Talos Vulnerability Report

TALOS-2022-1455

TCL LinkHub Mesh Wifi confsrv set_mf_rule stack-based buffer overflow vulnerability

AUGUST 1, 2022

CVE NUMBER

CVE-2022-23919,CVE-2022-23918

SUMMARY

A stack-based buffer overflow vulnerability exists in the confsrv set_mf_rule functionality of TCL LinkHub Mesh Wifi 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 WiFi system is a node-based mesh system designed for wifi deployments across large homes. These nodes include most features standard in current WiFi 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 `mf_lists`

```
message mf_rule {
    required string ethaddr = 1;                                    [1]
    optional string name = 2;                                       [2]
}
message mf_lists {
    required int32 mode = 1;
    repeated MESSAGE_NOT_RESOLVED rules = 2;    //This is not optional, so it must
be resolved by hand to compile to .proto
    optional uint64 timestamp = 3;
}
```

Using [1] and [2] we have control over both `ethaddr` and `name` in the packet, the parsing of the data within the protobuffer is `conf_set_mf_cfg`

```
004141f0  int32_t conf_set_mf_cfg(int32_t arg1, int32_t arg2, int32_t arg3)

00414210      arg_0 = arg1
0041421c      int32_t $a3
0041421c      arg_c = $a3
0041423c      void var_108
0041423c      memset(&var_108, 0, 0x80)
00414264      void var_88
00414264      memset(&var_88, 0, 0x80)
00414288      struct MfLists* pkt = mf_lists__unpack(0, arg3, arg2)
[3]
0041429c      int32_t $v0_2
0041429c      if (pkt == 0) {
004142c8          printf("[%s][%d][niuwu] Unpack failed %d…", "conf_set_mf_cfg",
0x103, arg3, 0x4ae4b0)
004142d4          $v0_2 = 0xffffffff
004142d4      } else {
004142e8          clear_all_mf_mib()
00414300          set_mf_rule(pkt: pkt)
[4]
00414314          if (pkt->is_timestamp_present != 0) {
00414344              sprintf(&var_88, "%llu", pkt->timestamp.d, pkt->timestamp:4.d,
0x4ae4b0)
00414368              SetValue(name: "sys.cfg.stamp", input_buffer: &var_88)
0041435c          }
...
```

At [3] the protobuffer is unpacked into a structure and then at [4] the structure is passed into `set_mf_rule`

```
00413b2c  int32_t set_mf_rule(struct MfLists* pkt)

00413b54      uint8_t ethAddrBuffer[0x12]
00413b54      ethAddrBuffer[0].d = 0
00413b58      ethAddrBuffer[4].d = 0
00413b5c      ethAddrBuffer[8].d = 0
00413b60      ethAddrBuffer[0xc].d = 0
00413b64      ethAddrBuffer[0x10].w = 0
00413b84      uint8_t nameBuffer[0x40]
00413b84      memset(&nameBuffer, 0, 0x40)
00413bac      uint8_t var_150[0x40]
00413bac      memset(&var_150, 0, 0x40)
00413bd4      uint8_t var_110[0x80]
00413bd4      memset(&var_110, 0, 0x80)
00413bfc      uint8_t var_90[0x80]
00413bfc      memset(&var_90, 0, 0x80)
00413c08      int32_t var_1a8 = 0
00413c0c      int32_t var_1ac = 0
00413c18      int32_t var_1b8 = 0
00413c1c      int32_t var_1ac_1 = 0
00414080      int32_t $v0_31
00414080      while (true) {
00414080          $v0_31 = var_1ac_1 s< 2 ? 1 : 0
00414084          if ($v0_31 == 0) {
00414084              break
00414084          }
00413c2c          int32_t var_1b0_1
00413c2c          if (var_1ac_1 != 0) {
00413c48              var_1b0_1 = 5
00413c48          } else {
00413c38              var_1b0_1 = 2
00413c38          }
00413f54          uint8_t (* var_1c8)[0x40]
00413f54          for (int32_t loop_idx = 0; loop_idx u< pkt->rules_count; loop_idx
= loop_idx + 1) {
00413c6c              struct MfRule* $v0_6 = *(pkt->rules + (loop_idx << 2))
00413c8c              memset(&ethAddrBuffer, 0, 0x12)
[5]
00413cb0              memset(&nameBuffer, 0, 0x40)
[6]
00413cc0              if ($v0_6 != 0) {
00413cd0                  if ($v0_6->ethAddr != 0) {
?00413d18                     memcpy(&ethAddrBuffer, $v0_6->ethAddr, strlen($v0_6-
>ethAddr))           [7]
00413ce4                  }
00413d2c                  if ($v0_6->name != 0) {
?00413d74                     memcpy(&nameBuffer, $v0_6->name, strlen($v0_6->name))
[8]
00413d40                  }
....
```

CVE-2022-23918 - ethAddr stack buffer overflow

As seen above at [5] the `ethAddr` `memcpy` occurs into a stack-based buffer of size 0x12.

```
00413cd8  2000c28f   lw      $v0, 0x20($fp) {var_1b8_1}
00413cdc  0c00508c   lw      $s0, 0xc($v0) {MfRule::ethAddr}
00413ce0  2000c28f   lw      $v0, 0x20($fp) {var_1b8_1}
00413ce4  0c00428c   lw      $v0, 0xc($v0) {MfRule::ethAddr}
00413ce8  21204000   move    $a0, $v0
00413cec  c08a828f   lw      $v0, -0x7540($gp)  {strlen}        [9]
00413cf0  21c84000   move    $t9, $v0
00413cf4  09f82003   jalr    $t9
00413cf8  00000000   nop
00413cfc  1800dc8f   lw      $gp, 0x18($fp) {var_1c0}
00413d00  3400c327   addiu   $v1, $fp, 0x34 {ethAddrBuffer}
00413d04  21206000   move    $a0, $v1 {ethAddrBuffer}
00413d08  21280002   move    $a1, $s0
00413d0c  21304000   move    $a2, $v0
00413d10  b88b828f   lw      $v0, -0x7448($gp)  {memcpy}
00413d14  21c84000   move    $t9, $v0
00413d18  09f82003   jalr    $t9
00413d1c  00000000   nop
```

At [7] and [9] we can see that the length of the `memcpy` is not the static size of the buffer, but instead the `strlen` of the user data provided from the protobuf packet. This results in a simple stack 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  : 0x0
$v1  : 0x2
$a0  : 0x11
$a1  : 0x2
$a2  : 0x200
$a3  : 0x0
$t0  : 0x1
$t1  : 0x41414141 ("AAAA"?)
$t2  : 0x41414141 ("AAAA"?)
$t3  : 0x41414141 ("AAAA"?)
$t4  : 0x41414141 ("AAAA"?)
$t5  : 0x41414141 ("AAAA"?)
$t6  : 0x41414141 ("AAAA"?)
$t7  : 0x41414141 ("AAAA"?)
$s0  : 0x41414141 ("AAAA"?)
$s1  : 0x7fe05d48  →  0x82011507
$s2  : 0x772f6a60  →  "uc_api_lib.c"
$s3  : 0x0
$s4  : 0x772f7be4  →  "_session_read_and_dispatch"
$s5  : 0x772dd090  →  0x3c1c0003
$s6  : 0x1b6
$s7  : 0x10
$t8  : 0x1
$t9  : 0x76ee752c  →  0x3c1c0002
$k0  : 0x0
$k1  : 0x0
$s8  : 0x41414141 ("AAAA"?)
$pc  : 0x41414141 ("AAAA"?)
$sp  : 0x7fe05af8  →  0x004bbfe8  →  0x772d1c28  →  0x28aaeef9
$hi  : 0x5
$lo  : 0x19999999
$fir : 0x0
$ra  : 0x41414141 ("AAAA"?)
$gp  : 0x004ae4b0  →  0x00000000
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────── stack ────
0x7fe05af8|+0x0000: 0x004bbfe8  →  0x772d1c28  →  0x28aaeef9     ← $sp
0x7fe05afc|+0x0004: 0x000001ae
0x7fe05b00|+0x0008: 0x7fe05d6c  →  0xa9120108
0x7fe05b04|+0x000c: 0x00000000
0x7fe05b08|+0x0010: 0x004ae4b0  →  0x00000000
0x7fe05b0c|+0x0014: 0x00000000
0x7fe05b10|+0x0018: 0x004bbfe8  →  0x772d1c28  →  0x28aaeef9
0x7fe05b14|+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 ────
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────
```

CVE-2022-23919 - name stack buffer overflow

As seen above at [5] the `name` `memcpy` occurs into a stack-based buffer of size 0x40.

```
00413d34  2000c28f   lw      $v0, 0x20($fp) {var_1b8_1}
00413d38  1000508c   lw      $s0, 0x10($v0) {MfRule::name}
00413d3c  2000c28f   lw      $v0, 0x20($fp) {var_1b8_1}
00413d40  1000428c   lw      $v0, 0x10($v0) {MfRule::name}
00413d44  21204000   move    $a0, $v0
00413d48  c08a828f   lw      $v0, -0x7540($gp)  {strlen}        [10]
00413d4c  21c84000   move    $t9, $v0
00413d50  09f82003   jalr    $t9
00413d54  00000000   nop
00413d58  1800dc8f   lw      $gp, 0x18($fp) {var_1c0}
00413d5c  4800c327   addiu   $v1, $fp, 0x48 {nameBuffer}
00413d60  21206000   move    $a0, $v1 {nameBuffer}
00413d64  21280002   move    $a1, $s0
00413d68  21304000   move    $a2, $v0
00413d6c  b88b828f   lw      $v0, -0x7448($gp)  {memcpy}
00413d70  21c84000   move    $t9, $v0
00413d74  09f82003   jalr    $t9
00413d78  00000000   nop
```

At [8] and [10] we can see that the length of the `memcpy` is not the static size of the buffer, but instead the `strlen` of the user data provided from the protobuf packet. This results in a simple stack 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  : 0x0
$v1  : 0x2
$a0  : 0x11
$a1  : 0x2
$a2  : 0x200
$a3  : 0x0
$t0  : 0x0
$t1  : 0x41414141 ("AAAA"?)
$t2  : 0x004bc828  →  0x0045d3e0  →  <add_results_timeout_check+0> lui gp, 0x5
$t3  : 0x5
$t4  : 0xfffffffc
$t5  : 0xfffffffe
$t6  : 0x770bd534  →  0x00000000
$t7  : 0x0
$s0  : 0x41414141 ("AAAA"?)
$s1  : 0x7f9a90b8  →  0x82011507
$s2  : 0x774a6a60  →  "uc_api_lib.c"
$s3  : 0x0
$s4  : 0x774a7be4  →  "_session_read_and_dispatch"
$s5  : 0x7748d090  →  0x3c1c0003
$s6  : 0x1b3
$s7  : 0x10
$t8  : 0x264
$t9  : 0x7709752c  →  0x3c1c0002
$k0  : 0x0
$k1  : 0x0
$s8  : 0x41414141 ("AAAA"?)
$pc  : 0x41414141 ("AAAA"?)
$sp  : 0x7f9a8e68  →  0x004bb610  →  0x77481c28  →  0x28aaeef9
$hi  : 0x31a
$lo  : 0x1cbe9
$fir : 0x0
$ra  : 0x41414141 ("AAAA"?)
$gp  : 0x004ae4b0  →  0x00000000
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────── stack ────
0x7f9a8e68|+0x0000: 0x004bb610  →  0x77481c28  →  0x28aaeef9     ← $sp
0x7f9a8e6c|+0x0004: 0x000001ab
0x7f9a8e70|+0x0008: 0x7f9a90dc  →  0xa6120108
0x7f9a8e74|+0x000c: 0x00000000
0x7f9a8e78|+0x0010: 0x004ae4b0  →  0x00000000
0x7f9a8e7c|+0x0014: 0x00000000
0x7f9a8e80|+0x0018: 0x004bb610  →  0x77481c28  →  0x28aaeef9
0x7f9a8e84|+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

2022-02-08 - Initial Vendor Contact

2022-02-09 - Vendor Disclosure

2022-08-01 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.
