Talos Vulnerability Report

TALOS-2022-1507

TCL LinkHub Mesh Wifi ucloud_del_node denial of service vulnerability

AUGUST 1, 2022

CVE NUMBER

CVE-2022-26346

SUMMARY

A denial of service vulnerability exists in the ucloud_del_node functionality of TCL LinkHub Mesh Wi-Fi MS1G_00_01.00_14. A specially-crafted network packet can lead to denial of service. 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.

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

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

CWE

CWE-284 - Improper Access Control

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, or wired network, 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 `MxpManageList`

```
message MxpManage {
    required string serialNum = 1;              [1]
    required int32 opt = 2;
}
message MxpManageList {
    repeated MxpManage mxp = 1;
    optional uint64 timestamp = 2;
}
```

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

```
00428a98  int32_t ucloud_del_node(int32_t arg1, int32_t arg2, int32_t arg3)

00428ab8      arg_0 = arg1
00428ac4      int32_t $a3
00428ac4      arg_c = $a3
00428ac8      int32_t var_440 = 0
00428acc      int32_t var_444 = 0
00428aec      void group_sn
00428aec      memset(&group_sn, 0, 0x100)
00428af8      int32_t var_448 = 0
00428afc      int32_t sn = 0
00428b00      int32_t var_338 = 0
00428b04      int32_t var_334 = 0
00428b08      int32_t var_330 = 0
00428b0c      int32_t var_32c = 0
00428b10      int32_t var_328 = 0
00428b14      int32_t var_324 = 0
00428b18      int32_t var_320 = 0
00428b38      void var_31c
00428b38      memset(&var_31c, 0, 0x100)
00428b60      void var_21c
00428b60      memset(&var_21c, 0, 0x210)
00428b70      int32_t $v0_1
00428b70      if (arg2 == 0) {
00428b98          _td_snprintf(3, "api/map_manage.c", 0x7a1, "    in is null !
\n", 0x4ae4b0)
00428ba4          $v0_1 = 0xffffffff
00428ba4      } else {
00428bc8          GetValue(name: "sys.mesh.groupsn", output_buffer: &group_sn)
00428bec          GetValue(name: "serial.number", output_buffer: &sn)
00428c14          struct MxpManageList* pkt = mxp_manage_list__unpack(0, arg3, arg2)
00428c28          if (pkt == 0) {
00428c50              _td_snprintf(3, "api/map_manage.c", 0x7a9, "     unpack failed
!     \n", 0x4ae4b0)
00428c5c              $v0_1 = 0xffffffff
00428c5c          } else {
00428c78              init_node_opt_hash_table(&var_21c)
00428c94              get_node_opt_hash_table(&var_21c)
00428ca0              int32_t loop_idx = 0
00428f40              while (true) {
00428f40                  if (loop_idx u>= pkt->mxp_manage_count) {
00428f50                      if (pkt->is_timestamp_present != 0) {
00428f80                          sprintf(&var_31c, "%llu", pkt->timestamp.d, pkt-
>timestamp:4.d, 0x4ae4b0)
00428fa4                          SetValue(name: "sys.cfg.stamp", input_buffer:
&var_31c)
00428f98                      }
00428fc0                      mxp_manage_list__free_unpacked(pkt, 0)
00428fdc                      save_all_mesh_node_opt(&var_21c)
00428ff8                      free_the_hash_table(&var_21c)
0042902c                      printf("[%s][%d][kg] groupsn = %s\n",
"ucloud_del_node", 0x7d0, &group_sn, 0x4ae4b0)
00429050                      SetValue(name: "sys.mesh.groupsn", input_buffer:
&group_sn)
00429064                      CommitCfm()
00429070                      $v0_1 = 0
00429070                      break
00429070                  }
```

```
00428cd8                    upload_one_node_basic_info(serial_number: *(*(pkt->p_mxp +
(loop_idx << 2)) + 0xc), 2)
00428d24                    if (strncmp(&sn, *(*(pkt->p_mxp + (loop_idx << 2)) + 0xc),
0x20) == 0) {                                                          [2]
00428d4c                        printf("[%s][%d][luminais] mpp is delete…",
"ucloud_del_node", 0x7b5)
00428d68                        monitor_stop("cmdsrv")
00428d84                        monitor_stop("mesh_status_check")
00428da0                        monitor_stop("pann")
00428dbc                        monitor_stop("netctrl")
00428dd8                        mxp_manage_list__free_unpacked(pkt, 0)
00428df4                        free_the_hash_table(&var_21c)
00428e08                        systool_handle_restore_zero()
00428e14                        $v0_1 = 0
00428e18                        break
00428e18                    }
00428e60                    printf("[%s][%d][kg] del sn = %s\n", "ucloud_del_node",
0x7c1, *(*(pkt->p_mxp + (loop_idx << 2)) + 0xc), 0x4ae4b0)
00428e94                    client_node_del(*(*(pkt->p_mxp + (loop_idx << 2)) + 0xc))
[3]
00428ed8                    str_list_del_item(&group_sn, &data_480a1c, *(*(pkt->p_mxp
+ (loop_idx << 2)) + 0xc))
00428f18                    update_node_opt_to_hash_table(&var_21c, 2, *(*(pkt->p_mxp
+ (loop_idx << 2)) + 0xc))
00428f2c                    loop_idx = loop_idx + 1
00428f28                }
00428f28            }
00428f28        }
00429084        return $v0_1
```

At [2] a check is done to see if the `serialNum` provided is of the current device. If it is, a factory reset will effectively occur, reverting all the network configurations. [3] represents if the `serialNum` provided is not the receiving base station. In this case the message is passed along the mesh to find the base station that is trying to be deleted, at which point the matching base station will perform a factory reset. This protobuf message does not require any authentication.

TIMELINE

2022-03-29 - Vendor Disclosure
2022-08-01 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.
