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+
2.5GBase-T
GbE
WAN / LAN
LAN


DrayTek
Vigor2927 Series
DrayTek
Vigor2927 Series
DrayTek

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

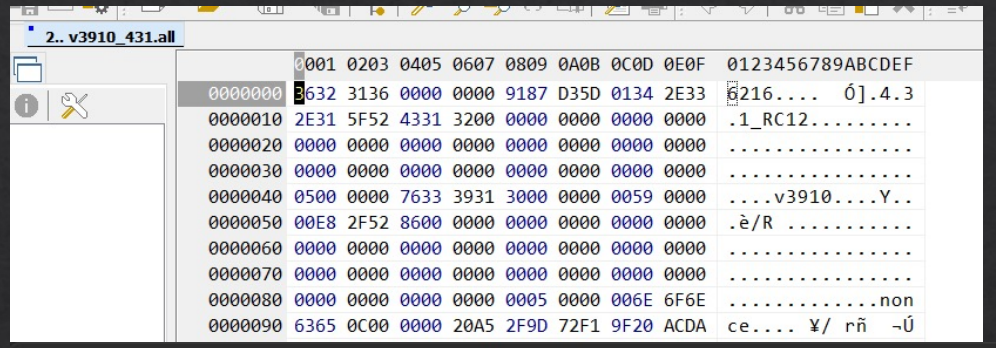
- Where are the Firmware?
  - Dumping flash, but we need a device….
- DrayTek website let you download firmware without registering
  - E.g. https://fw.draytek.com.tw/Vigor2862/Firmware/v3.9.8.1/Vigor2862_v3.9.8.1_STD_en.zip
  - Parent directory has older versions too!
    - E.g. https://fw.draytek.com.tw/Vigor2862/Firmware/

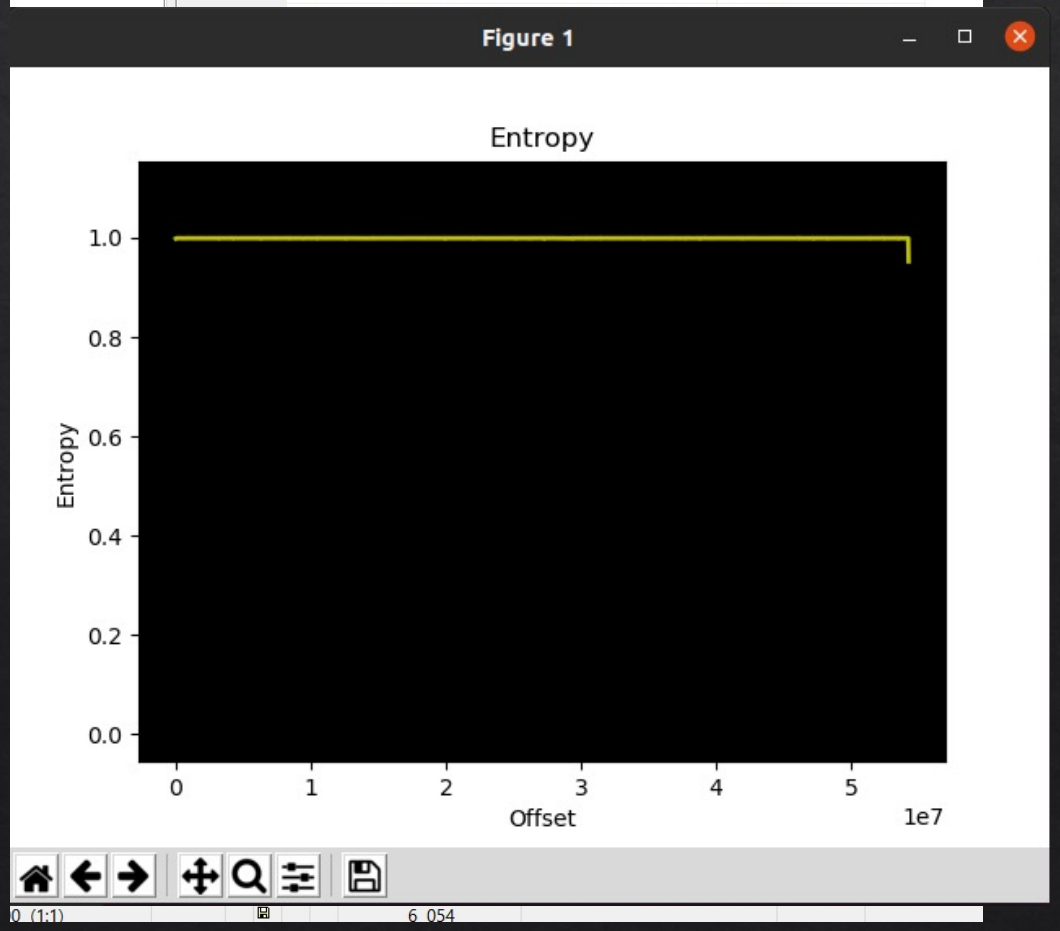
Index of /Vigor2862/Firmware

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)



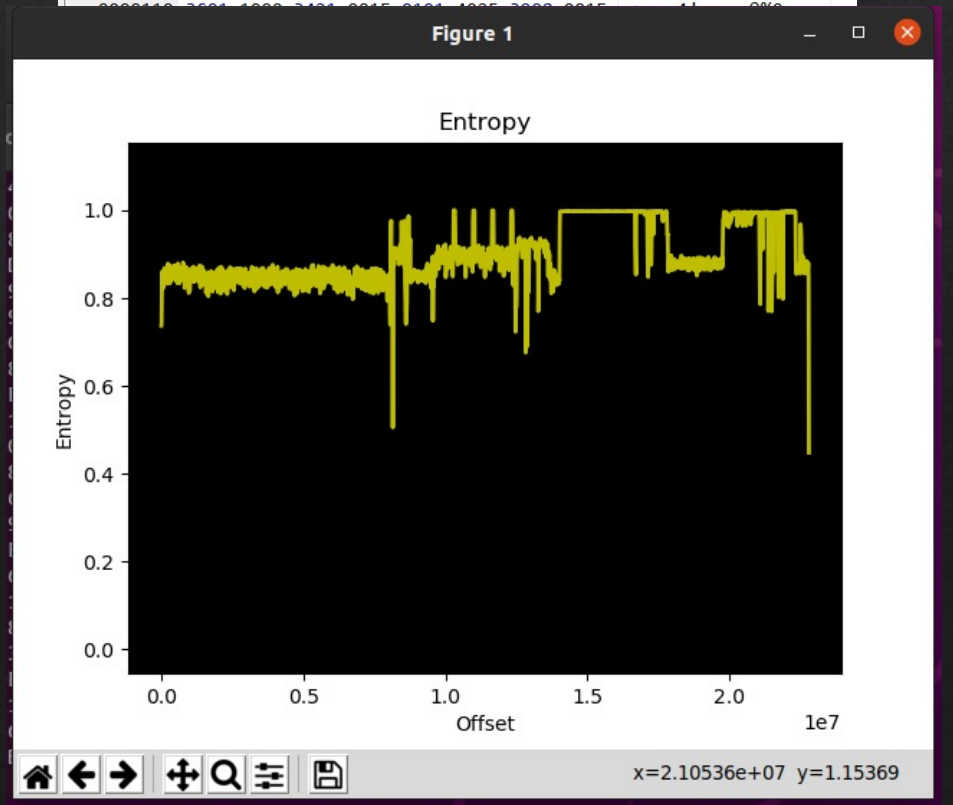


Getting started

- 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

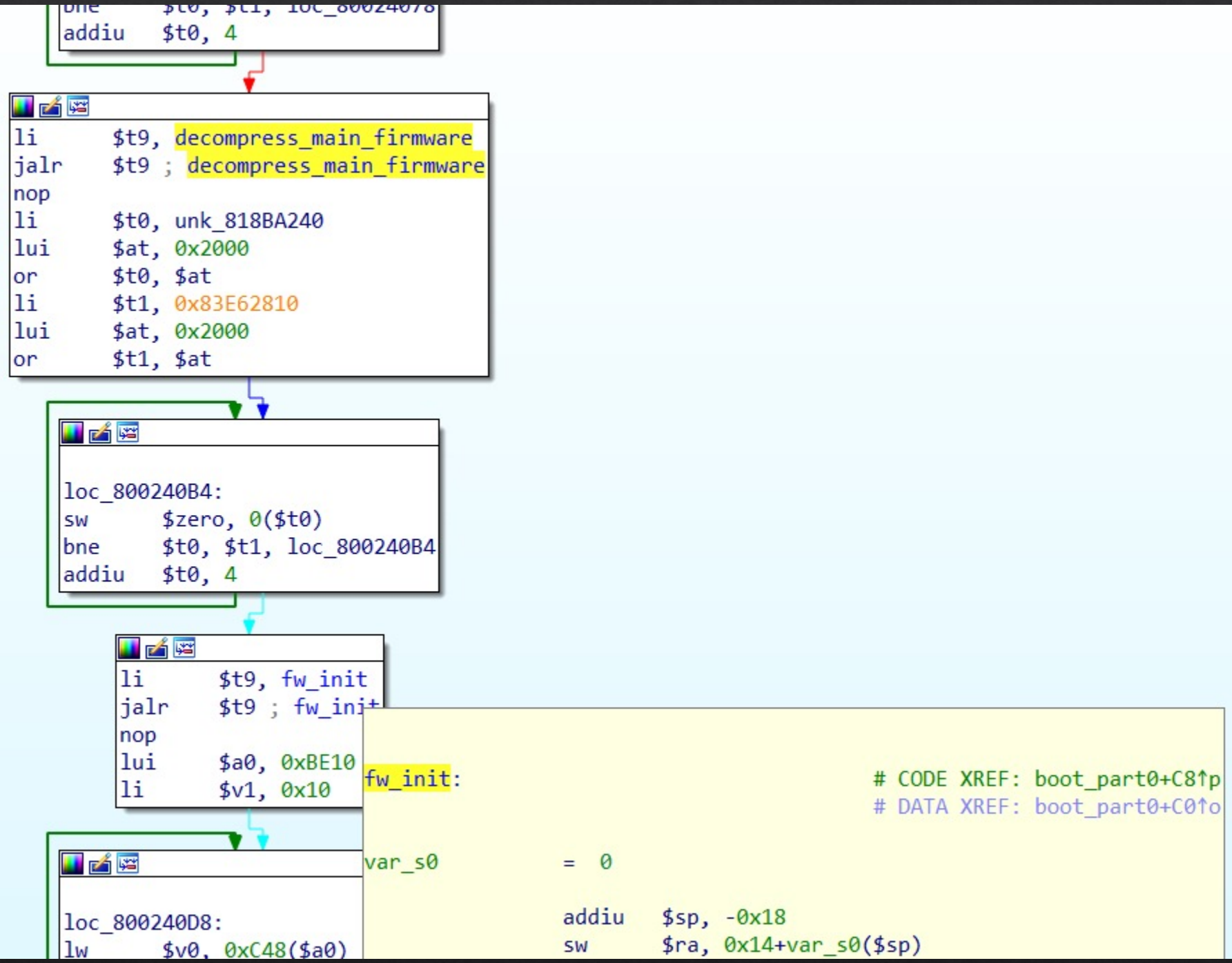
  ⇒ Decompression done in place happens before that

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

```
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