Major Vulnerabilities Discovered and Patched in Realtek RTL8195A Wi-Fi Module

By Uriya Yavniely | February 3, 2021
12 min read

In a recent supply chain security assessment, the JFrog security research team (formerly Vdoo) analyzed multiple networking devices for security vulnerabilities and exposures. During the analysis we discovered and responsibly disclosed six major vulnerabilities in Realtek's RTL8195A Wi-Fi module that these devices were based on. An attacker that exploits the discovered vulnerabilities can gain remote root access to the Wi-Fi module, and from there very possibly hop to the application processor as well (as the attacker has complete control of the device's wireless communications).

The RTL8195 module is an extremely compact, low-power Wi-Fi module targeted at embedded devices. It has supported software from major vendors such as ARM, Samsung, Google, Amazon and more. For example, according to AWS it is used in a myriad of industries such as:

- Agriculture
- Automotive
- Energy
- Gaming
- Healthcare
- Industrial
- Security
- Smart Home

In this blog post, we discuss the technical details of the vulnerabilities, share background on the vulnerable component, offer guidance on how to detect and resolve it, and discuss the context of how these vulnerabilities were discovered through supply chain security assessment.

Technical Overview

The RTL8195A is a standalone Wi-Fi hardware module which is being used in many low-power applications:

Realtek supplies their own "Ameba" API to be used with the device, which allows any developer to communicate easily via Wi-Fi, HTTP, mDNS, MQTT and more.

As part of the module's Wi-Fi functionality, the module supports the WEP, WPA and WPA2 authentication modes.

In our security assessment, we have discovered that the WPA2 handshake mechanism is vulnerable to various stack overflow and read out-of-bounds issues.

While the issues have been verified only on the RTL8195A module, we believe they are relevant for the following modules as well:

- RTL8711AM
- RTL8711AF
- RTL8710AF

The most severe issue we discovered is VD-1406, a remote stack overflow that allows an attacker in the proximity of an RTL8195 module to completely take over the module, without knowing the Wi-Fi network password (PSK) and regardless of whether the module is acting as a Wi-Fi access point or client. The attack scenarios are detailed in the next section: "Technical Deep-Dive".

VD-1407 and VD-1411 can also be exploited without knowing the network security key (the PSK, or more accurately the PMK which is derived from it) and by this, a remote code execution or denial of service can be performed on a WPA2 client that uses this Wi-Fi module.

VD-1408, VD-1409 and VD-1410 require the attacker to know the network's PSK as a prerequisite for the attack and can be abused for obtaining remote code execution on WPA2 clients that use this Wi-Fi module.

Realtek has already published a security bulletin and allocated a CVE on VD-1406.

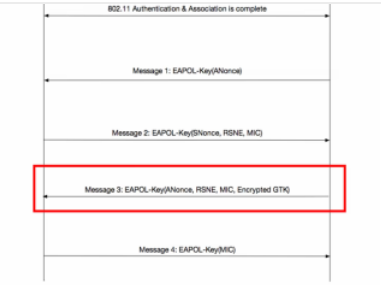
Technical Deep-Dive

VD-1406 (CVE-2020-9395) – Stack-based buffer overflow vulnerability

This vulnerability does not require knowledge of the network's PSK.

This issue allows exploitation of both Wi-Fi client and access point (AP) devices.

As part of the WPA2 4-way handshake, a key exchange occurs at the "EAPOL" frame:



In this key exchange, the Realtek WPA2 client/server calls the ClientEAPOLKeyRecvd and EAPOLKeyRecvd functions, respectively, to process the packet.

The above functions both call the CheckMIC() function, which is responsible for checking the integrity of the MIC part in the EAP packet.

In CheckMIC() an unsafe copy can be triggered:

```
rtl_memcpy(tmpbuf, EAPOLMsgRecvd.Octet, EAPOLMsgRecvd.Length);
```

from: decompilation of CheckMIC() function in lib_wlan.a

Both EAPOLMsgRecvd.Octet and EAPOLMsgRecvd.Length are attacker-controlled.

EAPOLMsgRecvd.Octet contains the Ethernet layer (14 bytes) and the 802.1X Authentication layer EAPOLMsgRecvd.Length comes directly from the 802.1X Authentication's Length field.

tmpbuf is a local buffer with a fixed size of 512 bytes.

It is possible to send a packet in size that is bigger than the size of tmpbuf (512 bytes stack buffer) and cause memcpy() to copy more bytes than allocated, thus causing a stack overflow.

This vulnerability can be exploited to gain remote code execution or denial of service by overwriting the return address of CheckMIC().

Note that there are no mitigating factors in place whatsoever (Stack canaries, ASLR or even non-executable stack) and as such exploitation is trivial.

The vulnerability can be triggered in two different scenarios, in order to attack WPA2 clients and WPA2 access points:

1. The Realtek device victim is the client (CheckMIC() is called from ClientEAPOLKeyRecvd()).
   In this scenario an attacker can exploit a victim client device by:
   1. Sniffing Wi-Fi packets to see which wireless network the victim device is connected to and getting the SSID of that network.
   2. Preparing a malicious access point which will perform the attack and has the exact SSID.
   3. Sending a deauth packet to the victim device and broadcasting louder than the original network in order that the device will connect to the malicious access point.
2. The Realtek device victim is the access point (CheckMIC() is called from EAPOLKeyRecvd())
   In this scenario the attacker client can simply connect to the victim AP and exploit it.

VD-1407 (CVE-2020-25853) – Read out of bounds vulnerability

This vulnerability does not require knowledge of the network's PSK.

This issue resides in the same CheckMIC() function that was mentioned in VD-1406:

```
v3__Octet = EAPOLMsgRecvd.Octet;
v4 = EAPOLMsgRecvd.Octet[20];
v5__key = key;
v6__tmpbuf_at_95 = &tmpbuf[95];
rtl_memcpy(tmpbuf, EAPOLMsgRecvd.Octet, EAPOLMsgRecvd.Length);
rtl_memset(&tmpbuf[95], 0, 0x10u);
v7 = v4 & 7;
// USER-CONTROLLED SIZE -->
v8__len = (unsigned __int16)(ntohs(*(uint16_t *)&tmpbuf[16]) + 4);
if ( v7 == 1 ) {
_rt_md5_hmac_veneer(&tmpbuf[14], v8__len, v5__key, 16, &tmpbuf[95]);
return rtl_memcmp(v6__tmpbuf_at_95, v3__Octet + 95, 0x10u) == 0;
}
if ( v7 != 2 )
return 0;
v6__tmpbuf_at_95 = sha1digest;
_rt_hmac_sha1_veneer((int)&tmpbuf[14], v8__len, (int)v5__key, 16,
(int)sha1digest);
```

from: decompilation of CheckMIC() function in lib_wlan.a

An unsigned short is read from the EAP packet – the 802.1X Authentication's Length field and it ends up in &tmpbuf[16].

v8__len is later used as a size parameter to _rt_md5_hmac_veneer() or _rt_hmac_sha1_veneer(). Those functions will read outside of tmpbuf bounds if v8__len is large enough (tmpbuf has a fixed size of 512 bytes). In some cases this can lead to a denial of service.

Note that there is also an integer overflow when calculating v8__len, but this does not cause any exploitable issues.

VD-1408 (CVE-2020-25854) – Stack-based buffer overflow vulnerability

Exploitation of this issue requires knowledge of the network's Pre-Shared-Key (PSK).

This issue allows exploitation of Wi-Fi client devices.

In DecWPA2KeyData() there is a stack overflow:

```
_rt_arc4_crypt_veneer(&rc4_ctx, tmp2, v7->EapolKeyMsgRecvd.Octet + 95,
v10__keylen);
```

from: decompilation of DecWPA2KeyData() function

Where EapolKeyMsgRecvd contains the 802.1x Authentication layer except the first 4 bytes, v10__keylen is a big endian unsigned short at &EapolKeyMsgRecvd[93] (when calling from ClientEAPOLKeyData()), and tmp2 is a local buffer in size of 257 bytes.

rt_arc4_crypt_veneer() or _AES_UnWRAP_veneer() will decrypt the data in v7->EapolKeyMsgRecvd.Octet + 95 into tmp2 with an encrypted text size of a user controlled keylen (the key is being decrypted, so it's actually the encrypted text size).

A similar issue happens if AES is being used:

```
_AES_UnWRAP_veneer(v8__key, keylen, kek, keklen, tmp2);
```

In both cases – the decryption process can write past of tmp2 size because of keylen is user controlled, which will cause a stack overflow that can lead to a denial of service or code execution.

In order to have controlled data after the decryption process (AES or RC4) we need to know the KEK parameter, otherwise we won't be able to decrypt the data in v7->EapolKeyMsgRecvd.Octet + 95 correctly for overflowing the stack, and then we won't be able to correctly rewrite the return address.

We can calculate the KEK, as ClientEAPOLKeyRecvd() passes the PTK as the KEK parameter. The PTK is derived from the PMK which is the passphrase of the Wi-Fi network. So, assuming we know the passphrase – we can run arbitrary code remotely on a device.

Since DecWPA2KeyData() is called only from the ClientEAPOLKeyRecvd() the attack scenario will be as follows:

1. The attacker knows the passphrase for the WPA2 Wi-Fi network that the victim device is connected to.
2. The attacker sends a deauth packet to the victim device, in most of the cases the device will try to connect again.
3. The attacker sniffs the EAP packets in order to capture anonce and snonce which are needed for calculating the PTK from the passphrase.
4. The attacker prepares a malicious AP which will perform the attack and has the exact SSID and the exact PTK (by using the known passphrase and the captures snonce and anonce). The malicious AP must broadcast louder than the original network.
5. The attacker sends an additional deauth packet to the victim device.
6. The victim device will try to connect to the malicious AP, and by doing so will parse our malicious EAP packet and run arbitrary code.

VD-1409 (CVE-2020-25855) – Stack-based buffer overflow vulnerability

Exploitation of this issue requires knowledge of the network's Pre-Shared-Key (PSK). This issue allows exploitation of Wi-Fi client devices.

In AES_UnWRAP() (from lib_rom.a) there is a stack overflow:

```
v5__cipher_len = (cipher_len + 7) & (cipher_len >> 32);
if ( cipher_len >= 0 )
v5__cipher_len = cipher_len;
v6__cipher = cipher;
v7__aligned_cipher_len = v5__cipher_len >> 3;
nblock = v7__aligned_cipher_len - 1;
aes_set_key(&ctx, key, 128);
memcpy(A, v6__cipher, 8u);
if ( v7__aligned_cipher_len - 1 > 0 )
{
v8__R = R;
v9__cipher_ptr = (int)(v6__cipher + 8);
v10__block_counter = 0;
do
{
v11__R = v8__R;
v12__cipher_ptr = (const void *)v9__cipher_ptr;
++v10__block_counter;
++v8__R;
v9__cipher_ptr += 8;
// STACK OVERFLOW -->
memcpy(v11__R, v12__cipher_ptr, 8u);
}
while ( v10__block_counter != nblock );
}
```

from: decompilation of AES_UnWRAP() function

There is a loop here that copies 8 bytes to array, which is a local buffer that is defined as:

```
unsigned __int8 R[32][8];
```

The problem is that the loop iterations number can be controlled by the user (it is exactly cipher_len/8) and isn't limited by the maximum entries in R (32).

This function is called from DecWPA2KeyData() with a user controlled cipher_len (keylen parameter from DecWPA2KeyData() – see VD-1408 for more details.

VD-1410 (CVE-2020-25856) – Stack-based buffer overflow vulnerability

Exploitation of this issue requires knowledge of the network's Pre-Shared-Key (PSK).

This issue allows exploitation of Wi-Fi client devices.

There is a memcpy in DecWPA2KeyData() that can cause a stack overflow for ClientEAPOLKeyRecvd()'s stack.

The memcpy:

```
rtl_memcpy(v11__kout, v12__tmp2, v10__keylen);
```

from: decompilation of DecWPA2KeyData() function

Where v12__tmp2 is a local buffer in size of 257 bytes, v10__keylen is user controlled (see the previous VD-1408 for more details and exploitability limitations) and v11__kout is passed as the kout parameter to DecWPA2KeyData() from ClientEAPOLKeyRecvd():

```
DecWPA2KeyData(
v5__WPA_STA_INFO,
v9__EapolKeyMsgRecvd + 95,
(unsigned __int16)(v9__EapolKeyMsgRecvd[94] + (v9__EapolKeyMsgRecvd[93] <<
8)),
&v5__WPA_STA_INFO->PTK[16],
16,
decrypted_data)
```

from: decompilation of ClientEAPOLKeyRecvd() function

Where decrypted_data is passed as the kout parameter to DecWPA2KeyData().

(decrypted_data is a local buffer with a fixed size of 128 bytes)

VD-1411 (CVE-2020-25857) – Stack-based buffer overflow vulnerability

This vulnerability does not require knowledge of the network's PSK.

There is a stack overflow in ClientEAPOLKeyRecvd(), which can be abused for denial of service:

```
v22,
v21__wpa_global_info->GTK[(v5__WPA_STA_INFO->EapolKeyMsgRecvd.Octet[2] >> 4)
& 3]) )
goto exit;
v23__EapolKeyMsgRecvd = v5__WPA_STA_INFO->EapolKeyMsgRecvd.Octet;
if ( *v23__EapolKeyMsgRecvd == 2 )
{
// STACK OVERFLOW -->
rtl_memcpy(
decrypted_data,
v21__wpa_global_info->GTK[(v23__EapolKeyMsgRecvd[2] >> 4) & 3],
v23__EapolKeyMsgRecvd[94] + (v23__EapolKeyMsgRecvd[93] << 8));
if ( decrypted_data[0] == 221 && !rtl_memcmp(&decrypted_data[2],
GTK_KDE_OUI_0, 4u) )
{
v24 = decrypted_data[1];
v25 = decrypted_data[6] & 3;
v4->securitypriv.wpa_global_info.GN = v25;
rtl_memcpy(v21__wpa_global_info->GTK[v25], &decrypted_data[8], (unsigned
__int8)(v24 - 6));
}
}
```

from: decompilation of ClientEAPOLKeyRecvd() function

Note that the length of the memcpy is completely user controlled (since v23__EapolKeyMsgRecvd is completely user controlled)

Where v23__EapolKeyMsgRecvd is EapolKeyMsgRecvd.Octet, which contains the 802.1x Authentication layer except the first 4 bytes, the destination buffer is decrypted_data which is a local buffer with a fixed size of 128 bytes. The source buffer is the GTK, but it is not user controlled, therefore we cannot overwrite the return address of ClientEAPOLKeyRecvd() with a valid address. Because of this limitation, this vulnerability cannot be exploited for code execution (only denial of service).

Acknowledgment

We would like to thank Realtek's security team for efficiently and promptly handling this security issue, and for their professional conduct of communication.

FAQ

Q1. How do I know if my device is vulnerable?

Any version built after April 21, 2020 is completely patched against all the above issues.

Any version built after March 3, 2020 is patched against VD-1406, but still vulnerable to all other issues.

The build date can usually be extracted as a simple string, from the binary firmware.

For example, look for any build dates in the firmware by running the following command, and observing a similar output:

```
# strings realtek_firmware.bin | grep -P '2021|2020|2019|2018|2017'
2020/09/30-17:14:47
```

Q2. Which patches can I apply to resolve the issue?

The updated versions of the Ameba SDK can be downloaded from Realtek's website.

The latest version of Ameba Arduino (2.0.8) contains patches for all the above issues.

Q3. How do I mitigate the risk if I cannot update the device's firmware?

Using a strong, private WPA2 passphrase will prevent exploitation of issues VD-1408, VD-1409 and VD-1410

Questions? Thoughts? Contact us at research@jfrog.com for any inquiries related to security vulnerabilities.

In addition to discovering and responsibly disclosing vulnerabilities as part of our day-to-day activities, the JFrog security research team works to enhance software security by empowering organizations to discover vulnerabilities through automated security analysis. For more information and updates on JFrog DevOps Platform security features – click here.

Tags: Realtek wi-fi RTL8195 buffer overflow how-to security-research vulnerability disclosure software vulnerability software supply chain vulnerability management