

# BinCAT

*Purrfecting binary static analysis*

8 juin 2017

Philippe Biondi, Raphaël Rigo, Sarah Zennou, Xavier Mehrenberger

# Plan

Introduction

Démonstration

Sous le capot

Conclusion

# Plan


Introduction

Démonstration

Sous le capot

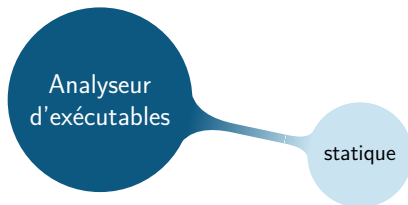
Conclusion

# BinCAT (*Binary Code Analysis Toolkit*)

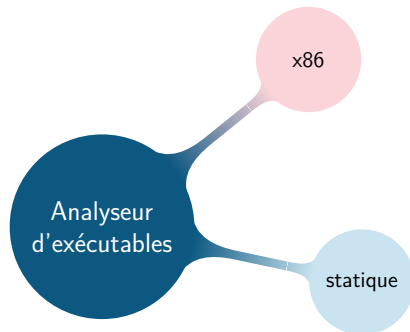


Analyseur  
d'exécutables

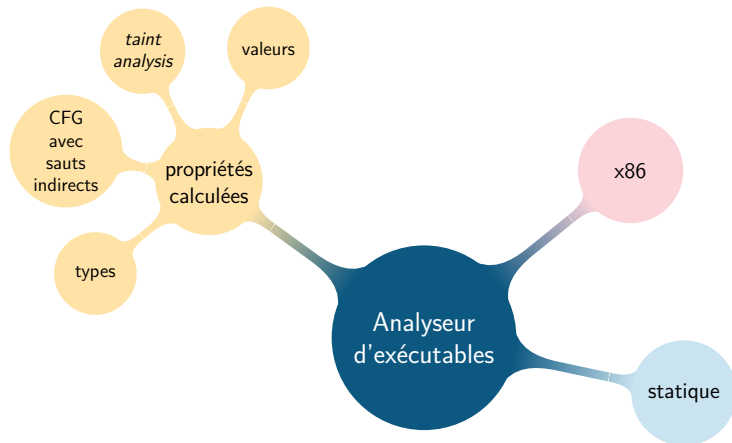
# BinCAT (*Binary Code Analysis Toolkit*)



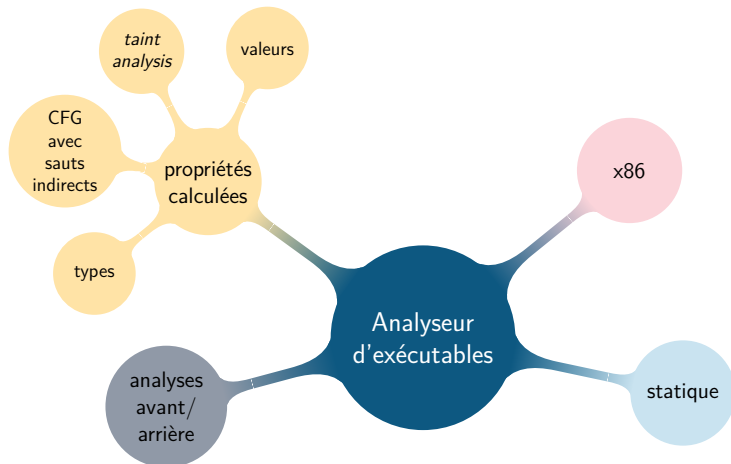
# BinCAT (*Binary Code Analysis Toolkit*)



# BinCAT (*Binary Code Analysis Toolkit*)

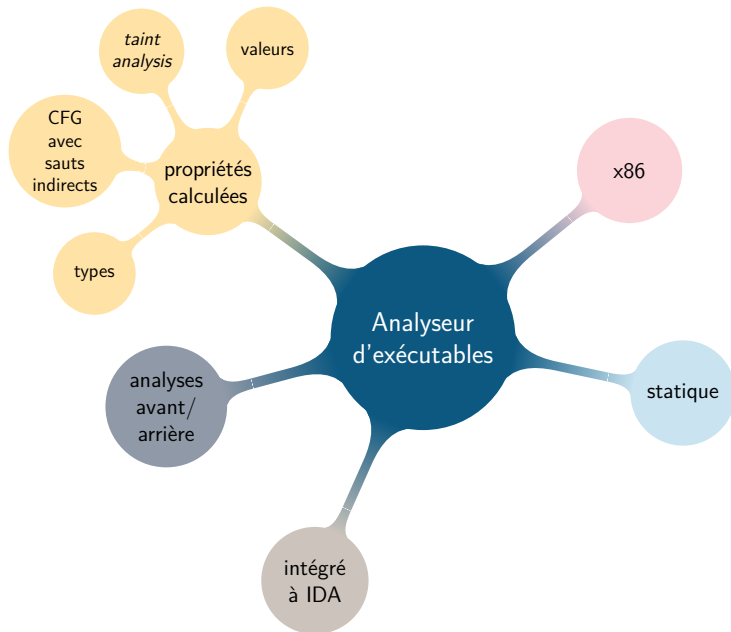


# BinCAT (*Binary Code Analysis Toolkit*)

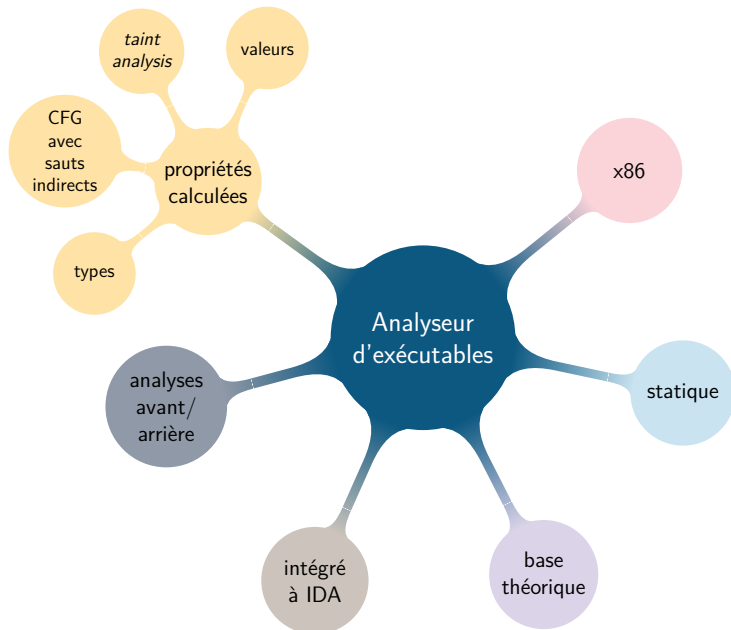




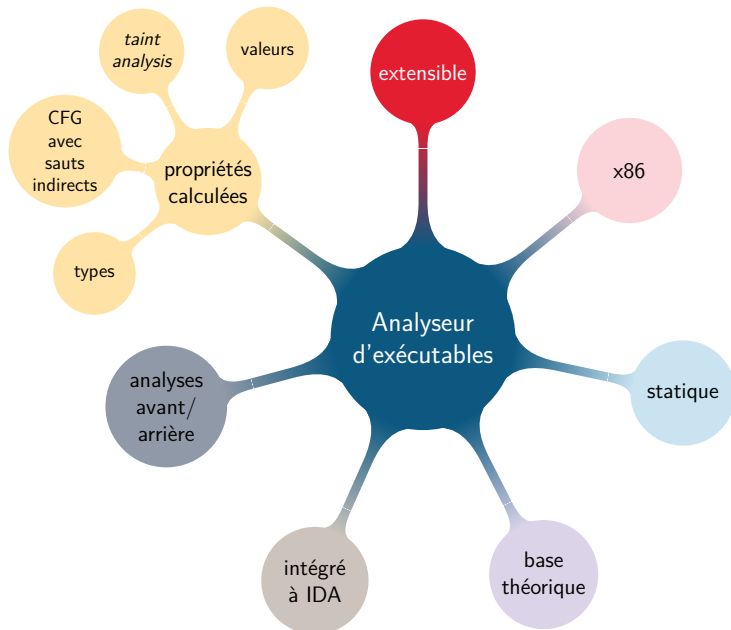
# BinCAT (*Binary Code Analysis Toolkit*)



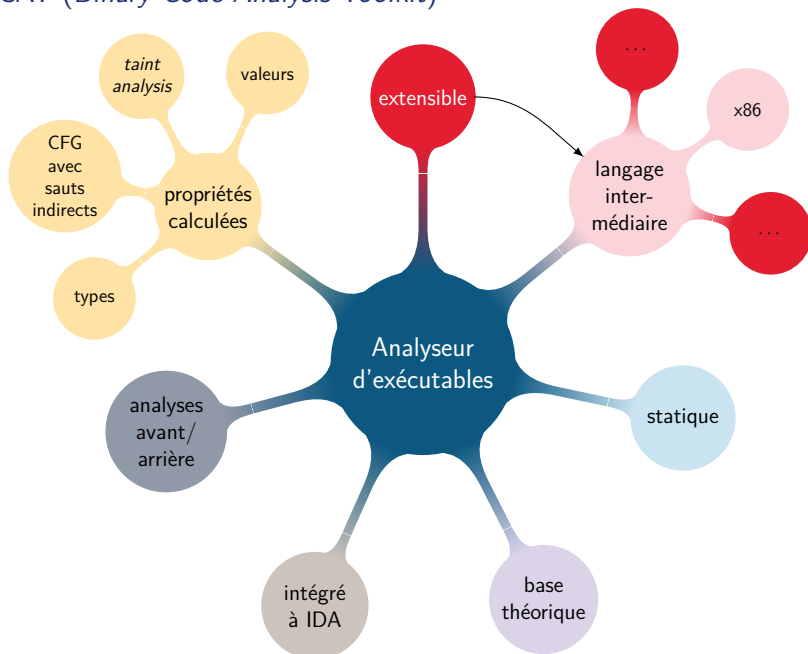
# BinCAT (*Binary Code Analysis Toolkit*)



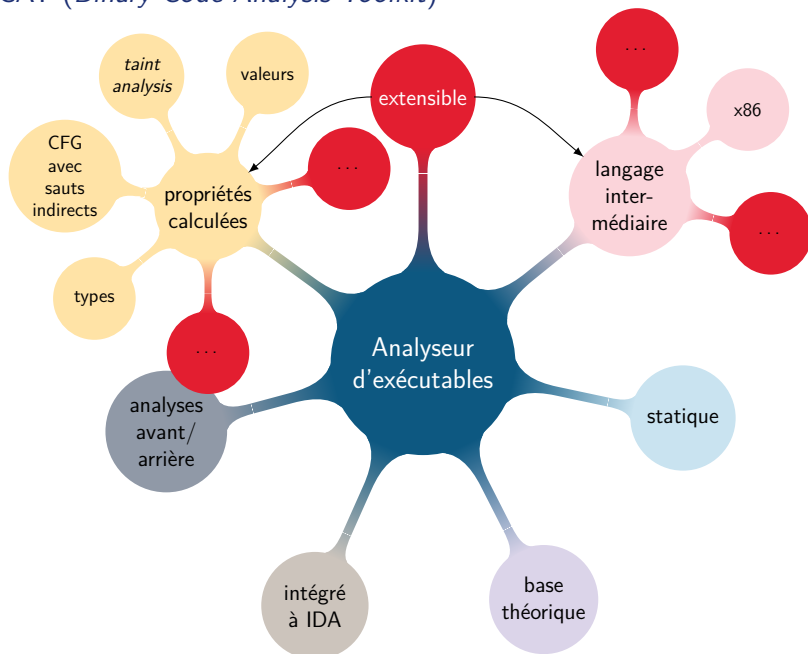
# BinCAT (Binary Code Analysis Toolkit)



# BinCAT (*Binary Code Analysis Toolkit*)



# BinCAT (Binary Code Analysis Toolkit)



# Plan

Introduction

Démonstration

Sous le capot

Conclusion

## Exemple : keygenme

```
$ ./get_key
```

```
Usage: ./get_key company department name licence
```

## Exemple : keygenme

```
$ ./get_key
```

```
Usage: ./get_key company department name licence
```

```
$ ./get_key company department name wrong_serial
```



## Exemple : keygenme

```
$ ./get_key
```

```
Usage: ./get_key company department name licence
```

```
$ ./get_key company department name wrong_serial
```

```
Licence=>[025E60CB08F00A1A23F236CC78FC819CE6590DD7]
```

```
Invalid serial licence
```

## Exemple : keygenme

```
$ ./get_key
```

```
Usage: ./get_key company department name licence
```

```
$ ./get_key company department name wrong_serial
```

```
Licence=>[025E60CB08F00A1A23F236CC78FC819CE6590DD7]
```

```
Invalid serial licence
```

```
$ ./get_key company department name 025E60CB0[...]
```

## Exemple : keygenme

```
$ ./get_key
```

```
Usage: ./get_key company department name licence
```

```
$ ./get_key company department name wrong_serial
```

```
Licence=>[025E60CB08F00A1A23F236CC78FC819CE6590DD7]
```

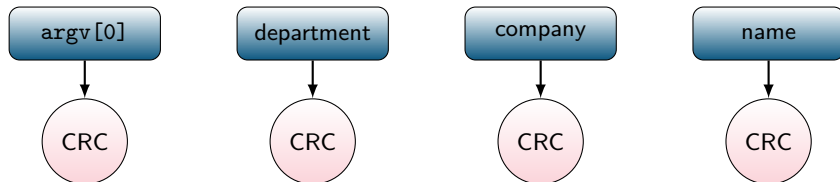
```
Invalid serial licence
```

```
$ ./get_key company department name 025E60CB0[...]
```

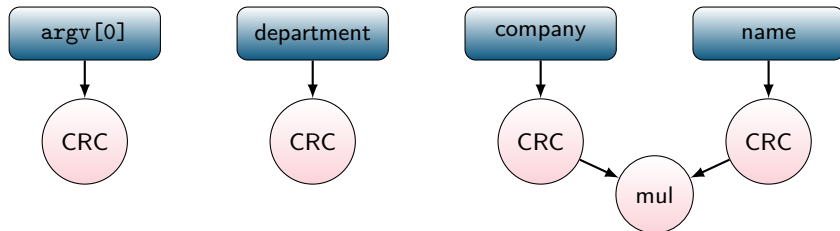
```
Licence=>[025E60CB08F00A1A23F236CC78FC819CE6590DD7]
```

```
Thank you for registering !
```

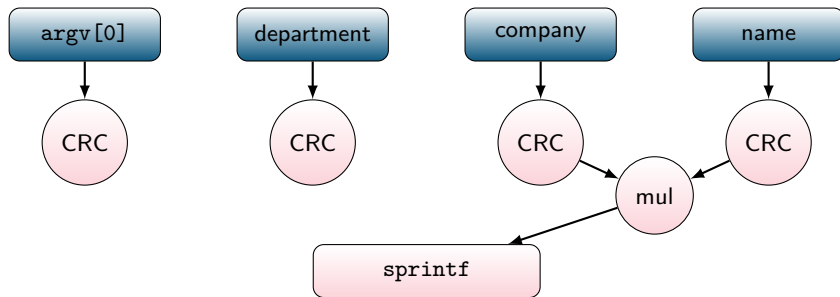
## Keygenme : principe



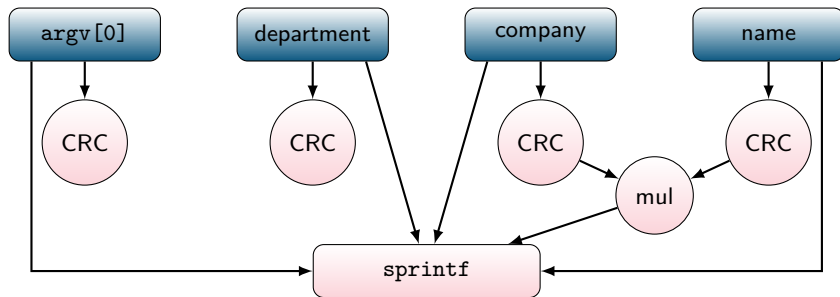
## Keygenme : principe



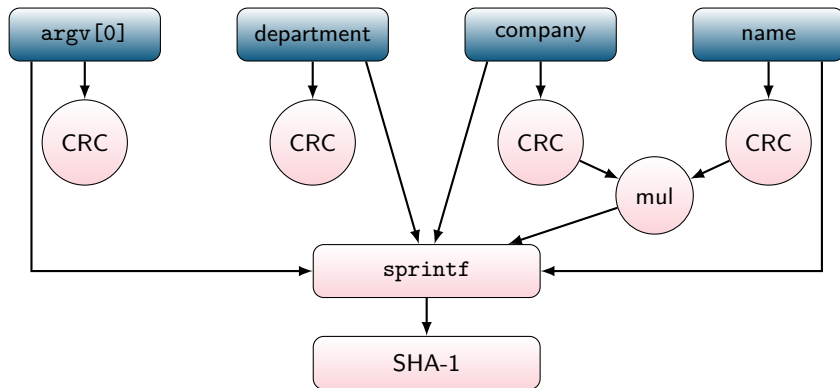
## Keygenme : principe



## Keygenme : principe

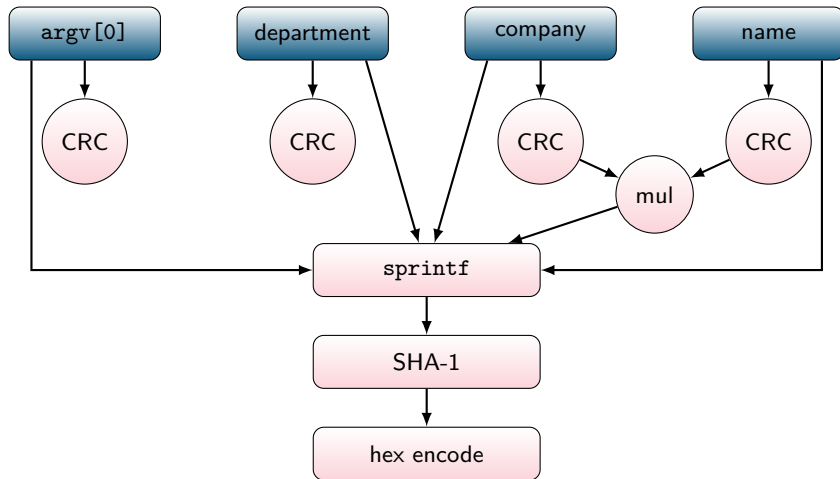


## Keygenme : principe

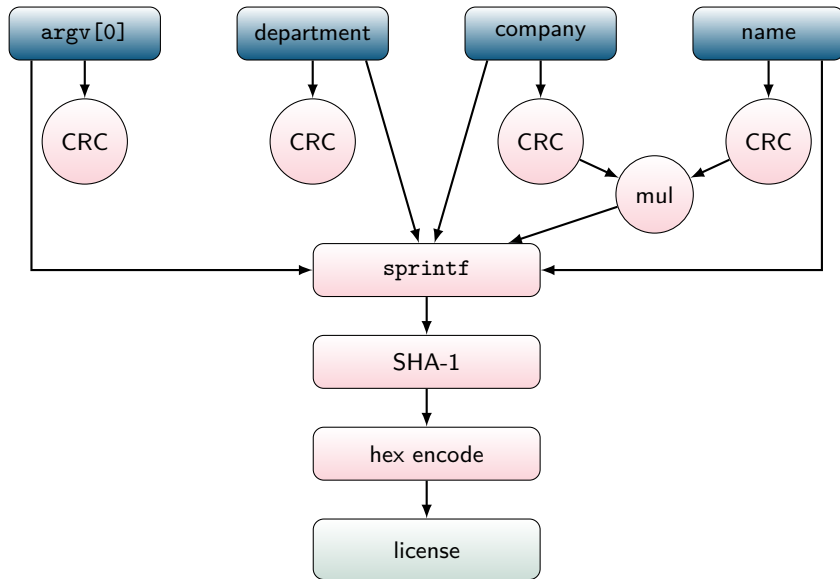




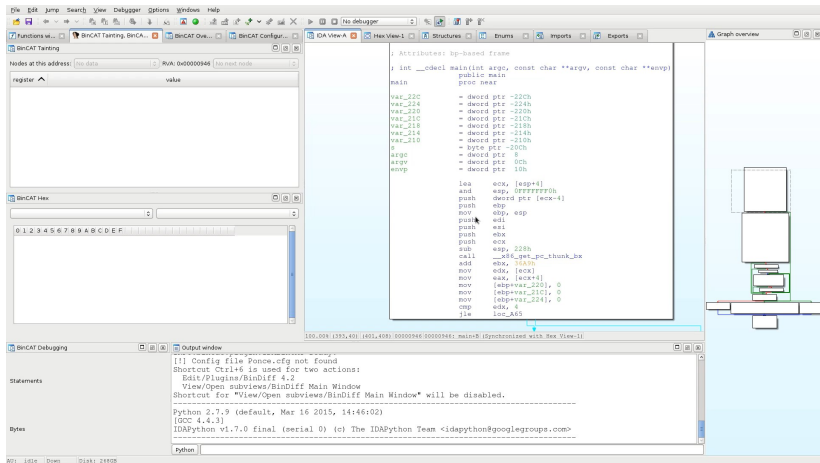
## Keygenme : principe



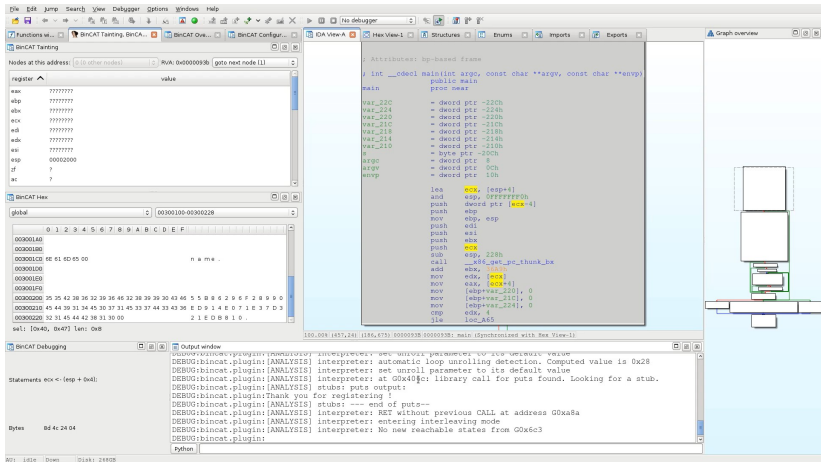
## Keygenme : principe



# Démo 1 : Utilisation de BinCAT



## Démo 2 : Teinte



# Plan

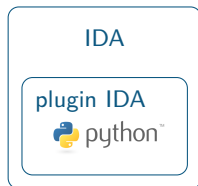
Introduction

Démonstration

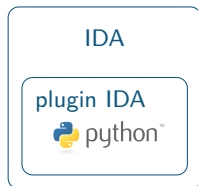
Sous le capot

Conclusion

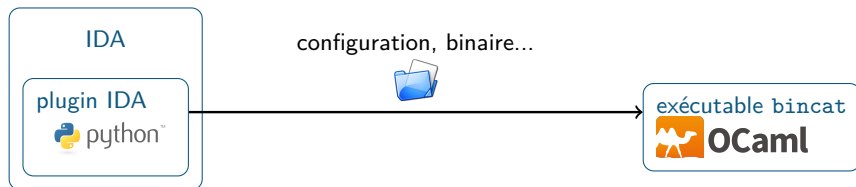
# Architecture



# Architecture

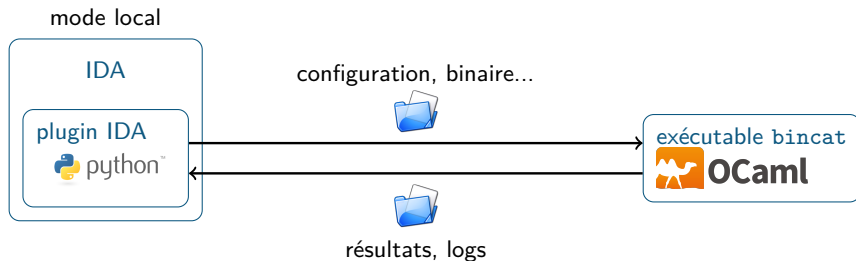


# Architecture

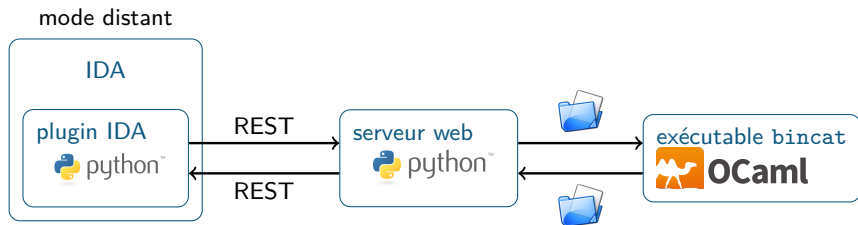




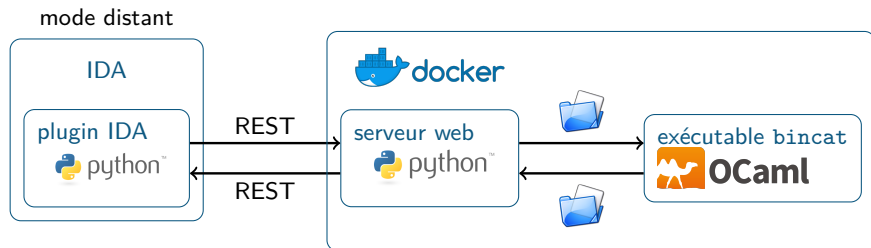
# Architecture



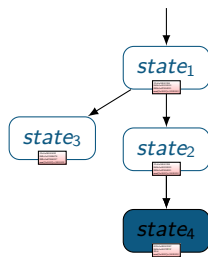
# Architecture



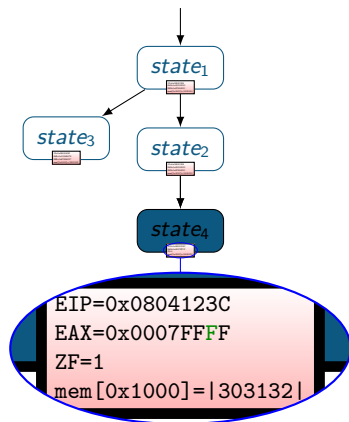
# Architecture



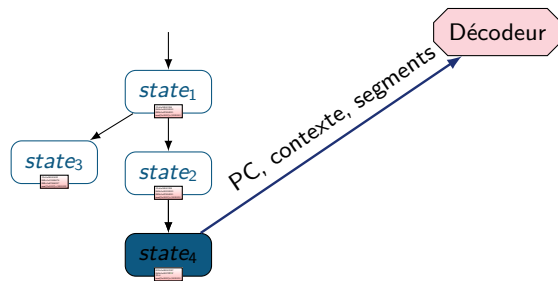
# Reconstruction du graphe de flot de contrôle



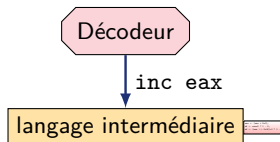
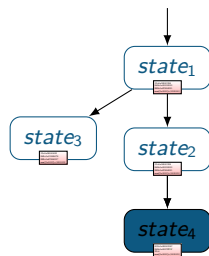
## Reconstruction du graphe de flot de contrôle



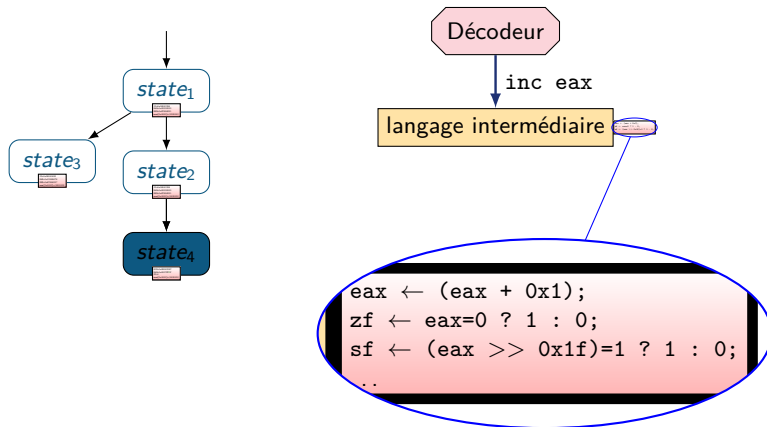
# Reconstruction du graphe de flot de contrôle



# Reconstruction du graphe de flot de contrôle

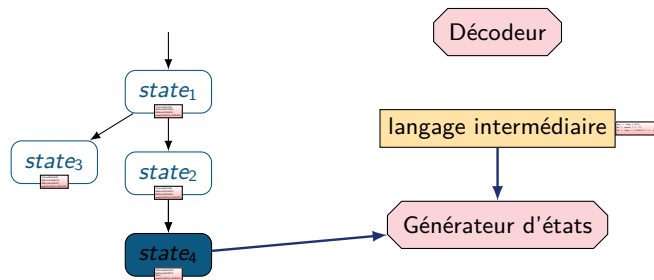


# Reconstruction du graphe de flot de contrôle

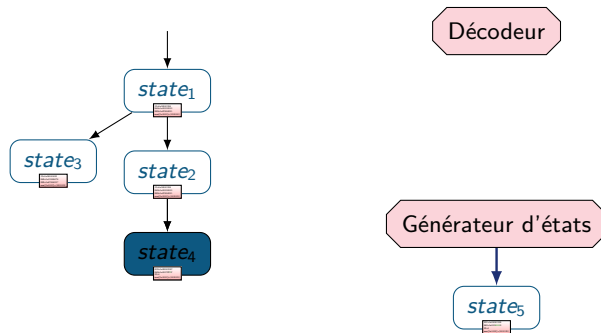




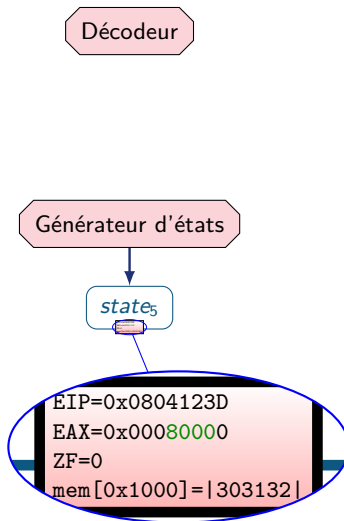
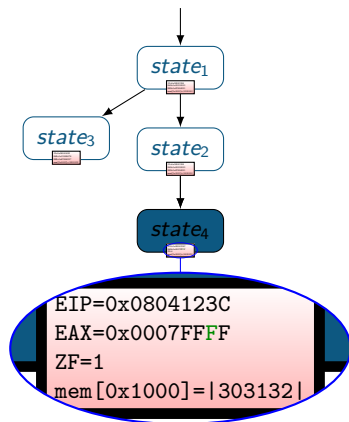
# Reconstruction du graphe de flot de contrôle



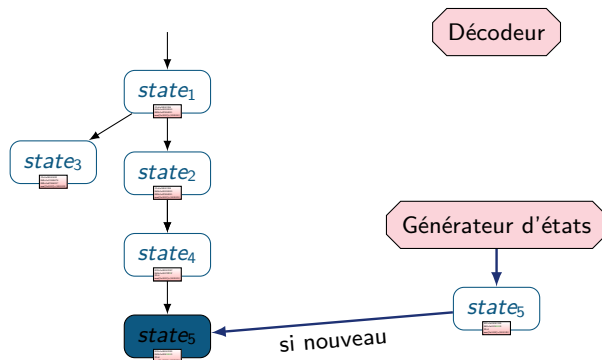
# Reconstruction du graphe de flot de contrôle



# Reconstruction du graphe de flot de contrôle



# Reconstruction du graphe de flot de contrôle

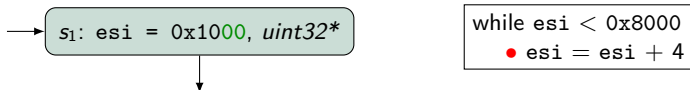


## Validation théorique : l'analyse statique par interprétation abstraite

- les opérations sur les valeurs/*taint*/type sont faites sur des objets abstraits qui représentent des ensembles de valeurs/*taint*/type  
ex :  $0 \equiv \{0\}$ ,  $? \equiv \{\text{entiers}\}$ ,  $\text{Struct} \equiv \{\text{structures C}\}$
- les calculs abstraits sont toujours une *surapproximation* du calcul réel
- exemple d'approximation : l'élargissement  $\nabla$  des boucles

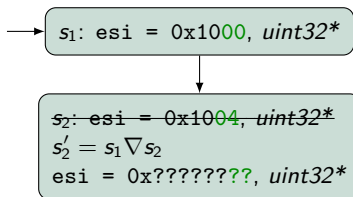
## Validation théorique : l'analyse statique par interprétation abstraite

- les opérations sur les valeurs/*taint*/type sont faites sur des objets abstraits qui représentent des ensembles de valeurs/*taint*/type  
ex :  $0 \equiv \{0\}$ ,  $? \equiv \{\text{entiers}\}$ ,  $\text{Struct} \equiv \{\text{structures C}\}$
- les calculs abstraits sont toujours une *surapproximation* du calcul réel
- exemple d'approximation : l'élargissement  $\nabla$  des boucles



## Validation théorique : l'analyse statique par interprétation abstraite

- les opérations sur les valeurs/*taint*/type sont faites sur des objets abstraits qui représentent des ensembles de valeurs/*taint*/type  
ex :  $0 \equiv \{0\}$ ,  $? \equiv \{\text{entiers}\}$ ,  $\text{Struct} \equiv \{\text{structures C}\}$
- les calculs abstraits sont toujours une *surapproximation* du calcul réel
- exemple d'approximation : l'élargissement  $\nabla$  des boucles

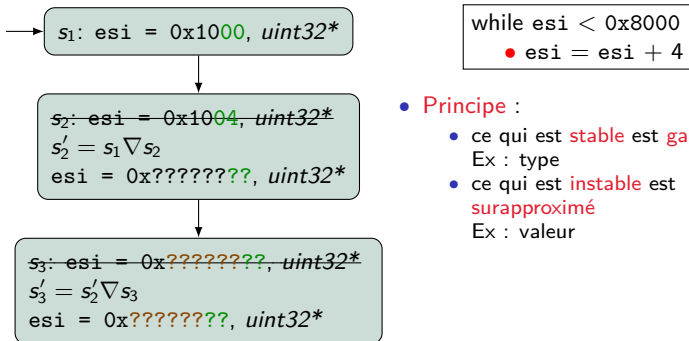


```
while esi < 0x8000
    esi = esi + 4
```

- Principe :**
  - ce qui est **stable** est **gardé**  
Ex : type
  - ce qui est **instable** est **surapproximé**  
Ex : valeur

## Validation théorique : l'analyse statique par interprétation abstraite

- les opérations sur les valeurs/*taint*/type sont faites sur des objets abstraits qui représentent des ensembles de valeurs/*taint*/type  
ex :  $0 \equiv \{0\}$ ,  $? \equiv \{\text{entiers}\}$ ,  $\text{Struct} \equiv \{\text{structures C}\}$
- les calculs abstraits sont toujours une *surapproximation* du calcul réel
- exemple d'approximation : l'élargissement  $\nabla$  des boucles



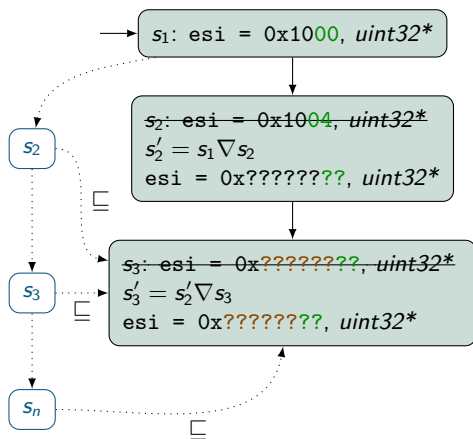
- Principe :

- ce qui est **stable** est **gardé**  
Ex : type
- ce qui est **instable** est **surapproximé**  
Ex : valeur



## Validation théorique : l'analyse statique par interprétation abstraite

- les opérations sur les valeurs/*taint*/type sont faites sur des objets abstraits qui représentent des ensembles de valeurs/*taint*/type  
ex :  $0 \equiv \{0\}$ ,  $? \equiv \{\text{entiers}\}$ ,  $\text{Struct} \equiv \{\text{structures } C\}$
- les calculs abstraits sont toujours une *surapproximation* du calcul réel
- exemple d'approximation : l'élargissement  $\nabla$  des boucles



while esi < 0x8000

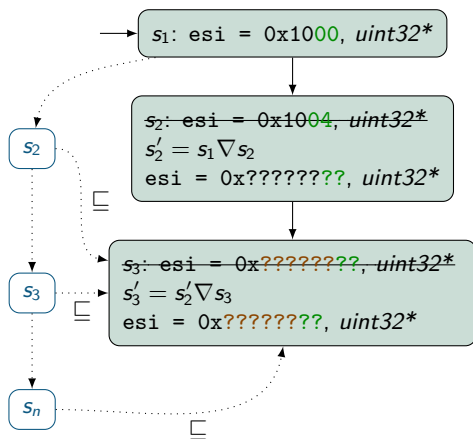
• esi = esi + 4

### Principe :

- ce qui est **stable** est **gardé**  
Ex : type
- ce qui est **instable** est **surapproximé**  
Ex : valeur
- Théorème 1** : la suite  $(s'_i)$  est ultimement stationnaire
- Théorème 2** : le point fixe  $s'_f$  est une surapproximation de la trace d'exécution réelle

## Validation théorique : l'analyse statique par interprétation abstraite

- les opérations sur les valeurs/*taint*/type sont faites sur des objets abstraits qui représentent des ensembles de valeurs/*taint*/type  
ex :  $0 \equiv \{0\}$ ,  $? \equiv \{\text{entiers}\}$ ,  $\text{Struct} \equiv \{\text{structures C}\}$
- les calculs abstraits sont toujours une *surapproximation* du calcul réel
- exemple d'approximation : l'élargissement  $\nabla$  des boucles



while  $\text{esi} < 0x8000$

•  $\text{esi} = \text{esi} + 4$

### Principe :

- ce qui est **stable** est **gardé**  
Ex : type
- ce qui est **instable** est **surapproximé**  
Ex : valeur
- Théorème 1** : la suite  $(s'_i)$  est ultimement stationnaire
- Théorème 2** : le point fixe  $s'_f$  est une surapproximation de la trace d'exécution réelle
- des techniques existent pour retrouver de la précision

- La théorie est correcte

# Validation empirique

- La théorie est correcte
- En théorie, l'implementation aussi

- La théorie est correcte
- En théorie, l'implémentation aussi
- En pratique, nombreuses sources de bug : décodeur, opérations abstraites, etc.

# Validation empirique

- La théorie est correcte
- En théorie, l'implementation aussi
- En pratique, nombreuses sources de bug : décodeur, opérations abstraites, etc.

⇒ nombreux tests unitaires

- La théorie est correcte
- En théorie, l'implementation aussi
- En pratique, nombreuses sources de bug : décodeur, opérations abstraites, etc.

⇒ nombreux tests unitaires

- tests BinCAT vs CPU :  $> 67.000$  tests sur  $\simeq 55$  instructions

- La théorie est correcte
- En théorie, l'implémentation aussi
- En pratique, nombreuses sources de bug : décodeur, opérations abstraites, etc.

⇒ nombreux tests unitaires

- tests BinCAT vs CPU :  $> 67.000$  tests sur  $\simeq 55$  instructions
- tests BinCAT vs tests QEMU : 87% des tests couverts sur  $\simeq 105$  instructions



- La théorie est correcte
- En théorie, l'implémentation aussi
- En pratique, nombreuses sources de bug : décodeur, opérations abstraites, etc.

⇒ nombreux tests unitaires

- tests BinCAT vs CPU :  $> 67.000$  tests sur  $\simeq 55$  instructions
- tests BinCAT vs tests QEMU : 87% des tests couverts sur  $\simeq 105$  instructions



# Performances de l'analyseur

Exemple : keygenme:

- 6407 instructions analysées
- espace mémoire : 90 Mo de mémoire vive
- temps d'analyse : 6 s
- moyenne:  $\simeq$  1060 instructions/s

Tests QEMU :

- 209 120 instructions analysées
- espace mémoire : 2,3 Go de mémoire vive
- temps d'analyse : 23 min 30 s
- moyenne:  $\simeq$  150 instructions/s

Intel Core i7-6700K CPU @ 4,00GHz

# Plan

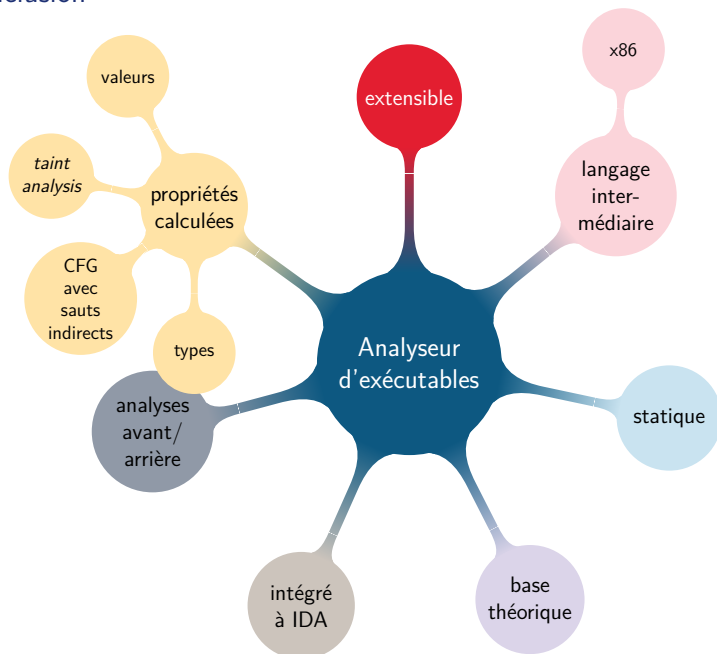
Introduction

Démonstration

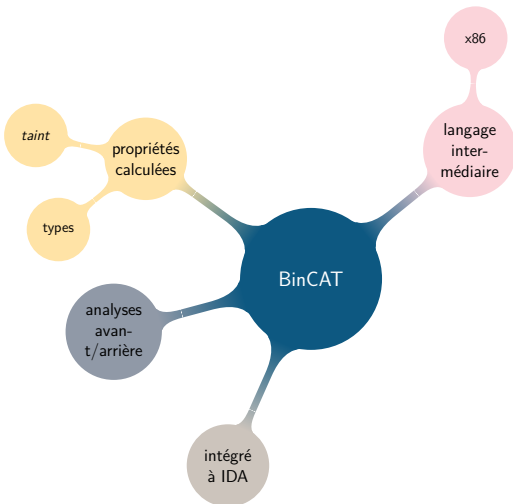
Sous le capot

Conclusion

## Conclusion

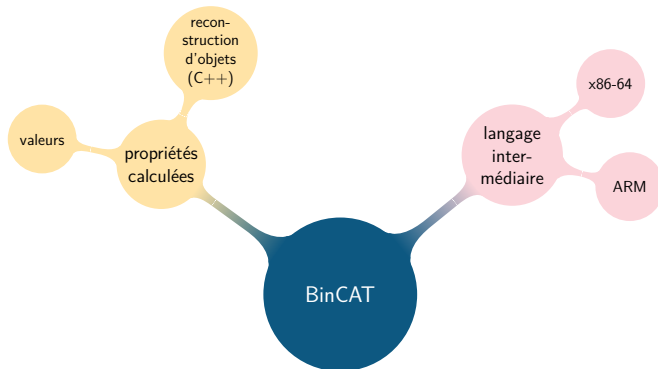


# Améliorations de l'existant



- reconstruction de types plus précise
  - insertion de nouveaux types à partir d'heuristiques. Ex : détection de structures sur la pile
- distinction de différentes sources de teinte
- raffinement des calculs lors d'une analyse arrière
- modélisation de fonctions de bibliothèques standard
- override de valeurs et de types dans IDA
- définitions mémoire dans IDA

## Futures fonctionnalités



- approximations moins grossières dans le calcul des valeurs : utilisation d'intervalles
- reconstruction d'objets complexes (C++)
- décodeurs x86-64 et ARM

# Merci !

- projet partiellement financé par la DGA MI
- sources disponibles (licence AGPL)

```
https://github.com/airbus-seclab/bincat  
docker run -p 5000:5000 airbusseclab/bincat
```

# Couverture x86

ADD			PUSH ES		POP ES		OR			PUSH CS		2 bytes																			
ADC			PUSH SS		POP SS		SBB			PUSH DS		POP DS																			
AND			ES:		DAA		SUB			CS:		DAS																			
XOR			SS:		AAA		CMP			DS:		AAS																			
INC			DEC																												
PUSH			POP																												
PUSHA		POPA		BOUND		ARPL		FS:		GS:		OPsize:		ADsize:		PUSH		IMUL		PUSH		IMUL		INSB		INSW		OUTSB		OUTSW	
JNO		JNO		JB		JNB		JZ		JNZ		JBE		JA		JS		JNS		JP		JNP		JL		JNL		JLE		JNLE	
Grp1		Grp1		Grp1		TEST		XCHG		MOV			LEA		MOV		POP														
NOP		XCHG		EAX		CWD		CDQ		CALL		WAIT		PUSHF		POPF		SAHF		LAHF											
MOV		EAX		MOVS		CMPS		TEST		STOS		LODS		SCAS																	
MOV			MOV																												
SHIFT		RETN		LES		LDS		MOV		ENTER		LEAVE		RETF		INT3		INT		INTO		IRETD									
Grp2		AAM		AAD		SALC		XLAT		FPU																					
LOOPNZ		LOOPZ		LOOP		JCXZ		IN		OUT		CALL		JMP		JMPF		JMPS		IN		OUT									
LOCK:		INT1		REPNE:		REP:		HLT		CMC		Grp3		CLC		STC		CLI		STI		CLD		STD		Grp4		Grp5			



# Couverture x86 - deuxième table

Grp6	Grp7	LAR	LSL		CLTS		INVD	WBINVD		UD2		NOP			
SSE							Prefix SSE	HINT NOP							
MOV CR DR								SSE							
WRMSR	RDTSC	RDMSR	RDPIC	SYSENTER	SYSEXIT		GETSEC SHX	MOVBE		SSE					
CMOV															
SSE															
MMX							SSE								
MMX SSE VMX									MMX SSE						
JNO	JNO	JB	JNB	JZ	JNZ	JBE	JA	JS	JNS	JP	JNP	JL	JNL	JLE	JNLE
SETNO	SETNO	SETB	SETNB	SETZ	SETNZ	SETBE	SETA	SETS	SETNS	SETP	SETNP	SETL	SETNL	SETLE	SETNLE
PUSH FS	POP FS	CPUID	BT	SHLD				PUSH GS	POP GS	RSM	BTS	SHRD	FENCE	IMUL	
CMPXCHG	LSS	BTR	LFS	LGS	MOVZX	POPCNT	UD	BTx	BTC	BSF	BSR	MOVXS			
XADD	SSE				CMPXCHG	BSWAP									
MMX							SSE								
MMX							SSE								
MMX							SSE								

# Treillis actuellement implémentés

