Talos Vulnerability Report

TALOS-2022-1480

Anker Eufy Homebase 2 libxm_av.so DemuxCmdInBuffer buffer overflow vulnerability

MAY 5, 2022

CVE NUMBER

CVE-2022-26073

SUMMARY

A denial of service vulnerability exists in the libxm_av.so DemuxCmdInBuffer functionality of Anker Eufy Homebase 2 2.1.8.5h. A specially-crafted set of network packets can lead to a device reboot. An attacker can send packets to trigger this vulnerability.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

Anker Eufy Homebase 2 2.1.8.5h

PRODUCT URLS

Eufy Homebase 2 - https://us.eufylife.com/products/t88411d1

CVSSV3 SCORE

7.4 - CVSS:3.0/AV:A/AC:L/PR:N/UI:N/S:C/C:N/I:N/A:H

CWE

CWE-190 - Integer Overflow or Wraparound

DETAILS

The Eufy Homebase 2 is the video storage and networking gateway that enables the functionality of the Eufy Smarthome ecosystem. All Eufy devices connect back to this device, and this device connects out to the cloud, while also providing assorted services to enhance other Eufy Smarthome devices.

Among the `home_security` binary's responsibilities, communications with the cloud and with smarthome devices is the most important. While the binary itself is somewhat opaque with regards to the actual implementation of this, a good chunk of the network functionality is within an imported `libxm_av.so` library. This library normally creates five different network servers, as so:

```
  tcp
    32392 - UDPRecvClient path
    32293 - WifiComSend_Pth
    32295 - WifiComRecv_Pth
    32290 - DspComSvr_Path - recv   ** not seen **
    32292 - DspComSvr_Path - send   ** not seen **

  udp
    32380 - UDPComCreate => UdpRecvSvr_pth
    32392 - UdpSndSvr
```

While the code paths of some of these seem to converge or complement each other, these servers are all related to communications between the Homebase 2 and the smarthome devices. For today's writeup we take a brief look into the `WifiComRecv_Pth` on TCP port 32295. Just like all the other ports, this codepath uses the `getpeermac()` function for defacto authentication, in that a network connection's IP address must have an arp table entry on the `br0` interface to be allowed to talk to any of these ports. Assuming that one has a vulnerabilty to bypass this check (see TALOS-2022-1479, libxm_av.so getpeermac() authentication bypass vulnerability), or if one has compromised one of the smarthome devices that are resident on the `br0` `192.168.32.0/24` network, then one can essentially talk to the Eufy Homebase like any other device that has been paired. So what can one do with this?

Looking at the `WifiComRecv_Pth`, let's take a look at what the message structure looks like:

```
struct WifiComPkt{
    uint32_t magic; //[1]
    uint32_t opcode;
    uint32_t datalen; // [2]
    uint8_t data[]// [3]
}
```

The packet must start with the magic bytes `"\x55\x55\x00\xff"` at [1]. We have an opcode as expected, and then there's a datasize at [2] which determines the length of the array at [3]. All pretty standard. So let's now examine the code handling these messages:

```
void*  WifiComRecv_client_pth(struct mall_20* malloced_arg) {
    //[...]
    0002d068      while (true)
    0002d068          if (g_XM_RUNNING != 0)  // [4]
    0002d14c              int32_t recv_ret
    0002d14c              uint32_t bytes_demuxed
    0002d14c              char* rbufptr
    0002d14c              do
    0002d088                  recv_ret = recv(sockfd: mall20.fd, buf:
&buffer[pWriteOffset], len: 0x400 - pWriteOffset, flags: 0x40) // [5]
    0002d094                  if (recv_ret s> 0)
    0002d0a0                      pWriteOffset = pWriteOffset + recv_ret
    0002d0b4                      bytes_demuxed = DemuxCmdInBuffer(buffer:
&buffer[pReadOffset], inpsize: pWriteOffset - pReadOffset, mall_20: &mall20) // [6]
}
```

The server runs for as long as the `g_XM_RUNNING` global is switched on at [4] and constantly reads packets into a size 0x400 buffer on the stack at [5], which is subsequently parsed inside the `DemuxCmdInBuffer` function at [6].

```
0002cd04  void* DemuxCmdInBuffer(char* buffer, uint32_t inpsize, struct mall_20*
mall_20)
0002cd6c      uint8_t* var_3c
// [...]
0002cde8      int32_t size_m4 = inpsize - 4
0002cdf4      uint32_t bytes_left = 0
0002cdf0      if (size_m4 s> 0)
0002ce00          int32_t size_mc = inpsize - 0xc
0002ce04          uint32_t bytesread = 0
0002ce24          do
0002ce2c              struct WifiPkt* wifipkt = &buffer[bytesread]
0002ce38              while (wifipkt->magic == 0xff005555)                  // [7]
0002ce54                  if (bytesread + 0xb s>= inpsize)
0002cefc                      return inpsize - bytesread
0002ce74                  bytes_left = inpsize - bytesread
0002ce70                  if (bytesread + wifipkt->size + 0xb s>= inpsize)   // [8]
0002cefc                      return bytes_left
0002ce7c                  XM_LOG(fname: "Xm_WifiComServer.c", line: 0x8a2, 2, 0,
fmtstr: "cmd offset:%d\n", values: bytesread)
0002ce94                  int32_t wificmd_ret = WifiCommandRespProc(wifipkt:
wifipkt, mall_20: mall_20)  // [9]
```

After iterating through our data packet until finding the `0xff005555` magic bytes at [7], we finally get to our vulnerability: a lack of checking of the `Wifipkt->size` field inside the `DemuxCmdInBuffer` function, which allows us to cause an integer overflow at [8]. Unfortunately for attacker purposes, this `WifiPkt->datalen` field does not actually do that much inside `WifiCommandRespProc` [9] besides being checked and also potentially used as the length in an `fwrite` call. The best we can really get out of this vulnerability is setting the `datalen` large enough such that an out-of-bounds read occurs after advancing its read pointer by `WifiPkt->datalen+0xb`. This results in an unmapped read and a binary crash. If the `home_security` process crashes enough times within a given period, a device reboot will also occur, resulting in a denial of service.

Crash Information

```
[ 5886.888000] do_page_fault() #2: sending SIGSEGV to home_security(23710) for
invalid read access from
[ 5886.888000] 282e7aa4 (pc == 778c4e30, ra == 778c4e9c)

<(^.^)>#info reg
          zero       at       v0       v1       a0       a1       a2       a3
 R0   00000000 1100ff00 487f5f11 00000001 00000001 00000001 00000000 00000001
            t0       t1       t2       t3       t4       t5       t6       t7
 R8   00000000 fffffffe 00000000 00000000 9b64c2b0 8758a11c 00000000 00000007
            s0       s1       s2       s3       s4       s5       s6       s7
 R16  b780a0ff 33e09af7 7c5ff9f8 00000010 0000000c ff005555 7700a4d1 7700b920
            t8       t9       k0       k1       gp       sp       s8       ra
 R24  00000000 773d36ec 00000000 00000000 77026560 7c5ff960 7c5ff9d8 76ff6e9c
            sr       lo       hi      bad    cause       pc
      0100ff13 067e836c 001154dc 33e09af7 00800010 76ff6e30
           fsr      fir
      00000000 00000000

<(^.^)>#x/10i $pc-0x10
   0x76d3ae20 <WifiComRecv_client_pth+504>:     li      a1,2280
   0x76d3ae24 <WifiComRecv_client_pth+508>:     li      a2,2
   0x76d3ae28 <WifiComRecv_client_pth+512>:     jalr    t9
   0x76d3ae2c <WifiComRecv_client_pth+516>:     move    a3,zero
=> 0x76d3ae30 <WifiComRecv_client_pth+520>:     lw      gp,32(sp)
   0x76d3ae34 <WifiComRecv_client_pth+524>:     move    s4,s2
   0x76d3ae38 <WifiComRecv_client_pth+528>:     move    s0,s2
   0x76d3ae3c <WifiComRecv_client_pth+532>:     move    s1,zero
   0x76d3ae40 <WifiComRecv_client_pth+536>:     lw      t9,-31148(gp)
```

VENDOR RESPONSE

Fixed version 3.1.8.7 and 3.1.8.7h is in grayscale on Homebase2

TIMELINE

2022-03-11 - Vendor disclosure

2022-04-15 - Vendor patched

2022-05-05 - Public disclosure

CREDIT

Discovered by Lilith >_> of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2022-1479

NEXT REPORT

TALOS-2022-1475