TrendNet Wireless Camera buffer overflow vulnerability

Munawwar
11-May-2020

CVE Details

ID : CVE-2020-12763

Advisory

Description

TrendNet ProView Wireless camera TV-IP512WN (version v1.0R) is vulnerable to buffer overflow in handling RTSP packet in firmware version 1.0.4 which may result in remote code execution or denial of service. The issue is in the binary rtspd which resides in /sbin folder which is responsible for serving rtsp connection received by the device. The problem arises in parsing "Authorization: Basic" RTSP header which could be arbitrarily long, the value of this header is copied onto stack memory without any bounds check which could lead to a buffer overflow. What makes this vulnerability more severe is that the user need not be authenticated to trigger the overflow.

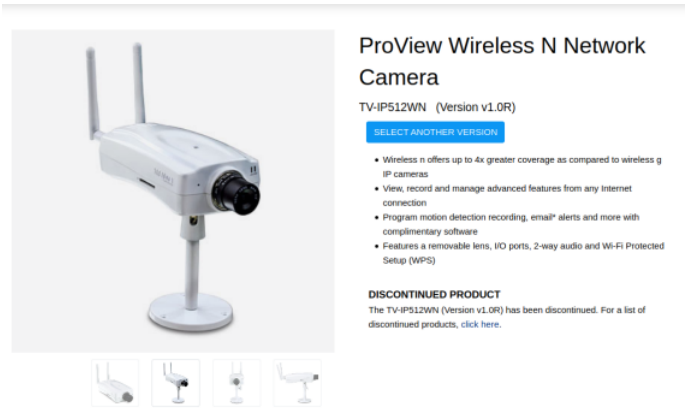
Before we look at the technical details it is important to note that:

1. The vulnerability was discovered in the latest version of the firmware.
2. The vulnerability has been reported to TrendNet.
3. TrendNet disagreed to verify the vulnerability on the basis that the device has reached End-Of-Life.

Vulnerable Product

| | | |----------|---| | Manufacturer | TrendNet | | Affected Model | ProView Wireless N Network Camera TV-IP512WN (Version v1.0R) | | Firmware | FW_TV-IP512P_512WN(1.0.4).zip | | SHA-1 Checksum | B27998BBB4C8503E9314730F504E389B56AD6F6C | | Product URL | Link |

Product Images



Firmware Details

Firmware

DO NOT upgrade firmware on any TRENDnet product using wireless connection.
Firmware upgrade over wireless connection may damage the product.
Please perform firmware upgrade with "wired" network connection only

Date	File Size	
7/27/2017	6.71 mb	Download

Firmware Version: 1.0.4
Firmware Release Date: 5/2013
Note:

1. Fixes security vulnerabilities

Filename: FW_TV-IP512P_512WN(1.0.4).zip
SHA-1 Checksum: B27998BBB4C8503E9314730F504E389B56AD6F6C

Vulnerability Details

One thing to note is that the bug is discovered by doing static analysis, as I am faced issue in procuring the device due to covid-19 situation, producing the crash will little difficult. I will present many screenshots of the decompiled code in the post and to make the code more readable I have renamed lots of variable/function names.

The vulnerability a CWE-120 classic stack overflow which can lead to remote code execution(RCE) or denial of service(DOS). The overflow is in rtspd binary which is in sbin folder. This binary is responsible for handling rtsp connection received by the device.

The overflow occurs while parsing *Authorization: Basic* header in function(*address 0x11a18*) let refer to it as *parse_auth_header*. This function receives two parameters when called, first is the buffer which holds the request RTSP header and the second parameter is where the parsed header value is copied, below is the disassembly of that function.

```
 2 int parse_auth_header(char *src_buf,void *dest_buf)
 3
 4 {
 5   int is_equal;
 6   char *newline_ptr;
 7   char *src_buf_ptr;
 8
 9   src_buf_ptr = src_buf;
10   while( true ) {
11     if (*src_buf_ptr == '\0') {
12       return 0;
13     }
14     is_equal = strncmp(src_buf_ptr,"Authorization: Basic ",0x15);
15     if (is_equal == 0) break;
16     src_buf_ptr = src_buf_ptr + 1;
17   }
18   src_buf_ptr = src_buf_ptr + 0x15;
19   newline_ptr = strstr(src_buf_ptr,"\r\n");
20   if (newline_ptr != (char *)0x0) {
21     memcpy(dest_buf,src_buf_ptr,(size_t)(newline_ptr + -(int)src_buf_ptr));
22     return 1;
23   }
24   return 0;
25 }
26
```

Figure A: Disassembly of function which does the Authorization header parsing (parse_auth_header)

Let try to understand this function a little bit :

1. As you can see from the above code, **line 10-17** is a loop which searchers for the start of "Authorization: Basic" header and if found breaks out of the loop and forwards the pointer header by 0x15(which is the length of header key),
2. Then *strstr function* call is made to search for `\r\n` (end-of-line character) which returns a (char *)pointer.
3. The pointers created in above two points are used to calculate the size of the *Authorization Basic* header value by subtracting those two pointers. This value is used to find the size of the header value.
4. Then the *memcpy* is done from *char* * pointer from point 1 into the second parameter *dest_buf* of the function. It is at this point where the overflow could take place as no precaution is taken to check if the *dest_buf* is large enough to hold the auth header value. If the Authorization Basic header is very long then it can overflow the *dest_buf* buffer. I will prove it in a while.

Now let's look at what are the parameter that is passed to this function. The function at **0x11ae8**, let call it *check_authentication*, call *parse_auth_header* function. Let look at the disassembly of the *check_authentication* function.

```
 2 int check_authentication(undefined4 param_1,undefined4 param_2,undefined4 param_3,char *param_4)
 3
 4 {
 5   int is_auth_header_found;
 6   undefined4 uStack00000000;
 7   int local_480;
 8   undefined auStack1144 [1024];
 9   undefined auth_header [64];
10   int local_38;
11   void *local_34;
12   void *local_30;
13   void *local_2c;
14   void *local_28;
15   void *local_24;
16   char *http_header_ptr;
17   undefined4 local_1c;
18   undefined4 local_18;
19   undefined4 local_14;
20   undefined4 uStack4;
21
22   uStack4 = 0x11af4;
23   local_24 = (void *)0x0;
24   local_28 = (void *)0x0;
25   local_2c = (void *)0x0;
26   local_30 = (void *)0x0;
27   local_34 = (void *)0x0;
28   local_38 = 0;
29   uStack00000000 = 0;
30   http_header_ptr = param_4;
31   local_1c = param_3;
32   local_18 = param_2;
33   local_14 = param_1;
34   memset(auth_header,0,64);
35   is_auth_header_found = parse_auth_header(http_header_ptr,auth_header);
36   if (is_auth_header_found == 0) {
37     if (local_24 != (void *)0x0) {
38       operator.delete[](local_24);
39     }
40     if (local_28 != (void *)0x0) {
41       operator.delete[](local_28);
```

```
int __stdcall parse_auth_header(char * src_buf, void * dest_buf)
  int      r0:4  <RETURN>
  char *   r0:4  src_buf
  void *   r1:4  dest_buf
```

Figure B: The decompiled code of *check_authentication* function

We can see on **line 36** is call *parse_auth_header function*. The first parameter is the pointer of the first parameter of *check_authentication* function itself and the second parameter is the pointer to a char buffer which is of size 64. I hope you can see the problem here.

The second parameter(*dest_buf*) to *parse_auth_header* is of fixed size (64 bytes) and then is buffer is used in copying the data in *parse_auth_header* function at that point no bounds check done to ensure that size of the header value is less than or equal to *dest_buf* size.

This is all good but how do we know that this function processes the data received from the socket? good question. Let's see what function is calling *check_authentication* function. Let's call this function as *is_client_authenticated*, below is the disassembly of it:

```
 1
 2  int is_client_authenticated(astruct_3 *param_1,char *command,undefined4 param_3,char *param_4)
 3
 4  {
 5    int is_authenticated;
 6    undefined4 uVar1;
 7    size_t buf_size;
 8    undefined4 local_1034;
 9    char res_buf [4096];
10    undefined4 local_28;
11    char *local_24;
12    undefined4 local_20;
13    char *local_1c;
14    astruct_3 *local_18;
15
16    if (*(char *)&param_1->field_0x4 != '\0') {
17      local_28 = 1;
18      local_1034 = 1;
19      local_24 = param_4;
20      local_20 = param_3;
21      local_1c = command;
22      local_18 = param_1;
23      is_authenticated = check_authentication(param_1,command,param_3,param_4);
24      if (is_authenticated == 0) {
25        printf("%s %s %d\n","rtsphandler.cpp","checkAuthentication",0x38a,local_1034);
26        FUN_00019834(local_18 + 1,"\"/\"");
27        memset(res_buf,0,4096);
28        local_1034 = FUN_0000e068(local_18);
29        uVar1 = FUN_00012788(local_18 + 1);
30        snprintf(res_buf,0x1000,
31                 "RTSP/1.0 401 Unauthorized\r\nCSeq: %s\r\n%sWWW-Authenticate: Basic realm=%s\r\n\r\n"
32                 ,local_20,local_1034,uVar1);
33        buf_size = strlen(res_buf);
34        send(local_18->client_sock_fd,res_buf,buf_size,0);
35        return 1;
36      }
37    }
38    return 0;
39  }
```

Figure C : The decompiled code of *is_client_authenticated* function

On **line 23** *check_authentication* function is called and based on the return value if block is executed. In *if block (line24 - 35)* some error message is printed on the console with the file and function name, and also there is a string which is appended to the buffer which has the string **RTSP/1.0 401 Unauthorized**, this buffer is then used in *send* call, which is a function that writes data to the socket. This proves these series of function are executing authentication and an overflow can be triggered by providing very long *Authorization Basic* header.

Let's investigate it further, find all the reference to is_client_authenticate function will help us understand when is authentication check is done. Let's look at those references

```
16    local_14 = param_1;
17    iVar1 = is_client_authenticated(param_1,"PAUSE",param_2,param_3);
18    if (iVar1 == 0) {
19      memset(acStack4124,0,0x1000);
20      sprintf(acStack4124,"RTSP/1.0 405 PAUSE\r\nCSeq: %s\r\nConnection: Close\r\n\r\n",local_18);
21      __n = strlen(acStack4124);
22      send(*(int *)(local_14 + 8),acStack4124,__n,0);
23      printf("\n[RTSPd]: handleCmd_Pause Stop VideoSesseion...m_amr_is_on:%d\n",
24             *(undefined4 *)(local_14 + 0x130));
25      FUN_00016450(local_14 + 0x40);
26      printf("\n[RTSPd]: handleCmd_Pause Stop AudioSesseion...m_amr_is_on:%d\n",
27             *(undefined4 *)(local_14 + 0x130));
28      FUN_00016450(local_14 + 0x94);
29    }
30    return;
31  }
```

Figure D

```
18    iVar1 = is_client_authenticated(param_1,"GET_PARAMETER",param_2,param_3);
19    if (iVar1 == 0) {
20      memset(acStack4124,0,0x1000);
21      uVar2 = FUN_0000e068(local_14);
22      sprintf(acStack4124,"RTSP/1.0 200 OK\r\nCSeq: %s\r\n%sSession: %d\r\n\r\n",local_18,uVar2,
23              *(undefined4 *)(local_14 + 0x38));
24      __n = strlen(acStack4124);
25      send(*(int *)(local_14 + 8),acStack4124,__n,0);
26    }
27    return;
```

Figure E

```
      iVar1 = is_client_authenticated(local_30,"DESCRIBE",local_34,param_5);
      if (iVar1 == 0) {
        memset(acStack4252,0,0x1000);
        uStack8392 = 0xffffffff;
        FUN_00010472c(local_30,acStack4252);
        memset(&uStack8352,0,0x1000);
        local_211c = FUN_0000e068(local_30);
        local_2128 = strlen(acStack4252);
        local_2130 = local_38;
        local_212c = local_3c;
        local_2124 = acStack4252;
        sprintf((char *)&uStack8352,

                "RTSP/1.0 200 OK\r\nCSeq: %s\r\n%sContent-Base: rtsp://%s/%s/\r\nContent-Type:
                application/sdp\r\nContent-Length: %d\r\n\r\n%s"
                ,local_34,local_211c);
        __n = strlen((char *)&uStack8352);
        send(*(int *)(local_30 + 8),&uStack8352,__n,0);
      }
    }
    else {
```

Figure F

```
iVar1 = is_client_authenticated(param_1,"SETUP",param_2,param_4);
if (iVar1 == 0) {
  iStack4136 = 0;
  uStack4140 = 0;
  iStack4144 = 0;
  iStack4148 = 0;
  if (local_20 != 0) {
    iStack4148 = FUN_00011c5c(local_18,local_20);
```

Figure G

```
iVar1 = is_client_authenticated(param_1,"PLAY",param_2,param_5);
if (iVar1 == 0) {
  local_24 = 0;
  local_28 = 0;
  memset(acStack4136,0,0x1000);
  if (*(int *)(local_14 + 0x130) == 0) {
    FUN_000162bc(local_14 + 0x40,&local_28);
    FUN_000162bc(local_14 + 0x94,&local_24);
    uVar2 = FUN_0000e068(local_14);
    sprintf(acStack4136,

            "RTSP/1.0 200 OK\r\nCSeq: %s\r\n%sSession: %d;timeout=80\r\nServer: PVSS\r\nRTP-Info
            url=rtsp://%s/%s/trackID=%d;seq=%d,url=rtsp://%s/%s/trackID=%d;seq=%d\r\nRange:
            npt=now-\r\n\r\n"
            ,local_18,uVar2,*(undefined4 *)(local_14 + 0x38),local_1c,local_20,1,0,local_1c,
            local_20,2,0);
  }
```

Figure H

```
iVar1 = is_client_authenticated(param_1,"TEARDOWN",param_2,param_3);
if (iVar1 == 0) {
  memset(acStack4124,0,0x1000);
  uVar2 = FUN_0000e068(local_14);
  sprintf(acStack4124,"RTSP/1.0 200 OK\r\nCSeq: %s\r\n%s\r\n",local_18,uVar2);
  printf("\n[RTSPd %s %s %d]: handleCmd_TEARDOWN\n","rtsphandler.cpp","handleCmd_TEARDOWN",0x50a);
  FUN_00016450(local_14 + 0x40);
  FUN_00016598(local_14 + 0x40);
  FUN_00016450(local_14 + 0x94);
  FUN_00016598(local_14 + 0x94);
  sleep(1);
}
return;
}
```

Figure I

As you can see from the above code when various camera functionalities are triggered like PAUSE, SETUP, PLAY, TEARDOWN, etc *is_client_authenticated* function call is made to check if the received request has appropriate Authorization to execute the functionality. The return value of zero indicates authenticated, and then the functionality is triggered, else it simply returns.

This wikipedia page show the format of RTSP protocol which also further confirms different functionality which we saw in above Figures D - I.

Another very important point which we need to be aware of is that this bug is triggered while parsing for Authorization header which means this RCE can be triggered without authentication which raised the severity of this bug.

Conclusion

We looked at how a simple bug in packet parsing code can cause a buffer overflow. We also looked at how the attacker can trigger the bug without authentication. We couldn't produce a working PoC of the RCE but in the second part of this post, I will try to emulate the binary with qemu and try to trigger to overflow by creating remote socket connect and create an RCE, until then happy hacking!

EXPLIoT.io

At EXPLIoT.io, we build tools for security testing of Internet of Things (IoT) infrastructure and products.

1. EXPLIoT: IoT security testing and exploitation framework
2. Firmware Auditor: a pre-emptive firmware security analysis tool
3. DIVA [Damn Insecure and Vulnerable Application] Board
4. NANO: a compact hacker-friendly multi-purpose, multi-protocol hardware tool
5. ZigBee Auditor

These are products of our years of experience and expertise in the field of IoT security. We are here to help you better your security posture. For more information, visit https://expliot.io or drop us an email at !

GET STARTED

☷ All Blogs › ✍ Latest Blogs

Gagan.Aggarwal
24-November-2022

Understand the Data Protection Bill 2022 in under 5 minutes

This blog contains a summary of the bill and what organizations need to keep in mind before collecting personal data of their customers.

P3n7a90n
14-October-2022

Code Injection and SQLi in WP ALL Export Pro

This blog contains details on finding Code Injection and SQLi in WP All Export Pro(CVE-2022-3394 and CVE-2022-3395).

Gagan.Aggarwal
3-October-2022

Starters Guide To Cyber Threat Intelligence

This blog explains what CTI is and how it works and its basic components

☷ All News › ⚑ Latest News

Talk, Online
2022-05-28 14:36:10Z

Aseem Jakhar will be giving a talk at cyberstartersconference.

Workshop, Online
2022-05-13 14:42:16Z

Kartheek Lade will be conducting a workshop on "Car hacking 101"

Webinar, Online
2022-04-29 00:47:52Z

Amit prajapat will be delivering a webinar on "Gaining Access to Protected Components In Android".

Subscribe to Our Newsletter

Your E-Mail Address

SUBSCRIBE

or

Follow our Social Media Handles

Payatu

Research Powered Cybersecurity Services and Training. Eliminate security threats through our innovative and extensive security assessments.

Subscribe to our newsletter

Enter your email address.

Services

- IoT Security Testing
- Red Team Assessment
- Product Security
- AI/ML Security Audit
- Web Security Testing
- Mobile Security Testing
- DevSecOps Consulting
- Code Review
- Cloud Security
- Critical Infrastructure

Products

- EXPLIoT
- CloudFuzz

Conference

- NullCon
- Hardwear.io

Resources

- Blog
- E-Book
- Advisory
- Media
- Case Studies
- MasterClass Series
- Securecode.wiki
- Checklist

About

- About Us
- Career
- News
- Contact Us
- Payatu Bandits
- Hardware-Lab
- Disclosure Policy