

BinCAT

Purrfecting binary static analysis

June 16th 2017 - REcon

Philippe Biondi, Raphaël Rigo, Sarah Zennou, Xavier Mehrenberger

Plan

Introduction

Demo

Under the hood

Conclusion


Plan

Introduction

Demo

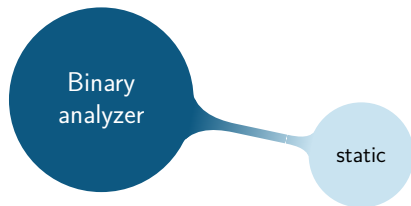
Under the hood

Conclusion

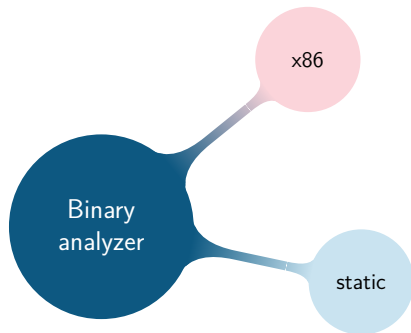


Binary
analyzer

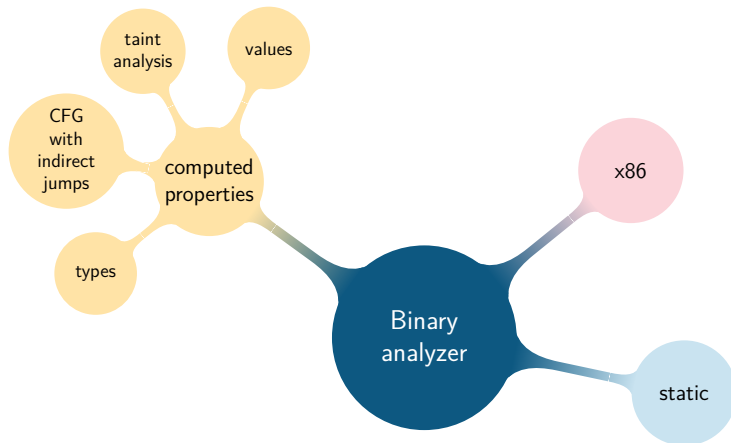
BinCAT (*Binary Code Analysis Toolkit*)



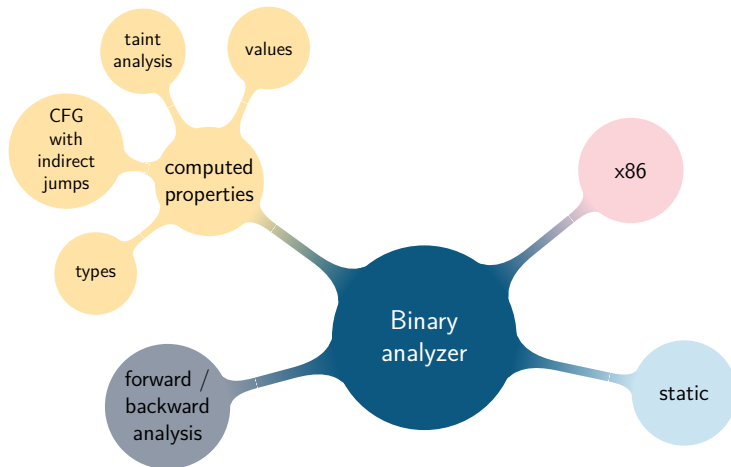
BinCAT (*Binary Code Analysis Toolkit*)



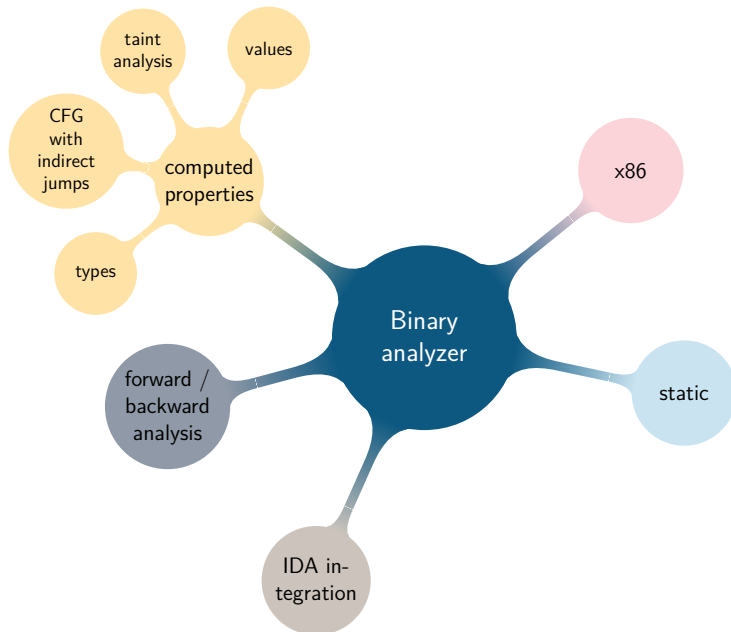
BinCAT (*Binary Code Analysis Toolkit*)



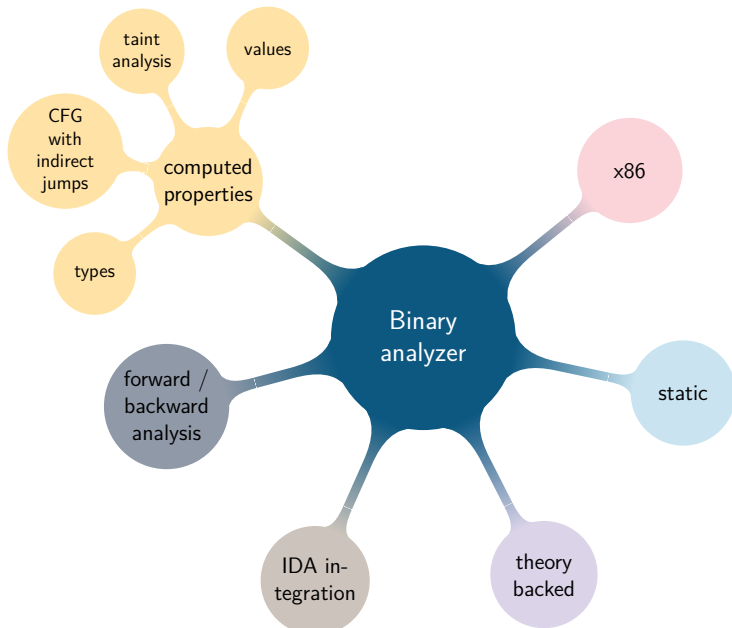
BinCAT (*Binary Code Analysis Toolkit*)



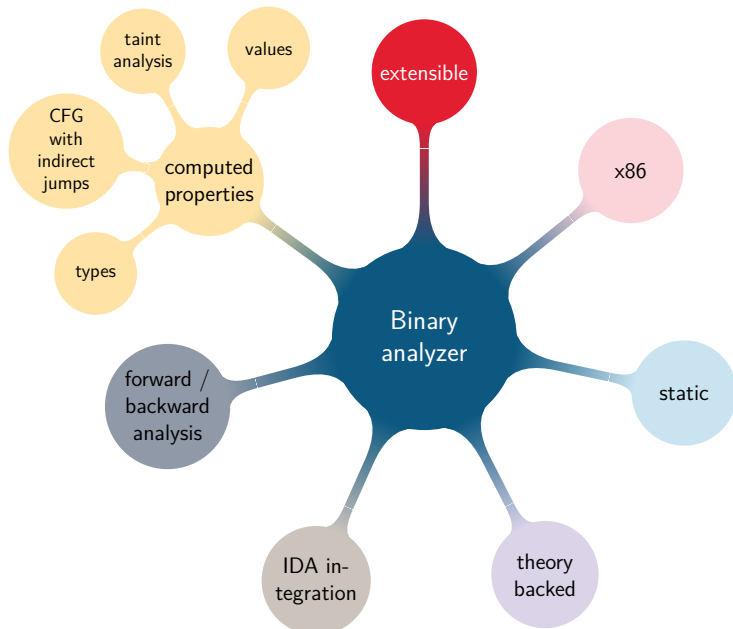
BinCAT (*Binary Code Analysis Toolkit*)



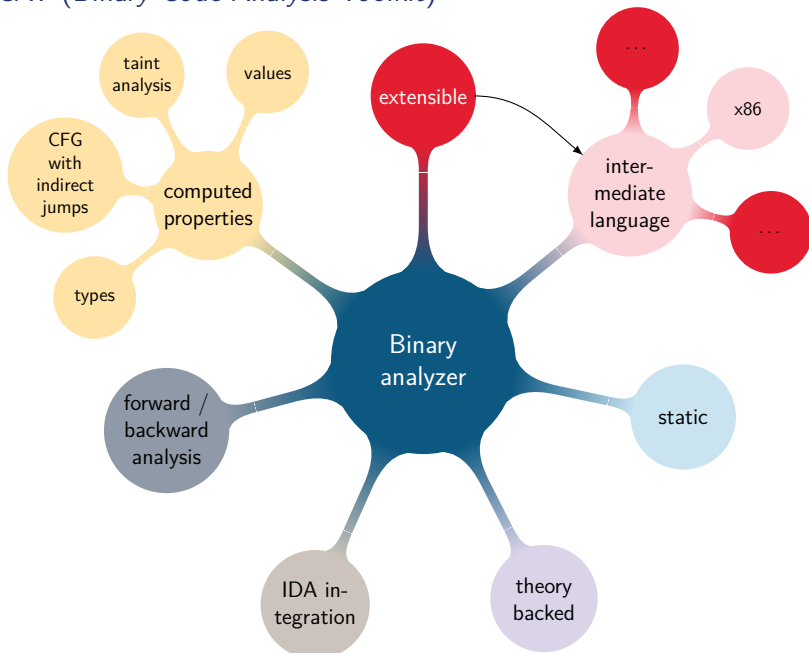
BinCAT (Binary Code Analysis Toolkit)



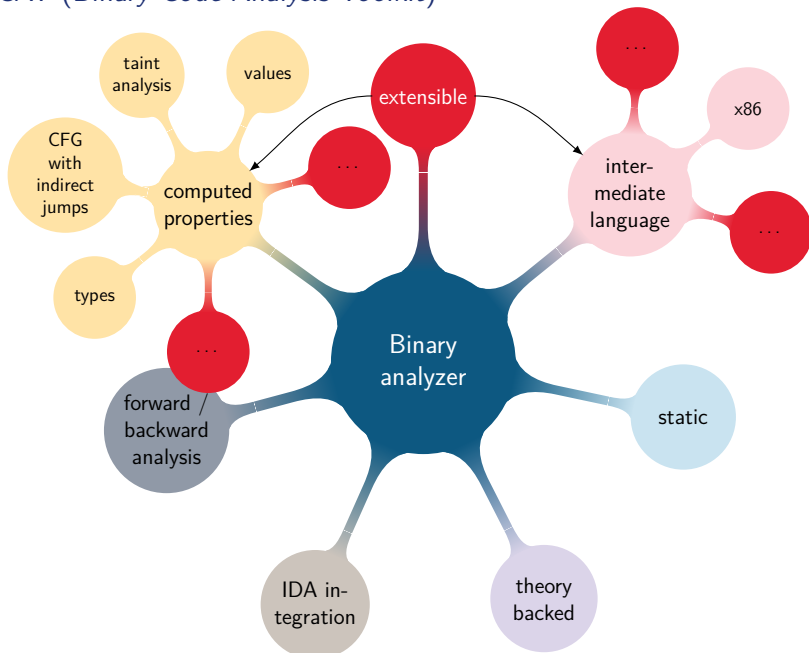
BinCAT (*Binary Code Analysis Toolkit*)



BinCAT (*Binary Code Analysis Toolkit*)



BinCAT (*Binary Code Analysis Toolkit*)



Plan

Introduction

Demo

Under the hood

Conclusion

Example: keygenme

```
$ ./get_key
```

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

Example: keygenme

```
$ ./get_key
```

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

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


Example: keygenme

```
$ ./get_key
```

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

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

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

```
Invalid serial wrong_serial
```

Example: keygenme

```
$ ./get_key
```

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

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

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

```
Invalid serial wrong_serial
```

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

Example: keygenme

```
$ ./get_key
```

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

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

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

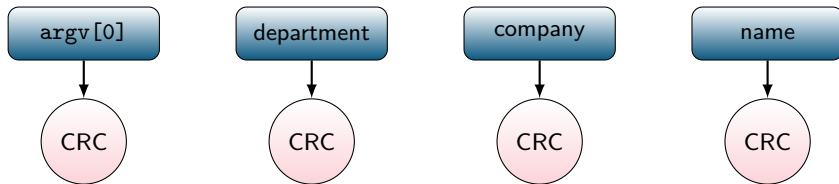
```
Invalid serial wrong_serial
```

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

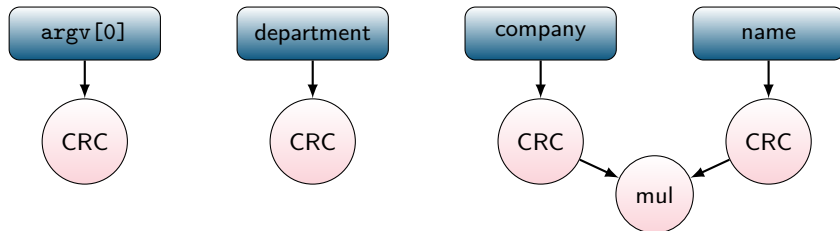
```
Licence=>[025E60CB08F00A1A23F236CC78FC819CE6590DD7]
```

```
Thank you for registering !
```

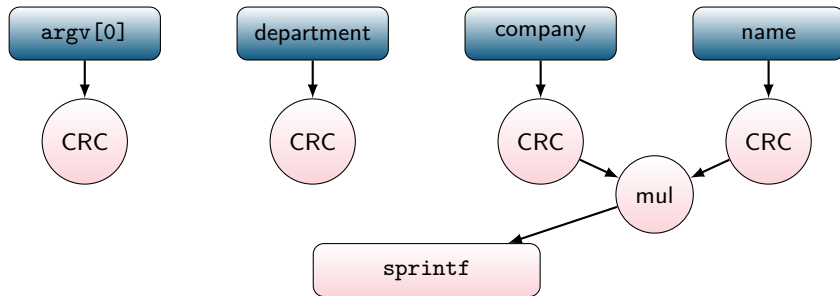
Keygenme: data flow



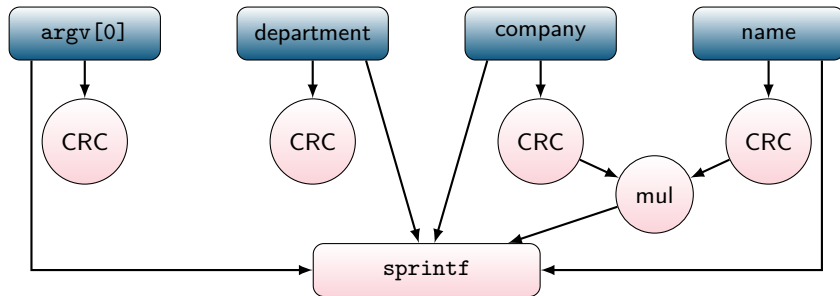
Keygenme: data flow



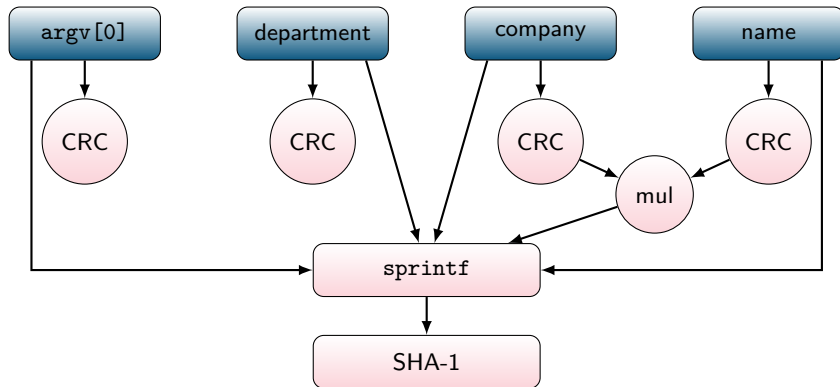
Keygenme: data flow



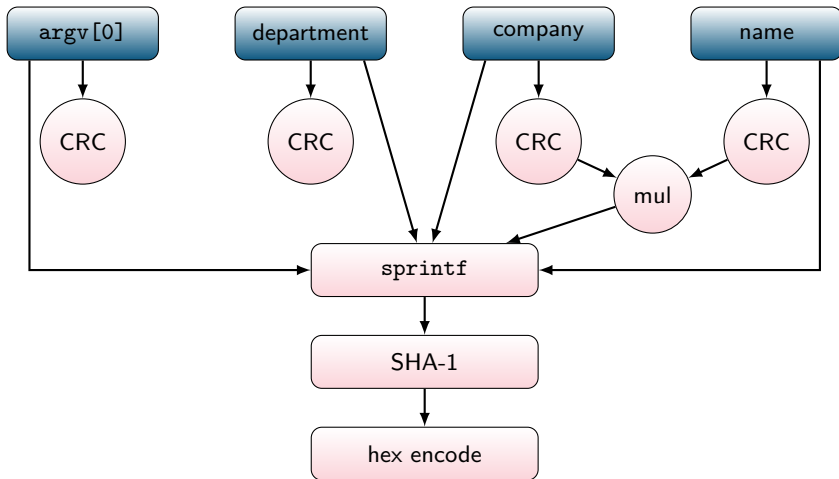
Keygenme: data flow



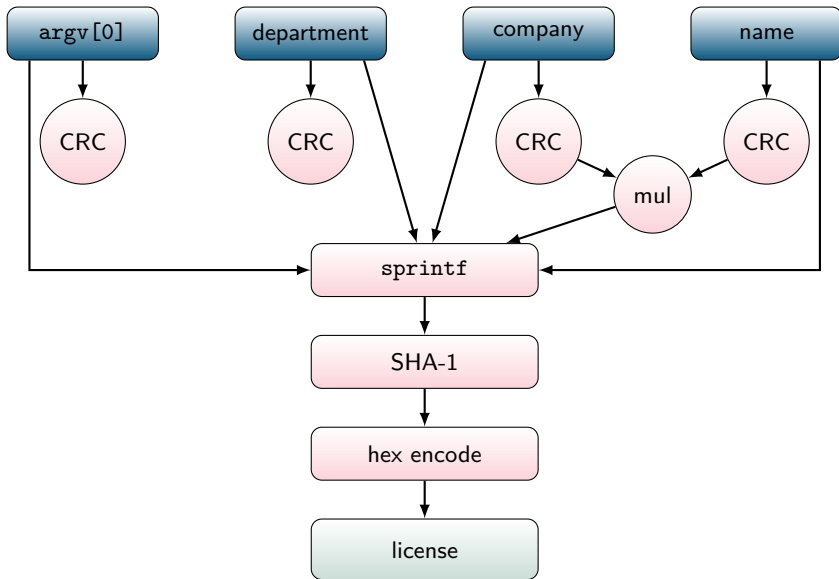
Keygenme: data flow



Keygenme: data flow



Keygenme: data flow



Demo 1: BinCAT usage

The screenshot shows the IDA Pro interface with the following components:

- Functions window:** Shows BinCAT Tainting, BinCAT Overview, and BinCAT Configuration.
- Register window:** Shows the register values for the current instruction.
- BinCAT Hex:** A hex dump window showing the memory contents of the instruction.
- Main window:** Displays assembly code for a function named 'main'. The code includes variable declarations and instructions like 'lea', 'and', 'push', 'mov', 'sub', 'call', 'add', 'mov', 'mov', 'mov', 'cmp', and 'jle'. A blue arrow points to the instruction 'jle loc_A65'.
- BinCAT Debugging:** Shows the output of a debugger, including statements and bytes.

```
Attributes: hp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
proc near

VAR_22C      = dword ptr -22Ch
VAR_224      = dword ptr -224h
VAR_220      = dword ptr -220h
VAR_21C      = dword ptr -21Ch
VAR_218      = dword ptr -218h
VAR_214      = dword ptr -214h
VAR_210      = dword ptr -210h
S            = byte ptr -20Ch
ARG0         = dword ptr  8
ARGV         = dword ptr 0Ch
ENVPP        = dword ptr 10h

lea     ecx, [esp+4]
and     esp, 0FFFFFFF0h
push   dword ptr [ecx-4]
push   ebp
mov     ebp, esp
push   esi
push   esi
push   ebx
push   ecx
sub     esp, 228h
call   __std_get_pc_thunk_bx
add     ebx, 36Ah
mov     edx, [ecx]
mov     eax, [ecx+4]
mov     [ebp+var_220], 0
mov     [ebp+var_21C], 0
mov     [ebp+var_224], 0
cmp     edx, 8
jle     loc_A65
```

Output window:

```
[!] Config file Ponce.cfg not found
Shortcut Ctrl+6 is used for two actions:
  Edit/Plugins/BinDiff 4.2
  View/Open subviews/BinDiff Main Window
Shortcut for "View/Open subviews/BinDiff Main Window" will be disabled.
-----
Python 2.7.9 (default, Mar 16 2015, 14:46:02)
[GCC 4.4.3]
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
-----
tython
```

Demo 2: Tainting

The screenshot displays the BinCAT IDE interface with several windows open:

- BinCAT Tainting:** Shows the current address (RVA: 00010093b) and the instruction "goto next node [I]".
- Register:** Lists registers and their values, such as eax: 77777777, ebp: 77777777, ebx: 77777777, etc.
- BinCAT Hex:** Shows a hex dump of memory starting at address 00300100-00300228. The hex data is: 00300100: 00 01 02 03 04 05 06 07 08 09 A B C D E F, 00300108: 00 00 00 00 00 00 00 00 00 00 00 00, 0030010C: 6E 61 6D 65 00 n a m e ., 00300110: 00 00 00 00, 00300100: 00 00 00 00, 003001F0: 00 00 00 00, 00300200: 25 35 42 38 36 32 39 36 46 32 38 39 39 30 43 46 5 5 8 8 6 2 9 6 F 2 8 9 9 0, 00300210: 45 44 39 31 34 45 30 37 31 45 33 37 44 33 43 36 E D 9 1 4 6 0 7 1 E 3 7 D 3, 00300220: 52 31 45 44 42 38 31 30 00 2 1 E 0 B 8 1 0., 00300228: 54: [0x40, 0x47] Len: 0x8
- Hex View:** Shows the disassembled assembly code for the function `int __cdecl main(int argc, const char **argv, const char **envp)`. The code includes variable declarations, stack frame setup, and a loop that prints the string "name".
- BinCAT Debugging:** Shows the output window with the following log:

```
DEBUG:binicat.plugin:[ANALYSIS] interpreter: set unroll parameter to its default value
DEBUG:binicat.plugin:[ANALYSIS] interpreter: set unroll parameter to its default value
DEBUG:binicat.plugin:[ANALYSIS] interpreter: at 00x4048: library call for puts found. Looking for a stub.
DEBUG:binicat.plugin:[ANALYSIS] stubs: puts output:
DEBUG:binicat.plugin:Thank you for registering !
DEBUG:binicat.plugin:[ANALYSIS] stubs: -- end of puts--
DEBUG:binicat.plugin:[ANALYSIS] interpreter: RET without previous CALL at address 00x40A8
DEBUG:binicat.plugin:[ANALYSIS] interpreter: entering interleaving mode
DEBUG:binicat.plugin:[ANALYSIS] interpreter: No new reachable states from 00x6C3
DEBUG:binicat.plugin:
```

Plan

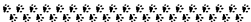
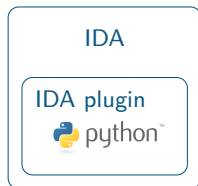
Introduction

Demo

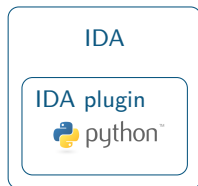
Under the hood

Conclusion

Architecture



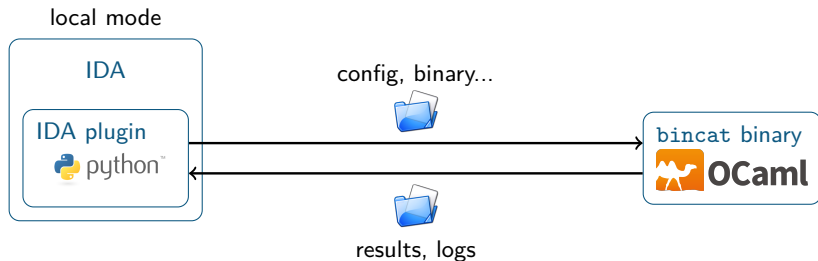
Architecture



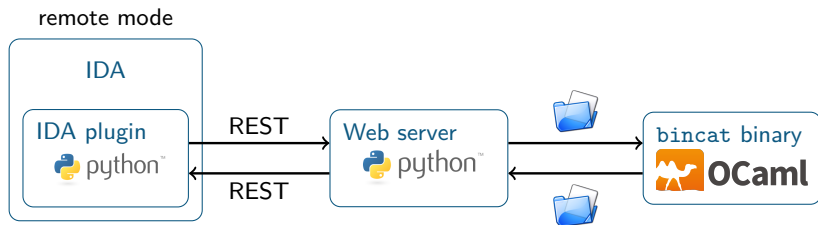
Architecture



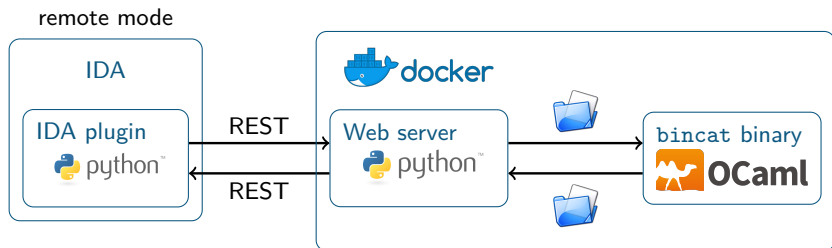
Architecture



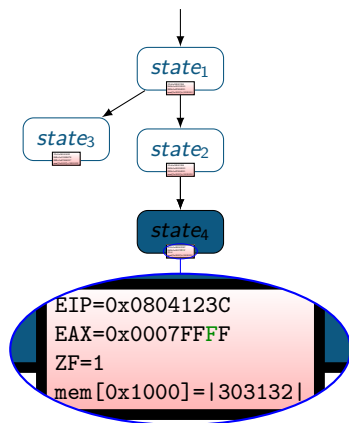
Architecture



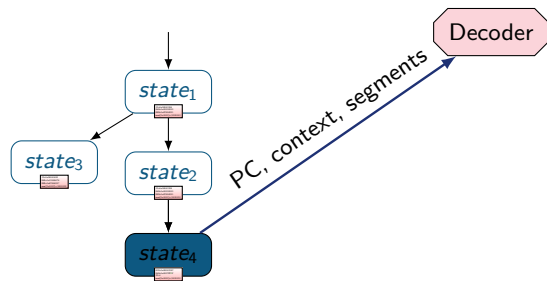
Architecture



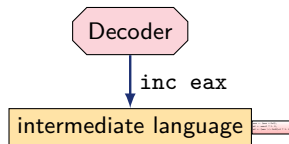
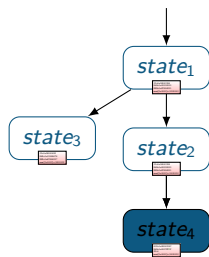
Control flow graph reconstruction



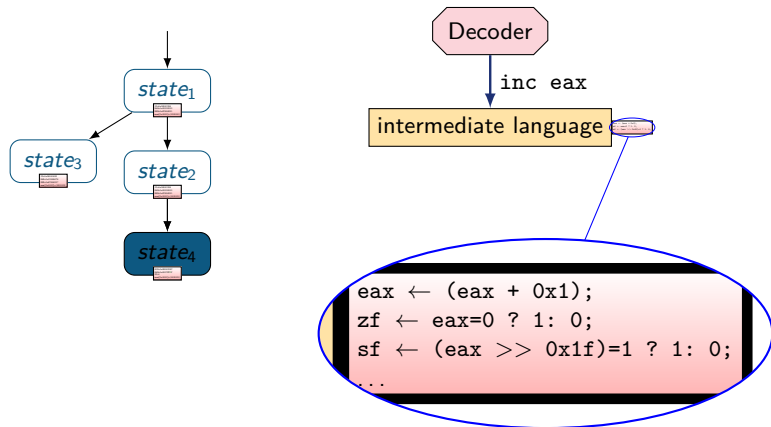
Control flow graph reconstruction



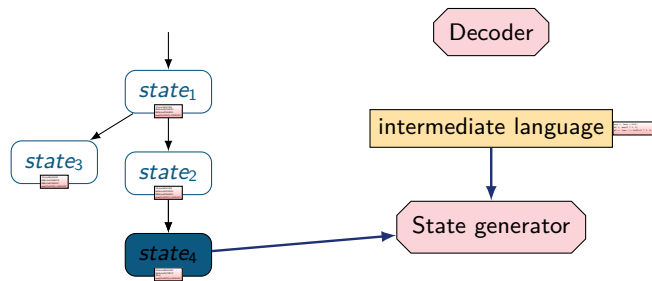
Control flow graph reconstruction



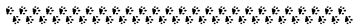
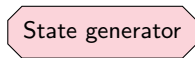
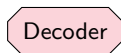
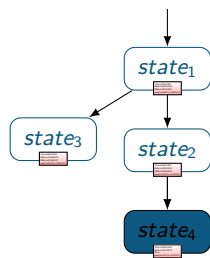
Control flow graph reconstruction



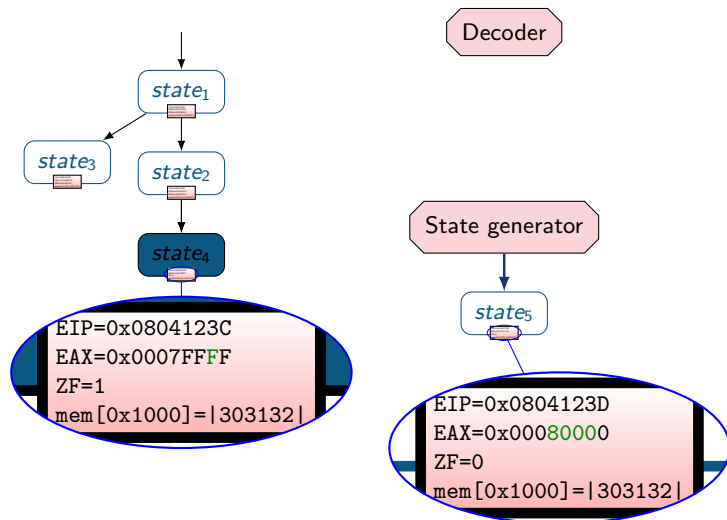
Control flow graph reconstruction



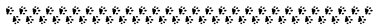
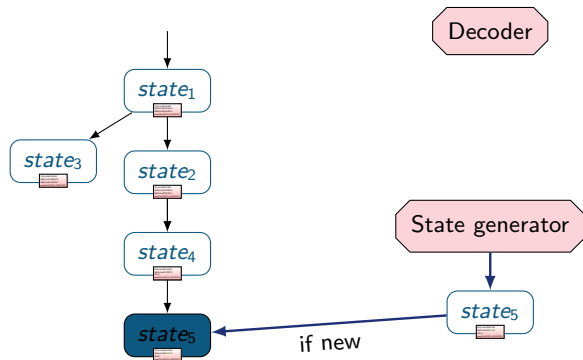
Control flow graph reconstruction



Control flow graph reconstruction



Control flow graph reconstruction

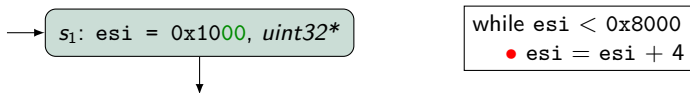


Formal correctness: static analysis by abstract interpretation

- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)

Formal correctness: static analysis by abstract interpretation

- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)



Formal correctness: static analysis by abstract interpretation

- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)

→ $s_1: \text{esi} = 0x1000, \text{uint32}^*$

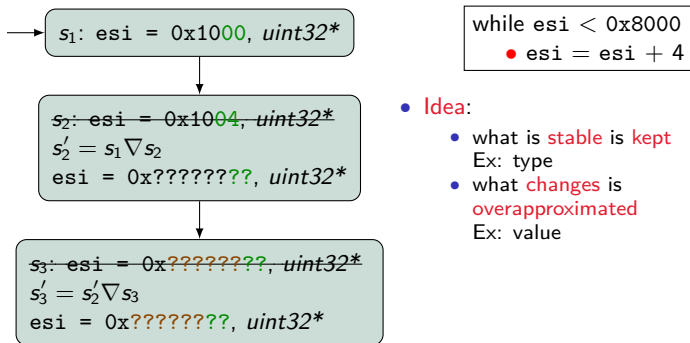
~~$s_2: \text{esi} = 0x1004, \text{uint32}^*$~~
 $s'_2 = s_1 \nabla s_2$
 $\text{esi} = 0x????????, \text{uint32}^*$

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

- **Idea:**
 - what is **stable** is **kept**
Ex: type
 - what **changes** is **overapproximated**
Ex: value

Formal correctness: static analysis by abstract interpretation

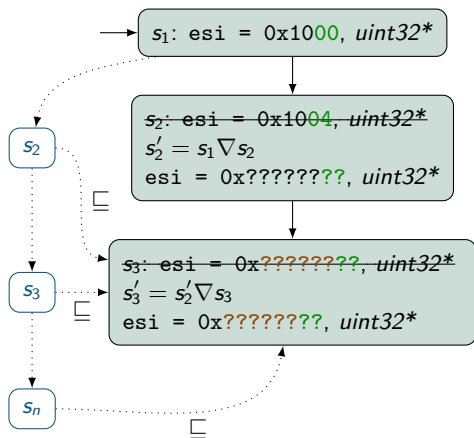
- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)



- Idea:**
 - what is **stable** is **kept**
Ex: type
 - what **changes** is **overapproximated**
Ex: value

Formal correctness: static analysis by abstract interpretation

- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)



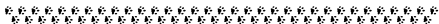
- Idea:**
 - what is **stable** is **kept**
Ex: type
 - what **changes** is **overapproximated**
Ex: value
- Theorem 1:** (s_i') sequence is ultimately stationary
- Theorem 2:** fixpoint s_f' is an overapproximation of the real execution trace

Empirical testing

- Theory is correct.

Empirical testing

- Theory is correct.
- *In theory*, the implementation too.



Empirical testing

- Theory is correct.
- *In theory*, the implementation too.
- In practice, a lot of things are complex and bug prone: the decoder, abstract operations, etc.

Empirical testing

- Theory is correct.
- *In theory*, the implementation too.
- In practice, a lot of things are complex and bug prone: the decoder, abstract operations, etc.

⇒ lots of unit tests

Empirical testing

- Theory is correct.
- *In theory*, the implementation too.
- In practice, a lot of things are complex and bug prone: the decoder, abstract operations, etc.

⇒ lots of unit tests

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



Empirical testing

- Theory is correct.
- *In theory*, the implementation too.
- In practice, a lot of things are complex and bug prone: the decoder, abstract operations, etc.

⇒ lots of unit tests

- BinCAT vs CPU: > 67.000 tests for $\simeq 55$ instructions
- BinCAT vs QEMU test-i386: 87% test coverage over $\simeq 105$ instructions

Empirical testing

- Theory is correct.
- *In theory*, the implementation too.
- In practice, a lot of things are complex and bug prone: the decoder, abstract operations, etc.

⇒ lots of unit tests

- BinCAT vs CPU: > 67.000 tests for \simeq 55 instructions
- BinCAT vs QEMU test-i386: 87% test coverage over \simeq 105 instructions



Analyzer's performance

Example: keygenme

- 6407 instructions analyzed
- RAM usage: 90 MiB
- running time: 6s
- average: $\simeq 1060$ insn/s

QEMU tests:

- 209 120 instructions analyzed
- RAM usage: 2.3 GiB
- running time: 23 min 30 s
- average: $\simeq 150$ insn/s

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

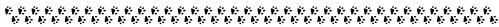
Plan

Introduction

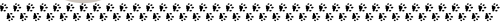
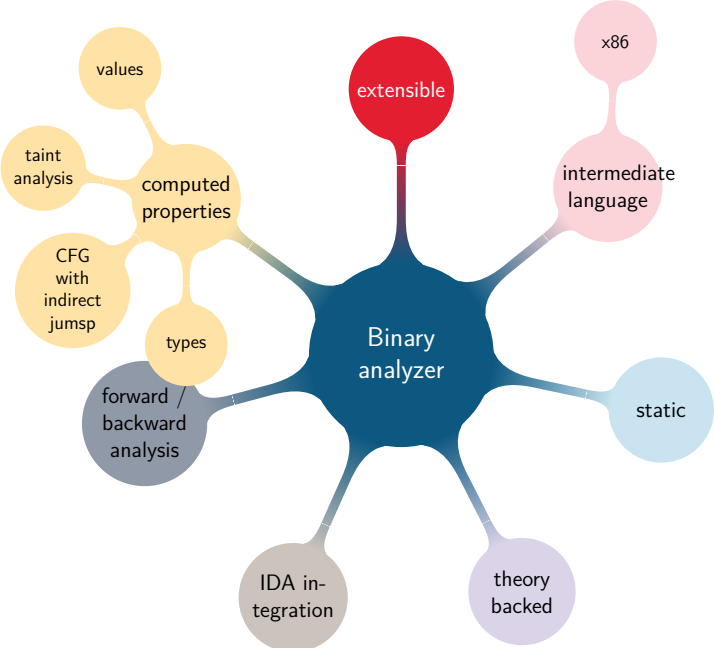
Demo

Under the hood

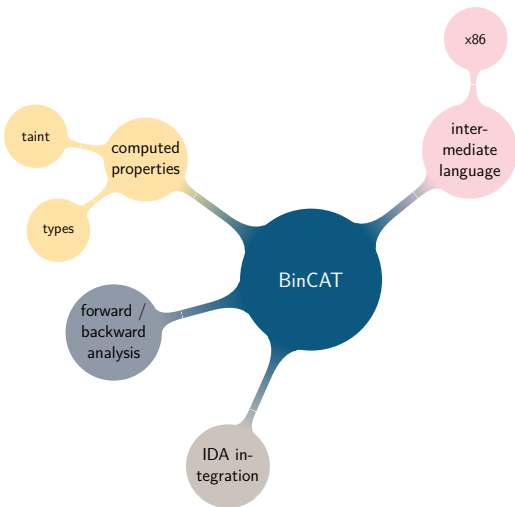
Conclusion



Conclusion

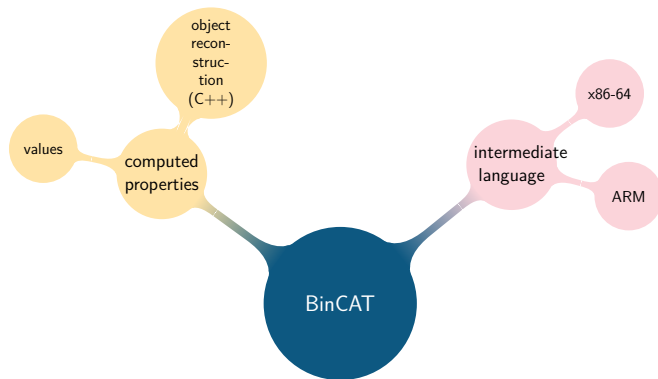


Current features improvements (planned)



- better type reconstruction
 - new types from heuristics. Ex: structures detection on stack
- several distinct taint sources
- more precise computations in backward analysis
- more standard library functions models
- type and value override in IDA
- memory definition directly in IDA

Future features



- finer approximations in values computation by using intervals
- complex objects reconstruction (C++)
- x86-64 and ARM decoders

Thanks!

Full paper (link in README):

https://www.sstic.org/media/SSTIC2017/SSTIC-actes/bincat_purrfecting_binary_static_analysis/SSTIC2017-Article-binocat_purrfecting_binary_static_analysis-biondi_rigo_zennou_mehrenberger.pdf

- project was partially financed by DGA-MI
- Get it! (AGPL licence)

<https://github.com/airbus-seclab/binocat>

```
docker run -p 5000:5000 airbusseclab/binocat
```

tutorial in doc/tutorial.md



x86 coverage

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															
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	

x86 coverage - Second table



Currently implemented lattices

