radare2#danacon

–pancake

Quien Soy?

Quien Soy?

Me han criticado un par de veces para que me ahorre está slide.

Así que os voy a dejar con la duda y pasaré a la siguiente slide

Seguramente me conoceréis por dar la brasa con la r2con

Otra charla de IA?

...

r2ai

Si atendiste a Navajas o a la r2con ya sabréis lo guay que es y que es capaz:

- Decompilar cualquier arquitectura soportada por R2
  - Hasta Swift, SwiftUI, Unity, STM8, …
- Propagación de tipos
- Renombrar functions
- Explicar funciones
- Buscar vulnerabilidades
- Utilizar radare2 sin intervención humana \o/
- Resolver crackmes automaticamente
- Tenemos un modelo propio con un dataset en desarrollo

Podría reutilizar alguna charla ya

Y utilizar funcionalidades que ya existen..

Pero siempre es más divertido empezar las slides 1h antes del stream

Mientras..

- Implementar las nuevas funcionalidades y subirlas al git
- Fuzzear el parser de DWARF y corregir los bugs
- Secar la ropa de la lavadora a mano con un radiador

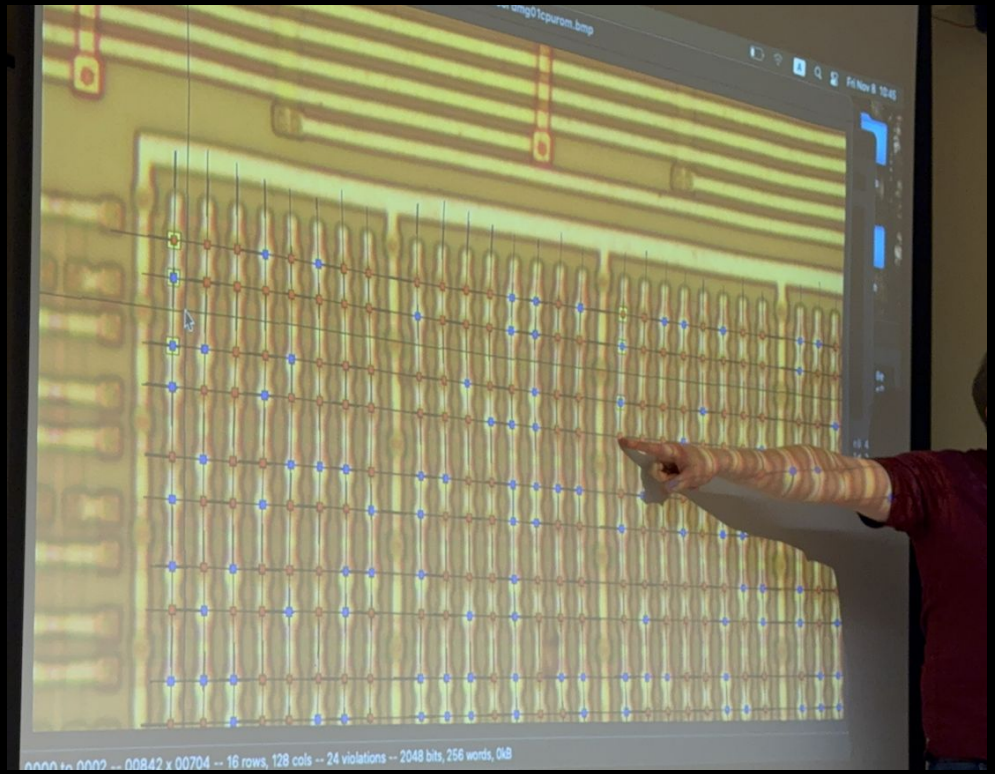
Instruction Decoding

Estamos Leyendo Código Válido?

Problemas no resueltos

Durante la charla de Travis Goodspeed aprendimos cómo recuperar bit a bit juegos de gameboy a partir de fotos con microscopio de las memorias ROM de los cartuchos.

https://rada.re/con



Problemas no Resueltos

Cada fabricante de memorias ROM crea sus propias configuraciones de cableados para definir los bits de cada byte.

No. No están ordenados de izquierda a derecha.

Las imágenes no siempre están alineadas, no se puede automatizar.

https://github.com/travisgoodspeed/maskromtool

Problemas no resueltos

Existen infinidad de combinaciones de secuencias de bits.

Cada volcado que hagamos tendremos que comprobarlo con varias configuraciones de arquitecturas (en este caso solo gameboy aka z80)..

Pero.. como sabemos que el código desensamblado es correcto?

Código No Válido

Una forma fácil sería sacar un desensamblado lineal

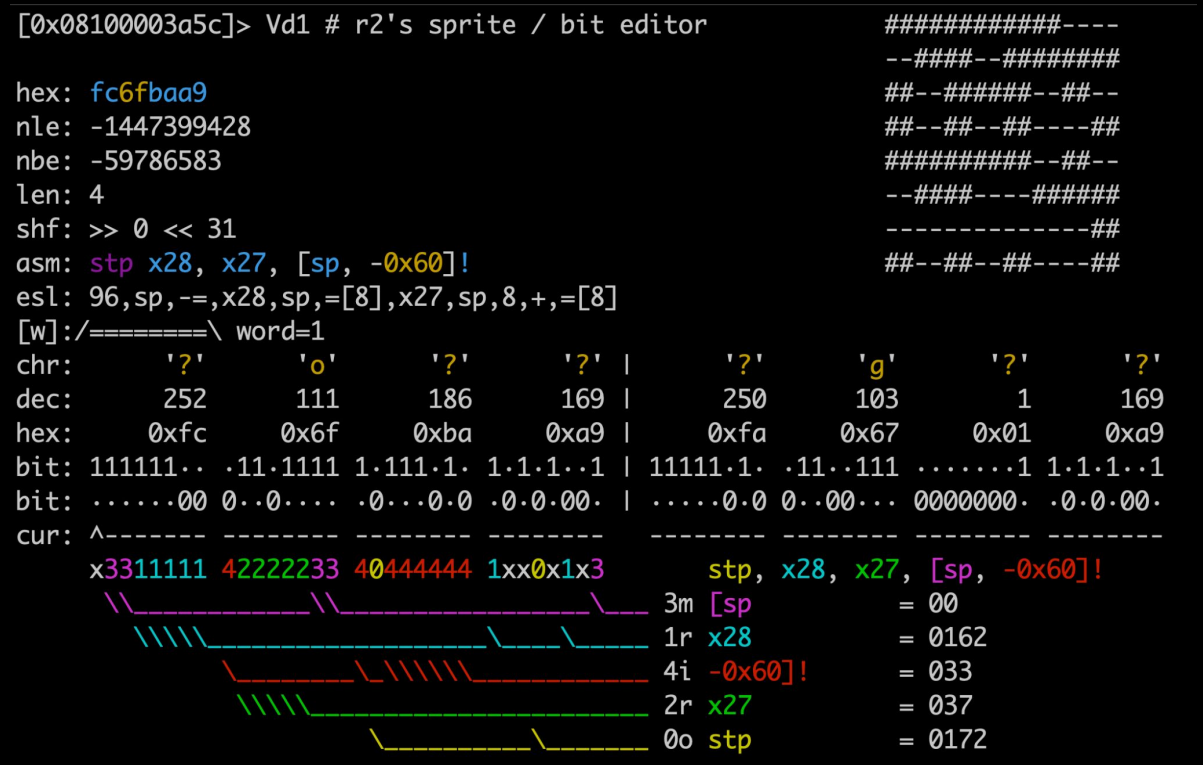
- Desde la dirección cero? El entrypoint?
- Buscar patrones de principios de función? (preludes)

A partir del output

- Contar instrucciones inválidas
- Hacer estadística de aparición por tipos de instrucciones
- Comprobar referencias a memoria no mapeada
- Secuencias de instrucciones con sentido

Decoding Instructions en ARM64

Decoding Instructions en ARM64



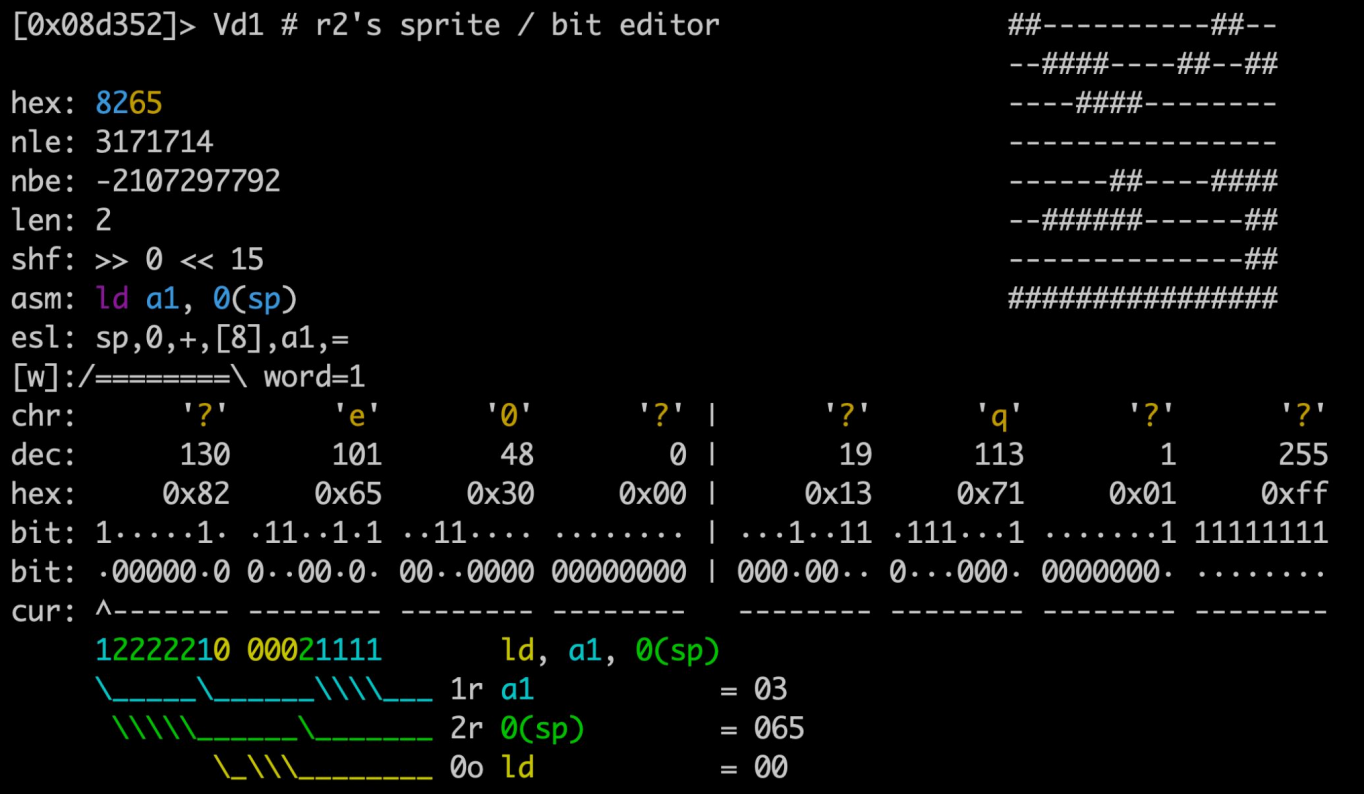
Decoding Instructions en X86

```
[0x085be3]> Vd1 # r2's sprite / bit editor
hex: 31d2
nle: -146222543
nbe: 835864823
len: 2
shf: >> 0 << 15
asm: xor edx, edx
esl: edx,rdx,^,0xffffffff,&,rdx,=,$z,zf,:=,$p,pf,:=,31,$s,sf,:=,0,cf,:=,0,of,:=
[w]:/========\ word=1
chr:      '1'      '?'      'H'      '?' |      '?'      'H'      '?'      '?'
dec:       49      210       72      247 |      246       72      137      208
hex:     0x31     0xd2     0x48     0xf7 |     0xf6     0x48     0x89     0xd0
bit: ..11...1 11.1..1. .1..1... 1111.111 | 1111.11. .1..1... 1...1..1 11.1....
bit: 00..000. ..0.00.0 0.00.000 ....0... | ....0..0 0.00.000 .000.00. ..0.0000
cur: ^------- -------- -------- --------   -------- -------- -------- --------
     00000101 11222111       xor, edx, edx
     \\\\\_\_____________ 0o xor          = 014
          \_\_\\___\\\___ 1r edx          = 072
                \\\_____ 2r edx          = 02
```

Decoding Instructions en X86

```
[0x085be0]> Vd1 # r2's sprite / bit editor
hex: 488b07
nle: 822577992
nbe: 1217070897
len: 3
shf: >> 0 << 23
asm: mov rax, qword [rdi]
esl: rdi,[8],rax,=
[w]:/========\ word=1
chr:      'H'      '?'      '?'      '1' |      '?'      'H'      '?'      '?'
dec:       72      139        7       49 |      210       72      247      246
hex:     0x48     0x8b     0x07     0x31 |     0xd2     0x48     0xf7     0xf6
bit: .1..1... 1...1.11 .....111 ..11...1 | 11.1..1. .1..1... 1111.111 1111.11.
bit: 0.00.000 .000.0.. 00000... 00..000. | ..0.00.0 0.00.000 ....0... ....0..0
cur: ^------- -------- -------- --------   -------- -------- -------- --------
     00001103 00000011 33111333       mov, rax, qword, [rdi]
     \\\\__\__\\\\\\_____________ 0o mov          = 01042
         \\_________\\___\\\_____ 1r rax          = 0130
           \__________\\___\\\___ 3m [rdi]        = 00
```

Decoding Instructions en RISCV



Decoding Instructions en RISCV

Estadística y Entropía

Vamos a sacar la calculadora (r2js)

Desensamblemos y busquemos direcciones inválidas

```
ARCH=x86 mips riscv sparc arm

BITS=8 16 32 64

CPU=...

$ r2 -a $ARCH -b $BITS -qc ‘pd~invalid?’ shellcode.bin 2> /dev/null
```

Pero.. podemos utilizar

```
ARCH=x86 mips riscv sparc arm

BITS=8 16 32 64

CPU=...

$ r2 -a $ARCH -b $BITS -qc ‘pd~invalid?’ shellcode.bin 2>
/dev/null
```

Pero.. podemos utilizar métricas más detalladas?

```
> aobj~{}
{"opstr":"ld a1, 0(sp)","size":2,
"bytes":[[1,2,2,2,2,2,1,0],[0,0,0,2,1,1,1,1]],
"flipstr":"12222210 00021111 ",
"Args":[[0,15,14,13],[1,7,8,9,10,11],[2,3,4,5,6,12]],
"Vals":[0,53,3]}
```

Y a partir de aqui ya podemos empejar a jugar con r2js! \o/

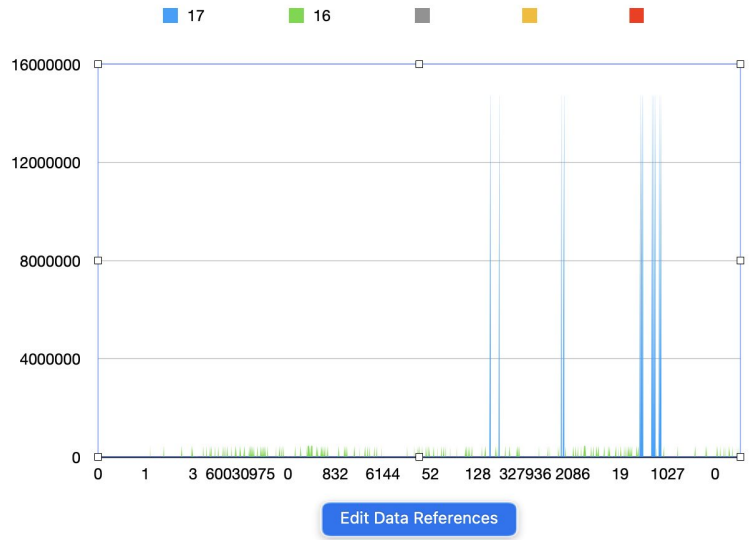
r2js demo

```
r2.call("aa");
const funcs = r2.cmd("aflq").split(/\n/g);
const valid = [];

const verbose = false;
function log(x) {
        if (verbose) {
                console.log(x);
        }
}
for (let func of funcs) {
        const bbs = r2.cmdj("afbj@" + func);
        log(func);
        for (let bb of bbs) {
                log("bb");
                const ops = r2.cmdj("pdbj@"+bb.addr);
                for (let op of ops) {
                        log ("op : " + op.disasm);
                        const bits = r2.cmd("aob @ " + op.offset)
                        const jbits = r2.cmdj("aobj @ " + op.offset)
                        log(bits);
                        valid.push(jbits.vals);
                }
        }
}

console.log(valid.join("\n"))
```

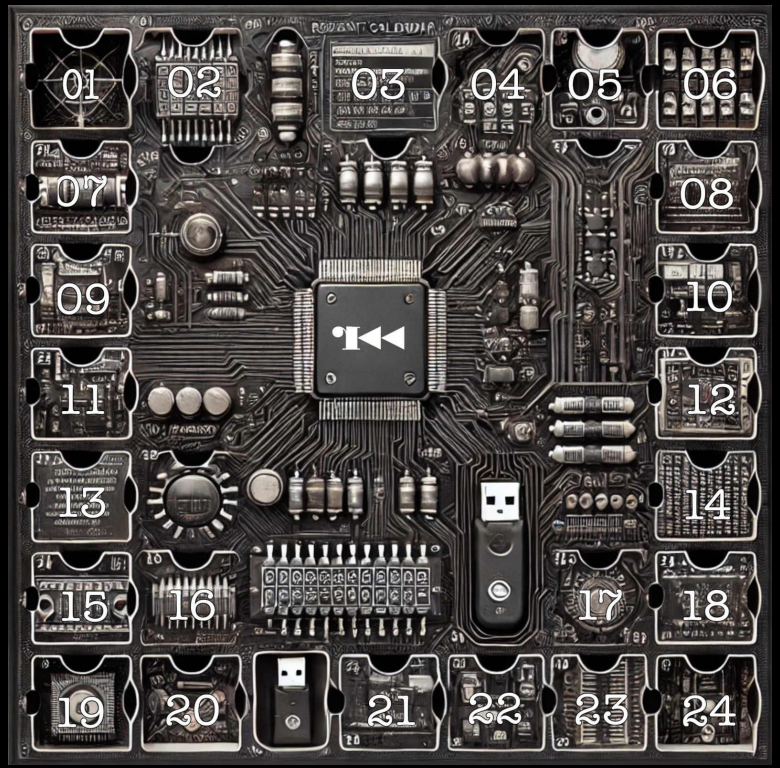
Graphing valid arm64 code

a

1	17	16			
0	6	35	1		
16512	66	31			
17	2	3			
1	17	16			
1024	6	35	1		
16512	66	31			
17	2	3			
1	17	16			
2048	6	35	1		
16512	66	31			
17	2	3			
1	17	16			
3072	6	35	1		
16512	66	31			
17	2	3			
1	17	16			
4096	6	35	1		
16512	66	31			
17	2	3			
1	17	16			
5120	6	35	1		
16512	66	31			



<SPAM> Advent Of Radare2 </SPAM>



Preguntas? ^^