Talos Vulnerability Report

TALOS-2022-1484

TCL LinkHub Mesh Wi-Fi confsrv ucloud_set_node_location buffer overflow vulnerability

AUGUST 1, 2022

CVE NUMBER

CVE-2022-26342

SUMMARY

A 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 a 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-120 - Buffer Copy without Checking Size of Input (‘Classic 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;
    required string location = 2;           [1]
    optional uint64 timestamp = 3;
}
```

Using [1] we have control over `location` 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($v0_5 + 0x30, pkt->location)
[2]
0042950c              if (sx.d(*($v0_5 + 0x30)) == 0) {
00429544                  *($v0_5 + 4) = *($v0_5 + 4) & 0xfffffffd
0042953c              } else {
00429524                  *($v0_5 + 4) = *($v0_5 + 4) | 2
0042951c              }
0042951c          }
...
```

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

```
004294a4  1c00c28f   lw      $v0, 0x1c($fp) {var_12c_1}
004294a8  0c00428c   lw      $v0, 0xc($v0) {NodeLocationDescriptor::serial_number}
004294ac  21204000   move    $a0, $v0
004294b0  4883828f   lw      $v0, -0x7cb8($gp)  {client_sn_lkup}  {data_4a67f8}
004294b4  21c84000   move    $t9, $v0  {client_sn_lkup}
004294b8  09f82003   jalr    $t9  {client_sn_lkup}
004294bc  00000000   nop

004294c0  1000dc8f   lw      $gp, 0x10($fp) {var_138}
004294c4  1800c2af   sw      $v0, 0x18($fp) {var_130_1}
[3]
004294c8  1800c28f   lw      $v0, 0x18($fp) {var_130_1}
004294cc  1e004010   beqz    $v0, 0x429548
004294d0  00000000   nop

004294d4  1800c28f   lw      $v0, 0x18($fp) {var_130_1}
[4]
004294d8  30004324   addiu   $v1, $v0, 0x30
004294dc  1c00c28f   lw      $v0, 0x1c($fp) {var_12c_1}
004294e0  1000428c   lw      $v0, 0x10($v0) {NodeLocationDescriptor::location}
[5]
004294e4  21206000   move    $a0, $v1
004294e8  21284000   move    $a1, $v0
004294ec  7c86828f   lw      $v0, -0x7984($gp)  {strcpy}
004294f0  21c84000   move    $t9, $v0
004294f4  09f82003   jalr    $t9
[6]
004294f8  00000000   nop
```

At [3] we see the return value of `client_sn_lkup` being saved onto the stack that is loaded at [4] to be the destination argument of `strcpy`. At [5] the `location` is loaded from the protobuf data and used as the source for the `strcpy`. At [6] we see that `strcpy` is called with no further validation or verification that the buffer is large enough to hold the data, or that the source is small enough to fit in the buffer. This leads to a simple buffer overflow using `strcpy`.

The destination buffer for the overflow is retrieved from `client_sn_lkup`, seen below.

```
00422c30  void* client_sn_lkup(char* sn)

00422c54      int32_t var_10 = 0
00422c60      uint32_t var_10_1 = g_online_map_head
00422cb8      uint32_t $v0_4
00422cb8      while (true) {
00422cb8          if (var_10_1 == 0) {
00422cc0              $v0_4 = 0
00422cc0              break
00422cc0          }
00422c94          if (strncmp(sn, var_10_1 + 8, 0x20) == 0) {
00422c9c              $v0_4 = var_10_1
00422ca0              break
00422ca0          }
00422cb0          var_10_1 = *var_10_1
00422cac      }
00422cd4      return $v0_4
```

`g_online_map_head` is located within the BSS section of the binary. This means that with this buffer overflow we can target any other statically allocated variable within the binary, which can lead to remote code execution.

Crash Information

```
Program received signal SIGSEGV, Segmentation fault.
0x779493f0 in strncmp () from target:/lib/libc.so.0
[ Legend: Modified register | Code | Heap | Stack | String ]
────────────────────────────────────────────────────────────────────────────────
──────────────────────────── registers ────
$zero: 0x0
$at  : 0x806f0000
$v0  : 0x32
$v1  : 0x41
$a0  : 0x004ba150  →  "21012304710"
$a1  : 0x41414149 ("IAAA"?)
$a2  : 0x20
$a3  : 0x7
$t0  : 0x0
$t1  : 0x0
$t2  : 0x200
$t3  : 0x100
$t4  : 0x807
$t5  : 0x800
$t6  : 0x400
$t7  : 0x8
$s0  : 0x7fddaaa8  →  0x82031107
$s1  : 0x7fddaaa8  →  0x82031107
$s2  : 0x77d7aa60  →  "uc_api_lib.c"
$s3  : 0x0
$s4  : 0x77d7bbe4  →  "_session_read_and_dispatch"
$s5  : 0x77d61090  →  0x3c1c0003
$s6  : 0x1018
$s7  : 0x10
$t8  : 0x0
$t9  : 0x779493d0  →  0x2cc20004
$k0  : 0x0
$k1  : 0x0
$s8  : 0x7fdda6d0  →  0x00000000
$pc  : 0x779493f0  →  0x1040001d
$sp  : 0x7fdda6d0  →  0x00000000
$hi  : 0x169
$lo  : 0x25512
$fir : 0x0
$ra  : 0x00422c90  →  <client_sn_lkup+96> lw gp, 16(s8)
$gp  : 0x004ae4b0  →  0x00000000
────────────────────────────────────────────────────────────────────────────────
──────────────────────────────── stack ────
0x7fdda6d0│+0x0000: 0x00000000   ← $s8, $sp
0x7fdda6d4│+0x0004: 0x00000000
0x7fdda6d8│+0x0008: 0x00000000
0x7fdda6dc│+0x000c: 0x00000000
0x7fdda6e0│+0x0010: 0x004ae4b0  →  0x00000000
0x7fdda6e4│+0x0014: 0x00000000
0x7fdda6e8│+0x0018: "AAAA"
0x7fdda6ec│+0x001c: 0x00000000
────────────────────────────────────────────────────────────────────────────────
────────────── code:mips:MIPS32 ────
   0x779493e4 <strncmp+20>     move   v0, zero
   0x779493e8 <strncmp+24>     lbu    v0, 0(a0)
   0x779493ec <strncmp+28>     addiu  a3, a3, -1
```

TIMELINE

2022-03-16 - Vendor Disclosure

2022-08-01 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.
