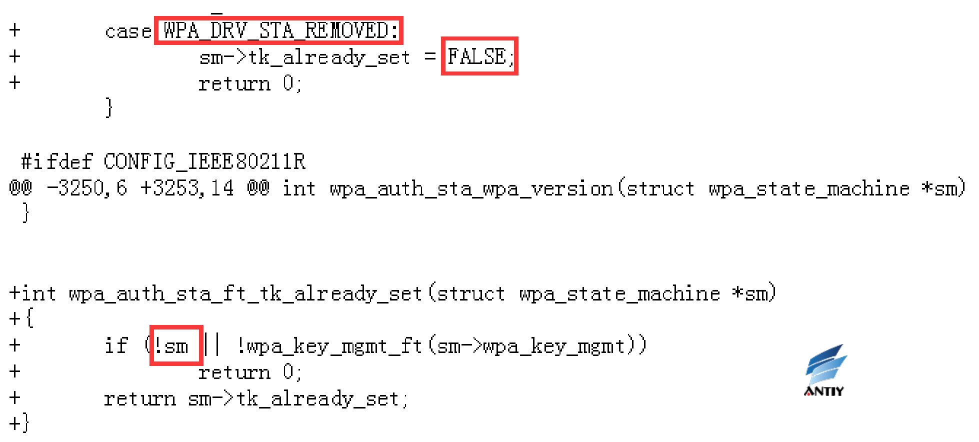
1. rebased-v2.6-0001-hostapd-Avoid-key-reinstallation-in-FT-handshake.patch

TK should be avoided being reinstall to the driver part in the handshake stage Reassociation-Response, thus to prevent the key reinstallation attack.



```
+           * Skip this if the STA has already completed FT reassociation and the
+           * TK has been configured since the TX/RX PN must not be reset to 0 for
+           * the same key.
            */
-          if (!sta->added_unassoc)
+          if (!sta->added_unassoc &&
+              (!(sta->flags & WLAN_STA_AUTHORIZED) ||
+               !wpa_auth_sta_ft_tk_already_set(sta->wpa_sm))) {
                  hostapd_drv_sta_remove(hapd, sta->addr);
+                 wpa_auth_sm_event(sta->wpa_sm, WPA_DRV_STA_REMOVED);
+                 set = 0;
+          }
```

In addition, the configuration is allowed only when the TK confirmation has been uninstalled and the reinstallation is not allowed (only one configuration is allowed)

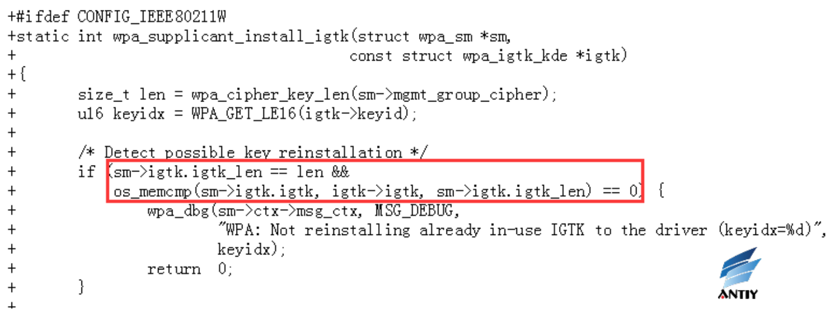


```
+       case WPA_DRV_STA_REMOVED:
+               sm->tk_already_set = FALSE;
+               return 0;
        }

 #ifdef CONFIG_IEEE80211R
@@ -3250,6 +3253,14 @@ int wpa_auth_sta_wpa_version(struct wpa_state_machine *sm)
 }


+int wpa_auth_sta_ft_tk_already_set(struct wpa_state_machine *sm)
+{
+       if (!sm || !wpa_key_mgmt_ft(sm->wpa_key_mgmt))
+               return 0;
+       return sm->tk_already_set;
+}
```

2. rebased-v2.6-0002-Prevent-reinstallation-of-an-already-in-use-group-ke.patch

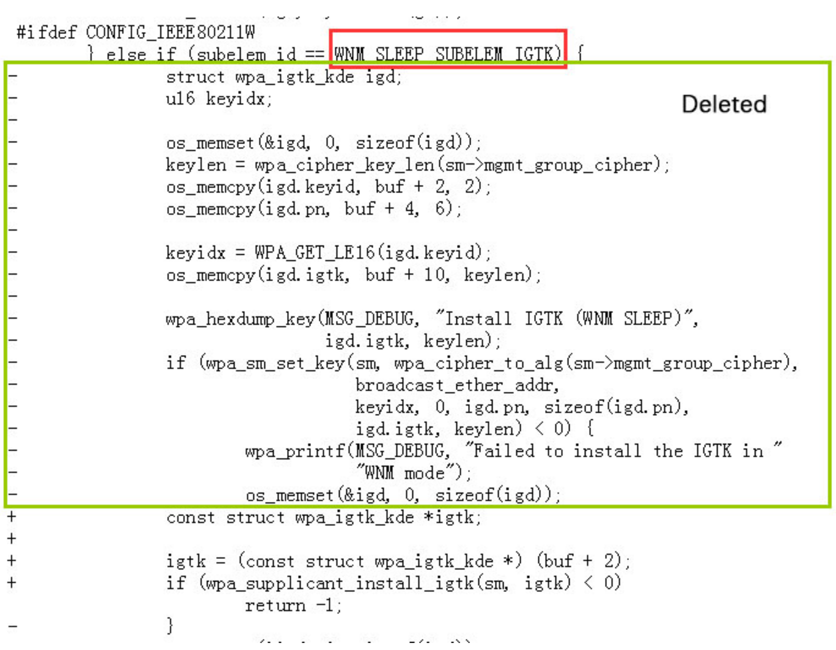
Tracking the GTK and IGTK currently in use, when a message of the first handshake phase or a WNM-sleep mode response is received (possibly retransmitted), not installing a new key if the key is already in use. This prevents the attacker from cheating the client to reset or change the behavior of the sequence counter associated with the group key.

When installing igtk:



```
+#ifdef CONFIG_IEEE80211W
+static int wpa_supplicant_install_igtk(struct wpa_sm *sm,
+                                      const struct wpa_igtk_kde *igtk)
+{
+       size_t len = wpa_cipher_key_len(sm->mgmt_group_cipher);
+       u16 keyidx = WPA_GET_LE16(igtk->keyid);
+
+       /* Detect possible key reinstallation */
+       if (sm->igtk.igtk_len == len &&
+           os_memcmp(sm->igtk.igtk, igtk->igtk, sm->igtk.igtk_len) == 0) {
+               wpa_dbg(sm->ctx->msg_ctx, MSG_DEBUG,
+                       "WPA: Not reinstalling already in-use IGTK to the driver (keyidx=%d)",
+                       keyidx);
+               return  0;
+       }
+
```

When WNM - sleep mode responds：

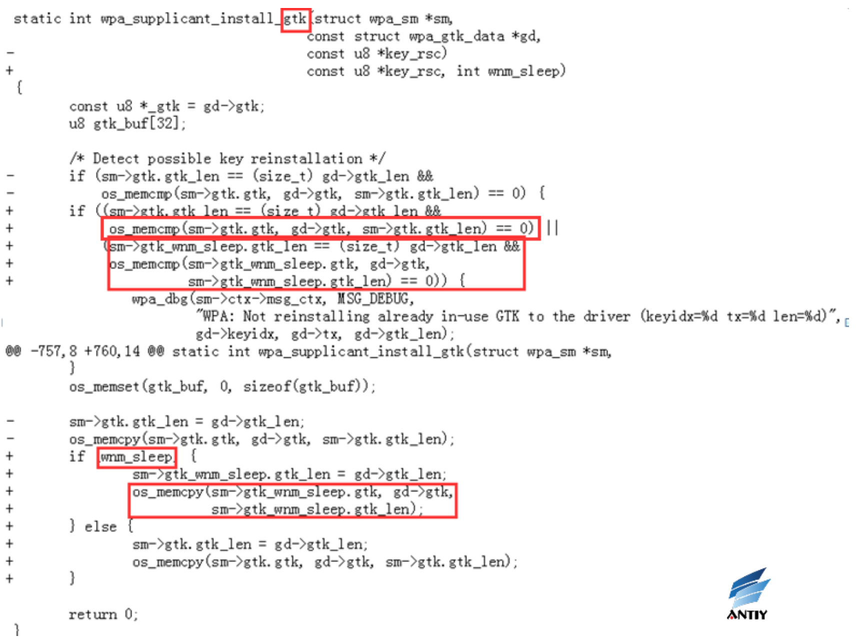


```
#ifdef CONFIG_IEEE80211W
        } else if (subelem_id == WNM_SLEEP_SUBELEM_IGTK) {
-               struct wpa_igtk_kde igd;
-               u16 keyidx;
-
-               os_memset(&igd, 0, sizeof(igd));
-               keylen = wpa_cipher_key_len(sm->mgmt_group_cipher);
-               os_memcpy(igd.keyid, buf + 2, 2);
-               os_memcpy(igd.pn, buf + 4, 6);
-
-               keyidx = WPA_GET_LE16(igd.keyid);
-               os_memcpy(igd.igtk, buf + 10, keylen);
-
-               wpa_hexdump_key(MSG_DEBUG, "Install IGTK (WNM SLEEP)",
-                               igd.igtk, keylen);
-               if (wpa_sm_set_key(sm, wpa_cipher_to_alg(sm->mgmt_group_cipher),
-                                  broadcast_ether_addr,
-                                  keyidx, 0, igd.pn, sizeof(igd.pn),
-                                  igd.igtk, keylen) < 0) {
-                       wpa_printf(MSG_DEBUG, "Failed to install the IGTK in "
-                                  "WNM mode");
-                       os_memset(&igd, 0, sizeof(igd));
+               const struct wpa_igtk_kde *igtk;
+
+               igtk = (const struct wpa_igtk_kde *) (buf + 2);
+               if (wpa_supplicant_install_igtk(sm, igtk) < 0)
                        return -1;
-               }
```

3. **rebased-v2.6-0003-Extend-protection-of-GTK-IGTK-reinstallation-of-WNM-.patch**

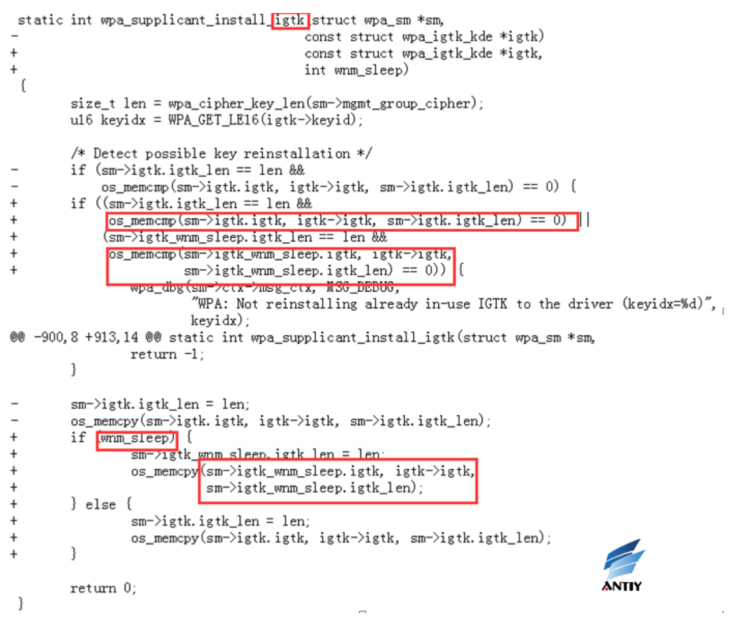
This patch traces the last configured GTK / IGTK value, in conjunction with the EAPOL-Key frame and the WNM-sleep mode frame, because when the GTK / IGTK of these two different mechanisms changes, tracking a single value is not sufficient to detect the possible behavior of the key reconfiguration.

Record Igtk in both modes:



```
 static int wpa_supplicant_install_gtk(struct wpa_sm *sm,
                                       const struct wpa_gtk_data *gd,
-                                      const u8 *key_rsc)
+                                      const u8 *key_rsc, int wnm_sleep)
 {
        const u8 *_gtk = gd->gtk;
        u8 gtk_buf[32];

        /* Detect possible key reinstallation */
-       if (sm->gtk.gtk_len == (size_t) gd->gtk_len &&
-           os_memcmp(sm->gtk.gtk, gd->gtk, sm->gtk.gtk_len) == 0) {
+       if ((sm->gtk.gtk_len == (size_t) gd->gtk_len &&
+            os_memcmp(sm->gtk.gtk, gd->gtk, sm->gtk.gtk_len) == 0) ||
+           (sm->gtk_wnm_sleep.gtk_len == (size_t) gd->gtk_len &&
+            os_memcmp(sm->gtk_wnm_sleep.gtk, gd->gtk,
+                      sm->gtk_wnm_sleep.gtk_len) == 0)) {
                wpa_dbg(sm->ctx->msg_ctx, MSG_DEBUG,
                        "WPA: Not reinstalling already in-use GTK to the driver (keyidx=%d tx=%d len=%d)",
                        gd->keyidx, gd->tx, gd->gtk_len);
@@ -757,8 +760,14 @@ static int wpa_supplicant_install_gtk(struct wpa_sm *sm,
        }
        os_memset(gtk_buf, 0, sizeof(gtk_buf));

-       sm->gtk.gtk_len = gd->gtk_len;
-       os_memcpy(sm->gtk.gtk, gd->gtk, sm->gtk.gtk_len);
+       if (wnm_sleep) {
+               sm->gtk_wnm_sleep.gtk_len = gd->gtk_len;
+               os_memcpy(sm->gtk_wnm_sleep.gtk, gd->gtk,
+                         sm->gtk_wnm_sleep.gtk_len);
+       } else {
+               sm->gtk.gtk_len = gd->gtk_len;
+               os_memcpy(sm->gtk.gtk, gd->gtk, sm->gtk.gtk_len);
+       }

        return 0;
 }
```

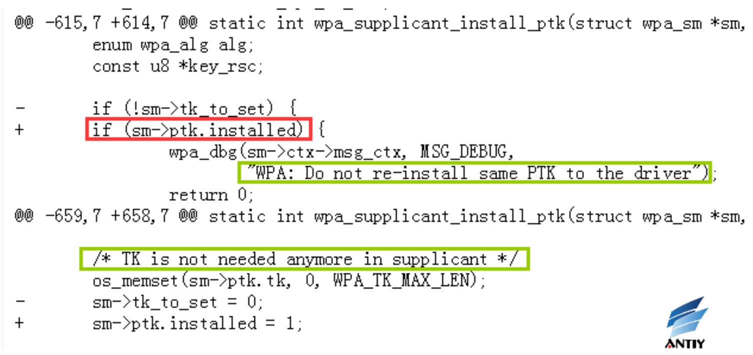
Record Igtk in both modes:



```
 static int wpa_supplicant_install_igtk(struct wpa_sm *sm,
-                                       const struct wpa_igtk_kde *igtk)
+                                       const struct wpa_igtk_kde *igtk,
+                                       int wnm_sleep)
 {
        size_t len = wpa_cipher_key_len(sm->mgmt_group_cipher);
        u16 keyidx = WPA_GET_LE16(igtk->keyid);

        /* Detect possible key reinstallation */
-       if (sm->igtk.igtk_len == len &&
-           os_memcmp(sm->igtk.igtk, igtk->igtk, sm->igtk.igtk_len) == 0) {
+       if ((sm->igtk.igtk_len == len &&
+            os_memcmp(sm->igtk.igtk, igtk->igtk, sm->igtk.igtk_len) == 0) ||
+           (sm->igtk_wnm_sleep.igtk_len == len &&
+            os_memcmp(sm->igtk_wnm_sleep.igtk, igtk->igtk,
+                      sm->igtk_wnm_sleep.igtk_len) == 0)) {
                wpa_dbg(sm->ctx->msg_ctx, MSG_DEBUG,
                        "WPA: Not reinstalling already in-use IGTK to the driver (keyidx=%d)",
                        keyidx);
@@ -900,8 +913,14 @@ static int wpa_supplicant_install_igtk(struct wpa_sm *sm,
                return -1;
        }

-       sm->igtk.igtk_len = len;
-       os_memcpy(sm->igtk.igtk, igtk->igtk, sm->igtk.igtk_len);
+       if (wnm_sleep) {
+               sm->igtk_wnm_sleep.igtk_len = len;
+               os_memcpy(sm->igtk_wnm_sleep.igtk, igtk->igtk,
+                         sm->igtk_wnm_sleep.igtk_len);
+       } else {
+               sm->igtk.igtk_len = len;
+               os_memcpy(sm->igtk.igtk, igtk->igtk, sm->igtk.igtk_len);
+       }

        return 0;
 }
```

4. rebased-v2.6-0004-Prevent-installation-of-an-all-zero-TK.patch

Track whether the PTK has been installed to the driver, and TK has been cleared from memory. This prevents the attacker from cheating the client to install all-zero TK.

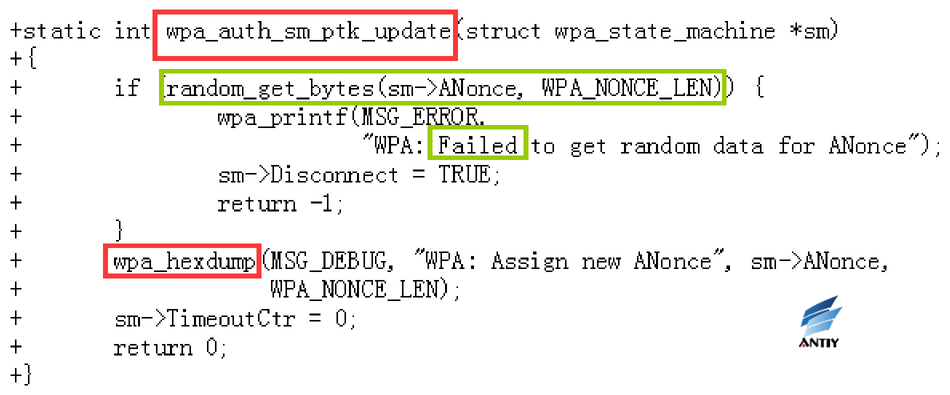


```
@@ -615,7 +614,7 @@ static int wpa_supplicant_install_ptk(struct wpa_sm *sm,
        enum wpa_alg alg;
        const u8 *key_rsc;

-       if (!sm->tk_to_set) {
+       if (sm->ptk.installed) {
                wpa_dbg(sm->ctx->msg_ctx, MSG_DEBUG,
                        "WPA: Do not re-install same PTK to the driver");
                return 0;
@@ -659,7 +658,7 @@ static int wpa_supplicant_install_ptk(struct wpa_sm *sm,

        /* TK is not needed anymore in supplicant */
        os_memset(sm->ptk.tk, 0, WPA_TK_MAX_LEN);
-       sm->tk_to_set = 0;
+       sm->ptk.installed = 1;
```

5. rebased-v2.6-0005-Fix-PTK-rekeying-to-generate-a-new-ANonce.patch

The authorization state machine for PTK rekeying bypasses the authenication2 state while the nonce is generated, and enters the PKT-START state directly, since there is no need to re-confirm the PMK at this time, it may result in nonce not being "random" or other problems.

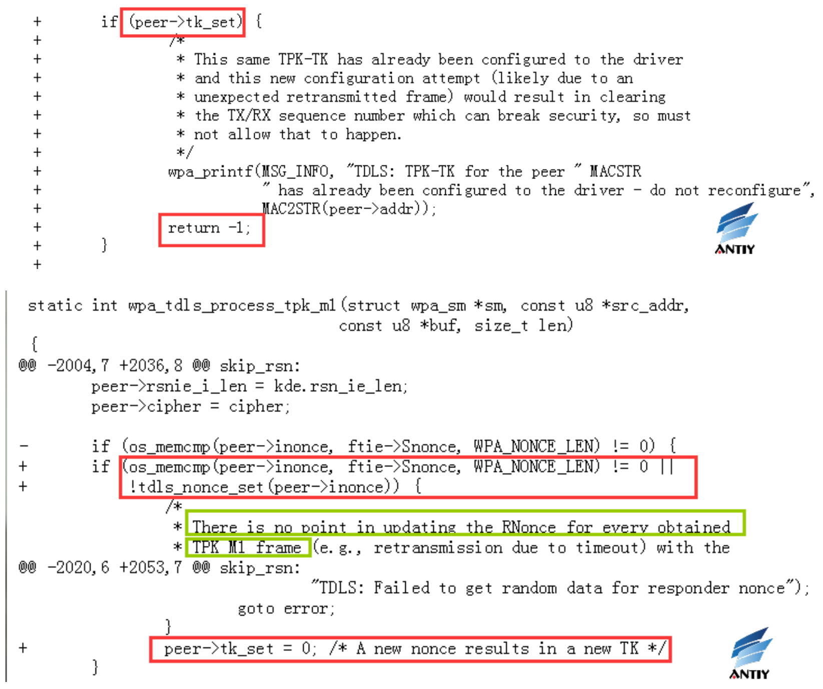
For this problem, when you switch to the PTKSTART state, a new ANonce is generated.



```
+static int wpa_auth_sm_ptk_update(struct wpa_state_machine *sm)
+{
+       if (random_get_bytes(sm->ANonce, WPA_NONCE_LEN)) {
+               wpa_printf(MSG_ERROR,
+                          "WPA: Failed to get random data for ANonce");
+               sm->Disconnect = TRUE;
+               return -1;
+       }
+       wpa_hexdump(MSG_DEBUG, "WPA: Assign new ANonce", sm->ANonce,
+                   WPA_NONCE_LEN);
+       sm->TimeoutCtr = 0;
+       return 0;
+}
```

6. rebased-v2.6-0006-TDLS-Reject-TPK-TK-reconfiguration.patch

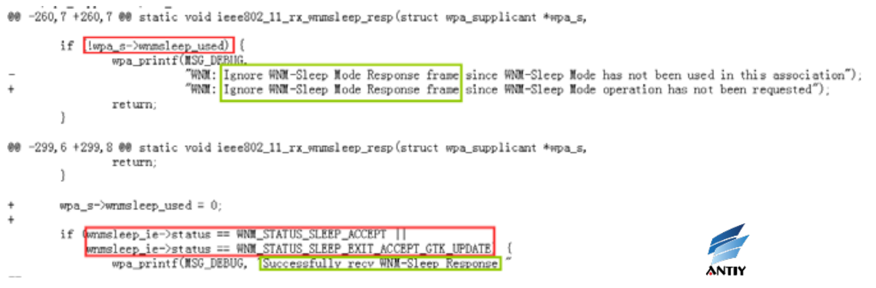
When TPK-TK is successfully configured, it is forbidden to reconfigure the same parameters to the drive.



```
+       if (peer->tk_set) {
+               /*
+                * This same TPK-TK has already been configured to the driver
+                * and this new configuration attempt (likely due to an
+                * unexpected retransmitted frame) would result in clearing
+                * the TX/RX sequence number which can break security, so must
+                * not allow that to happen.
+                */
+               wpa_printf(MSG_INFO, "TDLS: TPK-TK for the peer " MACSTR
+                          " has already been configured to the driver - do not reconfigure",
+                          MAC2STR(peer->addr));
+               return -1;
+       }
+

 static int wpa_tdls_process_tpk_m1(struct wpa_sm *sm, const u8 *src_addr,
                                    const u8 *buf, size_t len)
 {
@@ -2004,7 +2036,8 @@ skip_rsn:
        peer->rsnie_i_len = kde.rsn_ie_len;
        peer->cipher = cipher;

-       if (os_memcmp(peer->inonce, ftie->Snonce, WPA_NONCE_LEN) != 0) {
+       if (os_memcmp(peer->inonce, ftie->Snonce, WPA_NONCE_LEN) != 0 ||
+           !tdls_nonce_set(peer->inonce)) {
                /*
                 * There is no point in updating the RNonce for every obtained
                 * TPK M1 frame (e.g., retransmission due to timeout) with the
@@ -2020,6 +2053,7 @@ skip_rsn:
                                   "TDLS: Failed to get random data for responder nonce");
                        goto error;
                }
+               peer->tk_set = 0; /* A new nonce results in a new TK */
        }
```

7. rebased-v2.6-0007-WNM-Ignore-WNM-Sleep-Mode-Response-without-pending-r.patch

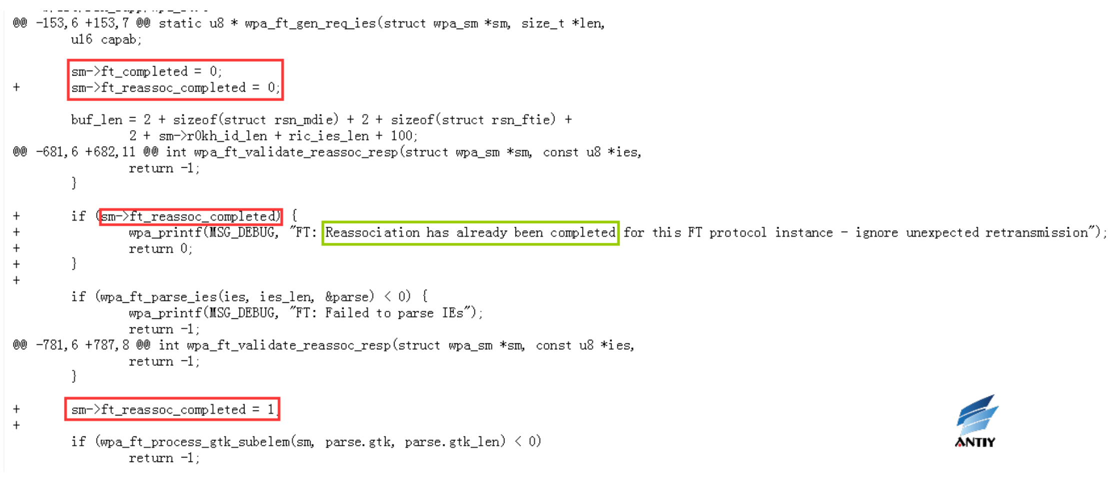
If WNM-sleep mode is not already used, the corresponding WNM-sleep mode request is ignored. This avoids unexpected retransmission of data frames.



```
@@ -260,7 +260,7 @@ static void ieee802_11_rx_wnmsleep_resp(struct wpa_supplicant *wpa_s,

        if (!wpa_s->wnmsleep_used) {
                wpa_printf(MSG_DEBUG,
-                          "WNM: Ignore WNM-Sleep Mode Response frame since WNM-Sleep Mode has not been used in this association");
+                          "WNM: Ignore WNM-Sleep Mode Response frame since WNM-Sleep Mode operation has not been requested");
                return;
        }

@@ -299,6 +299,8 @@ static void ieee802_11_rx_wnmsleep_resp(struct wpa_supplicant *wpa_s,
                return;
        }

+       wpa_s->wnmsleep_used = 0;
+
        if (wnmsleep_ie->status == WNM_STATUS_SLEEP_ACCEPT ||
            wnmsleep_ie->status == WNM_STATUS_SLEEP_EXIT_ACCEPT_GTK_UPDATE) {
                wpa_printf(MSG_DEBUG, "Successfully recv WNM-Sleep Response "
```

8. rebased-v2.6-0008-FT-Do-not-allow-multiple-Reassociation-Response-fram.patch

The driver part opens a connection event unless the client explicitly requests a new one. However, reconfiguring the same pair of keys or group keys can lead to nonce reuse problems, so additional checks are required to avoid malicious attacks, including accidental receipt of retransmitted packets due to some environmental factors.



```
@@ -153,6 +153,7 @@ static u8 * wpa_ft_gen_req_ies(struct wpa_sm *sm, size_t *len,
        u16 capab;

        sm->ft_completed = 0;
+       sm->ft_reassoc_completed = 0;

        buf_len = 2 + sizeof(struct rsn_mdie) + 2 + sizeof(struct rsn_ftie) +
                2 + sm->r0kh_id_len + ric_ies_len + 100;
@@ -681,6 +682,11 @@ int wpa_ft_validate_reassoc_resp(struct wpa_sm *sm, const u8 *ies,
                return -1;
        }

+       if (sm->ft_reassoc_completed) {
+               wpa_printf(MSG_DEBUG, "FT: Reassociation has already been completed for this FT protocol instance - ignore unexpected retransmission");
+               return 0;
+       }
+
        if (wpa_ft_parse_ies(ies, ies_len, &parse) < 0) {
                wpa_printf(MSG_DEBUG, "FT: Failed to parse IEs");
                return -1;
@@ -781,6 +787,8 @@ int wpa_ft_validate_reassoc_resp(struct wpa_sm *sm, const u8 *ies,
                return -1;
        }

+       sm->ft_reassoc_completed = 1
+
        if (wpa_ft_process_gtk_subelem(sm, parse.gtk, parse.gtk_len) < 0)
                return -1;
```

4 Conclusion

KRACK vulnerabilities are mainly used to attack the four-way handshake process of WPA / WPA2. Instead of using AP access point, this kind of attack is against the client. Therefore, it is possible for the user not to update their routers. For ordinary home users, they should pay more attention to the security bulletins from endpoint vendors, and update the configuration or patch, giving priority to update the laptop and smart phones and other clients.

The use of the vulnerability did not destroy the cryptosystem itself, what is actually attacked is the implementation process, so this kind of attack can almost bypass all the security monitoring equipment. In a well-implemented network environment, this vulnerability can be used to make foundation for the follow-up attacks through a good implementation of Wi-Fi.

It is relatively easy for most WPA2 home and commercial wireless application users to upgrade the client. However, it will have a huge impact on millions of IoT wireless devices that are difficult to update. Please keep vigilant, we will continue to pay attention to the relevant events and make active response.

Appendix 1: References

[1] Mathy Vanhoef, Frank Piessens. Key Reinstallation Attacks.

https://www.krackattacks.com/

[2] Mathy Vanhoef, Frank Piessens. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2.

https://papers.mathyvanhoef.com/ccs2017.pdf

[3] Wikipedia. WPA.

https://zh.wikipedia.org/wiki/WPA

[4] Kristate, Github. Vendor Response.

https://github.com/kristate/krackinfo#vendor-response-complete

Appendix 2: About Antiy

Antiy is a leading provider of threat detection and defense technology, whose enterprise mission is to enhance users' ability to deal with cyberspace threats and improve their awareness of threats. Relying on independent advanced core technologies including series products of threat detection engines and experts team, Antiy can provide users with products, solutions and services related to endpoint protection, traffic monitoring, threat intelligence and situational awareness.

Antiy has fostered nationwide detection and monitoring capability with our products and services covering multiple countries. With effective combination of techniques and products of both big data analysis and security visualization, Antiy expands the group work competence of engineers and shortens the product response cycle by massive automation sample analysis platform. With years’ continual accumulation of massive security threat knowledge library, Antiy promotes the solution of situational awareness and monitoring and early warning that targets against APT and at scale network and critical infrastructure, combining with the experience of integrated application of big data analysis and security visualization.

More than a hundred famous security vendors and IT vendors select Antiy as their partner of detection capability. The antivirus engine of Antiy has provided security protection for nearly a hundred thousand network devices and security devices and more than eight hundred million mobile phones. The mobile detection engine of Antiy was the first Chinses product that won AV-TEST reward in the world, it is the only one whose detection rate is 100% for twice in the Mobile Security Product Test 2015 organized by the international certified authority AV-C.

The technical strength of Antiy has been recognized by industry management organizations, customers and partners. Antiy has consecutively been awarded the qualification of national security emergency support unit five times and one of the six of CNNVD first-level support units.

Antiy is the significant enterprise node of China emergency response system, which has provided early warning, in-depth analysis or systematic solutions for major security incidents, such as Code Red II, Dvldr, Stuxnet, Bash Shellcode, Sandworm, Equation, White Elephant, WannaCry, etc.

More information about Antiy Labs:	http://www.antiy.net
More information about enterprise security company:	http://www.antiy.cn
More information about Antiy AVL TEAM:	http://www.avlsec.com

Appendix 2: About Lianshi

Lianshi Networks is an innovative service provider focused on business application security and data security. It started the CASB implementation model based on the entrusted security agent technology, and independently developed the CipherGateway service application security gateway, which can integrate the security mechanism (closely related to businesses) to the application system without modifying it. This can nearly build the security capabilities inside for the application, so that enterprises can enrich the application system security features in an unprecedented way, to deal with business application security and data security risks from the start.

More information about Lianshi: http://www.ciphergateway.com

Wechat ID : CipherGateway