

Vulnerabilities identified in Wyze cam pan v3

Vulnerabilities at a glance

For the pwn2own 2025 ireland competition, I targeted Wyze cam pan v3 and found two issues, tracked as CVE-2026-38698 and CVE-2026-38699.

These vulnerabilities are present in a communication framework called ThroughTek Kalay (TUTK). This solution is used in a variety of IoT devices, including Wyze Cam Pan v3.

CVE-2026-38698 allows a local authenticated attacker with the DTLS PSK to execute arbitrary code by exploiting a heap overflow bug in `tutk_packet_alloc` function.

CVE-2026-38699 allows a local unauthenticated attacker without the DTLS PSK can DOS the camera by exploiting an Out of Bounds Read bug in `__Fill_ReadBuf` function while copying the DTLS encrypted payload offset in `0x407` Handler.

Technical Walkthrough

CVE-2026-38698

First, wait for the AuthKey to be transmitted over the local network during device discovery, network transition, or client reconnection events. Capture the UDP traffic and decrypt the payload using the `ReverseTransCodePartial` routine, which is the same payload decoding function implemented by the camera firmware and previously reversed in public research. The AuthKey can be extracted from the decrypted payload at offset `0x4A`.

Using the recovered AuthKey, establish a custom session by invoking the `0x601` handler three times. The first invocation is used to leak the device UID. The second initializes `PreSessionInfo` using the captured AuthKey. The third upgrades `PreSessionInfo` into `SessionInfo` by setting the privilege byte at offset `0x19` to `0x2`.

After session establishment, use the `0x407` handler to negotiate the protected communication session. On newer versions, SSL pinning must first be bypassed in the mobile application to inspect the connection flow and extract the required `ENR` value, which is then used to derive the session PSK from its SHA-256 hash.

Once the session is established, transmit a crafted `0x407` payload containing an encrypted AV packet with an AVPacket header length larger than the expected limit (`> 0x438` up to `65536` bytes). Due to insufficient validation of the supplied AVPacket length, the iCamera service performs an out-of-bounds heap operation, resulting in heap corruption and process termination with attacker-controlled contents.

Repeated execution of the proof-of-concept demonstrates reliable crashes at multiple locations caused by corruption from sprayed `0x41414141` heap data. Since multiple heap structures contain function pointers and function arguments, successful corruption of these objects may allow escalation from denial of service to arbitrary code execution on the target device.

Attacker-controlled packet length propagation from encrypted `0x407` payload parsing. The highlighted code extracts `uVar19` from the encrypted payload at offset `0x18` and forwards it as the `size_to_copy` argument without validating whether the supplied length exceeds the expected AV packet boundaries. The source buffer is derived from `dtls_encrypted_payload + iVar14 + 0x24`, allowing attacker-controlled payload data and length fields to be passed directly into downstream packet processing routines.

```

Decompile: _doAVTransNew - (iCamera_v3_pro_symbols)
28 float fVar24;
29 double dVar25;
30 float fVar26;
31 timeval local_540;
32 undefined4 local_538;
33 undefined4 local_534;
34 undefined4 local_530;
35 uint local_38;
36 byte *local_30;
37
38 iVar14 = *(int *)(param_1 + 0x3c);
39 uVar12 = *(ushort *)(dtls_encrypted_payload + 2);
40 uVar18 = (uint)*dtls_encrypted_payload;
41 bVar1 = dtls_encrypted_payload[1];
42 uVar19 = (uint)bVar1;
43 (**(code **)(iVar14 + 0x14))(iVar14);
44 (**(code **)(iVar14 + 0x10))(iVar14);
45 if (uVar18 == 0xc) {
46     pbVar10 = dtls_encrypted_payload + 0x18;
47     if ((bVar1 & 8) == 0) {
48         pbVar10 = dtls_encrypted_payload + 0x10;
49         uVar18 = (uint)*pbVar10;
50         iVar14 = 0;
51         puVar15 = (ushort *)(dtls_encrypted_payload + 8);
52         uVar22 = 8;
53         iVar8 = 8;
54         uVar19 = (uint)*(ushort *)(dtls_encrypted_payload + 0x18);
55         if (uVar18 == 8) goto LAB_0062f9c8;
56 LAB_0062f924:
57         if (uVar18 - 3 < 4) goto LAB_0062f9c8;
58         if (uVar18 == 0xc) {
59             avPutdecodeDataToFifo
60                 (*(undefined4 *) (param_1 + 0x1784), dtls_encrypted_payload + iVar14 + 0x24, uVar19,
61                 puVar15, 0xc);
62             goto switchD_0062f868_caseD_c;
63

```

Propagation of attacker-controlled size and payload buffer into `tutk_packet_Alloc()`. The function receives the untrusted payload pointer (`param_2`) together with the unchecked `size_to_copy` value, forwarding both directly into `tutk_packet_Alloc()` for packet allocation and copy operations. Because the supplied size is not constrained before allocation logic is reached, malformed packet metadata can influence subsequent heap operations.

```
Decompile: avPutdecodeDataToFifo - (ICamera_v3_pro_symbols)
1
2 undefined4
3 avPutdecodeDataToFifo
4 (undefined4 param_1, char *param_2, int size_to_copy, ushort *param_4, uint param_5)
5
6 {
7     ushort uVar1;
8     undefined2 uVar2;
9     int iVar3;
10    int iVar4;
11
12    if (((param_5 & 0xffffffff) == 4) || (param_5 == 8)) {
13        iVar3 = tutk_packet_FifoGetFrmCount(param_1);
14        if (0 < iVar3) {
15            uVar2 = tutk_packet_FifoGetMinFrmNo(param_1);
16            iVar3 = comparePacketNo(uVar2, *param_4);
17            if (iVar3 == -1) {
18                return 0xffffb1dd;
19            }
20        }
21        *param_2 = *param_2 + -1;
22        iVar3 = tutk_packet_FifoSeekByFrmNoPos(param_1, *param_4, (char)param_4[1]);
23    }
24    else {
25        iVar3 = tutk_packet_FifoSeekByFrmNoPos(param_1, *param_4, (char)param_4[1]);
26    }
27    if (iVar3 == 0) {
28        if ((char)param_4[2] == '\0') {
29            return 0;
30        }
31        iVar3 = tutk_packet_Alloc(param_2, size_to_copy, 0, 0, 0x414 - size_to_copy);
32        if (iVar3 == 0) {
33            tutk_packet_FifoRemoveFrameByFrmNo(param_1, *param_4);
34            return 0xffffb1dd;
35        }
36    }
```

Heap overflow condition in `tutk_packet_Alloc()`. The allocation size is calculated with a fixed packet buffer layout (`malloc(param_5 + 0x24 + param_4 + param_2)` / effective bounded packet allocation around `0x414` bytes), while the subsequent `memcpy()` operation copies attacker-controlled data using the corrupted size value without ensuring it fits within the allocated destination region. Supplying a packet length larger than the expected `0x414` boundary results in a heap-based buffer overflow, allowing overwrite of adjacent heap memory structures, including potential function pointers and other sensitive objects.

```
Decompile: tutk_packet_Alloc - (iCamera_v3_pro_symbols)
4
5 {
6     undefined4 *puVar1;
7     void *__dest;
8
9     puVar1 = (undefined4 *)malloc(param_5 + 0x24 + param_4 + param_2);
10    if (puVar1 != (undefined4 *)0x0) {
11        *puVar1 = 0;
12        puVar1[1] = 0;
13        puVar1[2] = 0;
14        puVar1[3] = 0;
15        puVar1[4] = 0;
16        puVar1[5] = 0;
17        puVar1[6] = 0;
18        puVar1[7] = 0;
19        puVar1[8] = 0;
20        if (param_5 != 0) {
21            memset((void *)((int)puVar1 + param_4 + param_2 + 0x24), 0, param_5);
22        }
23        if ((param_4 == 0) || (param_3 == (void *)0x0)) {
24            if ((param_2 != 0) && (param_1 != (void *)0x0)) {
25                puVar1[7] = puVar1 + 9;
26                memcpy(puVar1 + 9, param_1, param_2);
27                puVar1[6] = param_2;
28                return puVar1;
29            }
30        }
31        else {
32            puVar1[8] = puVar1 + 9;
33            *(short *)((int)puVar1 + 0xe) = (short)param_4;
34            memcpy(puVar1 + 9, param_3, param_4);
35            if ((param_2 != 0) && (param_1 != (void *)0x0)) {
36                __dest = (void *)((int)puVar1 + param_4 + 0x24);
37                puVar1[7] = __dest;
38                memcpy(__dest, param_1, param_2);
39            }
40        }
41    }
42}
```

CVE-2026-38699

The attacker first waits for the mobile client to broadcast the AuthKey over the local network during device discovery or reconnection events. After capturing this traffic, the UDP payload is decrypted using the `ReverseTransCodePartial` routine, which is the same function implemented by the camera firmware for payload decoding and was previously reversed and incorporated into Blasty's unwyze exploit. This allows extraction of the AuthKey located at offset `0x4A` within the decoded packet.

Using the recovered credential, a custom session is established by invoking the `0x601` handler three times. The first request leaks the device UID, the second initializes `PreSessionInfo` using the captured AuthKey, and the third upgrades `PreSessionInfo` into `SessionInfo` while setting the privilege byte at offset `0x19` to `0x2`. Once a valid session is created, a crafted `0x407` payload is transmitted with the encrypted payload offset set to `0x90000000`, triggering an out-of-bounds read condition that crashes the iCamera process and results in a denial-of-service condition.

Structure of the malformed `0x407` packet used to trigger the out-of-bounds read vulnerability

0x407 Exploit Packet

Transporting DTLS Encrypted Payload

0x00	0x03	0x04	0x08	0x10	0x14
0x204	Trigger Byte	0x0048	Command ID	0x90000000	49 49 49 49 49 49 49 49 49 49 49 ...
Magic Bytes	Trigger Byte	Length Field	Command ID	Corruption Offset	Generated Session ID

Attacker-controlled offset calculation during 0x407 packet parsing. The highlighted code derives a new pointer by adding a user-controlled offset from `payload + 0x10` to the base payload pointer, without validating whether the resulting address remains within the bounds of the received packet. This enables construction of an out-of-bounds pointer for later processing.

```
Decompile: _IOTC_Packet_Handler - (iCamera_v3_pro_symbols)
1369 if (cmd_id == 0x407) {
1370     cmd_id = (uint)*(ushort*)(payload + 4);
1371     iVar13 = payload + 0x10;
1372     if ((* (byte *) (payload + 3) & 4) != 0) {
1373         return;
1374     }
1375     if ((* (byte *) (payload + 3) & 8) == 0) {
1376         if (*(short *) (payload + 0xc) == 0) {
1377             iVar11 = __Search_Session(ip_addr, port_number);
1378         }
1379         else {
1380             RandomIDValueUpdate(&local_78, 1, *(short *) (payload + 0xc), 0);
1381             if (local_78._0_4_ == 0) {
1382                 return;
1383             }
1384             iVar11 = __Search_SessionByClientRandomID.part.38(&local_78, 1);
1385         }
1386     }
1387     else {
1388         if (cmd_id < 0xc) {
1389             return;
1390         }
1391         RandomIDValueUpdate(&local_78, 0, *(undefined4 *) (payload + 0x14),
1392                             *(undefined4 *) (payload + 0x18));
1393         if ((local_78._0_4_ == 0) ||
1394             (iVar11 = __Search_SessionByClientRandomID.part.38(&local_78, 2), iVar11 < 0
1395             iVar11 = _IOTC_P2P_Connection_Check(payload, sock_ctx, ip_addr, port_number, &lo
1396         )
1397         else {
1398             _IOTC_LAN_Connection_Check.isra.32(iVar11, sock_ctx, ip_addr, port_number);
1399         }
1400         iVar13 = payload + *(int *) (payload + 0x10) + 0x10;
1401         cmd_id = (uint)*(ushort *) (payload + 4) - *(int *) (payload + 0x10);
1402     }
1403     if (iVar11 == -1) {
1404         return;
1405     }
```

Propagation of the out-of-bounds pointer into `__Fill_ReadBuf()`. The derived pointer (`iVar13`), calculated from attacker-controlled packet fields, is passed as the second argument to `__Fill_ReadBuf()` despite potentially referencing memory outside the valid packet region. This allows subsequent operations to read from an invalid memory location.

```
Decompile: _IOTC_Packet_Handler - (iCamera_v3_pro_symbols)
1386     }
1387     else {
1388         if (cmd_id < 0xc) {
1389             return;
1390         }
1391         RandomIDValueUpdate(&local_78,0,*(undefined4 *) (payload + 0x14),
1392                             *(undefined4 *) (payload + 0x18));
1393         if ((local_78._0_4_ == 0) ||
1394             (iVar11 = __Search_SessionByClientRandomID.part.38(&local_78,2), iVar11 < 0
1395             iVar11 = _IOTC_P2P_Connection_Check(payload,sock_ctx,ip_addr,port_number,&lo
1396         )
1397         else {
1398             _IOTC_LAN_Connection_Check.isra.32(iVar11,sock_ctx,ip_addr,port_number);
1399         }
1400         iVar13 = payload + *(int *) (payload + 0x10) + 0x10;
1401         cmd_id = (uint)*(ushort *) (payload + 4) - *(int *) (payload + 0x10);
1402     }
1403     if (iVar11 == -1) {
1404         return;
1405     }
1406     uVar31 = (uint)*(byte *) (payload + 0xe);
1407     if (0x1f < uVar31) {
1408         return;
1409     }
1410     iVar29 = iVar11 * 0x1508;
1411     local_30 = *(undefined ***) (gSessionInfo + (uVar31 + 0xa8) * 4 + iVar29 + 4);
1412     if (local_30 == (undefined **)0x0) {
1413         if (0 < (int)cmd_id) {
1414             __Fill_ReadBuf(iVar11,iVar13,cmd_id & 0xffff,*(undefined2 *) (payload + 6),uV
1415                             (int)*(char *) (payload + 0xf));
1416         }
1417     }
1418     else {
1419         uVar12 = IOTC_Check_Session_Status(iVar11);
1420         (*(code *)local_30)(iVar11,uVar31,iVar13,cmd_id,uVar12);
1421         pcVar18 = gSessionInfo + (*(byte *) (payload + 0xe) + 0xe8) * 4 + iVar29 + 4;
1422     }
```

Out-of-bounds read inside `__Fill_ReadBuf()`. Memory is allocated using `malloc(param3)`, and `memcpy()` copies exactly `param3` bytes into the destination buffer. However, the source pointer (`param_2`) is attacker-influenced and may reference an invalid or out-of-bounds address derived from the malformed `0x407` packet, causing an out-of-bounds read that can lead to process crash or denial of service.


```
Decompile: __Fill_ReadBuf - (iCamera_v3_pro_symbols)
1
2 undefined4
3 __Fill_ReadBuf(int param_1, void *param_2, size_t param_3, undefined2 param_4, byte param_5,
4
5 )
6 {
7     undefined2 *__ptr;
8     void *__dest;
9     int *piVar1;
10    int iVar2;
11    int iVar3;
12    undefined4 uVar4;
13
14    pthread_mutex_lock((pthread_mutex_t *)gSessionLock);
15    iVar3 = gSessionInfo + param_1 * 0x1508;
16    uVar4 = 0xffffffff;
17    if (*(char *) (iVar3 + (uint)param_5 + 0x1f8) != '\x01') goto LAB_00412f94;
18    uVar4 = 0xffffffff;
19    __ptr = (undefined2 *)malloc(0x14);
20    if (__ptr == (undefined2 *)0x0) goto LAB_00412f94;
21    __dest = malloc(param_3);
22    *(void **)(__ptr + 4) = __dest;
23    if (__dest == (void *)0x0) {
24        free(__ptr);
25        goto LAB_00412f94;
26    }
27    memcpy(__dest, param_2, param_3);
28    iVar3 = iVar3 + (uint)param_5 * 4;
29    iVar2 = *(int *) (iVar3 + 0x178);
30    *(byte *) (__ptr + 6) = param_6 & 3;
31    piVar1 = *(int **) (iVar3 + 0x21c);
32    *__ptr = (short)param_3;
33    *(undefined4 *) (__ptr + 8) = 0;
34    *(int *) (__ptr + 2) = iVar2;
35    __ptr[1] = param_4;
36    *(int *) (iVar3 + 0x178) = iVar2 + 1;
```