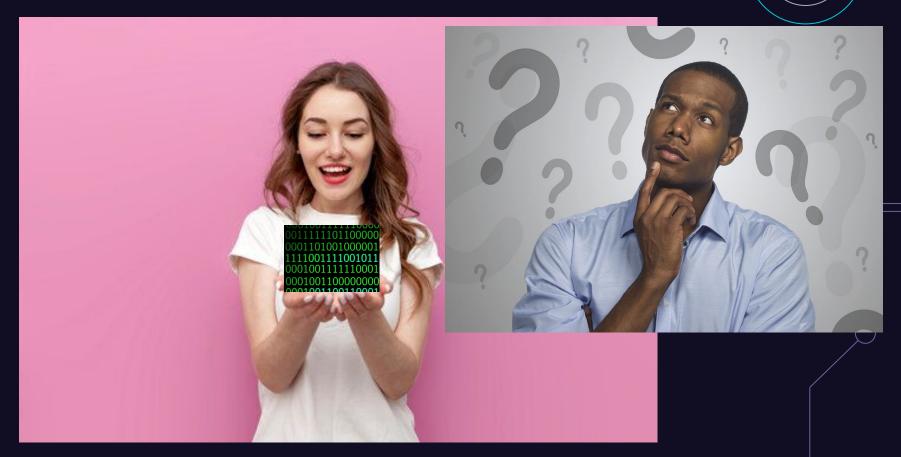
# ENCONTRANDO TTPs DE MALWARE CON EJECUCIÓN SIMBÓLICA

Lucía Martinez Gavier











# 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**





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



#### ► TIMELINE DEL WORKSHOP













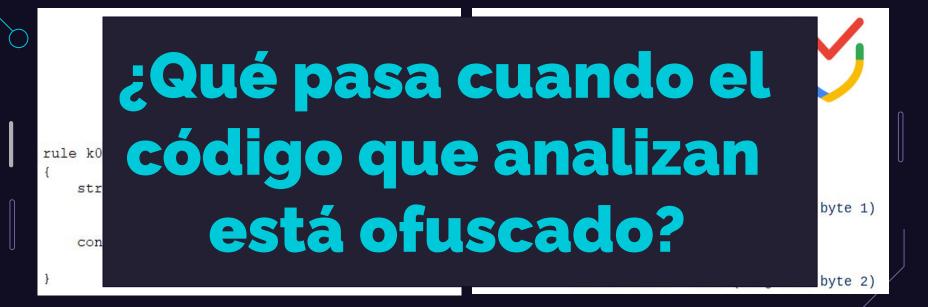
```
rule k0adic_test
{
    strings:
        $k0adic_hex = {66 51 62 41 77}
        $k0adic_text = "k0adic" fullword nocase
    condition:
        $k0adic_text or $k0adic_hex
```

#### mandiant/caparules



- 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)













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

**BARATO** (🔊) Buena performance



<sup>\*</sup>Para alguna definición de automático

#### **► TRADEOFF**



**BONITO** (🕏) Basado en reglas (automático\*)

**BARATO** (X) Buena performance



<sup>\*</sup>Para alguna definición de automático







## ANÁLISIS DE BINARIOS



Una forma de análisis de programas





**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





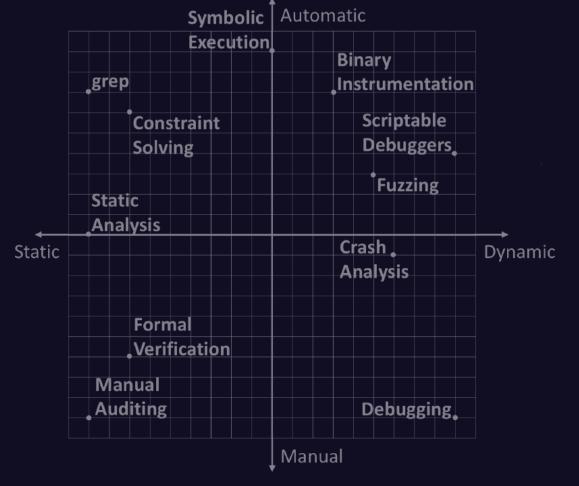
# TIPOS DE ANÁLISIS







# TIPOS DE ANÁLISIS





Graph from: Contemporary Automatic Program Analysis ~ By Julian Cohen

## **EJECUCIÓN SIMBÓLICA**



```
1 void foo(int x, int y) {
    int t = 0;
     if (x > y) {
         t = x;
     } else {
        t = y;
     if (t < x)
         assert(false);
10
11 }
```

# ¿Puede ejecutarse el assert?

Ejemplo de: Armando Solar-Lezama



```
1 void foo(int x, int y) {
     int t = 0;
     if (x > y) {
         t = x;
 5
     } else {
         t = y;
     if (t < x) {
         assert(false);
10
11 }
```





```
1 void foo(int x, int y) {
    int t = 0;
     if (x > y) {
                                  X
         t = x;
 5
     } else {
      t = y;
     if (t < x) {
         assert(false);
10
11 }
```



```
x_s \leq y_s
 1 void foo(int x, int y) {
    int t = 0;
     if (x > y) {
                                                      X
                                   X
         t = x;
 5
     } else {
     t = y;
     if (t < x) {
         assert(false);
10
11 }
```



```
X_s \leq y
                                       X_s > \overline{Y}_s
 1 void foo(int x, int y) {
     int t = 0;
     if (x > y) {
                                                          X
                                      X
          t = x;
     } else {
      t = y;
     if (t < x) {
          assert(false);
10
11 }
```

```
1 void foo(int x, int y) {
       int t = 0;
       if (x > y) {
                                                X
             t = x;
       } else {
            t = y;
       if (t < x) {
             assert(false);
                                            x_s \overline{si(x_s > y_s)}

y_s \overline{si(x_s \le y_s)}
10
11 }
```



```
t < x<sub>e</sub>
 1 void foo(int x, int y) {
       int t = 0;
       if (x > y) {
                                              X
                                                                       X
            t = x;
       } else {
      t = y;
       if (t < x) {
            assert(false);
                                           x_s si(x_s > y_s)
                                                                   x_s si(x_s > y_s)
10
                                           y_s si (x_s \le y_s)
                                                                    y<sub>s</sub> si (x<sub>s</sub>≤y<sub>s</sub>)
11 }
```



```
1 void foo(int x, int y) {
     int t = 0;
     if (x > y) {
         t = x;
     } else {
        t = y;
     if (t < x) {
         assert(false);
10
11 }
```

t < x<sub>s</sub>

x

Xs

Y

y<sub>s</sub>

t

 $x_s si(x_s > y_s)$  $y_s si(x_s \le y_s)$  ¿Es un estado posible?







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

$$\mathbf{x}_{s} > \mathbf{y}_{s}$$

$$\Rightarrow t = x_s$$

$$\Rightarrow \neg (t < x_s)$$









$$\Rightarrow$$
 t =  $y_s \ge x_s$ 

# ¿QUIÉN HACE ESTAS DEMOSTRACIONES?





# Responden si una fórmula es satisfacible



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

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





# Satisfiability modulo theories (SMT)

### Fórmulas más complejas

🔏 Enteros y reales

A Bit vectors

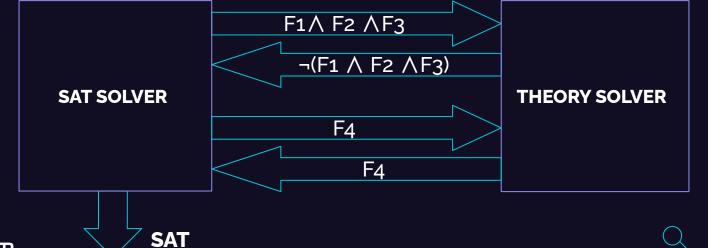
🔏 Listas y arreglos

Cuantificadores



### **SMT**





Eclypsium

3





angr



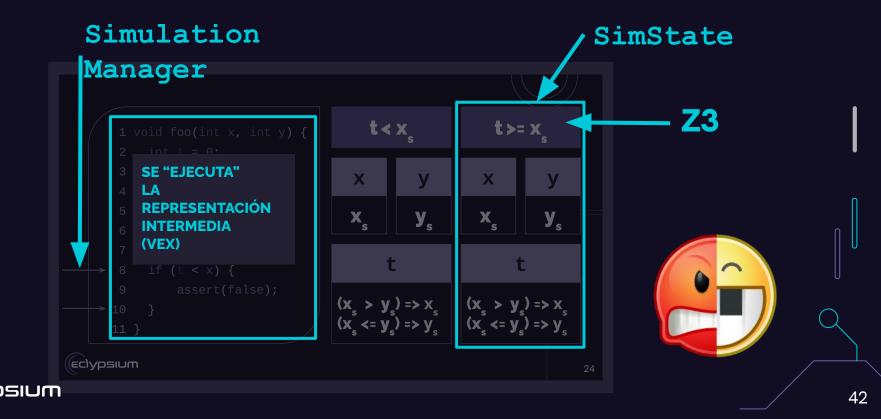
### Un framework hecho en Python para el análisis de binarios

### Usa VEX como representación intermedia

"When something is vexing it makes you angry"



### Ejecución simbólica con angr



### Análisis estático con angr

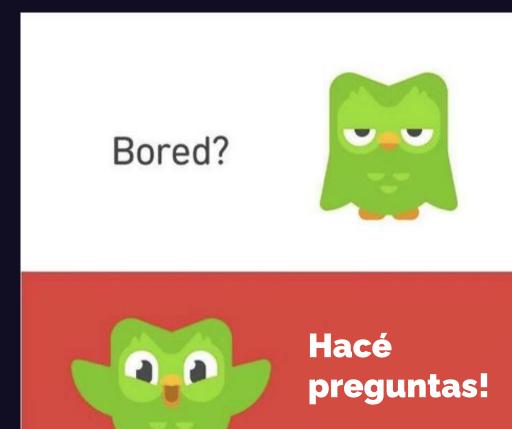


Bloque básico









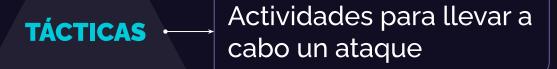
duolingo





### TACTICS, TECHNIQUES AND PROCEDURES





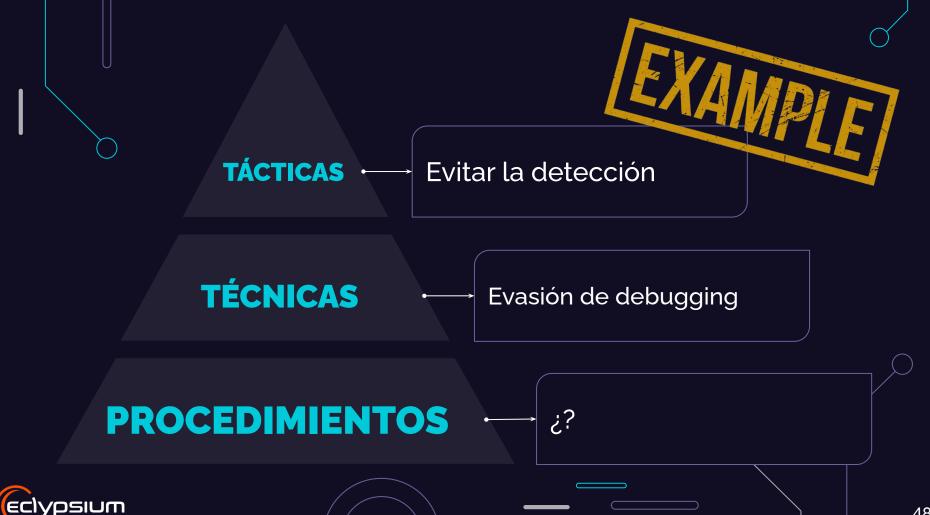
**TÉCNICAS** 

Métodos para alcanzar las metas

### **PROCEDIMIENTOS**

Implementación de las técnicas





### **BACKDOOR EN XZ**









# **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







# INTERRUPCIÓN POR SOFTWARE

Usada por los debuggers para poner breakpoints



# int3 endbr64

<del>f3 Of le fa</del>

48 83 ec 08

6a 01

ff 74 24 18

. . .

sub \$0x8,%rsp

push \$0x1

push 0x18(%rsp)





# ¿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;
```



#### UNA FORMA OFUSCADA DE DECIR

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

\*Solo si la función es llamada con ciertos parámetros



### UNA FORMA OFUSCADA DE DECIR

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

\*Solo si la función es llamada con ciertos parámetros









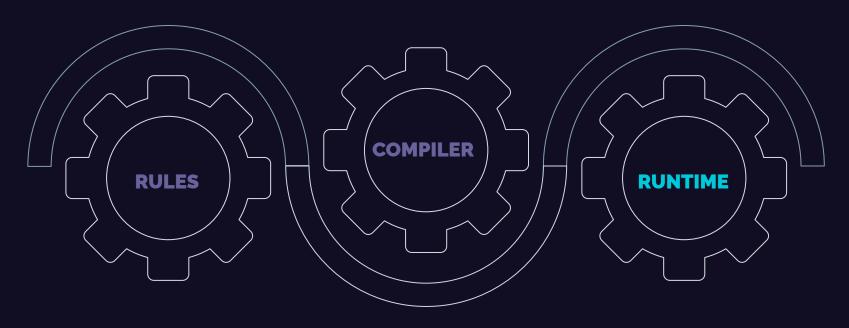


### **▶ DECLARATIVE** angr





### **DANGR**





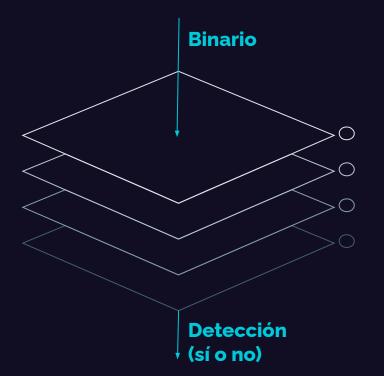
### **▶ DECLARATIVE** angr



```
$ pip install dangr-rt
$ python
```

>>> import dangr\_rt



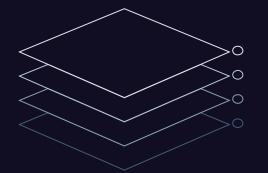


FILTRO ESTRUCTURAL
DEPENDENCIA DE DATOS
CONSTRAINTS
EJECUCIÓN Y RESOLUCIÓN





# EL PRIMER FILTRO ES ESTRUCTURAL







#### **JASM**

### Pattern matcher del assembler

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

```
macros:
pattern:
 - $or:
   - add:
     - $deref:
         main reg: "@any"
 - "@any":
     times:
       min: 0
       max: 3
```





### **JASM**

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

https://github.com/JukMR/JASM

```
macros:
 - name: "@any"
pattern:
 - $or:
   - add:
     - $deref:
         main reg: "@any"
 - "@any":
     times:
       min: 0
       max: 3
```



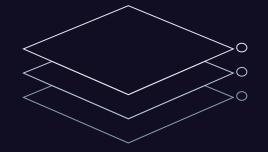
```
from dangr rt.jasm findings\
import structural filter
jasm matches = structural filter(
       binary path,
       jasm rule)
```

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





### FILTRO SEGÚN DEPENDENCIA DE DATOS

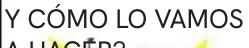






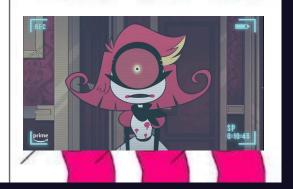














# ENCONTRANDO CAMINOS EN EL GRAFO DE DEPENDENCIA DE DATOS

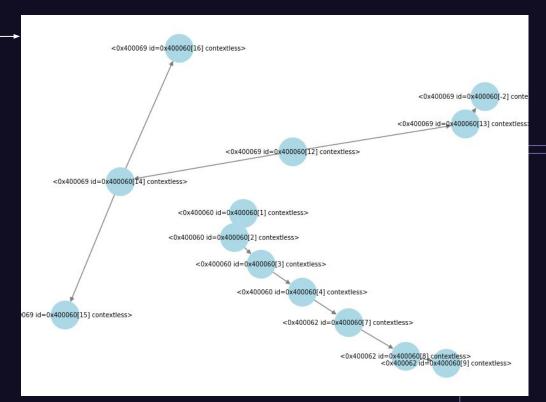


```
BINARIO movl (%rdi), %eax movl %eax, 1073741824 ret
```

#### REPRESENTACÓN INTERMEDIA

```
----- IMark(0x400060, 2, 0) -----
  t0 = GET: I64(rdi)
02 \mid t6 = LDle: I32(t0)
03 \mid t5 = 32Uto64(t6)
04 \mid PUT(rax) = t5
05 \mid PUT(rip) = 0x000000000400062
06 \mid ----- \text{IMark}(0x400062, 7, 0) -----
  1 t7 = 64to32(t5)
08 \mid STle(0x000000040000000) = t7
09 \mid PUT(rip) = 0x0000000000400069
10 \mid ----- \text{IMark}(0x400069, 1, 0) -----
  t2 = GET: 164 (rsp)
12 \mid t3 = LDle: 164(t2)
14 \mid PUT(rsp) = t4
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)
```



# CONSTRAINTS





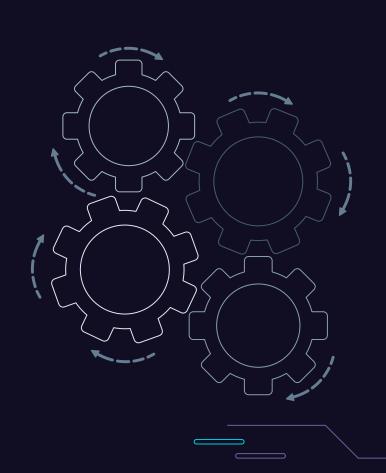




**USAMOS** CONSTRAINTS **PARA DESCRIBIR LA PARTE** "SIMBÓLICA" **DE NUESTRO** ANÁLISIS

Agregamos symbolic goals







```
# 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

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

Para cada estado, si no vale P, entonces el estado no es satisfacible





# ¿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

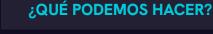






# ¿CÓMO HACER LA EJECUCIÓN SIMBÓLICA USABLE?

#### ¿QUÉ LA HACE LENTA?





Muchos estados



Ejecutar secciones relativamente chicas



Muchas variables simbólicas



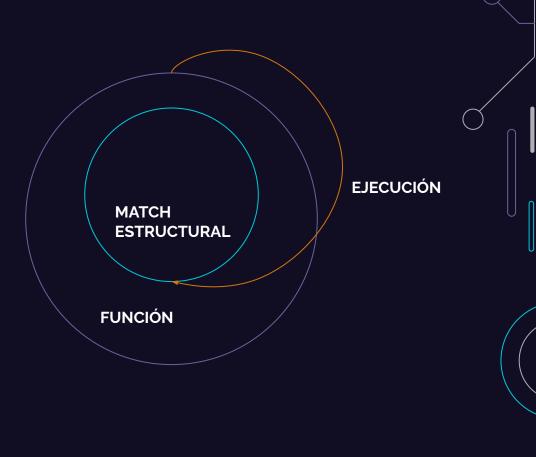
Concretizar todo lo posible\*





<sup>\*</sup>Depende cómo concretizamos, también ese proceso puede ser lento.

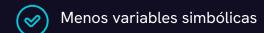
# ANALIZAR POR FUNCIÓN

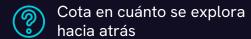


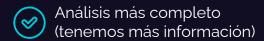


## **CONCRETIZAR LOS ARGUMENTOS**

concrete\_values = dangr.concretize\_fn\_args()









# 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

**eclypsium** 

# Simulación con Check Points

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

En qué punto en el código son relevantes.





# Nobody: Bugs right before a demo:





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

# > REGLAS DANGR 04

eclypsium



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



#### **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









#### TAKEAWAYS











# **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 <u>Slidesgo</u>, and includes icons by <u>Flaticon</u>, and infographics & images by <u>Freepik</u>

