Talos Vulnerability Report

TALOS-2022-1454

# TCL LinkHub Mesh Wifi confsrv set_port_fwd_rule stack-based buffer overflow vulnerability

AUGUST 1, 2022

## CVE NUMBER

CVE-2022-23399

## SUMMARY

A stack-based buffer overflow vulnerability exists in the confsrv set_port_fwd_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 `PortFwdLIst`

```
message PortFwdCfg {
    required string ethaddr = 1;
    required int32 protocol = 2;
    required int32 in_port = 3;
    required int32 ext_port = 4;
    optional string ipadr = 5;                  [1]
    optional string name = 6;                   [2]
    optional int32 in_port_end = 7;
    optional int32 ext_port_end = 8;
    optional int32 wan_interface = 9;
}
message PortFwdList {
    repeated PortFwdCfg rule = 1;    //This is not optional, so it must be resolved
by hand to compile to .proto
    optional uint64 timestamp = 2;
}
```

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

```
00415a44  int32_t conf_set_port_fwd_cfg(int32_t arg1, int32_t arg2, int32_t arg3)

00415a64      arg_0 = arg1
00415a70      int32_t $a3
00415a70      arg_c = $a3
00415a90      void var_108
00415a90      memset(&var_108, 0, 0x80)
00415ab8      void var_88
00415ab8      memset(&var_88, 0, 0x80)
00415adc      struct PortFwdList* pkt = port_fwd_list__unpack(0, arg3, arg2)
[3]
00415af0      int32_t $v0_2
00415af0      if (pkt == 0) {
00415b1c          printf("[%s][%d][niuwu] Unpack failed %d…",
"conf_set_port_fwd_cfg", 0x145, arg3)
00415b28          $v0_2 = 0xffffffff
00415b28      } else {
00415b3c          clear_all_port_fwd_mib()
00415b54          set_port_fwd_rule(pkt: pkt)
[4]
...
```

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

```
004148c0  int32_t set_port_fwd_rule(struct PortFwdList* pkt)

004148e4      int32_t var_170 = 0
004148e8      int32_t var_16c = 0
004148ec      int32_t var_168 = 0
004148f0      int32_t var_164 = 0
004148f4      int16_t var_160 = 0
004148f8      int32_t var_15c = 0
004148fc      int32_t var_158 = 0
00414900      int32_t var_154 = 0
00414904      int32_t var_150 = 0
00414908      int16_t var_14c = 0
00414928      uint8_t var_148[0x40]
00414928      memset(&var_148, 0, 0x40)
00414950      uint8_t var_108[0x80]
00414950      memset(&var_108, 0, 0x80)
00414978      uint8_t var_88[0x80]
00414978      memset(&var_88, 0, 0x80)              [5]
00414984      int32_t var_174 = 0
00414988      int32_t var_180 = 0
0041498c      int32_t var_184 = 0
00414990      int32_t var_188 = 0
00414d20      int32_t pkt_in_port
00414d20      int32_t pkt_in_port_end
00414d20      char* pkt_ipAddr
00414d20      int32_t pkt_protocol
00414d20      char* pkt_name
00414d20      for (int32_t var_174_1 = 0; var_174_1 u< pkt->rule_count; var_174_1 =
var_174_1 + 1) {
004149b4          struct PortFwdCfg* $v0_5 = *(pkt->rules + (var_174_1 << 2))
004149d0          if ($v0_5 != 0 && $v0_5->ipAddr != 0) {
004149e0              if ($v0_5->is_in_port_end_present == 0) {
004149f4                  $v0_5->in_port_end = $v0_5->in_port
004149ec              }
00414a00              if ($v0_5->is_ext_port_end_present == 0) {
00414a14                  $v0_5->ext_port_end = $v0_5->ext_port
00414a0c              }
00414a5c              pkt_in_port = $v0_5->in_port
00414a60              pkt_in_port_end = $v0_5->in_port_end
00414a64              pkt_ipAddr = $v0_5->ipAddr
00414a68              pkt_protocol = $v0_5->protocol
00414a6c              pkt_name = $v0_5->name
00414a80              sprintf(&var_88, "0;%d-%d;%d-%d;%s;%d;1;%s;", $v0_5->ext_port,
$v0_5->ext_port_end, pkt_in_port, pkt_in_port_end, pkt_ipAddr, pkt_protocol,
pkt_name)                   [6]
...
```

At [5] we can see that the static buffer used for `sprintf` is 0x80 bytes. At [6] (below in ASM) we can see that the user provided data from the protobuf packet is being used directly in the `sprintf` which can result in a simple stack-based buffer overflow.

```
00414a18  b480828f   lw      $v0, -0x7f4c($gp)  {data_4a6564}
00414a1c  30b94524   addiu   $a1, $v0, -0x46d0  {data_47b930, "0;%d-%d;%d-
%d;%s;%d;1;%s;"}       [8]
00414a20  4000c28f   lw      $v0, 0x40($fp) {var_178_1}
00414a24  1800438c   lw      $v1, 0x18($v0) {PortFwdCfg::ext_port}
00414a28  4000c28f   lw      $v0, 0x40($fp) {var_178_1}
00414a2c  3000428c   lw      $v0, 0x30($v0) {PortFwdCfg::ext_port_end}
00414a30  4000c48f   lw      $a0, 0x40($fp) {var_178_1}
00414a34  14008a8c   lw      $t2, 0x14($a0) {PortFwdCfg::in_port}
00414a38  4000c48f   lw      $a0, 0x40($fp) {var_178_1}
00414a3c  2800898c   lw      $t1, 0x28($a0) {PortFwdCfg::in_port_end}
00414a40  4000c48f   lw      $a0, 0x40($fp) {var_178_1}
00414a44  1c00888c   lw      $t0, 0x1c($a0) {PortFwdCfg::ipAddr}
[9]
00414a48  4000c48f   lw      $a0, 0x40($fp) {var_178_1}
00414a4c  1000878c   lw      $a3, 0x10($a0) {PortFwdCfg::protocol}
00414a50  4000c48f   lw      $a0, 0x40($fp) {var_178_1}
00414a54  2000868c   lw      $a2, 0x20($a0) {PortFwdCfg::name}
[10]
00414a58  3001c427   addiu   $a0, $fp, 0x130 {var_88}
[7]
00414a5c  1000aaaf   sw      $t2, 0x10($sp) {pkt_in_port}
00414a60  1400a9af   sw      $t1, 0x14($sp) {pkt_in_port_end}
00414a64  1800a8af   sw      $t0, 0x18($sp) {pkt_ipAddr}
00414a68  1c00a7af   sw      $a3, 0x1c($sp) {pkt_protocol}
00414a6c  2000a6af   sw      $a2, 0x20($sp) {pkt_name}
00414a70  21306000   move    $a2, $v1
00414a74  21384000   move    $a3, $v0
00414a78  0088828f   lw      $v0, -0x7800($gp)  {sprintf}
00414a7c  21c84000   move    $t9, $v0
00414a80  09f82003   jalr    $t9
00414a84  00000000   nop
```

Here we can see at [7] that the stack buffer is the first argument of `sprintf`, [8] shows the format string that is being used, while [9] and [10] show both controllable inputs into the format string with the `%s` formatter. Both `ipAddr` and `name` can be used to trigger this stack-based buffer overflow.

### Crash Information

```
Program received signal SIGSEGV, Segmentation fault.
0x00414d18 in set_port_fwd_rule ()
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────── registers ────
$zero: 0x0
$at  : 0x771ee3ab  →  "anage][472][luminais] Login [0]\nfo [0]\ny = devic[...]"
$v0  : 0x41414141 ("AAAA"?)
$v1  : 0x1
$a0  : 0x771eb1bc  →  0x00000000
$a1  : 0x771ee3ab  →  "anage][472][luminais] Login [0]\nfo [0]\ny = devic[...]"
$a2  : 0x0
$a3  : 0x0
$t0  : 0x0
$t1  : 0x2
$t2  : 0x1
$t3  : 0xfffffff8
$t4  : 0x807
$t5  : 0x800
$t6  : 0x0
$t7  : 0x8
$s0  : 0x7fdd2ba8  →  0x82011707
$s1  : 0x7fdd2ba8  →  0x82011707
$s2  : 0x775daa60  →  "uc_api_lib.c"
$s3  : 0x0
$s4  : 0x775dbbe4  →  "_session_read_and_dispatch"
$s5  : 0x775c1090  →  0x3c1c0003
$s6  : 0xa2
$s7  : 0x10
$t8  : 0x10
$t9  : 0x771cb52c  →  0x3c1c0002
$k0  : 0x0047b8d1  →  0x61000000
$k1  : 0x0
$s8  : 0x7fdd2798  →  0x004bc428  →  0x00000000
$pc  : 0x00414d18  →  0x8c42000c ("
                                  "?)
$sp  : 0x7fdd2798  →  0x004bc428  →  0x00000000
$hi  : 0x0
$lo  : 0x0
$fir : 0x0
$ra  : 0x00414b28  →  0x8fdc0028 ("("?)
$gp  : 0x004ae4b0  →  0x00000000
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────── stack ────
0x7fdd2798|+0x0000: 0x004bc428  →  0x00000000    ← $s8, $sp
0x7fdd279c|+0x0004: 0x7fdd28c8  →  "0;8088-8088;8088-
8088;;1;1;AAAAAAAAAAAAAAAAAAAAAA[...]"
0x7fdd27a0|+0x0008: 0x00000001
0x7fdd27a4|+0x000c: 0x0047b949  →  0x31000000
0x7fdd27a8|+0x0010: 0x00001f98
0x7fdd27ac|+0x0014: 0x00001f98
0x7fdd27b0|+0x0018: 0x004bc2d8  →  0x771f1500  →  0x00000000
0x7fdd27b4|+0x001c: 0x00000001
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────── code:mips:MIPS32 ────
```

```
    0x414d0c <set_port_fwd_rule+1100> sw     v0, 68(s8)
    0x414d10 <set_port_fwd_rule+1104> lw     v1, 68(s8)
    0x414d14 <set_port_fwd_rule+1108> lw     v0, 440(s8)
 →  0x414d18 <set_port_fwd_rule+1112> lw     v0, 12(v0)
    0x414d1c <set_port_fwd_rule+1116> sltu   v0, v1, v0
    0x414d20 <set_port_fwd_rule+1120> bnez   v0, 0x4149a0 <set_port_fwd_rule+224>
    0x414d24 <set_port_fwd_rule+1124> nop
    0x414d28 <set_port_fwd_rule+1128> lw     v0, -32588(gp)
    0x414d2c <set_port_fwd_rule+1132> addiu  v0, v0, -18220
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────── threads ────
[#0] Id 1, stopped 0x414d18 in set_port_fwd_rule (), reason: SIGSEGV
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────── trace ────
[#0] 0x414d18 → set_port_fwd_rule()
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────-
```

## 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.

---