Talos Vulnerability Report

TALOS-2022-1502

TCL LinkHub Mesh Wifi confctl_set_guest_wlan denial of service vulnerability

AUGUST 1, 2022

CVE NUMBER

CVE-2022-27660

SUMMARY

A denial of service vulnerability exists in the confctl_set_guest_wlan 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

9.3 - CVSS:3.0/AV:A/AC:L/PR:N/UI:N/S:C/C:N/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 `WlanCfgAll`

```
enum MESH_WIFI_TYPE {
    MESH_WIFI_2G = 0;
    MESH_WIFI_5G = 1;
    MESH_WIFI_MAX = 2;
}
message WlanTimeChoice {
    repeated int32 option = 1;
}
message WlanSecChoice {
    repeated string option = 1;
}
message WlanLimitChoice {
    repeated int32 option = 1;
}
message WlanCfg {
    required MESH_WIFI_TYPE band = 1;
    required string ssid = 2;
    required string passwd = 3;
    optional string sec = 4;
    optional int32 left = 5;
    optional int32 limite = 6;
    optional int32 timeout = 7;
    optional bool enable = 8;
}
message WlanCfgAll {
    repeated WlanCfg wlan = 1;
    optional WlanTimeChoice timeout = 2;
    optional WlanSecChoice security = 3;
    optional WlanLimitChoice limits = 4;
    optional uint64 timestamp = 5;
    optional bool enable = 6;                          [1]
    optional bool from_app = 7;                        [2]
}
```

Utilizing [1] and [2], direct control of `enable` and `from_app` is obtained. At least one entry of `wlan` also needs to be populated for parsing to occur; this parsing is done in `confctl_set_guest_wlan`:

```
0045700c  int32_t confctl_set_guest_wlan(int32_t arg1, int32_t arg2, int32_t arg3)

...
00457198          int32_t var_29c_1 = 0
004571b8          struct WlanCfgAll* pkt = wlan_cfg_all__unpack(0, arg3, arg2)
004571cc          if (pkt == 0) {
004571e4              puts("      wlan_cfg_all__unpack error…")
004571f0              $v0_1 = 0xffffffff
004571f0          } else {
00457228              printf("wlan_guest->from_app = %d__%s(%d…", pkt->from_app,
"confctl_set_guest_wlan", 0x349)
0045724c              GetValue(name: "wl.guest.dhcps_enable", output_buffer:
&wl_guest_dhcps_enable)
00457270              GetValue(name: "wl.guest.dhcps_ip", output_buffer:
&wl_guest_dhcps_ip)
00457294              GetValue(name: "wl.guest.dhcps_mask", output_buffer:
&wl_guest_dhcps_mask)
004572a0              int32_t var_294_1 = 0
00457898              while (true) {
00457898                  if (var_294_1 u>= pkt->wlan_count) {
[3]
004578d8                      printf("djc___wlan_guest->timestamp = %l…", 0x399,
pkt->timestamp.d, pkt->timestamp:4.d, "confctl_set_guest_wlan", 0x399, 0x4ae4b0)
...
00457954                      if (pkt->enable == 0) {
00457974                          SetValue(name: "wl.guest.dhcps_enable",
input_buffer: "0")
[6]
00457998                          SetValue(name: "wl2g.ssid1.enable", input_buffer:
"0")
004579bc                          SetValue(name: "wl5g.ssid1.enable", input_buffer:
"0")
004579d8                          doSystemCmd("ifconfig br0:1 0.0.0.0")
004579cc                      }
...
00457a28                      if (pkt->enable == 1 && (pkt->from_app == 1 || (pkt-
>from_app != 1 && *(*pkt->wlan + 0x30) == 0xffffffff))) {
[4]
00457a64                          printf("djc____timeout is %d____%s(%d)\n", *(*pkt-
>wlan + 0x30), "confctl_set_guest_wlan", 0x3ae)
00457a90                          printf("djc____start____wl.guets.____%s(…",
"confctl_set_guest_wlan", 0x3af)
00457ab4                          SetValue(name: "wl.guest.dhcps_enable",
input_buffer: "1")
[5]
00457ad8                          SetValue(name: "wl2g.ssid1.enable", input_buffer:
"1")
00457afc                          SetValue(name: "wl5g.ssid1.enable", input_buffer:
"1")
00457af0                      }
...
```

At [3] the requirement for at least one `wlan` entry to be filled out is enforced in order to enter the parsing logic. At [4] we can see that if both `enable` and `from_app` are set, we continue on to the logic to enable the guest wireless at [5]. The opposite can be done as well at [6], disabling the guest wireless. All of these actions can be done by an unauthenticated user on the wired or main wireless, by sending a packet to port 9003.

TIMELINE

2022-03-29 - Vendor Disclosure

2022-08-01 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

NEXT REPORT

TALOS-2022-1503

TALOS-2022-1533