Talos Vulnerability Report

TALOS-2022-1457

TCL LinkHub Mesh Wifi confsrv ucloud_add_node_new OS command injection vulnerability

AUGUST 1, 2022

CVE NUMBER

CVE-2022-21178

SUMMARY

An os command injection vulnerability exists in the confsrv ucloud_add_new_node functionality of TCL LinkHub Mesh Wifi 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 `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)
[3]
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
```

At [2] the command is built using `sprintf`. The data used is directly from the user packet, then passed into `doSystemCmd` at [3].

```
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 `serialNumMd5` directly.

TIMELINE

2022-04-27 - Vendor Disclosure

2022-08-01 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.