Talos Vulnerability Report

TALOS-2022-1482

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

AUGUST 1, 2022

CVE NUMBER

CVE-2022-25996

SUMMARY

A stack-based buffer overflow vulnerability exists in the confsrv addTimeGroup functionality of TCL LinkHub Mesh Wi-Fi MS1G_00_01.00_14. A specially-crafted network packet can lead to a 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 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 `TimeGroup` and `TimeRule`

```
message TimeRule {
    required int32 id = 1;
    required string desc = 2;
    required bool enable = 3;
    required string week = 4;              [1]
    required int32 begin_in_min = 5;
    required int32 end_in_min = 6;
}
message TimeGroup {
    repeated TimeRule tm_rule = 1;
    optional uint64 timestamp = 2;
}
```

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

```
0041acb8  int32_t set_time_group_rule(struct TimeGroup* pkt)

0041acf8      void var_e4
0041acf8      memset(&var_e4, 0, 0x80)
0041ad04      int32_t var_e8 = 0
0041ad08      int32_t var_ec = 0
0041ad0c      int32_t var_f0 = 0
0041ad10      int32_t var_64 = 0
0041ad14      int32_t var_60 = 0
0041ad18      int32_t var_5c = 0
0041ad1c      int32_t var_58 = 0
0041ad20      int32_t var_54 = 0
0041ad24      int32_t var_50 = 0
0041ad40      GetValue(name: "log.time.enable", output_buffer: &var_5c)
0041ad70      if (atoi(&var_5c) == 1) {
0041ada0          printf("Debug->%s: %s(%d)--\n", "../conf_time_group_api.c",
"set_time_group_rule", 0xba)
0041ad94      }
0041af58      uint32_t var_100
0041af58      for (int32_t loop_idx = 0; loop_idx u< pkt->tm_rule_count; loop_idx =
loop_idx + 1) {
0041adcc          struct TimeRule* $v0_5 = *(pkt->tm_rules + (loop_idx << 2))
0041add4          int32_t var_4c = 0
0041add8          int32_t var_48_1 = 0
0041addc          int32_t var_44_1 = 0
0041ade0          int32_t var_40_1 = 0
0041adfc          GetValue(name: "log.time.enable", output_buffer: &var_4c)
0041ae2c          if (atoi(&var_4c) == 1) {
0041ae44              var_100 = pkt->tm_rule_count
0041ae68              printf("Debug->%s: %s(%d)--\nn_tm_rule =…",
"../conf_time_group_api.c", "set_time_group_rule", 0xbf, var_100)
0041ae68          }
0041ae78          if ($v0_5 != 0) {
0041ae90              addTimeGroup($v0_5, var_ec)                [2]
...
```

At [2] the parsing is passed off to `addTimeGroup` if the `TimeGroup` reports having more than 0 `tm_rule`. The `TimeRule` protobuffer is passed into `addTimeGroup` for further parsing.

```
00419eb4  int32_t addTimeGroup(struct TimeRule* arg1, int32_t arg2)

00419ef8      uint8_t var_108[0x80]
00419ef8      memset(&var_108, 0, 0x80)
00419f04      uint8_t var_88[0x20]
00419f04      var_88[0].d = 0
00419f08      var_88[4].d = 0
00419f0c      var_88[8].d = 0
00419f10      var_88[0xc].d = 0
00419f14      var_88[0x10].d = 0
00419f18      var_88[0x14].d = 0
00419f1c      var_88[0x18].d = 0
00419f20      var_88[0x1c].d = 0
00419f24      int32_t var_68 = 0
00419f28      int32_t var_64 = 0
00419f2c      int32_t var_60 = 0
00419f30      int32_t var_5c = 0
00419f34      int32_t var_58 = 0
00419f38      int32_t var_54 = 0
00419f44      char var_50 = 0
00419f48      char var_4f = 0
00419f4c      char var_4e = 0
00419f50      char var_4d = 0
00419f54      char var_4c = 0
00419f58      char var_4b = 0
00419f5c      char var_4a = 0
00419f60      char var_49 = 0
00419f64      int32_t var_10c = 0
00419f68      int32_t var_110 = 0
00419f6c      int32_t var_48 = 0
00419f70      int32_t var_44 = 0
00419f74      int32_t var_40 = 0
00419f78      int32_t var_3c = 0
00419f7c      int32_t var_38 = 0
00419f80      int32_t var_34 = 0
00419f84      int32_t var_30 = 0
00419f88      int32_t var_2c = 0
00419f8c      int32_t var_28 = 0
00419f90      int32_t var_24 = 0
00419f94      int32_t var_20 = 0
00419f98      int32_t var_1c = 0
00419fb4      GetValue(name: "log.time.enable", output_buffer: &var_28)
00419fe4      if (atoi(&var_28) == 1) {
0041a014          printf("Debug->%s: %s(%d)--\n", "../conf_time_group_api.c",
"addTimeGroup", 0x38)
0041a008      }
0041a028      if (arg1->week != 0) {
0041a04c          strcpy(&var_68, arg1->week)               [3]
0041a034      }
...
```

At [3] we can clearly see that if `week` is populated within the `TimeRule` protobuf, a `strcpy` is performed without any validation of buffer or input length, which can lead to a stack-based buffer overflow. Below we can verify the issue in ASM:

```
0041a024  1800428c   lw      $v0, 0x18($v0) {TimeRule::week}      [4]
0041a028  0b004010   beqz    $v0, 0x41a058
0041a02c  00000000   nop

0041a030  4001c28f   lw      $v0, 0x140($fp) {arg_0}
0041a034  1800428c   lw      $v0, 0x18($v0) {TimeRule::week}
0041a038  d800c327   addiu   $v1, $fp, 0xd8 {var_68}
0041a03c  21206000   move    $a0, $v1 {var_68}                    [5]
0041a040  21284000   move    $a1, $v0
0041a044  7c86828f   lw      $v0, -0x7984($gp)  {strcpy}
0041a048  21c84000   move    $t9, $v0
0041a04c  09f82003   jalr    $t9                                [6]
0041a050  00000000   nop
```

At [4] we see that the `week` value is loaded from the protobuf and checked to see if it is non-zero. At [5] the stack-buffer is being loaded as the `dst` argument for `strcpy`, and at [6] we see that `strcpy` is being called with no additional validation or verification of the buffer or input length. This leads to a simple stack-based buffer overflow.

Crash Information

```
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
[ Legend: Modified register | Code | Heap | Stack | String ]
─────────────────────────────────────────────────────────────────────────────────────
───────────────────── registers ────
$zero: 0x0
$at  : 0x806f0000
$v0  : 0x1
$v1  : 0x0
$a0  : 0x0
$a1  : 0x7fffffff
$a2  : 0xa
$a3  : 0x1
$t0  : 0x7fadd019  →  0x00000000
$t1  : 0x7fadd018  →  0x00000000
$t2  : 0x5
$t3  : 0x19999999
$t4  : 0x777fa8e4  →  0x0000000b ("
                                   "?)
$t5  : 0x0
$t6  : 0x22
$t7  : 0x0
$s0  : 0x7fadd348  →  0x82011607
$s1  : 0x7fadd348  →  0x82011607
$s2  : 0x77612a60  →  "uc_api_lib.c"
$s3  : 0x0
$s4  : 0x77613be4  →  "_session_read_and_dispatch"
$s5  : 0x775f9090  →   lui gp, 0x3
$s6  : 0x7e
$s7  : 0x10
$t8  : 0x1
$t9  : 0x0
$k0  : 0x0
$k1  : 0x0
$s8  : 0x41414141 ("AAAA"?)
$pc  : 0x41414141 ("AAAA"?)
$sp  : 0x7fadd030  →  0x004bb900  →  0x004bba68  →  "CMD_MESH_WLAN_SET"
$hi  : 0x5
$lo  : 0x19999999
$fir : 0x0
$ra  : 0x41414141 ("AAAA"?)
$gp  : 0x004ae4b0  →  0x00000000
─────────────────────────────────────────────────────────────────────────────────────
───────────────────────── stack ────
0x7fadd030│+0x0000: 0x004bb900  →  0x004bba68  →  "CMD_MESH_WLAN_SET"    ← $sp
0x7fadd034│+0x0004: 0x00000000
0x7fadd038│+0x0008: 0x00000000
0x7fadd03c│+0x000c: 0x00000000
0x7fadd040│+0x0010: 0x00000000
0x7fadd044│+0x0014: 0x00000000
0x7fadd048│+0x0018: 0x004ae4b0  →  0x00000000
0x7fadd04c│+0x001c: 0x00000000
─────────────────────────────────────────────────────────────────────────────────────
────────────── code:mips:MIPS32 ────
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x41414140
─────────────────────────────────────────────────────────────────────────────────────
─────────────────────── threads ────
```

```
[#0] Id 1, stopped 0x41414141 in ?? (), reason: SIGSEGV
────────────────────────────────────────────────────────────────────────────────────
───────────────────────── trace ────
────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────
```

TIMELINE

2022-03-16 - Vendor Disclosure

2022-08-01 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2022-1463

NEXT REPORT

TALOS-2022-1484