Talos Vulnerability Report

TALOS-2022-1462

TCL LinkHub Mesh Wi-Fi confsrv confctl_set_app_language stack-based buffer overflow vulnerability

AUGUST 1, 2022

CVE NUMBER

CVE-2022-23103

SUMMARY

A stack-based buffer overflow vulnerability exists in the confsrv confctl_set_app_language 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 WiFi 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 `APPLang`

```
message APPLang {
    required string lang = 1;                          [1]
    optional uint64 timestamp = 2;
}
```

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

```
00416b4c  int32_t confctl_set_app_language(int32_t arg1, int32_t arg2, int32_t arg3)

00416b6c      arg_0 = arg1
00416b78      int32_t $a3
00416b78      arg_c = $a3
00416b80      int32_t $v0_1
00416b80      if (arg2 == 0) {
00416ba8          printf("[%s][%d][luminais] invalid param…",
"confctl_set_app_language", 0x114)
00416bb4          $v0_1 = 0xffffffff
00416bb4      } else {
00416bc0          int32_t var_224_1 = 0
00416bc4          int32_t var_228_1 = 0
00416be4          uint8_t var_21c[0x100]
00416be4          memset(&var_21c, 0, 0x100)
00416c0c          uint8_t var_11c[0x100]
00416c0c          memset(&var_11c, 0, 0x100)
00416c18          int32_t var_1c = 0
00416c1c          int32_t var_18_1 = 0
00416c20          int32_t var_14_1 = 0
00416c24          int32_t var_10_1 = 0
00416c38          unlink("/var/wan_detect_rst")
00416c60          struct AppLang* pkt = applang__unpack(0, arg3, arg2)
[2]
00416c74          if (pkt == 0) {
00416c9c              printf("[%s][%d][luminais] applang__unpa…",
"confctl_set_app_language", 0x123)
00416ca8              $v0_1 = 0xffffffff
00416ca8          } else {
00416cbc              if (pkt->lang != 0) {
00416ce0                  strcpy(&var_11c, pkt->lang)
[3]
00416d18                  var_224_1 = set_if_changed("sys.app.lang", &var_11c,
&var_21c)
...
```

At [2] the protobuffer is unpacked into a structure, and at [3] the packet data is used directly as the source for a `strcpy`. Below we can verify the issue in ASM:

```
00416cb4  2000c28f   lw      $v0, 0x20($fp) {var_220_1}
00416cb8  0c00428c   lw      $v0, 0xc($v0) {AppLang::lang}
00416cbc  17004010   beqz    $v0, 0x416d1c
[4]
00416cc0  00000000   nop

00416cc4  2000c28f   lw      $v0, 0x20($fp) {var_220_1}
00416cc8  0c00428c   lw      $v0, 0xc($v0) {AppLang::lang}
00416ccc  2401c327   addiu   $v1, $fp, 0x124 {var_11c}
00416cd0  21206000   move    $a0, $v1 {var_11c}
[5]
00416cd4  21284000   move    $a1, $v0
[6]
00416cd8  7c86828f   lw      $v0, -0x7984($gp)  {strcpy}
00416cdc  21c84000   move    $t9, $v0
00416ce0  09f82003   jalr    $t9
00416ce4  00000000   nop
```

Here we can see that at [4] we have a brief check to make sure that the pointer for the packet `lang` value is not NULL, and we see at [5] that a stack-based buffer is being loaded as the destination of `strcpy`. Finally at [6] we see that `AppLang::Lang` which is the user data provided in the protobuffer, is being used directly as the source with no validation of size. This leads to a simple stack-based buffer overflow using `strcpy`.

Crash Information

```
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────── registers ────
$zero: 0x0
$at  : 0x806f0000
$v0  : 0xffffffff
$v1  : 0x0
$a0  : 0x16
$a1  : 0x0
$a2  : 0x0
$a3  : 0x0
$t0  : 0x0
$t1  : 0x0
$t2  : 0x4
$t3  : 0x0
$t4  : 0x8785dd64
$t5  : 0x8000
$t6  : 0x0
$t7  : 0x0
$s0  : 0x7f962e88  →  0x82031007
$s1  : 0x7f962e88  →  0x82031007
$s2  : 0x77dcaa60  →  "uc_api_lib.c"
$s3  : 0x0
$s4  : 0x77dcbbe4  →  "_session_read_and_dispatch"
$s5  : 0x77db1090  →  0x3c1c0003
$s6  : 0x127
$s7  : 0x10
$t8  : 0x0
$t9  : 0x77980750  →  0x3c1c0006
$k0  : 0x0
$k1  : 0x0
$s8  : 0x41414141 ("AAAA"?)
$pc  : 0x41414141 ("AAAA"?)
$sp  : 0x7f962b20  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
$hi  : 0x98
$lo  : 0x1b2a
$fir : 0x0
$ra  : 0x41414141 ("AAAA"?)
$gp  : 0x77e833f0  →  0x00000000
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────── stack ────
0x7f962b20|+0x0000: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"    ←
$sp
0x7f962b24|+0x0004: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x7f962b28|+0x0008: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x7f962b2c|+0x000c: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x7f962b30|+0x0010: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x7f962b34|+0x0014: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x7f962b38|+0x0018: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x7f962b3c|+0x001c: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────── 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.
