SHIKATA GA NAI

Radare2 workshop

October 22, 2015

hack.lu 2015

WHOAMI

Julien (jvoisin) Voisin

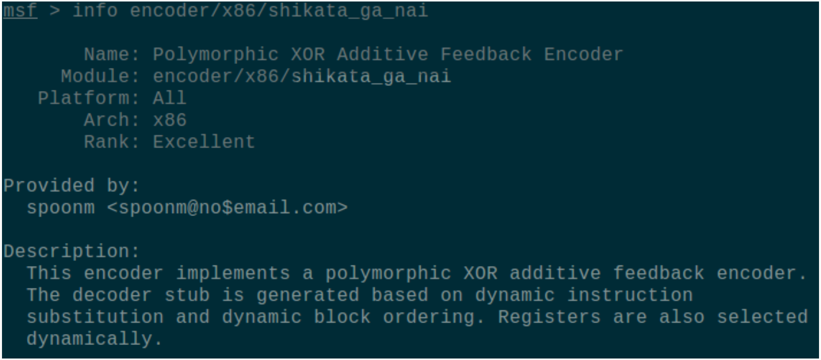
- French
- Working at FIXME
- I don’t know Ruby

DISCLAIMER

This workshop is based on ideas and scripts from Jaime (@NighetMan) Peñalba.

WHAT ARE WE GOING TO DO?

Unpack Shikata ga nai!



```
msf > info encoder/x86/shikata_ga_nai

       Name: Polymorphic XOR Additive Feedback Encoder
     Module: encoder/x86/shikata_ga_nai
   Platform: All
       Arch: x86
       Rank: Excellent

Provided by:
  spoonm <spoonm@no$email.com>

Description:
  This encoder implements a polymorphic XOR additive feedback encoder.
  The decoder stub is generated based on dynamic instruction
  substitution and dynamic block ordering. Registers are also selected
  dynamically.
```

SHIKATA GA NAI

- Polymorphic
- 320 lines of msf-powered OOP Ruby
- We want the unpacked shellcode

HOW DO WE DO IT?

SOLUTIONS

- Run it on your machine and see what happens

SOLUTIONS

- Run it on your machine and see what happens
- Step-step-step-step-step-… in gdb

SOLUTIONS

- Run it on your machine and see what happens
- Step-step-step-step-step-… in gdb
- Trace the execution in a virtual machine

SOLUTIONS

- Run it on your machine and see what happens
- Step-step-step-step-step-… in gdb
- Trace the execution in a virtual machine
- Use radare2 with ESIL!

BUT WHAT IS ESIL?

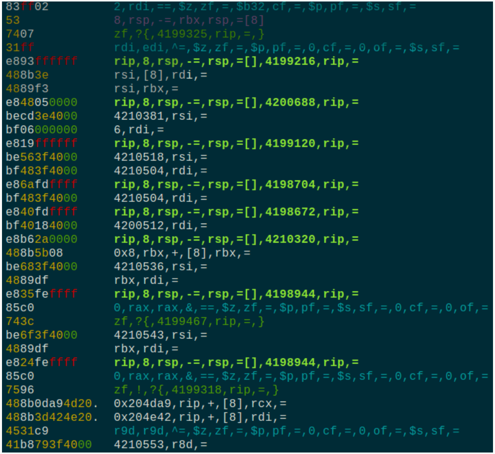
ESIL

- Evaluable String Intermediary Language
- Yet another intermediary language
- RPN-ish
- $jz\ 0xaabbccdd : zf, ?, 0xaabbccdd, eip, =,$

WHAT CAN WE DO WITH THIS *ESIL*?

ESIL

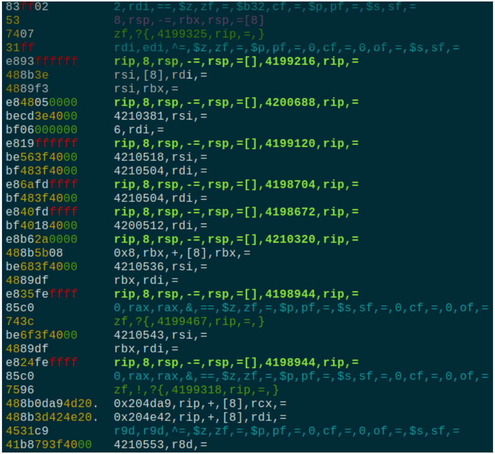
- Used for
  - Emulation



```
83ff02          2,rdi,==,$z,zf,=,$b32,cf,=,$p,pf,=,$s,sf,=
53              8,rsp,-=,rbx,rsp,=[8]
7407            zf,?{,4199325,rip,=,}
31ff            rdi,edi,^=,$z,zf,=,$p,pf,=,0,cf,=,0,of,=,$s,sf,=
e893ffffff      rip,8,rsp,-=,rsp,=[],4199216,rip,=
488b3e          rsi,[8],rdi,=
4889f3          rsi,rbx,=
e848050000      rip,8,rsp,-=,rsp,=[],4200688,rip,=
becd3e4000      4210381,rsi,=
bf06000000      6,rdi,=
e819ffffff      rip,8,rsp,-=,rsp,=[],4199120,rip,=
be563f4000      4210518,rsi,=
bf483f4000      4210504,rdi,=
e86afdffff      rip,8,rsp,-=,rsp,=[],4198704,rip,=
bf483f4000      4210504,rdi,=
e840fdffff      rip,8,rsp,-=,rsp,=[],4198672,rip,=
bf40184000      4200512,rdi,=
e8b62a0000      rip,8,rsp,-=,rsp,=[],4210320,rip,=
488b5b08        0x8,rbx,+,[8],rbx,=
be683f4000      4210536,rsi,=
4889df          rbx,rdi,=
e835feffff      rip,8,rsp,-=,rsp,=[],4198944,rip,=
85c0            0,rax,rax,&,==,$z,zf,=,$p,pf,=,$s,sf,=,0,cf,=,0,of,=
743c            zf,?{,4199467,rip,=,}
be6f3f4000      4210543,rsi,=
4889df          rbx,rdi,=
e824feffff      rip,8,rsp,-=,rsp,=[],4198944,rip,=
85c0            0,rax,rax,&,==,$z,zf,=,$p,pf,=,$s,sf,=,0,cf,=,0,of,=
7596            zf,!,?{,4199318,rip,=,}
488b0da94d20.   0x204da9,rip,+,[8],rcx,=
488b3d424e20.   0x204e42,rip,+,[8],rdi,=
4531c9          r9d,r9d,^=,$z,zf,=,$p,pf,=,0,cf,=,0,of,=,$s,sf,=
41b8793f4000    4210553,r8d,=
```

ESIL

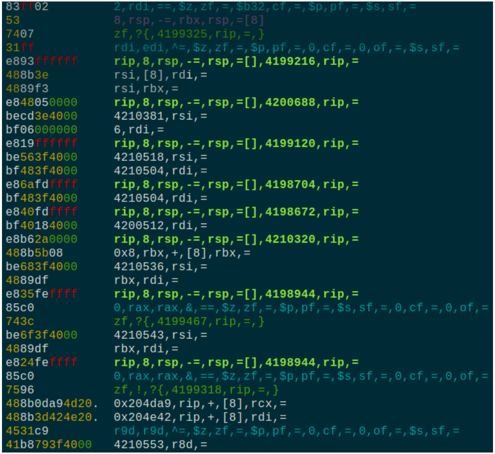
- Used for
    - Emulation
    - Decompilation



```
83ff02         2,rdi,==,$z,zf,=,$b32,cf,=,$p,pf,=,$s,sf,=
53             8,rsp,-=,rbx,rsp,=[8]
7407           zf,?{,4199325,rip,=,}
31ff           rdi,edi,^=,$z,zf,=,$p,pf,=,0,cf,=,0,of,=,$s,sf,=
e893ffffff     rip,8,rsp,-=,rsp,=[],4199216,rip,=
488b3e         rsi,[8],rdi,=
4889f3         rsi,rbx,=
e848050000     rip,8,rsp,-=,rsp,=[],4200688,rip,=
becd3e4000     4210381,rsi,=
bf06000000     6,rdi,=
e819ffffff     rip,8,rsp,-=,rsp,=[],4199120,rip,=
be563f4000     4210518,rsi,=
bf483f4000     4210504,rdi,=
e86afdffff     rip,8,rsp,-=,rsp,=[],4198704,rip,=
bf483f4000     4210504,rdi,=
e840fdffff     rip,8,rsp,-=,rsp,=[],4198672,rip,=
bf40184000     4200512,rdi,=
e8b62a0000     rip,8,rsp,-=,rsp,=[],4210320,rip,=
488b5b08       0x8,rbx,+,[8],rbx,=
be683f4000     4210536,rsi,=
4889df         rbx,rdi,=
e835feffff     rip,8,rsp,-=,rsp,=[],4198944,rip,=
85c0           0,rax,rax,&,==,$z,zf,=,$p,pf,=,$s,sf,=,0,cf,=,0,of,=
743c           zf,?{,4199467,rip,=,}
be6f3f4000     4210543,rsi,=
4889df         rbx,rdi,=
e824feffff     rip,8,rsp,-=,rsp,=[],4198944,rip,=
85c0           0,rax,rax,&,==,$z,zf,=,$p,pf,=,$s,sf,=,0,cf,=,0,of,=
7596           zf,!,?{,4199318,rip,=,}
488b0da94d20.  0x204da9,rip,+,[8],rcx,=
488b3d424e20.  0x204e42,rip,+,[8],rdi,=
4531c9         r9d,r9d,^=,$z,zf,=,$p,pf,=,0,cf,=,0,of,=,$s,sf,=
41b8793f4000   4210553,r8d,=
```

ESIL

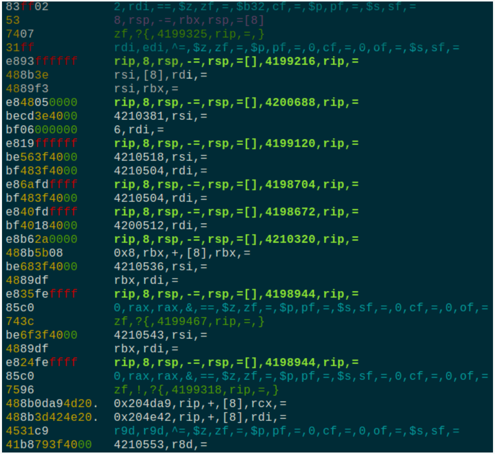
- Used for
  - Emulation
  - Decompilation
  - Analysis



```
83ff02          2,rdi,==,$z,zf,=,$b32,cf,=,$p,pf,=,$s,sf,=
53              8,rsp,-=,rbx,rsp,=[8]
7407            zf,?{,4199325,rip,=,}
31ff            rdi,edi,^=,$z,zf,=,$p,pf,=,0,cf,=,0,of,=,$s,sf,=
e893ffffff      rip,8,rsp,-=,rsp,=[],4199216,rip,=
488b3e          rsi,[8],rdi,=
4889f3          rsi,rbx,=
e848050000      rip,8,rsp,-=,rsp,=[],4200688,rip,=
becd3e4000      4210381,rsi,=
bf06000000      6,rdi,=
e819ffffff      rip,8,rsp,-=,rsp,=[],4199120,rip,=
be563f4000      4210518,rsi,=
bf483f4000      4210504,rdi,=
e86afdffff      rip,8,rsp,-=,rsp,=[],4198704,rip,=
bf483f4000      4210504,rdi,=
e840fdffff      rip,8,rsp,-=,rsp,=[],4198672,rip,=
bf40184000      4200512,rdi,=
e8b62a0000      rip,8,rsp,-=,rsp,=[],4210320,rip,=
488b5b08        0x8,rbx,+,[8],rbx,=
be683f4000      4210536,rsi,=
4889df          rbx,rdi,=
e835feffff      rip,8,rsp,-=,rsp,=[],4198944,rip,=
85c0            0,rax,rax,&,==,$z,zf,=,$p,pf,=,$s,sf,=,0,cf,=,0,of,=
743c            zf,?{,4199467,rip,=,}
be6f3f4000      4210543,rsi,=
4889df          rbx,rdi,=
e824feffff      rip,8,rsp,-=,rsp,=[],4198944,rip,=
85c0            0,rax,rax,&,==,$z,zf,=,$p,pf,=,$s,sf,=,0,cf,=,0,of,=
7596            zf,!,?{,4199318,rip,=,}
488b0da94d20.   0x204da9,rip,+,[8],rcx,=
488b3d424e20.   0x204e42,rip,+,[8],rdi,=
4531c9          r9d,r9d,^=,$z,zf,=,$p,pf,=,0,cf,=,0,of,=,$s,sf,=
41b8793f4000    4210553,r8d,=
```

ESIL

- Used for
  - Emulation
  - Decompilation
  - Analysis
  - Flamewars against other IL



```
83ff02         2,rdi,==,$z,zf,=,$b32,cf,=,$p,pf,=,$s,sf,=
53             8,rsp,-=,rbx,rsp,=[8]
7407           zf,?{,4199325,rip,=,}
31ff           rdi,edi,^=,$z,zf,=,$p,pf,=,0,cf,=,0,of,=,$s,sf,=
e893ffffff     rip,8,rsp,-=,rsp,=[],4199216,rip,=
488b3e         rsi,[8],rdi,=
4889f3         rsi,rbx,=
e848050000     rip,8,rsp,-=,rsp,=[],4200688,rip,=
becd3e4000     4210381,rsi,=
bf06000000     6,rdi,=
e819ffffff     rip,8,rsp,-=,rsp,=[],4199120,rip,=
be563f4000     4210518,rsi,=
bf483f4000     4210504,rdi,=
e86afdffff     rip,8,rsp,-=,rsp,=[],4198704,rip,=
bf483f4000     4210504,rdi,=
e840fdffff     rip,8,rsp,-=,rsp,=[],4198672,rip,=
bf40184000     4200512,rdi,=
e8b62a0000     rip,8,rsp,-=,rsp,=[],4210320,rip,=
488b5b08       0x8,rbx,+,[8],rbx,=
be683f4000     4210536,rsi,=
4889df         rbx,rdi,=
e835feffff     rip,8,rsp,-=,rsp,=[],4198944,rip,=
85c0           0,rax,rax,&,==,$z,zf,=,$p,pf,=,$s,sf,=,0,cf,=,0,of,=
743c           zf,?{,4199467,rip,=,}
be6f3f4000     4210543,rsi,=
4889df         rbx,rdi,=
e824feffff     rip,8,rsp,-=,rsp,=[],4198944,rip,=
85c0           0,rax,rax,&,==,$z,zf,=,$p,pf,=,$s,sf,=,0,cf,=,0,of,=
7596           zf,!,?{,4199318,rip,=,}
488b0da94d20.  0x204da9,rip,+,[8],rcx,=
488b3d424e20.  0x204e42,rip,+,[8],rdi,=
4531c9         r9d,r9d,^=,$z,zf,=,$p,pf,=,0,cf,=,0,of,=,$s,sf,=
41b8793f4000   4210553,r8d,=
```

HOW DOES EMULATION HELP US TO DUMP THE SHELLCODE?

WHERE TO STOP?

We can emulate the shellcode, but where do we stop?

- Instructions aren’t fixed.
- Blocks are permutated.
- Registers are dynamically selected.

So what can we do?

READING THE SOURCE CODE

It seems that the last instruction will always be loop.

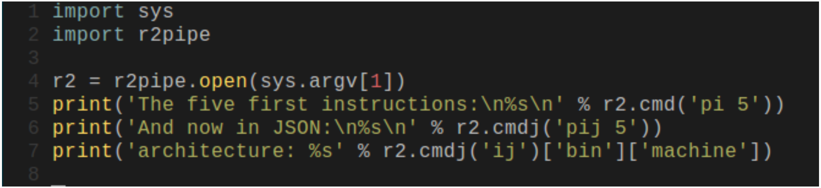
READING THE SOURCE CODE

It seems that the last instruction will always be loop.

So we can emulate the shellcode, and dump the result from the last loop instruction till then end.

HOW DO WE USE RADARE2/ESIL ANYWAY?

R2PIPE



```python
import sys
import r2pipe

r2 = r2pipe.open(sys.argv[1])
print('The five first instructions:\n%s\n' % r2.cmd('pi 5'))
print('And now in JSON:\n%s\n' % r2.cmdj('pij 5'))
print('architecture: %s' % r2.cmdj('ij')['bin']['machine'])

```

LANGUAGES

NodeJS

```
npm install r2pipe
```

Python

```
pip install r2pipe
```

Ruby

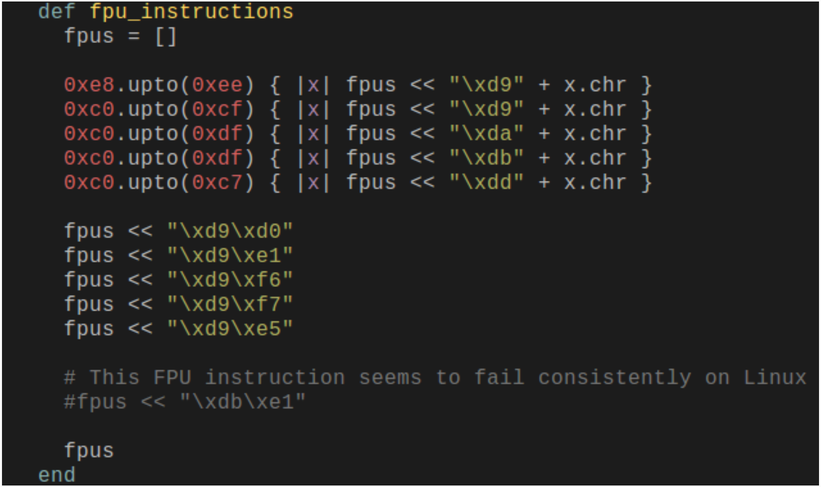
```
gem install r2pipe
```

so let's use esil?

PLOT TWIST

- FPU is currently not supported in ESIL :D
- FPU is used to get EIP with FNSTENV
- Polymorphic FPU instructions

PLOT TWIST



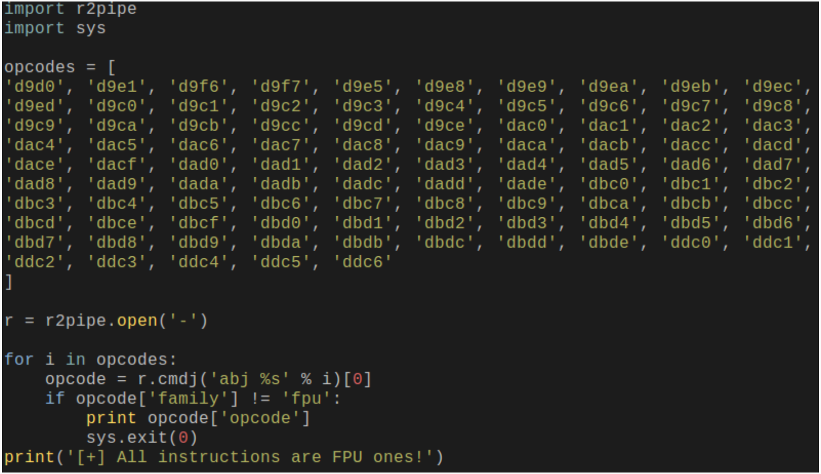
```
def fpu_instructions
  fpus = []

  0xe8.upto(0xee) { |x| fpus << "\xd9" + x.chr }
  0xc0.upto(0xcf) { |x| fpus << "\xd9" + x.chr }
  0xc0.upto(0xdf) { |x| fpus << "\xda" + x.chr }
  0xc0.upto(0xdf) { |x| fpus << "\xdb" + x.chr }
  0xc0.upto(0xc7) { |x| fpus << "\xdd" + x.chr }

  fpus << "\xd9\xd0"
  fpus << "\xd9\xe1"
  fpus << "\xd9\xf6"
  fpus << "\xd9\xf7"
  fpus << "\xd9\xe5"

  # This FPU instruction seems to fail consistently on Linux
  #fpus << "\xdb\xe1"

  fpus
end
```

CAN WE EMULATE THEM THE *GHETTO WAY*?

ARE THOSE DETECTED AS FPU BY R2?

- You’ve got the hello_world.py code
- Check if every opcode in the test_fpu.py one has the fpu family
- Feel free to do it in your favourite language!

MY SOLUTION



```
import r2pipe
import sys

opcodes = [
'd9d0', 'd9e1', 'd9f6', 'd9f7', 'd9e5', 'd9e8', 'd9e9', 'd9ea', 'd9eb', 'd9ec',
'd9ed', 'd9c0', 'd9c1', 'd9c2', 'd9c3', 'd9c4', 'd9c5', 'd9c6', 'd9c7', 'd9c8',
'd9c9', 'd9ca', 'd9cb', 'd9cc', 'd9cd', 'd9ce', 'dac0', 'dac1', 'dac2', 'dac3',
'dac4', 'dac5', 'dac6', 'dac7', 'dac8', 'dac9', 'daca', 'dacb', 'dacc', 'dacd',
'dace', 'dacf', 'dad0', 'dad1', 'dad2', 'dad3', 'dad4', 'dad5', 'dad6', 'dad7',
'dad8', 'dad9', 'dada', 'dadb', 'dadc', 'dadd', 'dade', 'dbc0', 'dbc1', 'dbc2',
'dbc3', 'dbc4', 'dbc5', 'dbc6', 'dbc7', 'dbc8', 'dbc9', 'dbca', 'dbcb', 'dbcc',
'dbcd', 'dbce', 'dbcf', 'dbd0', 'dbd1', 'dbd2', 'dbd3', 'dbd4', 'dbd5', 'dbd6',
'dbd7', 'dbd8', 'dbd9', 'dbda', 'dbdb', 'dbdc', 'dbdd', 'dbde', 'ddc0', 'ddc1',
'ddc2', 'ddc3', 'ddc4', 'ddc5', 'ddc6'
]

r = r2pipe.open('-')

for i in opcodes:
    opcode = r.cmdj('abj %s' % i)[0]
    if opcode['family'] != 'fpu':
        print opcode['opcode']
        sys.exit(0)
print('[+] All instructions are FPU ones!')
```

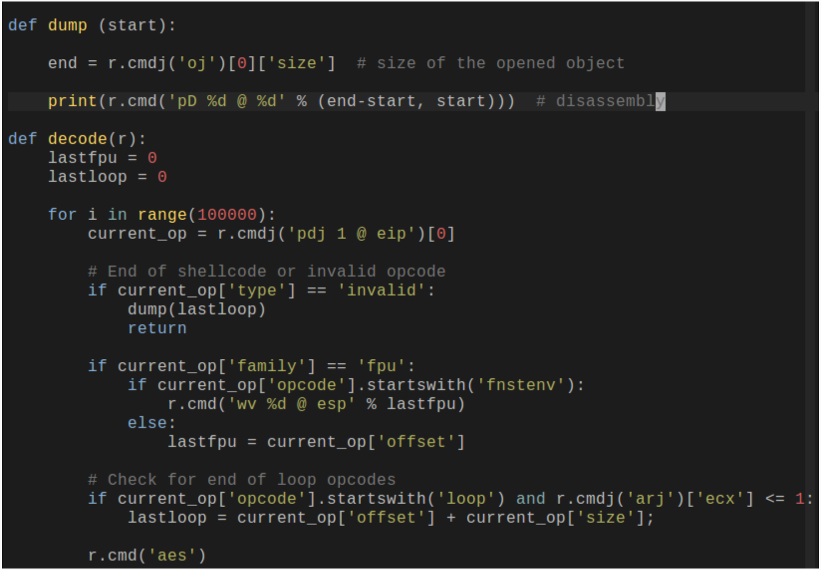
Ready to unpack Shikata Ga Nai?

SUM UP

1. Initialize the ESIL vm
2. If the instruction is invalid
    2.1 We're at the end!
    2.2 Dump from the last encountered loop instruction to the end
3. Else, if the instruction is an fpu one
    3.1 If it's fnstenv, write the previously stored eip at esp
    3.2 Else, store eip
4. Else, if the instruction is loop, store its location
5. Step and goto 2.

YOUR TURN!

MY SOLUTION



```python
def dump (start):

    end = r.cmdj('oj')[0]['size']  # size of the opened object

    print(r.cmd('pD %d @ %d' % (end-start, start)))  # disassembly

def decode(r):
    lastfpu = 0
    lastloop = 0

    for i in range(100000):
        current_op = r.cmdj('pdj 1 @ eip')[0]

        # End of shellcode or invalid opcode
        if current_op['type'] == 'invalid':
            dump(lastloop)
            return

        if current_op['family'] == 'fpu':
            if current_op['opcode'].startswith('fnstenv'):
                r.cmd('wv %d @ esp' % lastfpu)
            else:
                lastfpu = current_op['offset']

        # Check for end of loop opcodes
        if current_op['opcode'].startswith('loop') and r.cmdj('arj')['ecx'] <= 1:
            lastloop = current_op['offset'] + current_op['size'];

        r.cmd('aes')
```

CONCLUSION

CONCLUSION

- ESIL is cool
- Still WIP
- More to come!

CONCLUSION

Radare2 is nice.

You should use it.

RESOURCES

- Github repo
- Official website
- The r2 blog
- The r2 book
- Twitter