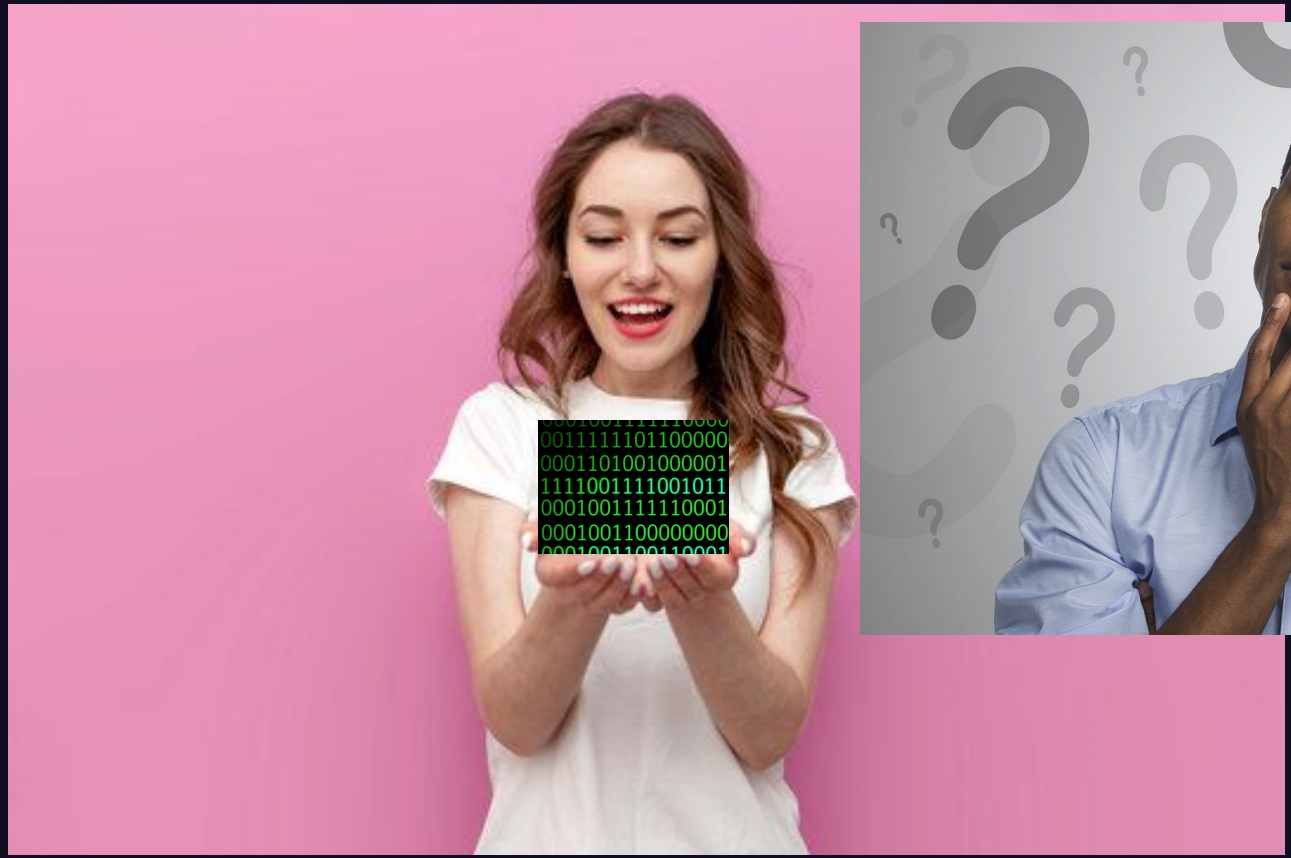
ENCONTRANDO TTPs DE MALWARE CON EJECUCIÓN SIMBÓLICA

Lucía Martinez Gavier


0011111101100000
0001101001000001
1111001111001011
0001001111110001
0001001100000000


0011111101100000
0001101001000001
1111001111001011
0001001111110001
0001001100000000

LUCÍA MARTINEZ GAVIER

- Estudiante de la Lic. Cs de la Computación de FAMAF en la UNC
- Me gusta que hagan preguntas en cualquier momento

DANIEL GUTSON

- Core Detection Team Leader, Engineering Director

- **Conocer y explorar una herramienta para detectar patrones de comportamiento en binarios**

TIMELINE DEL WORKSHOP

INTRODUCCIÓN 01

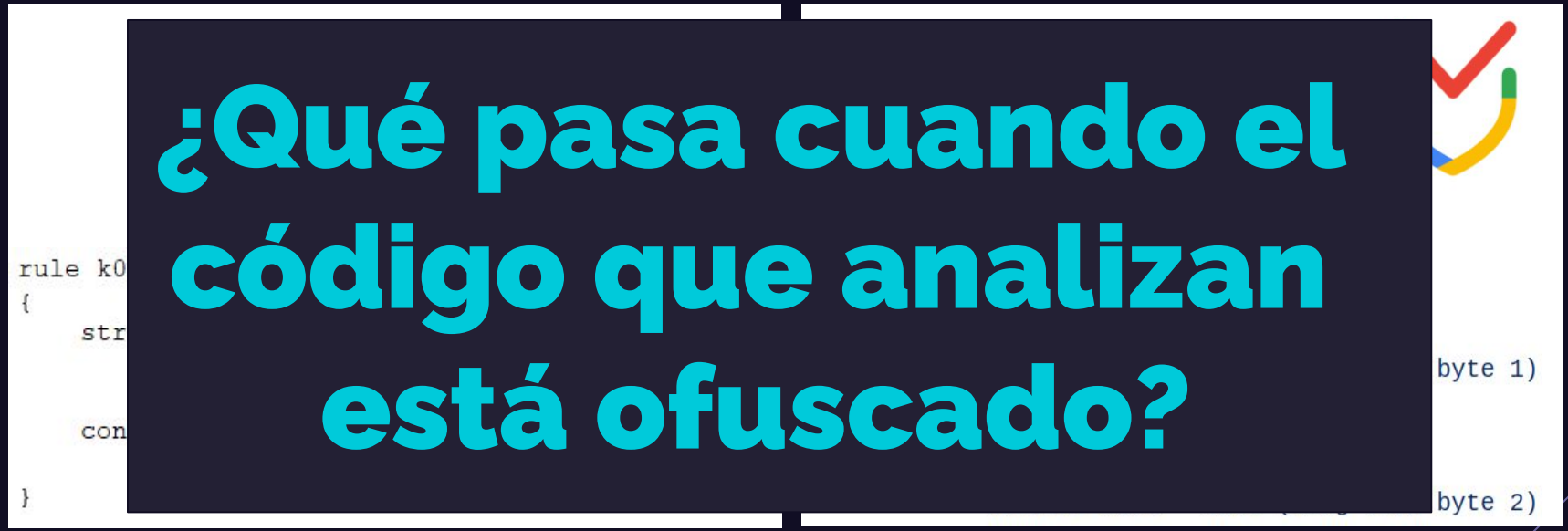
¿QUÉ EXISTE?


```
rule k0adic_test
{
    strings:
        $k0adic_hex = {66 51 62 41 77}
        $k0adic_text = "k0adic" fullword nocase
    condition:
        $k0adic_text or $k0adic_hex
}
```

mandiant/**capa-rules**

```
- and:
  - description: INT3 (long form)
  - instruction:
    - mnemonic: cmp
    - number: 0xCD = INT3 (long form byte 1)
  - instruction:
    - mnemonic: cmp
    - number: 0x03 = INT3 (long form byte 2)
```

¿QUÉ EXISTE?


rule k0
{
str
con
}
byte 1)
byte 2)
¿Qué pasa cuando el código que analizan está ofuscado?

¿QUÉ EXISTE?


yara
rule k0adic_test
{
strings:
$k0adic_hex = {66 51 62 41
$k0adic_text = "k0adic" fullwo
condition:
$k0adic_text or $k0adic_hex
}
ndiant/capa-
ription: INT3 (long form)
uction:
nemonic: cmp
- number: 0xCD = INT3 (long form byte 1)
instruction:
- mnemonic: cmp
- number: 0x03 = INT3 (long form byte 2)
FAILED

¿QUÉ EXISTE?

BUENO ✗ Behavioral

BONITO ✓ Basado en reglas (automático*)

BARATO ✓ Buena performance

*Para alguna definición de automático

TRADEOFF

BUENO ✓ **Behavioral**

BONITO ✓ **Basado en reglas (automático*)**

BARATO ✗ **Buena performance**

*Para alguna definición de automático

PRELIMINARES 02

ANÁLISIS DE BINARIOS

Una forma de análisis de programas

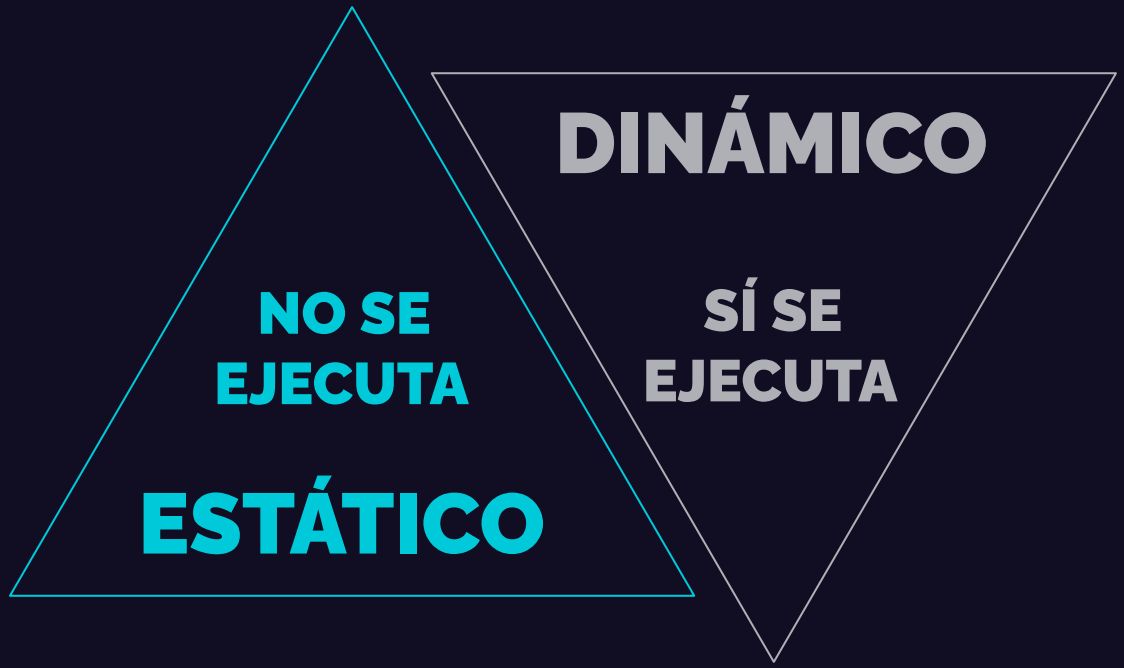
Código que analiza código

Examinar propiedades de binarios para conocer su comportamiento o estructura

¿Para qué?

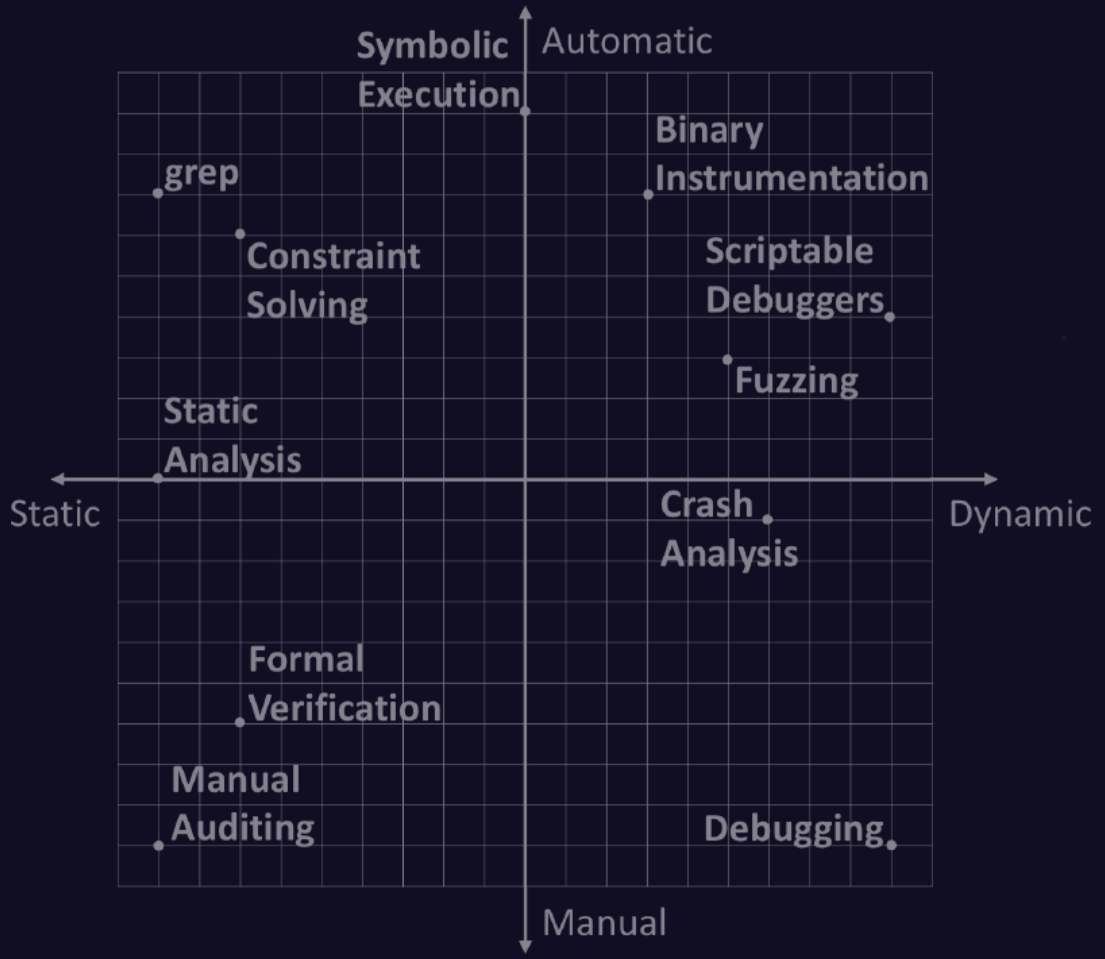
- Encontrar vulnerabilidades o bugs
- Chequear si el código es malicioso
- Hacer ingeniería reversa de la funcionalidad
- Profiling

TIPOS DE ANÁLISIS



TIPOS DE ANÁLISIS

TIPOS DE ANÁLISIS



Graph from: Contemporary Automatic Program Analysis ~ By Julian Cohen

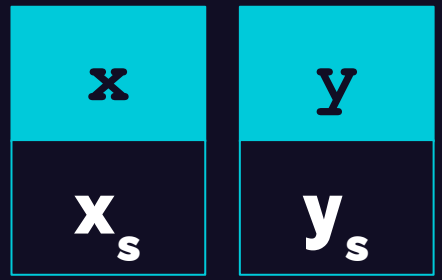
EJECUCIÓN SIMBÓLICA

```
1  void foo(int x, int y) {
2    int t = 0;
3    if (x > y) {
4        t = x;
5    } else {
6        t = y;
7    }
8    if (t < x) {
9        assert(false);
10   }
11 }
```

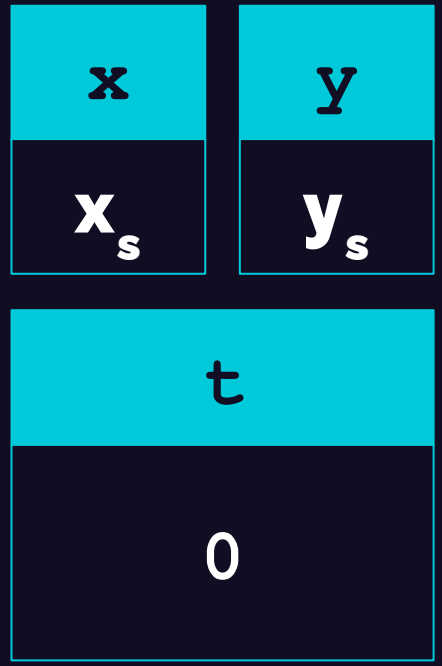
¿Puede ejecutarse el assert?

Ejemplo de: Armando Solar-Lezama

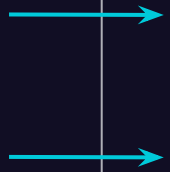
```
 1 void foo(int x, int y) {
 2   int t = 0;
 3   if (x > y) {
 4       t = x;
 5   } else {
 6       t = y;
 7   }
 8   if (t < x) {
 9       assert(false);
10   }
11 }
```



```
 1 void foo(int x, int y) {
 2   int t = 0;
 3   if (x > y) {
 4       t = x;
 5   } else {
 6       t = y;
 7   }
 8   if (t < x) {
 9       assert(false);
10   }
11 }
```

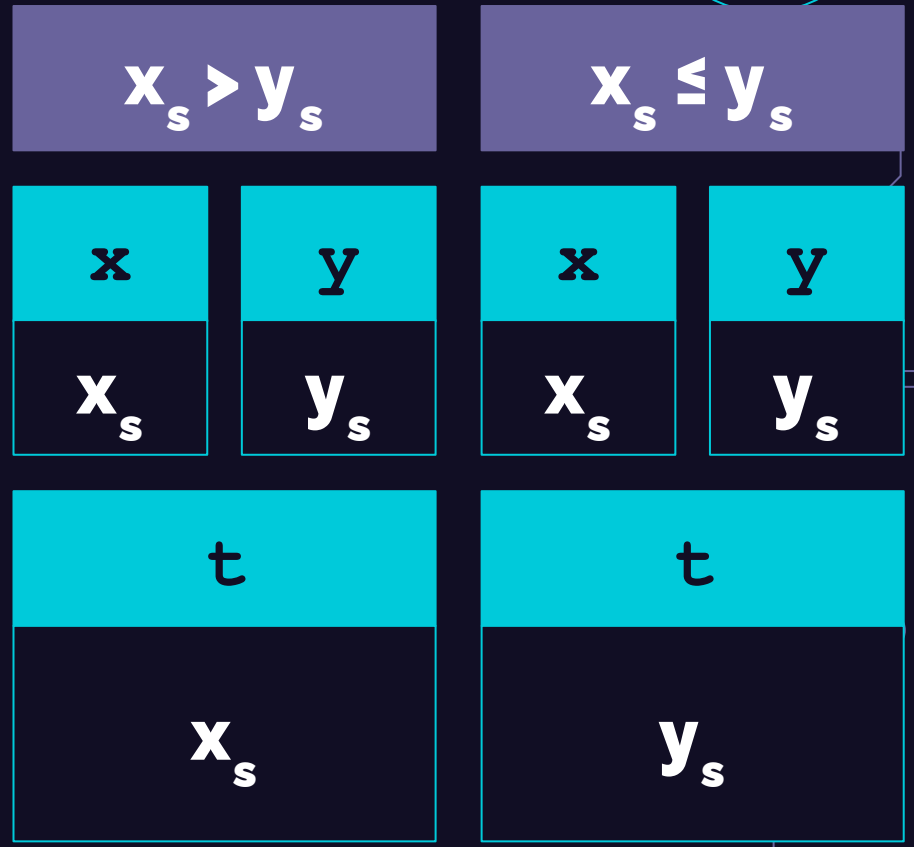


```
1  void foo(int x, int y) {
2    int t = 0;
3    if (x > y) {
4      t = x;
5    } else {
6      t = y;
7    }
8    if (t < x) {
9      assert(false);
10   }
11 }
```



$x_s > y_s$		$x_s \leq y_s$	
x	y	x	y
x_s	y_s	x_s	y_s
t		t	
0		0	

```
1  void foo(int x, int y) {
2    int t = 0;
3    if (x > y) {
4      t = x;
5    } else {
6      t = y;
7    }
8    if (t < x) {
9      assert(false);
10   }
11 }
```



```
 1 void foo(int x, int y) {
 2   int t = 0;
 3   if (x > y) {
 4       t = x;
 5   } else {
 6       t = y;
 7   }
 8   if (t < x) {
 9       assert(false);
10   }
11 }
```

x	y
x_s	y_s

t
x_s si $(x_s > y_s)$ y_s si $(x_s \leq y_s)$

```
1  void foo(int x, int y) {
2    int t = 0;
3    if (x > y) {
4        t = x;
5    } else {
6        t = y;
7    }
8    if (t < x) {
9        assert(false);
10   }
11 }
```

$t < x_s$		$t \geq x_s$	
x	y	x	y
x_s	y_s	x_s	y_s
t		t	
x_s si $(x_s > y_s)$ y_s si $(x_s \leq y_s)$		x_s si $(x_s > y_s)$ y_s si $(x_s \leq y_s)$	

```
1  void foo(int x, int y) {
2    int t = 0;
3    if (x > y) {
4        t = x;
5    } else {
6        t = y;
7    }
8    if (t < x) {
9        assert(false);
10   }
11 }
```

$t < x_s$

x	y
x_s	y_s

t
x_s si $(x_s > y_s)$ y_s si $(x_s \leq y_s)$

¿Es un estado posible?


NO

$$t = \begin{cases} x_s & \text{si } (x_s > y_s) \\ y_s & \text{si } (x_s \leq y_s) \end{cases}$$

- $x_s > y_s$

$\Rightarrow t = x_s$

$\Rightarrow \neg(t < x_s)$

- $x_s \leq y_s$

$\Rightarrow t = y_s$

$\Rightarrow t = y_s \geq x_s$

$\Rightarrow \neg(t < x_s)$

¿QUIÉN HACE ESTAS DEMOSTRACIONES?

SAT SOLVING

Responden si una fórmula es satisfacible

$$(a \land b) \lor (\neg c \land d)$$

¿Existe una asignación de valores para las *variables* que hagan a la **fórmula** verdadera?

Satisfiability modulo theories (SMT)

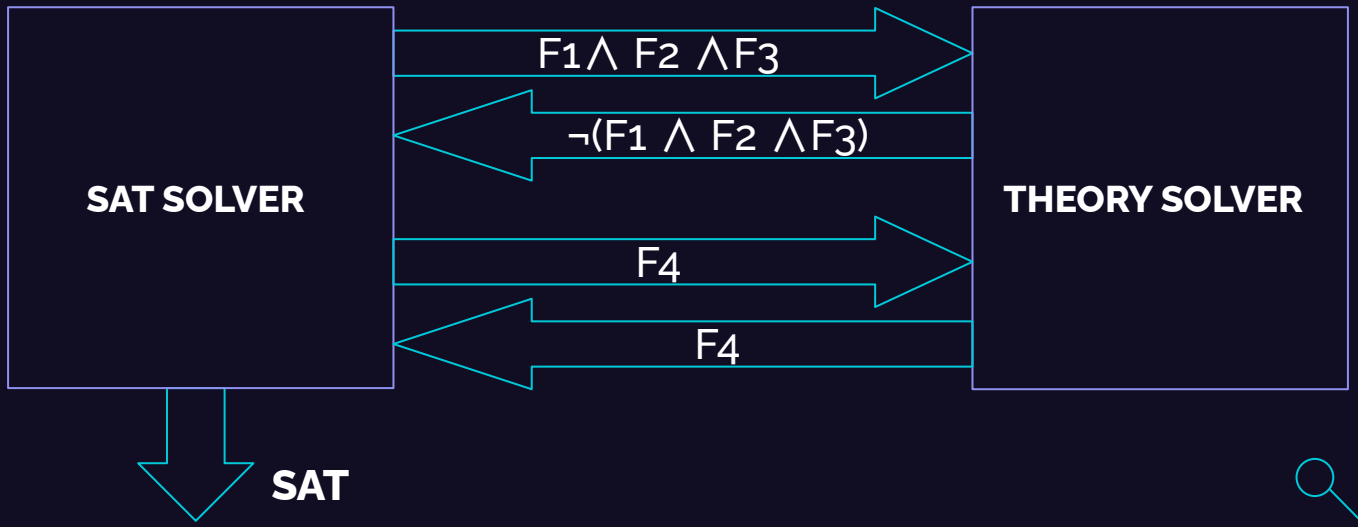
Fórmulas más complejas

- Enteros y reales
- Bit vectors
- Listas y arreglos
- Cuantificadores

SMT

$$(x_s > y_s \land t = x_s \land t < x_s) \lor x_s > y_s$$

F1 F2 F3 F4



angr

angr

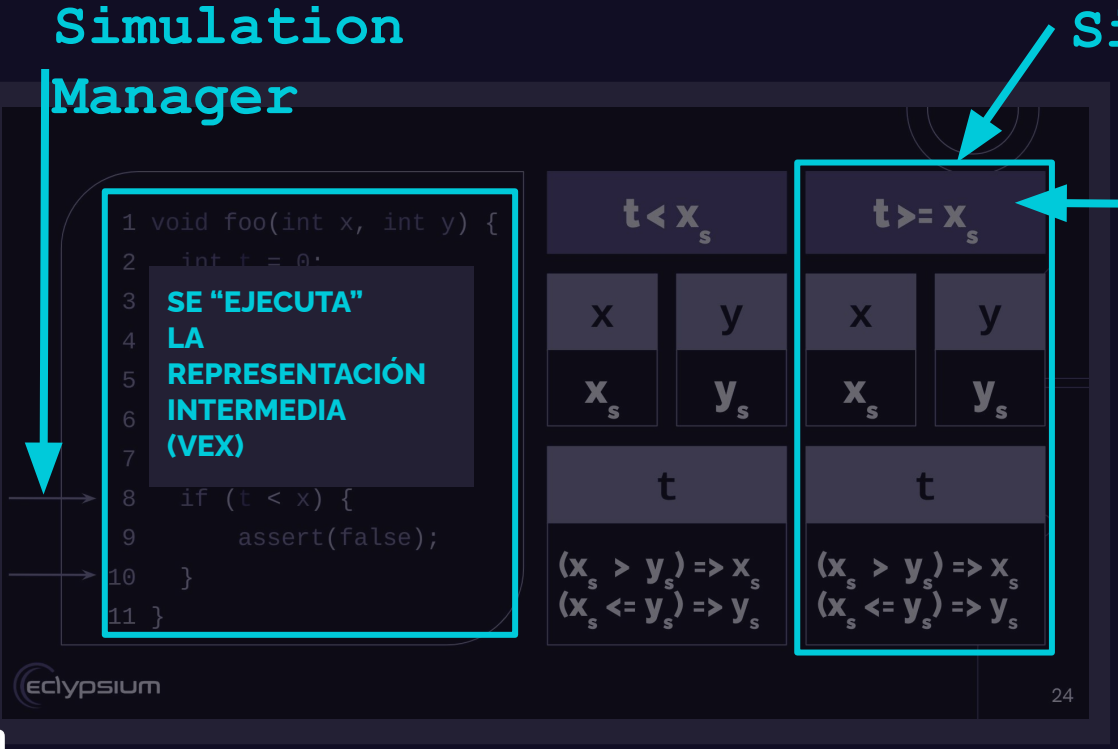

Análisis de binarios
Ejecución simbólica
Y muchas abstracciones

Un framework hecho en Python para el análisis de binarios

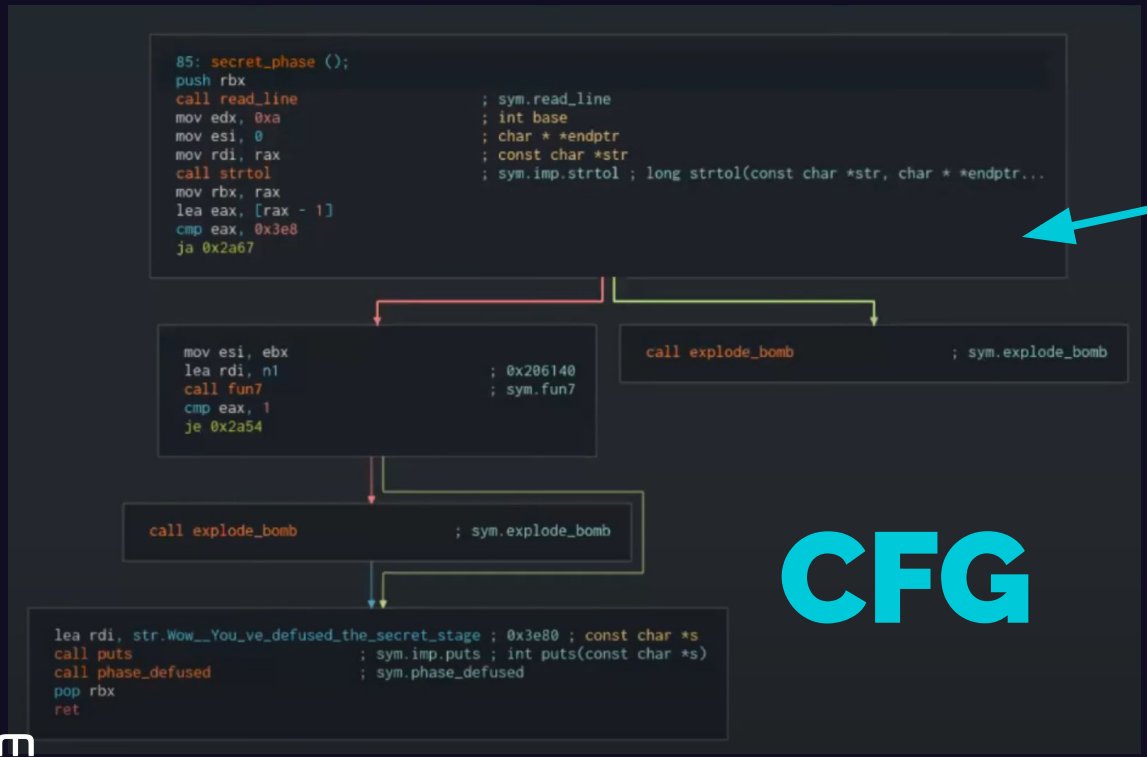
Usa VEX como representación intermedia

"When something is vexing it makes you angry"

Ejecución simbólica con angr



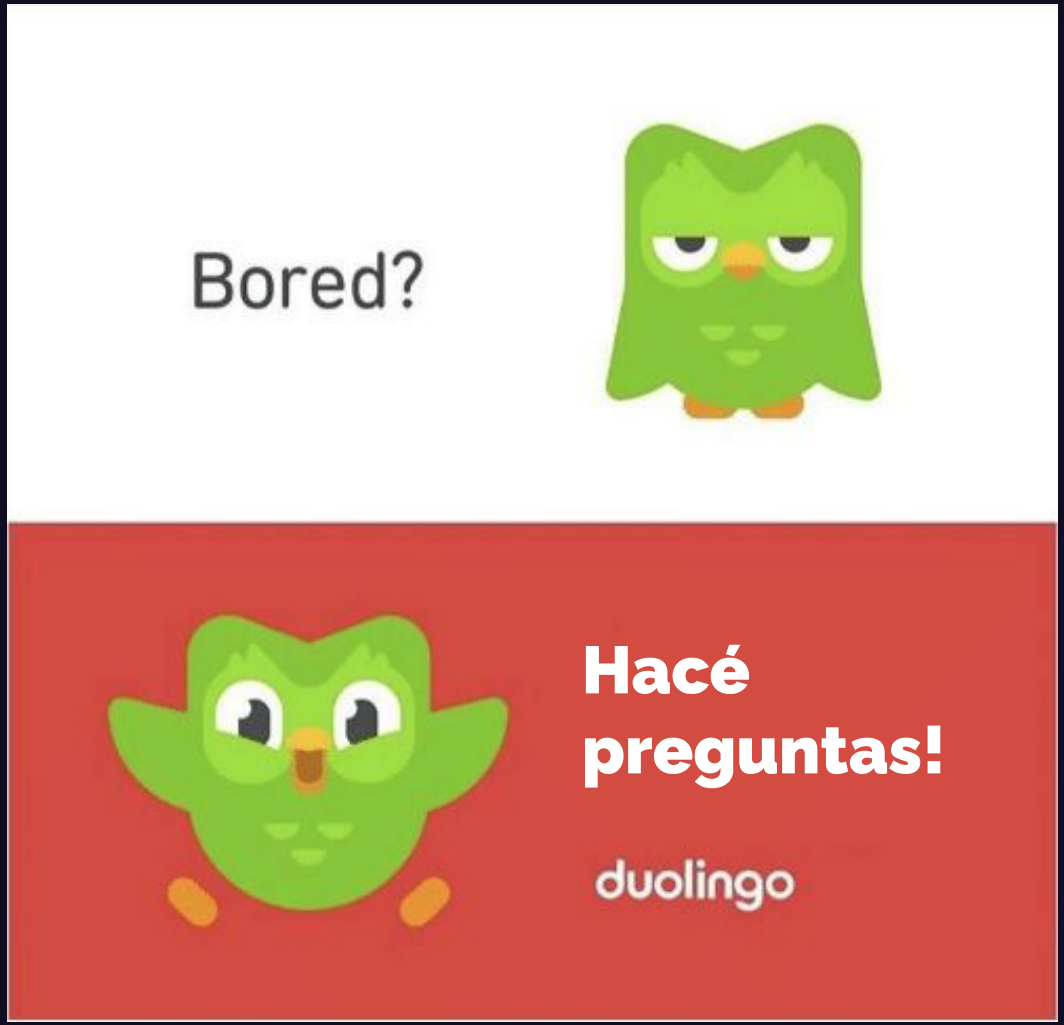
Análisis estático con angr



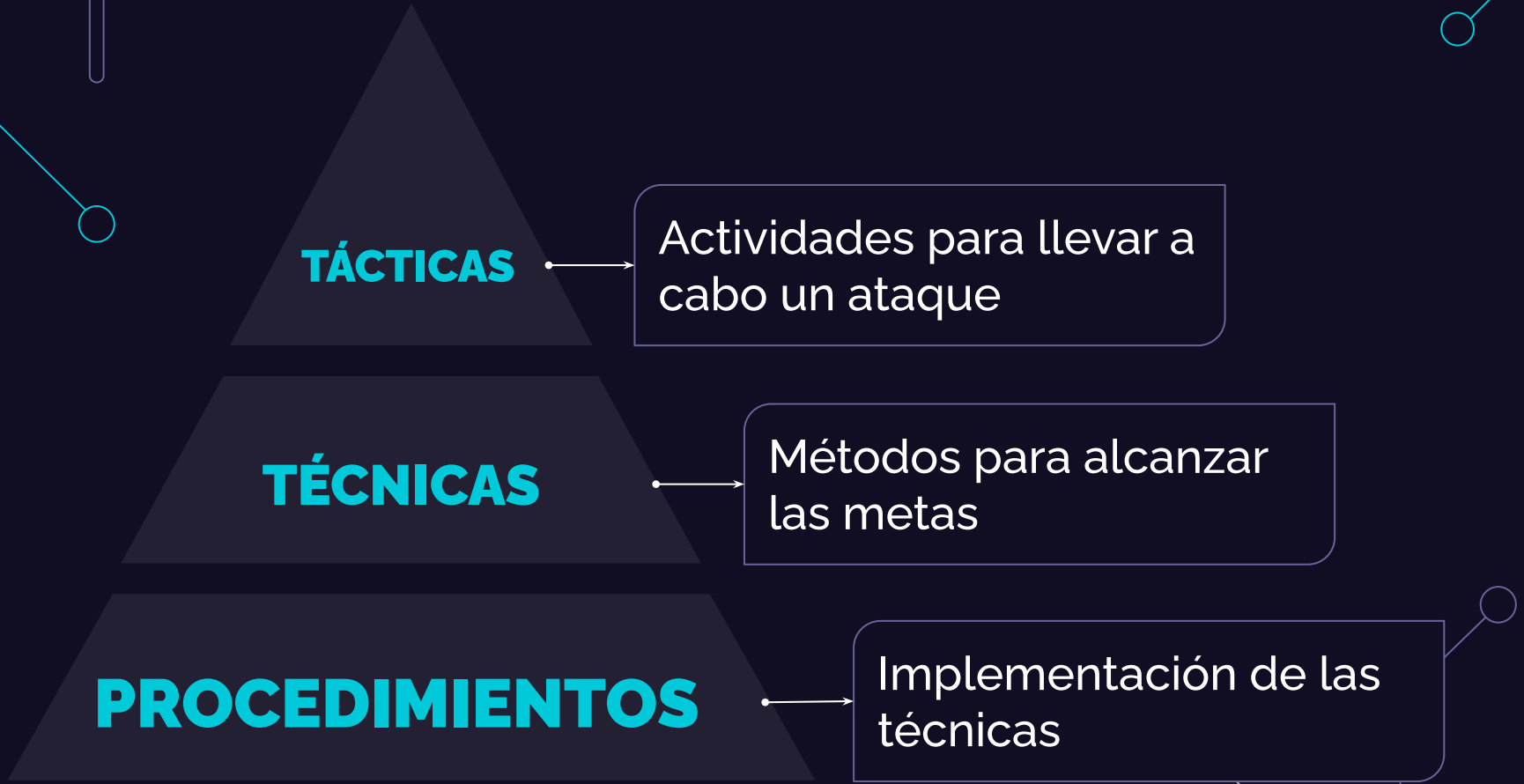
Bloque básico

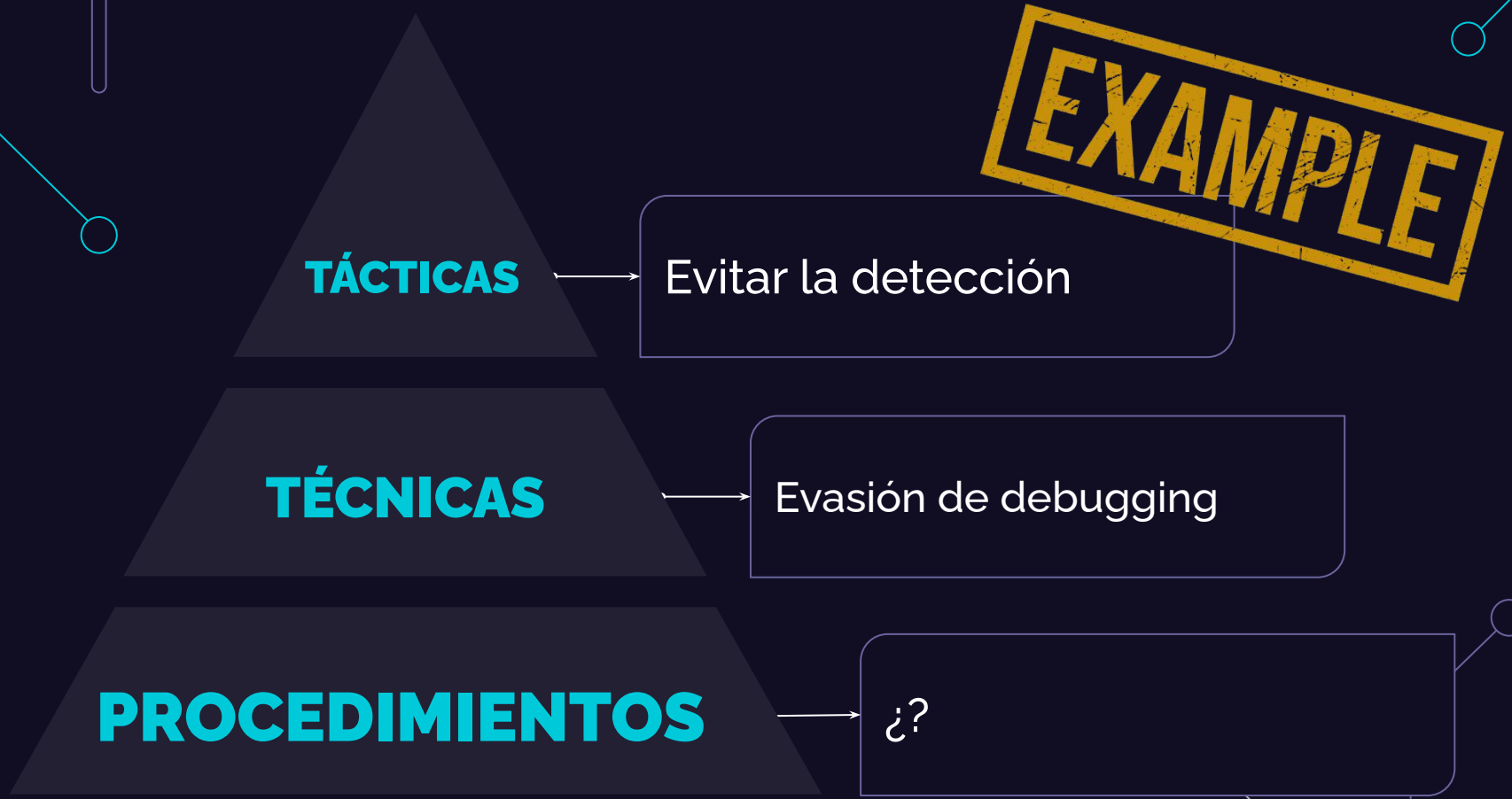
CFG


Bored?
Reestablish the Soviet Union
duolingo
MADE WITH COMMIE SMART FRIDGE


Bored?
Hacé preguntas!
duolingo

TACTICS, TECHNIQUES AND PROCEDURES


TÁCTICAS
Actividades para llevar a cabo un ataque
TÉCNICAS
Métodos para alcanzar las metas
PROCEDIMIENTOS
Implementación de las técnicas


EXAMPLE
TÁCTICAS
Evitar la detección
TÉCNICAS
Evasión de debugging
PROCEDIMIENTOS
¿?

BACKDOOR EN XZ


ONCE
UPON A TIME
THERE WAS A
LOVELY PRINCESS


endbr64

Control Hijacking Attacks

(Ataques de secuestro de control)

Intel's Control-flow Enforcement Technology

Control Hijacking Attacks

(Ataques de secuestro de control)

Intel's Control-flow Enforcement Technology

Siempre saltar a una instrucción válida

endbr64

Siempre saltar a una instrucción válida


int3

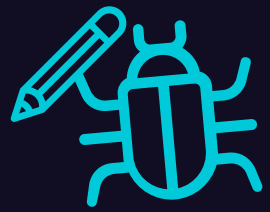
0xCC

INTERRUPCIÓN POR SOFTWARE

Usada por los debuggers para poner breakpoints

```
            cc
f3 0f 1e fa                 endbr64
48 83 ec 08                 sub     $0x8,%rsp
6a 01                       push    $0x1
ff 74 24 18                 push    0x18(%rsp)
...
```

int3



¿Cómo sabemos si estamos siendo debugueados?

Si asumimos que el binario fue compilado con soporte para Intel's Control-flow Enforcement (CET)

```
__int64 check_software_breakpoint(
    _DWORD *code_addr,
    __int64 a2, int a3) {

 unsigned int v4 = 0;
 if ( a2 - code_addr > 3 )
   return *code_addr + (a3 | 0x5E20000) == 0xF223;
 return v4;

}
```

Code from: smx-smx/XZ Backdoor Analysis

UNA FORMA OFUSCADA DE DECIR

```
*code_addr == 0xFA1E0FF3;
```

```
return *code_addr + (a3 | 0x5E20000) == 0xF223;
```

*Solo si la función es llamada con ciertos parámetros

Code from: smx-smx/XZ Backdoor Analysis

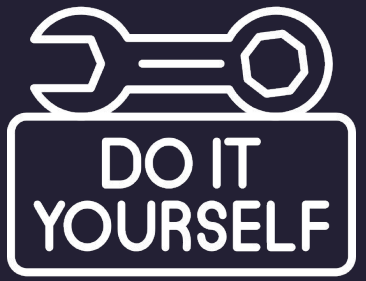
UNA FORMA OFUSCADA DE DECIR

```
*code_addr == endbr64;
```

```
return *code_addr + (a3 | 0x5E20000) == 0xF223;
```

*Solo si la función es llamada con ciertos parámetros

https://github.com/LuciaMartinezGavier/eko-dangr-rt

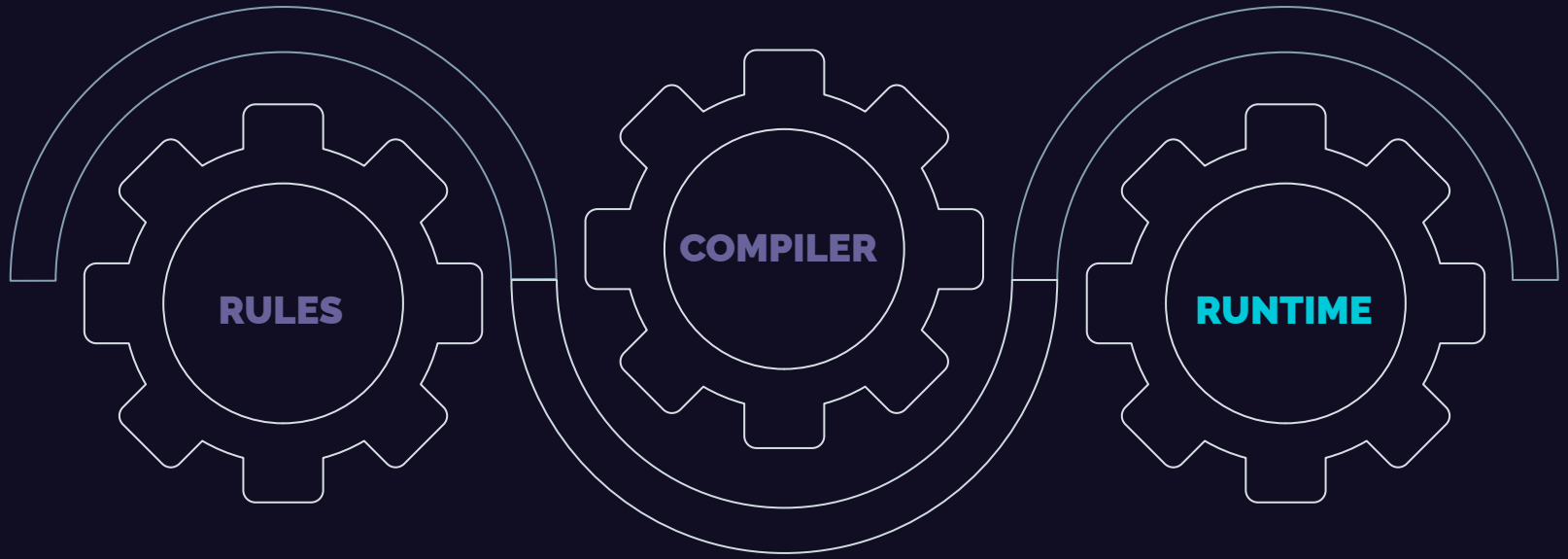


DANGR

03

DECLARATIVE angr

DANGR



DECLARATIVE angr

https://github.com/LuciaMartinezGavier/dangr

DANGR RUNTIME

```
$ pip install dangr-rt
$ python
>>> import dangr_rt
```

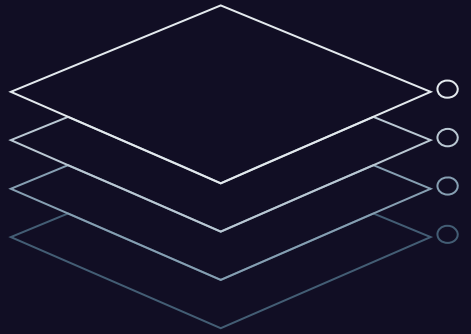
FILTRO ESTRUCTURAL

DEPENDENCIA DE DATOS

CONSTRAINTS

EJECUCIÓN Y RESOLUCIÓN

EL PRIMER FILTRO ES ESTRUCTURAL



JASM

Pattern matcher del assembler

- Rápido
- Provee el contexto
- Se desconoce el estado del programa

```
macros:
 - name: "@any"
   pattern: "[^, ]{0,1000}"
pattern:
 - $or:
   - add:
     - $deref:
         main_reg: "@any"
     - "@any"
 - "@any":
     times:
       min: 0
       max: 3
 - cmp
```

JASM

```
$ poetry shell
$ poetry install
$ python
src/jasm/main.py
-p jasm_rule.yaml
-b liblzma.so.5.6.1
--all-matches
```

```
macros:
 - name: "@any"
   pattern: "[^, ]{0,1000}"
pattern:
 - $or:
   - add:
     - $deref:
         main_reg: "@any"
     - "@any"
 - "@any":
     times:
       min: 0
       max: 3
 - cmp
```

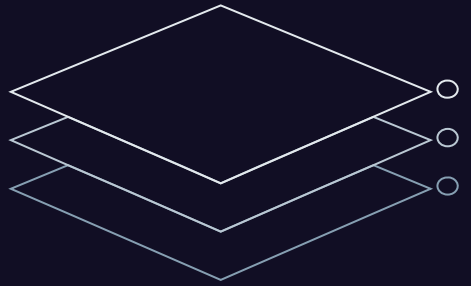
https://github.com/JukMR/JASM

```
from dangr_rt.jasm_findings\
import structural_filter

jasm_matches = structural_filter(
        binary_path,
        jasm_rule)
```

https://github.com/LuciaMartinezGavier/eko-dangr-rt

FILTRO SEGÚN DEPENDENCIA DE DATOS




QUÉ QUEREMOS!?
PODER IDENTIFICAR DEPENDENCIAS Y FILTRAR LOS MATCHES!!
Y CÓMO LO VAMOS A HACER?
REC
prime
SP
0:10:46

ENCONTRANDO CAMINOS EN EL GRAFO DE DEPENDENCIA DE DATOS

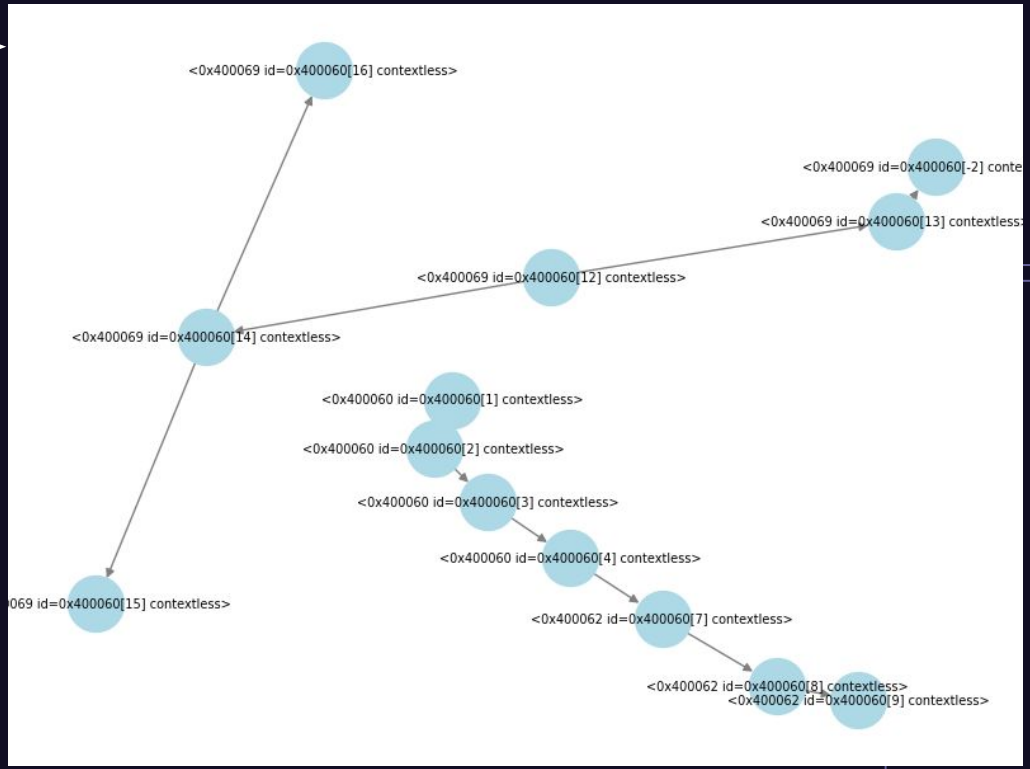
BINARIO

```
movl (%rdi), %eax
movl %eax, 1073741824
ret
```

REPRESENTACÓN INTERMEDIA

```
00 | ------ IMark(0x400060, 2, 0) ------
01 | t0 = GET:I64(rdi)
02 | t6 = LDle:I32(t0)
03 | t5 = 32Uto64(t6)
04 | PUT(rax) = t5
05 | PUT(rip) = 0x0000000000400062
06 | ------ IMark(0x400062, 7, 0) ------
07 | t7 = 64to32(t5)
08 | STle(0x0000000040000000) = t7
09 | PUT(rip) = 0x0000000000400069
10 | ------ IMark(0x400069, 1, 0) ------
11 | t2 = GET:I64(rsp)
12 | t3 = LDle:I64(t2)
13 | t4 = Add64(t2,0x0000000000000008)
14 | PUT(rsp) = t4
15 | t9 = SubA64(t4,0x0000000000000080)
16 | ===== AbiHint(0xt9, 128, t3) =====
   NEXT: PUT(rip) = t3; Ijk_Ret
```

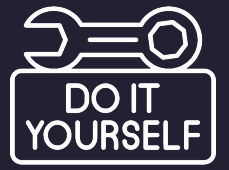
DATA DEPENDENCY GRAPH



ANÁLISIS DE DEPENDENCIAS

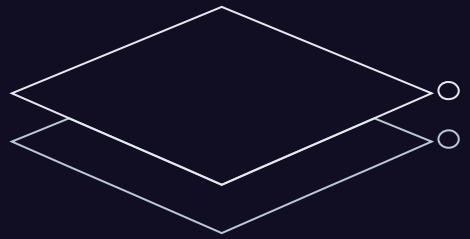
```
vf = dangr.get_variable_factory()
x = vf.create_from_capture(j_match.varmatch_from_name('x'))
y = vf.create_from_capture(j_match.varmatch_from_name('y'))

depends: bool = dangr.depends(x, y)
```

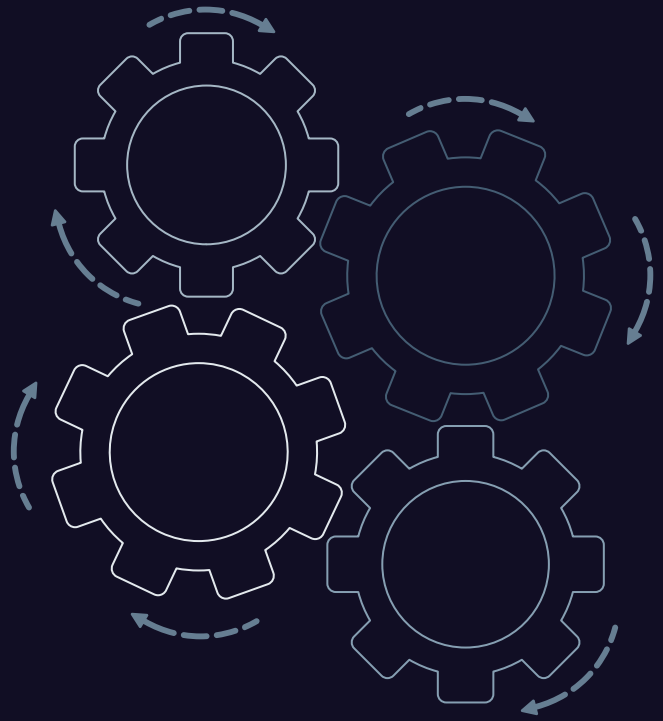


https://github.com/LuciaMartinezGavier/eko-dangr-rt

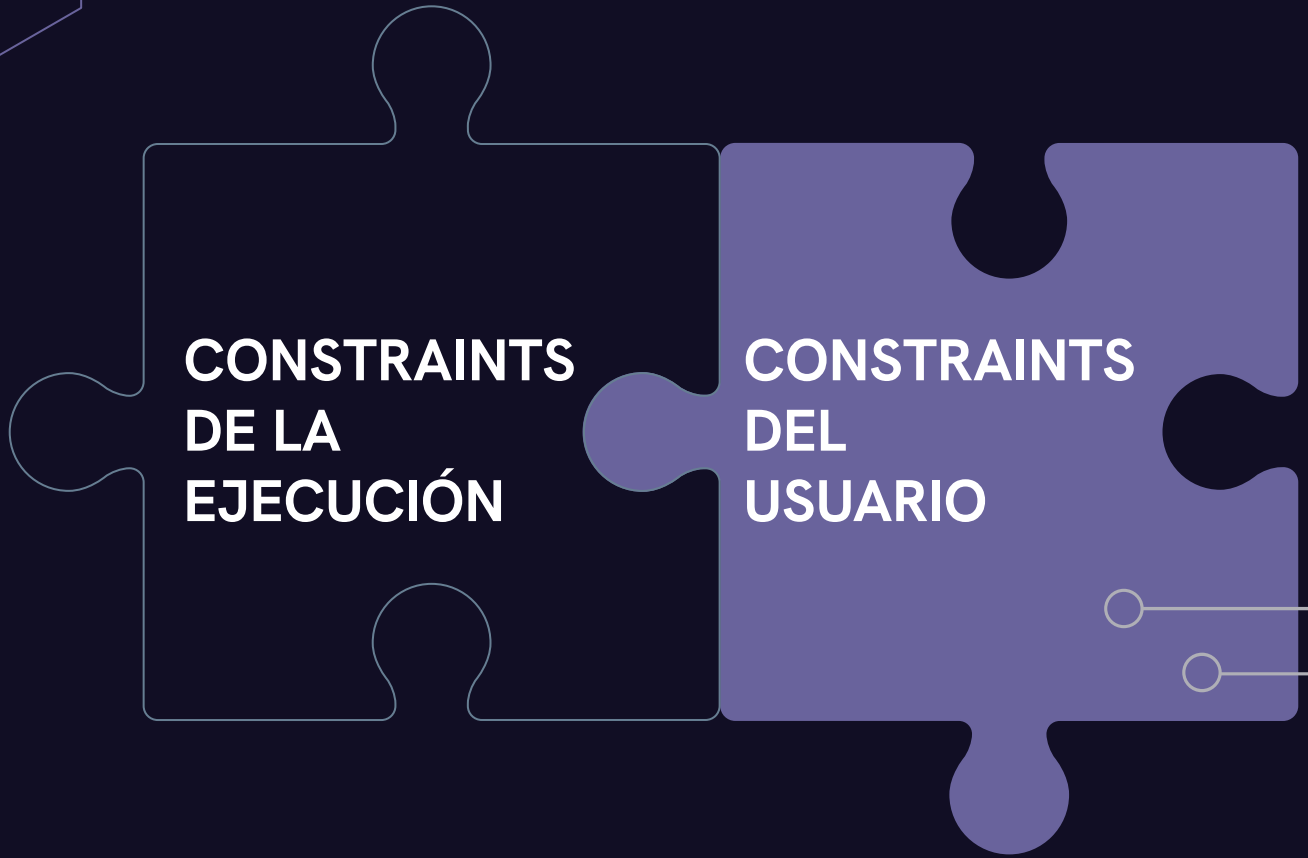
CONSTRAINTS



USAMOS CONSTRAINTS PARA DESCRIBIR LA PARTE “SIMBÓLICA” DE NUESTRO ANÁLISIS



Agregamos *symbolic goals*


CONSTRAINTS
DE LA
EJECUCIÓN
CONSTRAINTS
DEL
USUARIO

```
# Predicado P := IsMax(ptr)
dangr.add_constraint(IsMax(ptr))
[...]
if dangr.satisfiable(states):
    print("Detected!")
```

¿HAY ALGÚN ESTADO QUE SATISFAGA EL PREDICADO P?

$\exists s \in States: sat(s) : P_s(x)$

```
# Predicado P := IsMax(ptr)
dangr.add_constraint(IsMax(ptr))
[...]
if dangr.satisfiable(states):
    print("Detected!")
```

$$\forall s \in States: sat(s) : P_s(x)$$

Para todo estado satisfacible S, x satisface el predicado P en S

$$\forall s \in States:: \neg P_s(x) \Rightarrow \neg sat(s)$$

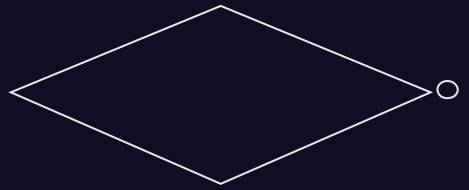
No existe un estado que sea satisfacible y que no valga el predicado P

¿TODOS LOS ESTADOS CUMPLEN P?

```
# Predicado P := dx == ENDNBR64_OPCODE
dangr.add_constraint(Not(Eq(dx, 0xFA1E0FF3)))

[ ... ]
if not dangr.satisfiable(states):
    print("Detected!")
```

EJECUCIÓN SIMBÓLICA

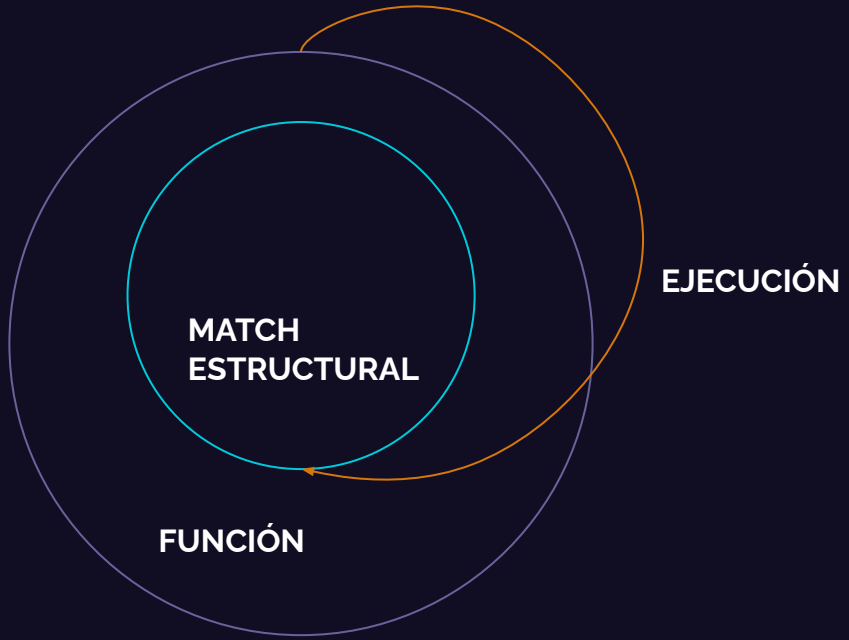


¿CÓMO HACER LA EJECUCIÓN SIMBÓLICA USABLE?

¿QUÉ LA HACE LENTA?	¿QUÉ PODEMOS HACER?
 Muchos estados	 Ejecutar secciones relativamente chicas
 Muchas variables simbólicas	 Concretizar todo lo posible*

*Depende cómo concretizamos, también ese proceso puede ser lento.

ANALIZAR POR FUNCIÓN



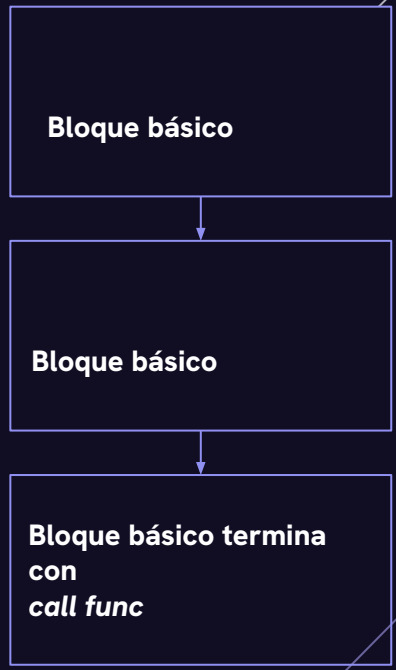
CONCRETIZAR LOS ARGUMENTOS

```
concrete_values = dangr.concretize_fn_args()
```

- Menos variables simbólicas
- Análisis más completo (tenemos más información)
- Cota en cuánto se explora hacia atrás

BACKWARD SIMULATION

- Búsqueda hacia atrás
- Basado en la suposición de que los argumentos se setean cerca de la llamada
- Se ejecutan solo los bloques relevantes

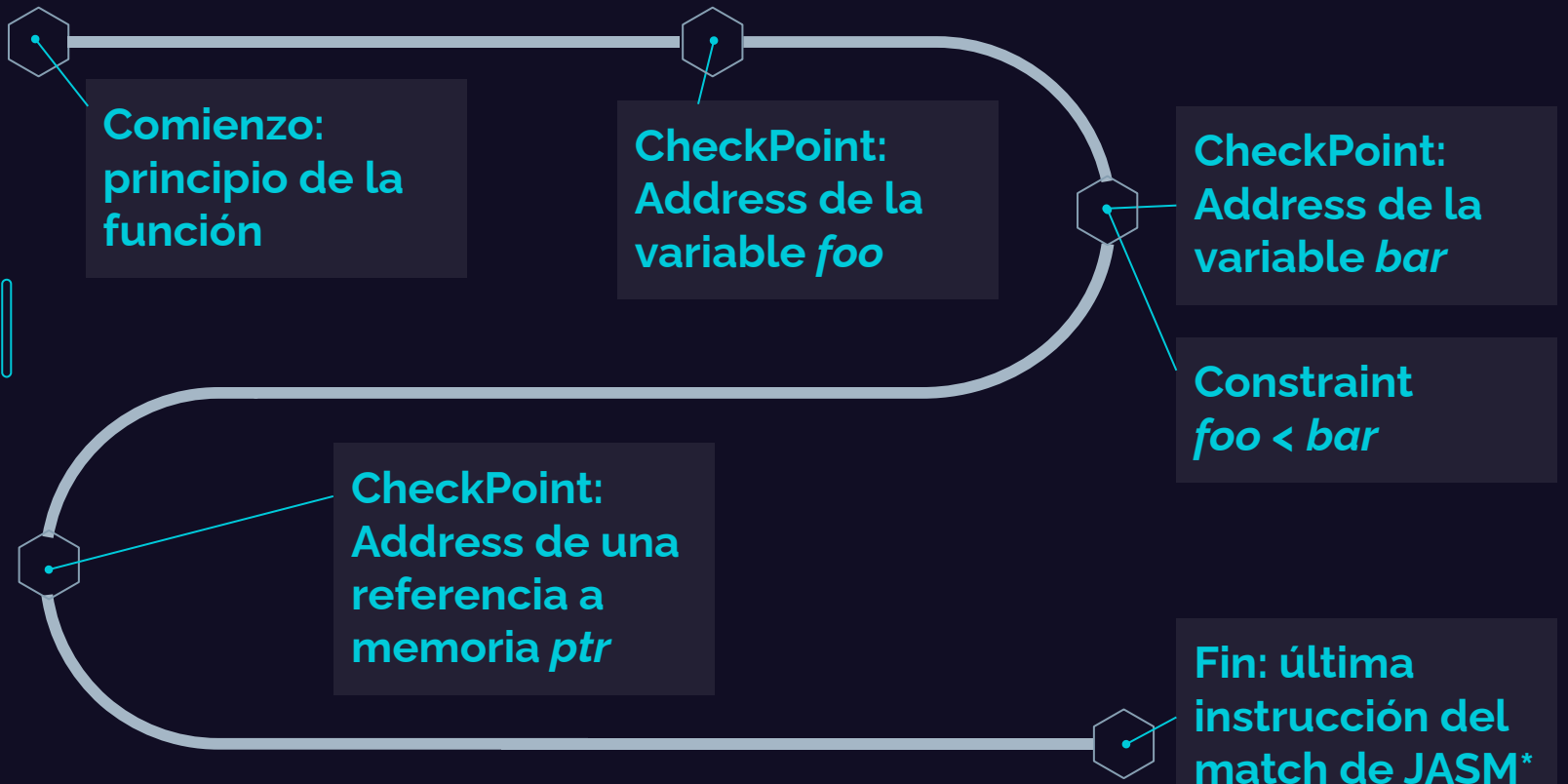


https://github.com/LuciaMartinezGavier/eko-dangr-rt

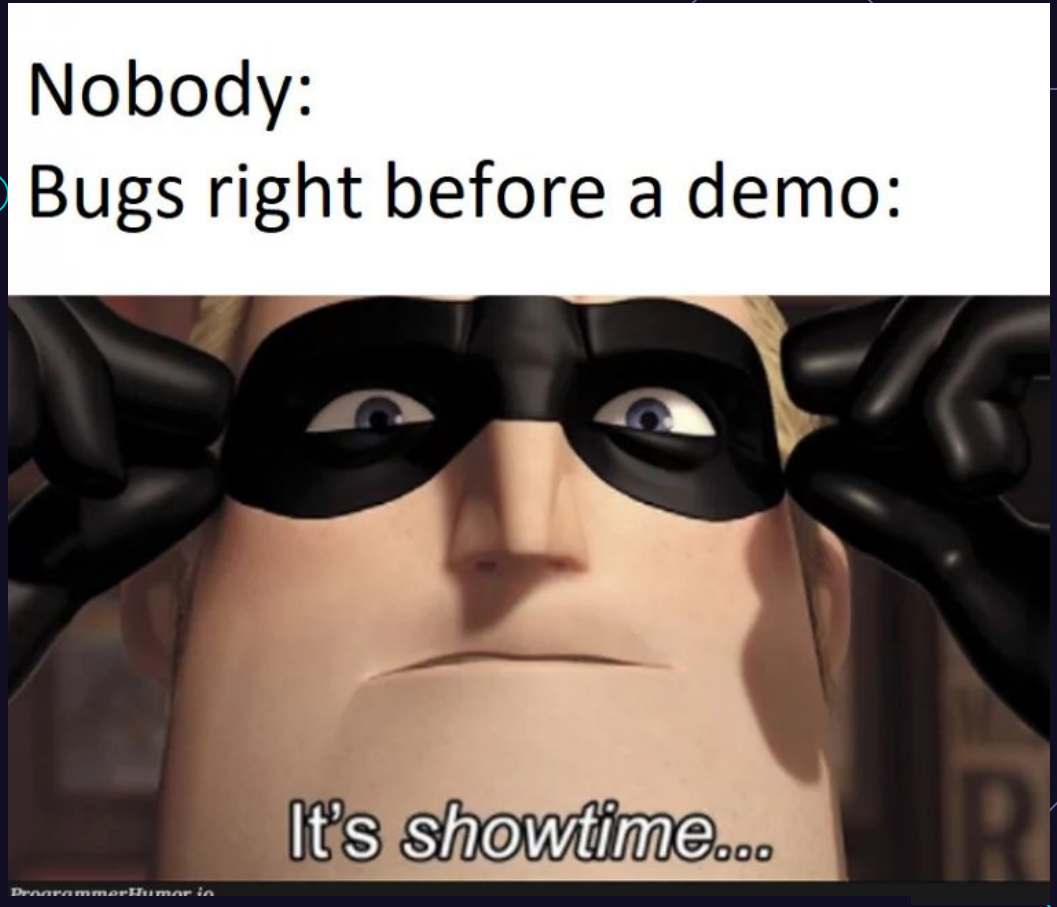
Simulación con Check Points

Las variables y los constraints tienen asociadas direcciones de memoria (del código).

En qué punto en el código son relevantes.



*La simulación podría terminar antes



https://github.com/LuciaMartinezGavier/eko-dangr-rt

REGLAS DANGR 04

```
config:
 solve_arguments: true
given:
 pattern:
 - add:
    - $deref:
      main_reg: "@any-ptr"
    - "@any"
 - cmp:
    - "@any-y"
    - "@any-z"
where:
 - dx = *ptr
 - (dx -> y) or (dx -> z)
such-that:
 - y = z
 - !(dx = 0xFA1E0FF3)
then: false
report: "Debugging evasion
through software breakpoint
detection"
```

ESTRUCTURA DE LA REGLA

CONFIG

Configuraciones generales que modifican el código generado

GIVEN

El pattern estructural: regla de JASM

WHERE

Dependencias e inicialización de variables

SUCH THAT

Constraints sobre la ejecución

THEN

Un booleano, true si se espera algún estado satisfactible

REPORT

Un mensaje que reporta la detección

TAKEAWAYS

05

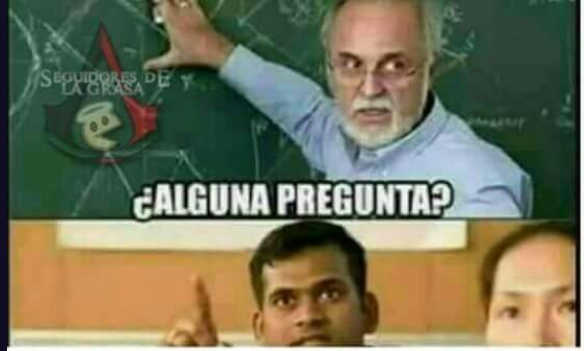
TAKEAWAYS

- Usamos ejecución simbólica
- Conocimos Dangr
- Detectamos el backdoor de XZ

TAKEAWAYS

- Usamos ejecución simbólica
- Conocimos Dangr
- Detectamos el backdoor de XZ
- Vimos a dangr fallar miserablemente

PREGUNTAS



En la película de shrek 2, cuando shrek se toma la pocima de felices por siempre, Fiona también se transforma. Entonces si burro también se la tomo ¿En qué se transformo la dragona?

GRACIAS

¿Más preguntas?
lucia@martinezgavier.com.ar
www.linkedin.com/in/luciamartinezgavier

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**