Talos Vulnerability Report

TALOS-2022-1456

TCL LinkHub Mesh Wifi confers ucloud_add_node_new stack-based buffer overflow vulnerability

AUGUST 1, 2022

CVE NUMBER

CVE-2022-21201

SUMMARY

A stack-based buffer overflow vulnerability exists in the confers ucloud_add_node_new 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 `ManualNodeInfo`

```
message ManualNodeInfo {
    required string serialNumMd5 = 1;        [1]
    optional uint64 timestamp = 2;
}
```

At [1] we have control over `serialNumMd5` in the packet. The parsing of the protobuffer data occurs in `ucloud_add_node_new`

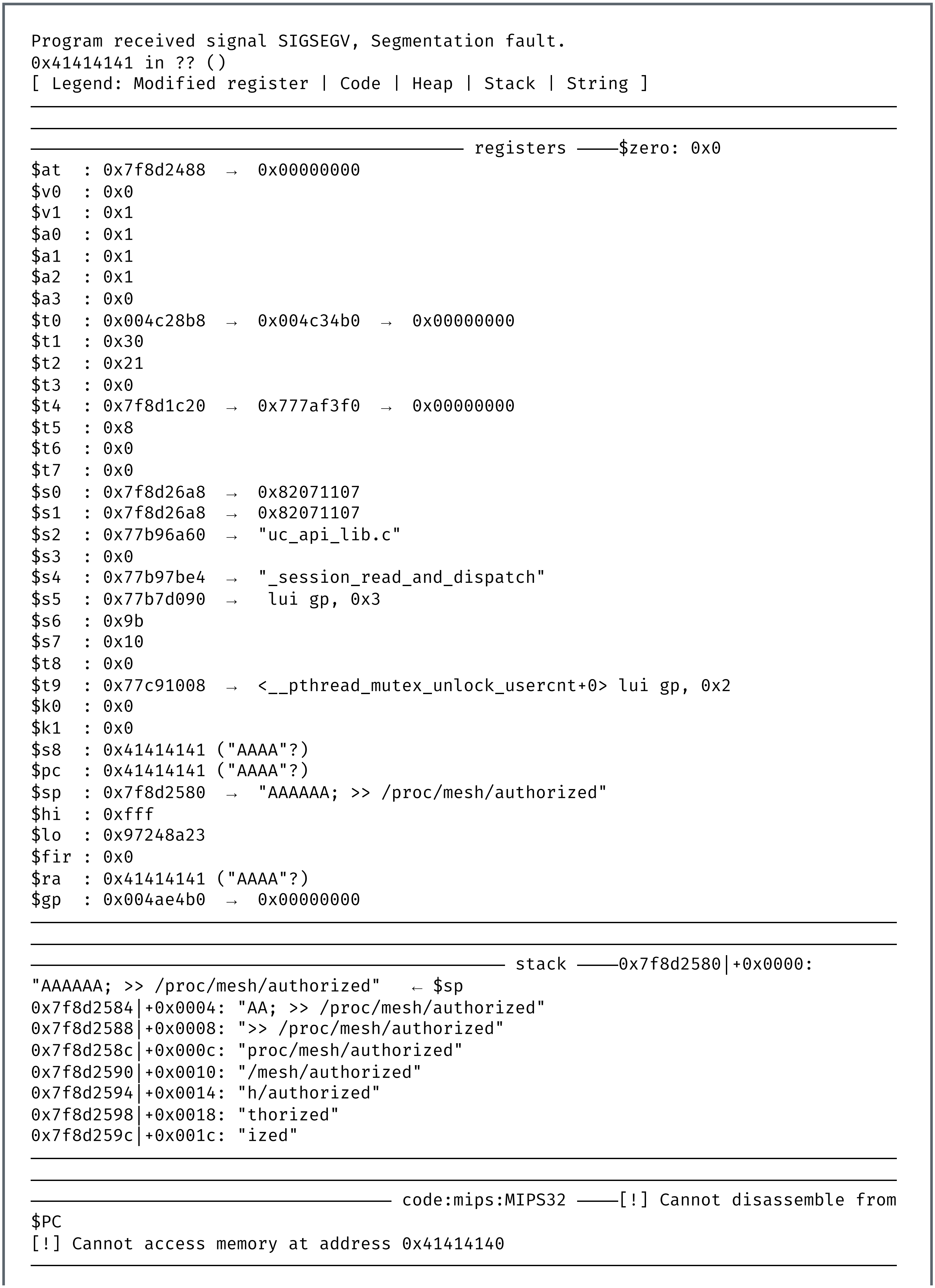
```
0042876c  int32_t ucloud_add_node_new(int32_t arg1, int32_t arg2, int32_t arg3)

0042878c      arg_0 = arg1
00428798      int32_t $a3
00428798      arg_c = $a3
004287bc      printf("%s(%d)\n", "ucloud_add_node_new", 0x756)
004287c8      int32_t var_b0 = 0
004287cc      int32_t var_ac = 0
004287d0      int32_t var_a8 = 0
004287d4      int32_t var_a4 = 0
004287d8      int32_t var_a0 = 0
004287dc      int32_t var_9c = 0
004287e0      int32_t var_98 = 0
004287e4      int32_t var_94 = 0
004287e8      int32_t var_90 = 0
00428808      void var_8c
00428808      memset(&var_8c, 0, 0x80)
00428818      int32_t $v0_1
00428818      if (arg2 == 0) {
00428840          printf("ManualNodeInfo is NULL%s(%d)\n", "ucloud_add_node_new",
0x75d)
0042884c          $v0_1 = 0xffffffff
0042884c      } else {
00428874          struct ManualNodeInfo* pkt = manual_node_info__unpack(0, arg3,
arg2)
00428888          if (pkt == 0) {
004288b0              printf("manual_node_info__unpack error%s…",
"ucloud_add_node_new", 0x766)
004288bc              $v0_1 = 0xffffffff
004288bc          } else {
004288d0              if (pkt->serialNumberMd5 == 0) {
00428938                  printf("[arainc][NodeInfo->serialnummd5 …",
"ucloud_add_node_new", 0x76f)
0042892c              } else {
00428904                  printf("[arainc][NodeInfo->serialnummd5 …", pkt-
>serialNumberMd5, "ucloud_add_node_new", 0x76d, 0x4ae4b0)
00428788              }
00428958              update_add_node_list(serial_number: pkt->serialNumberMd5)
00428988              sprintf(&var_8c, "echo %s >> /proc/mesh/authorized", pkt-
>serialNumberMd5)                                    [2]
004289bc              printf("[arainc][cmd_tmp = %s]%s(%d)\n", &var_8c,
"ucloud_add_node_new", 0x773, 0x4ae4b0)
004289d8              doSystemCmd(&var_8c)
004289ec              if (pkt->__offset(0x10).d != 0) {
00428a1c                  sprintf(&var_ac, "%llu", pkt->timestamp.d, pkt-
>timestamp:4.d)
00428a40                  SetValue(name: "sys.cfg.stamp", input_buffer: &var_ac)
00428a34              }
00428a54              CommitCfm()
00428a70              manual_node_info__free_unpacked(pkt, 0)
00428a7c              $v0_1 = 0
00428a7c          }
00428a7c      }
00428a90      return $v0_1
```

Looking at the assembly at [2]

```
0042896c  1800c28f   lw      $v0, 0x18($fp) {var_b0_1}
00428970  0c00428c   lw      $v0, 0xc($v0) {ManualNodeInfo::serialNumberMd5}
00428974  3c00c427   addiu   $a0, $fp, 0x3c {var_8c}
00428978  21286000   move    $a1, $v1  {data_4810d8, "echo %s >>
/proc/mesh/authorized"}
0042897c  21304000   move    $a2, $v0
00428980  0088828f   lw      $v0, -0x7800($gp)  {sprintf}
00428984  21c84000   move    $t9, $v0
00428988  09f82003   jalr    $t9
0042898c  00000000   nop
```

We see this is a straightforward heap buffer overflow. The `serialNumberMd5` is taken directly from the packet and passed into a `%s` formatter without any length limits. The buffer is 0x80 bytes long, and thus, an input of length 0x8C will take control of `$ra`

Crash Information



```
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────
────────────────────────────────────────── registers ────$zero: 0x0
$at  : 0x7f8d2488  →  0x00000000
$v0  : 0x0
$v1  : 0x1
$a0  : 0x1
$a1  : 0x1
$a2  : 0x1
$a3  : 0x0
$t0  : 0x004c28b8  →  0x004c34b0  →  0x00000000
$t1  : 0x30
$t2  : 0x21
$t3  : 0x0
$t4  : 0x7f8d1c20  →  0x777af3f0  →  0x00000000
$t5  : 0x8
$t6  : 0x0
$t7  : 0x0
$s0  : 0x7f8d26a8  →  0x82071107
$s1  : 0x7f8d26a8  →  0x82071107
$s2  : 0x77b96a60  →  "uc_api_lib.c"
$s3  : 0x0
$s4  : 0x77b97be4  →  "_session_read_and_dispatch"
$s5  : 0x77b7d090  →   lui gp, 0x3
$s6  : 0x9b
$s7  : 0x10
$t8  : 0x0
$t9  : 0x77c91008  →  <__pthread_mutex_unlock_usercnt+0> lui gp, 0x2
$k0  : 0x0
$k1  : 0x0
$s8  : 0x41414141 ("AAAA"?)
$pc  : 0x41414141 ("AAAA"?)
$sp  : 0x7f8d2580  →  "AAAAAA; >> /proc/mesh/authorized"
$hi  : 0xfff
$lo  : 0x97248a23
$fir : 0x0
$ra  : 0x41414141 ("AAAA"?)
$gp  : 0x004ae4b0  →  0x00000000
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────
────────────────────────────────────────────── stack ────0x7f8d2580│+0x0000:
"AAAAAA; >> /proc/mesh/authorized"   ← $sp
0x7f8d2584│+0x0004: "AA; >> /proc/mesh/authorized"
0x7f8d2588│+0x0008: ">> /proc/mesh/authorized"
0x7f8d258c│+0x000c: "proc/mesh/authorized"
0x7f8d2590│+0x0010: "/mesh/authorized"
0x7f8d2594│+0x0014: "h/authorized"
0x7f8d2598│+0x0018: "thorized"
0x7f8d259c│+0x001c: "ized"
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────
─────────────────────────────────── code:mips:MIPS32 ────[!] Cannot disassemble from
$PC
[!] Cannot access memory at address 0x41414140
──────────────────────────────────────────────────────────────────────────────
```

```
──────────────────────────────────────────────────────────────────────────────────────
───────────────────────────────────────────── threads ────[#0] Id 1, stopped
0x41414141 in ?? (), reason: SIGSEGV
──────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────────
─────────────────────────────────────────────── trace
──────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────
```

TIMELINE

2022-02-08 - Initial Vendor Contact

2022-02-09 - Vendor Disclosure

2022-08-01 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2022-1455

NEXT REPORT

TALOS-2022-1457