Talos Vulnerability Report

TALOS-2022-1483

TCL LinkHub Mesh Wi-Fi confsrv ucloud_set_node_location stack-based buffer overflow vulnerability

AUGUST 1, 2022

CVE NUMBER

CVE-2022-26009

SUMMARY

A stack-based buffer overflow vulnerability exists in the confsrv ucloud_set_node_location functionality of TCL LinkHub Mesh Wi-Fi MS1G_00_01.00_14. A specially-crafted network packet can lead to stack-based buffer overflow. An attacker can send a malicious packet 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.

TCL LinkHub Mesh Wifi MS1G_00_01.00_14

PRODUCT URLS

LinkHub Mesh Wifi - https://www.tcl.com/us/en/products/connected-home/linkhub/linkhub-mesh-wifi-system-3-pack

CVSSV3 SCORE

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

CWE

CWE-121 - Stack-based Buffer Overflow

DETAILS

The LinkHub Mesh Wi-Fi system is a node-based mesh system designed for Wi-Fi deployments across large homes. These nodes include most features standard in current Wi-Fi solutions and allow for easy expansion of the system by adding nodes. The mesh is managed solely by a phone application, and the routers have no web-based management console.

The LinkHub Mesh system uses protobuffers to communicate both internally on the device as well as externally with the controlling phone application. These protobuffers can be sent to port 9003 while on the Wi-Fi provided by the LinkHub Mesh in order to issue commands, much like the phone application would. Once the protobuffer is received, it is routed internally starting from the `ucloud` binary and is dispatched to the appropriate handler.

In this case, the handler is `confsrv` which handles many message types. In this case we are interested in `NodeLocation`

```
message NodeLocation {
    required string serialNum = 1;           [1]
    required string location = 2;
    optional uint64 timestamp = 3;
}
```

Using [1] we have control over `serialNum` in the packet. The parsing of the data in the protobuf is done in `ucloud_set_node_location`.

```
00429390  int32_t ucloud_set_node_location(int32_t arg1, int32_t arg2, int32_t arg3)

004293b0      arg_0 = arg1
004293bc      int32_t $a3
004293bc      arg_c = $a3
004293c0      int32_t var_12c = 0
004293c4      int32_t var_128 = 0
004293c8      int32_t var_124 = 0
004293cc      int32_t var_120 = 0
004293d0      int32_t var_11c = 0
004293d4      int32_t var_118 = 0
004293d8      int32_t var_114 = 0
004293dc      int32_t var_110 = 0
004293e0      int32_t var_10c = 0
004293e4      int32_t var_130 = 0
00429404      void var_108
00429404      memset(&var_108, 0, 0x100)
00429428      GetValue(name: "serial.number", output_buffer: &var_128)
00429450      struct NodeLocationDescriptor* pkt = node_location__unpack(0, arg3,
arg2)
00429464      int32_t $v0_2
00429464      if (pkt == 0) {
0042948c          _td_snprintf(3, "api/map_manage.c", 0x83a, "   unpack failed !
\n", 0x4ae4b0)
00429498          $v0_2 = 0xffffffff
00429498      } else {
004294b8          void* $v0_5 = client_sn_lkup(sn: pkt->serial_number)
004294cc          if ($v0_5 != 0) {
004294f4              strcpy($v0_5 + 0x30, pkt->location)
0042950c              if (sx.d(*($v0_5 + 0x30)) == 0) {
00429544                  *($v0_5 + 4) = *($v0_5 + 4) & 0xfffffffd
0042953c              } else {
00429524                  *($v0_5 + 4) = *($v0_5 + 4) | 2
0042951c              }
0042951c          }
00429574          sprintf(&var_108, "%s%s", "node.location@", pkt->serial_number,
0x4ae4b0)           [2]
...
```

At [2] we can clearly see that a `sprintf` is performed without any validation of buffer or input length, which can lead to a stack-based buffer overflow. Below we can verify the issue in ASM:

```
00429548  b480828f   lw      $v0, -0x7f4c($gp)  {data_4a6564}
0042954c  38084324   addiu   $v1, $v0, 0x838  {data_480838, "%s%s"}
00429550  1c00c28f   lw      $v0, 0x1c($fp) {var_12c_1}
00429554  0c00428c   lw      $v0, 0xc($v0) {NodeLocationDescriptor::serial_number}
00429558  4000c427   addiu   $a0, $fp, 0x40 {var_108}
[3]
0042955c  21286000   move    $a1, $v1  {data_480838, "%s%s"}
00429560  b480838f   lw      $v1, -0x7f4c($gp)  {data_4a6564}
00429564  40086624   addiu   $a2, $v1, 0x840  {data_480840, "node.location@"}
00429568  21384000   move    $a3, $v0
[4]
0042956c  0088828f   lw      $v0, -0x7800($gp)  {sprintf}
00429570  21c84000   move    $t9, $v0
00429574  09f82003   jalr    $t9
[5]
00429578  00000000   nop
```

At [3] we see that a stack buffer is being loaded as the destination argument of `sprintf`. At [4] the `serialNum` from the protobuf is being loaded as the second portion of the formation string. At [5] we see that `sprintf` is called with no further validation or verification that the buffer is large enough to hold the format string, or that the input is small enough to fit in the buffer. This leads to a simple stack-based buffer overflow using `sprintf`.

Crash Information

```
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────────
──────────────────── registers ────
$zero: 0x0
$at  : 0x806f0000
$v0  : 0x0
$v1  : 0x77c798e0
$a0  : 0x11
$a1  : 0x2
$a2  : 0x1
$a3  : 0x0
$t0  : 0x0
$t1  : 0x0
$t2  : 0x4
$t3  : 0x0
$t4  : 0x8785e654
$t5  : 0x8000
$t6  : 0x0
$t7  : 0x0
$s0  : 0x7f839ad8  →  0x82031107
$s1  : 0x7f839ad8  →  0x82031107
$s2  : 0x77a8aa60  →  "uc_api_lib.c"
$s3  : 0x0
$s4  : 0x77a8bbe4  →  "_session_read_and_dispatch"
$s5  : 0x77a71090  →  0x3c1c0003
$s6  : 0x115
$s7  : 0x10
$t8  : 0x0
$t9  : 0x7767b52c  →  0x3c1c0002
$k0  : 0x0
$k1  : 0x0
$s8  : 0x41414141 ("AAAA"?)
$pc  : 0x41414141 ("AAAA"?)
$sp  : 0x7f8399b0  →  "AAAAAAAAAAAAAA"
$hi  : 0x3a4f47
$lo  : 0x21743d08 ("\b=t!"?)
$fir : 0x0
$ra  : 0x41414141 ("AAAA"?)
$gp  : 0x004ae4b0  →  0x00000000
──────────────────────────────────────────────────────────────────────────────────
──────────────────────── stack ────
0x7f8399b0|+0x0000: "AAAAAAAAAAAAAA"     ← $sp
0x7f8399b4|+0x0004: "AAAAAAAAAA"
0x7f8399b8|+0x0008: "AAAAAA"
0x7f8399bc|+0x000c: 0x7f004141 ("AA"?)
0x7f8399c0|+0x0010: 0x7f839a2c  →  0x00000000
0x7f8399c4|+0x0014: 0x00000000
0x7f8399c8|+0x0018: 0x004ae4b0  →  0x00000000
0x7f8399cc|+0x001c: 0x7f839ad8  →  0x82031107
──────────────────────────────────────────────────────────────────────────────────
─────────────── code:mips:MIPS32 ────
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x41414140
──────────────────────────────────────────────────────────────────────────────────
──────────────────────── threads ────
[#0] Id 1, stopped 0x41414141 in ?? (), reason: SIGSEGV
```

```
──────────────────────────────────────────────────────────────────────────────
──────────────────────────── trace ─────
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────
```

TIMELINE

2022-03-16 - Vendor Disclosure

2022-08-01 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2022-1484

NEXT REPORT

TALOS-2022-1507