Talos Vulnerability Report

TALOS-2022-1458

TCL LinkHub Mesh Wifi confsrv ucloud_add_node OS command injection vulnerability

AUGUST 1, 2022

CVE NUMBER

CVE-2022-22140

SUMMARY

An os command injection vulnerability exists in the confsrv ucloud_add_node functionality of TCL LinkHub Mesh Wi-Fi MS1G_00_01.00_14. A specially-crafted network packet can lead to arbitrary command execution. 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

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

CWE

CWE-78 - Improper Neutralization of Special Elements used in an OS Command (‘OS Command Injection’)

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 `MxpManageList`.

```
message MxpManage {
    required string serialNum = 1;              [1]
    required int32 opt = 2;
}
message MxpManageList {
    repeated MxpManage mxp = 1;    //This is not optional, so it must be resolved by
hand to compile to .proto    //This is not optional, so it must be resolved by hand
to compile to .proto
    optional uint64 timestamp = 2;
}
```

Using [1] we have control over the `serialNum` in the packet. The parsing of the data within the protobuffer is `ucloud_add_node`.

```
00428478  int32_t ucloud_add_node(int32_t arg1, int32_t arg2, int32_t arg3)
00428498      arg_0 = arg1
004284a4      int32_t $a3
004284a4      arg_c = $a3
004284a8      int32_t var_12c = 0
004284ac      int32_t var_130 = 0
004284cc      void var_128
004284cc      memset(&var_128, 0, 0x100)
004284d8      int32_t var_28 = 0
004284dc      int32_t var_24 = 0
004284e0      int32_t var_20 = 0
004284e4      int32_t var_1c = 0
004284e8      int32_t var_18 = 0
004284ec      int32_t var_14 = 0
004284f0      int32_t var_10 = 0
004284f4      int32_t var_c = 0
004284fc      int32_t $v0_1
004284fc      if (arg2 == 0) {
00428524          _td_snprintf(3, "api/map_manage.c", 0x737, "    in is null !
\n", 0x4ae4b0)
00428530          $v0_1 = 0xffffffff
00428530      } else {
00428558          struct MxpManageList* $v0_3 = mxp_manage_list__unpack(0, arg3,
arg2)
0042856c          if ($v0_3 == 0) {
00428594              _td_snprintf(3, "api/map_manage.c", 0x73d, "     unpack failed
!     \n", 0x4ae4b0)
004285a0              $v0_1 = 0xffffffff
004285a0          } else {
004286b0              for (uint32_t var_130_1 = 0; var_130_1 u< $v0_3-
>mxp_manage_count; var_130_1 = var_130_1 + 1) {
004285d0                  if (confctl_module_debug_en(module_id: 9) != 0) {
00428618                      printf("\x1b[1;32m[%s][%d] : \x1b[0m\x1b…",
"ucloud_add_node", 0x743, *(*($v0_3->p_mxp + (var_130_1 << 2)) + 0xc), 0x4ae4b0)
00428494                  }
0042864c                  update_add_node_list(serial_number: *(*($v0_3->p_mxp +
(var_130_1 << 2)) + 0xc))
00428688                  doSystemCmd("echo %s >> /proc/mesh/authorized", *(*($v0_3-
>p_mxp + (var_130_1 << 2)) + 0xc))       [2]
00428670              }
004286c0              if ($v0_3->is_timestamp_present != 0) {
004286f0                  sprintf(&var_28, "%llu", $v0_3->timestamp.d, $v0_3-
>timestamp:4.d, 0x4ae4b0)
00428714                  SetValue(name: "sys.cfg.stamp", input_buffer: &var_28)
00428708              }
00428728              CommitCfm()
00428744              mxp_manage_list__free_unpacked($v0_3, 0)
00428750              $v0_1 = 0
00428750          }
00428750      }
00428764      return $v0_1
```

At [2] the `serialNum` is used directly in `doSystemCmd`.

```
000209b0  int32_t doSystemCmd(int32_t arg1, int32_t arg2)
000209d0      arg_4 = arg2
000209d4      int32_t $a2
000209d4      arg_8 = $a2
000209d8      int32_t $a3
000209d8      arg_c = $a3
000209fc      void var_408
000209fc      memset(&var_408, 0, 0x400)
00020a30      log_debug_print("doSystemCmd", &data_1b8d, 0, 0x80, 0x55500)
{"function entry!"}
00020a64      vsnprintf(&var_408, 0x400, arg1, &arg_4)
00020a80      int32_t $v0_1 = system(&var_408)
00020ab8      log_debug_print("doSystemCmd", &data_1b93, 0, 0x80, 0x55510)
{"function exit!"}
00020ad8      return $v0_1
```

With a quick look at `doSystemCmd` we can see that no special escaping is happening here and thus this is a simple command injection using `serialNum` directly.

TIMELINE

2022-04-27 - Vendor Disclosure

2022-08-01 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.
