

# Emulating a MIPS binary with radare2

This writeup shows how to use ESIL in radare2 to emulate a mips binary and solve a ctf challenge.

## Analysis

First we'll open and analyze the binary. We can see that its architecture is MIPS, which will prevent us from debugging it as normal (on my machine at least).

```
Mate Terminal
[0x00400670]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[x] Finding xrefs in noncode section with anal.in=io.maps
[x] Analyze value pointers (aav)
[x] Value from 0x00400000 to 0x00400b34 (aav)
[x] 0x00400000-0x00400b34 in 0x400000-0x400b34 (aav)
[x] Emulate code to find computed references (aae)
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x00400670]> i~arch
arch      mips
[0x00400670]> █
```

We can print a list of the functions, of which there are few, which will make solving this challenge much easier. Unsurprisingly, the key instructions can be found in the main function.

```
Mate Terminal
[0x00400670]> aflt
```

addr	size	name	nbbs	xref	calls	cc
0x00400670	96	entry0	2	3	2	0
0x00400a90	88	sym._fini	1	3	2	1
0x00400764	232	sym.__do_global_dtors_aux	8	7	1	4
0x004008c0	288	main	4	3	4	2
0x004005ec	124	sym._init	1	4	4	1
0x0040084c	428	sym.frame_dummy	10	10	0	7
0x004009e0	76	sym.__do_global_ctors_aux	4	5	1	2
0x004006d0	68	sym.deregister_tm_clones	4	5	0	4
0x00400a70	32	sym.imp.puts	1	3	0	1
0x00400a60	16	sym.imp.printf	1	3	0	1
0x00400a50	16	sym.imp.atoi	1	2	0	1
0x00400a40	16	sym.imp.__uclibc_main	1	2	0	1
0x00400a30	16	sym.imp.scanf	1	2	0	1

Towards the bottom of the main function there is a branch with what looks like a pass/fail condition, based on the user input earlier in the function. The condition tests for the equality of v1 and v0, so we'll look at how these are calculated.

```

Mate Terminal
[0x004008c0] 0x4008c0 # int main (int32_t arg_10h, int32_t arg_18h, int32_t arg_1ch, int
  move t9, v0
  jalr t9;[?]
  nop
  lw gp, (arg_10h)
  move v1, v0
  lw v0, (arg_18h)
  nop
  bne v1, v0, 0x4009a4
  nop

```

```

0x400974 [ob]
addiu v0, fp, 0x1c
; '@'
lui v1, 0x40
; argc ; str.FLAG_ak3j3ka_s
; 0x400b04
; "FLAG-ak3j3ka%s\n"
addiu a0, v1, 0xb04
move a1, v0

```

```

0x4009a4 [oc]
; CODE XREF from main @ 0x40096c
; '@'
lui v0, 0x40
; argc ; str.Wrong_password
; 0x400b14
; "Wrong password"
addiu a0, v0, 0xb14
; [0x410b99:4]=0x400a70 sym imp puts

```

It looks like v0 is calculated with a simple bit of arithmetic, and is later compared to v1. We could do this by hand without too much trouble, but I'm lazy, so we'll try another way.

```

Mate Terminal
[0x0040090c [xAdvC]0 0% 130 introtomips]> pd $r @ main+76 # 0x40090c
0x0040090c 4c80828f lw v0, -sym.imp.scanf(gp) ; [0x410bac:4]=0x4
0x00400910 00000000 nop
0x00400914 21c84000 move t9, v0
0x00400918 09f82003 jalr t9 ;[?]
0x0040091c 00000000 nop
0x00400920 1000dc8f lw gp, (arg_10h)
0x00400924 39050224 addiu v0, zero, 0x539
0x00400928 1800c2af sw v0, (arg_18h)
0x0040092c 1800c38f lw v1, (arg_18h)
0x00400930 2c00023c lui v0, 0x2c ; ', '
0x00400934 e01d4234 ori v0, v0, 0x1de0
0x00400938 21106200 addu v0, v1, v0
0x0040093c 1800c2af sw v0, (arg_18h)
0x00400940 1c00c227 addiu v0, fp, 0x1c
0x00400944 21204000 move a0, v0
0x00400948 3880828f lw v0, -sym.imp.atoi(gp) ; [0x410b98:4]=0x40
0x0040094c 00000000 nop
0x00400950 21c84000 move t9, v0
0x00400954 09f82003 jalr t9 ;[?]
0x00400958 00000000 nop
0x0040095c 1000dc8f lw gp, (arg_10h)
0x00400960 21184000 move v1, v0
0x00400964 1800c28f lw v0, (arg_18h)
0x00400968 00000000 nop

```

Using radare2's ESIL, we can emulate the instructions despite not being able to debug the binary like normal. First we'll seek to the main function and initialize the emulator using the three commands below. The first one starts the emulator, the second initializes the virtual memory, and the third initializes the program counter to the current seek address.

```
Mate Terminal
[0x0040090c]> aei
[0x0040090c]> aeim
[0x0040090c]> aeip
[0x0040090c]> █
```

We can then step through the binary like normal using F7 or the aes/aeso commands. We'll want to skip over library calls using the aess command since the libraries aren't actually currently linked.

```
Mate Terminal
[0x004008c8 [xAdvC]0 0% 132 introtomips]> pd $r @ main+8 # 0x4008c8
0x004008c8 2004beaf sw fp, (var_8h)
0x004008cc 21f0a003 move fp, sp
0x004008d0 42001c3c lui gp, 0x42 ; 'B'
0x004008d4 608b9c27 addiu gp, gp, -0x74a0
0x004008d8 1000bcaf sw gp, (var_418h)
0x004008dc 4000023c lui v0, 0x40 ; '@'
0x004008e0 f00a4424 addiu a0, v0, 0xaf0 ; 0x400af0 ; "Passw
0x004008e4 3480828f lw v0, -sym.imp.printf(gp) ; [0x410b94:4]=0x
0x004008e8 00000000 nop
0x004008ec 21c84000 move t9, v0
;-- pc:
0x004008f0 09f82003 jalr t9 ;[?]
0x004008f4 00000000 nop
0x004008f8 1000dc8f lw gp, (arg_10h)
0x004008fc 1c00c227 addiu v0, fp, 0x1c
0x00400900 4000033c lui v1, 0x40 ; '@'
0x00400904 fc0a6424 addiu a0, v1, 0xafc ; 0x400afc ; "%1023
0x00400908 21284000 move a1, v0
0x0040090c 4c80828f lw v0, -sym.imp.scanf(gp) ; [0x410bac:4]=0x4
0x00400910 00000000 nop
0x00400914 21c84000 move t9, v0
0x00400918 09f82003 jalr t9 ;[?]
0x0040091c 00000000 nop
0x00400920 1000dc8f lw gp, (arg_10h)
:> aess █
```

Once we get to the branch, examining the registers and stack shows that the v0 register is equal to 0x2c2319, which is equivalent to 2892569 in base 10.

```

Mate Terminal
0x00177bd8 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x00177be8 608b 4100 0000 0000 1923 2c00 0000 0000 ` .A.....# , .....
0x00177bf8 0000 0000 0000 0000 0000 0000 0000 0000 .....
0x00177c08 0000 0000 0000 0000 0000 0000 0000 0000 .....
zero 0x00000000          at 0x00000000          v0 0x002c2319
v1 0x00400a50          a0 0x00177bf4          a1 0x00177bf4
a2 0x00000000          a3 0x00000000          t0 0x00000000
t1 0x00000000          t2 0x00000000          t3 0x00000000
t4 0x00000000          t5 0x00000000          t6 0x00000000
t7 0x00000000          s0 0x00000000          s1 0x00000000
s2 0x00000000          s3 0x00000000          s4 0x00000000
s5 0x00000000          s6 0x00000000          s7 0x00000000
t8 0x00000000          t9 0x00400a50          k0 0x00000000
k1 0x00000000          gp 0x00418b60          sp 0x00177bd8
fp 0x00177bd8          ra 0x00000000          pc 0x0040096c
hi 0x00000000          lo 0x00000000          t 0x00000000

;-- pc:
< 0x0040096c          0d006214          bne v1, v0, 0x4009a4
0x00400970          00000000          nop
0x00400974          1c00c227          addiu v0, fp, 0x1c
0x00400978          4000033c          lui v1, 0x40
0x0040097c          040b6424          addiu a0, v1, 0xb04
; '@'
; 0x400b04 ; "FLAG-

:> ? 0x2c2319~int
int32 2892569
uint32 2892569
:> █

```

If we combine this with the rest of the flag in the success branch, we get the solution:

**FLAG-ak3j3ka2892569**