FROM RESEARCH TO INDUSTRY

cea tech

list

BINSEC/SE: A Dynamic Symbolic Execution Toolkit for Binary-level Analysis

Robin David
Sébastien Bardin
Thanh Dinh Ta
Josselin Feist
Laurent Mounier
Marie-Laure Potet
Jean-Yves Marion

SANER 2016, Osaka, Japan, March 16th

digiteo

INSTITUT CARNOT CEA LIST

Outline

Introduction

Dynamic Symbolic Execution

BINSEC/SE

Demo

Introduction

The need to reverse engineer an excutable : **malware, bug discovery, safety, testing** ..

Introduction

The need to reverse engineer an excutable : **malware, bug discovery, safety, testing** ..

Current approaches and limitations for binary-level understanding :

- **Static :**
  - allow to choose any path **[but not necessarily feasible]**
  - Easy to fool $\rightarrow$ indirect jumps, self-modification etc.

Introduction

The need to reverse engineer an excutable : **malware, bug discovery, safety, testing** ..

Current approaches and limitations for binary-level understanding :

- **Static :**
  - allow to choose any path **[but not necessarily feasible]**
  - Easy to fool $\rightarrow$ indirect jumps, self-modification etc.
- **Dynamic :**
  - only doable paths **[but depend on inputs]**
  - problem $\rightarrow$ possibly miss a lot of code areas

Introduction

The need to reverse engineer an excutable : **malware, bug discovery, safety, testing** ..

Current approaches and limitations for binary-level understanding :

- **Static :**
  - allow to choose any path **[but not necessarily feasible]**
  - Easy to fool → indirect jumps, self-modification etc.
- **Dynamic :**
  - only doable paths **[but depend on inputs]**
  - problem → possibly miss a lot of code areas

Symbolic : best of both world

- only doable paths
- can recover new paths **[regardless of path rarity]**

Running examples

Various problems occurs when trying to cover program paths :

Dynamic jumps

```
mov eax, var_x
shl eax, 2
add eax, off_y
mov eax, [eax]
jmp eax
```

Running examples

Various problems occurs when trying to cover program paths :

Dynamic jumps

```
mov eax, var_x
shl eax, 2
add eax, off_y
mov eax, [eax]
mov edx, eax
mov eax, edx
jmp eax
```

Heuristics limitations

IDA Pro 6.9 fooled by such trick..

Running examples

Various problems occurs when trying to cover program paths :

Dynamic jumps

```
mov eax, var_x
shl eax, 2
add eax, off_y
mov eax, [eax]
mov edx, eax
mov eax, edx
jmp eax
```

Call/Ret

```
1004002 : call 0x100400a
1004007 : (junk byte)
1004008 :
100400a : pop ebp
100400b : inc ebp
100400c : push ebp
100400d : ret
100400e : ...
```

Heuristics limitations

IDA Pro 6.9 fooled by such trick..

Running examples

Various problems occurs when trying to cover program paths :

Dynamic jumps

```
mov eax, var_x
shl eax, 2
add eax, off_y
mov eax, [eax]
mov edx, eax
mov eax, edx
jmp eax
```

Call/Ret

```
1004002 : call 0x100400a
1004007 : (junk byte)
1004008 : jmp 0x100400e
100400a : pop ebp
100400b : inc ebp
100400c : push ebp
100400d : ret
100400e : ...
```

Heuristics limitations

IDA Pro 6.9 fooled by such trick..

Running examples

Various problems occurs when trying to cover program paths :

Dynamic jumps

```
mov eax, var_x
shl eax, 2
add eax, off_y
mov eax, [eax]
mov edx, eax
mov eax, edx
jmp eax
```

Call/Ret

```
1004002 : call 0x100400a
1004007 : (junk byte)
1004008 : jmp 0x100400e
100400a : pop ebp
100400b : inc ebp
100400c : push ebp
100400d : ret
100400e : ...
```

Heuristics limitations

IDA Pro 6.9 fooled by such trick..

Heuristics limitations

Common disassemblers does not disassemble after unknown byte and ret instructions

Running examples

Various problems occurs when trying to cover program paths :

Dynamic jumps

```
mov eax, var_x
shl eax, 2
add eax, off_y
mov eax, [eax]
mov edx, eax
mov eax, edx
jmp eax
```

Call/Ret

```
1004002 : call 0x100400a
1004007 : (junk byte)
1004008 : jmp 0x100400e
100400a : pop ebp
100400b : inc ebp
100400c : push ebp
100400d : ret
100400e : ...
```

Heuristics limitations

IDA Pro 6.9 fooled by such trick..

Heuristics limitations

Common disassemblers does not disassemble after unknown byte and ret instructions

And many others..

Outline

Introduction

Dynamic Symbolic Execution

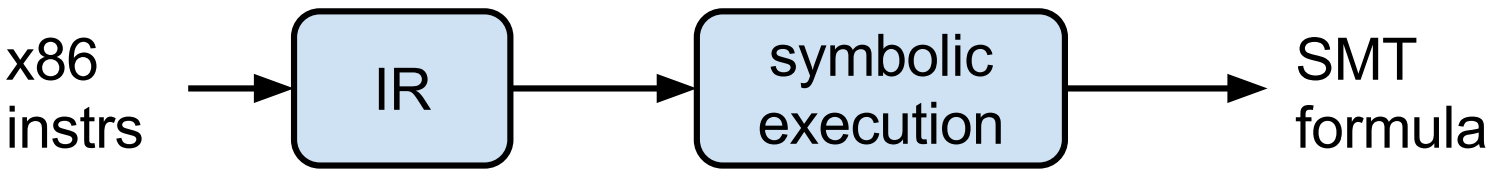
BINSEC/SE

Demo

DSE : In brief

Definition

Symbolic execution is the mean of executing a program using symbolic values (logical symbols) rather than actual values (bitvectors) in order to obtain in-out relationship of a path.



Dynamic Symbolic Execution [DSE] :

- precise reasoning on a single path
- sound execution of the program *(path necessarily feasible)*
- can recover new paths (goto eax, call/ret, etc.)
- thwart basic tricks *(code overlapping..)*

Outline

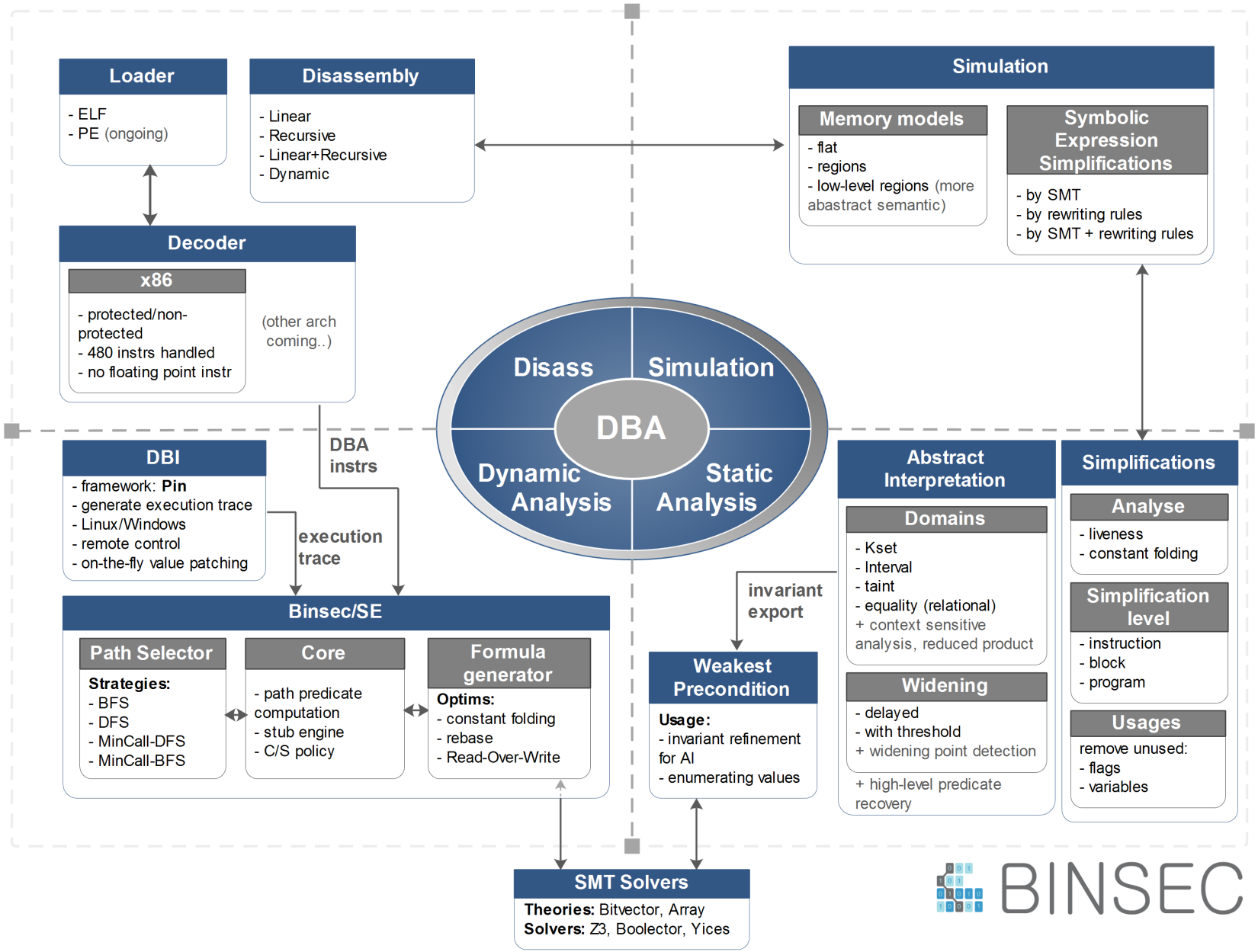
Introduction

Dynamic Symbolic Execution

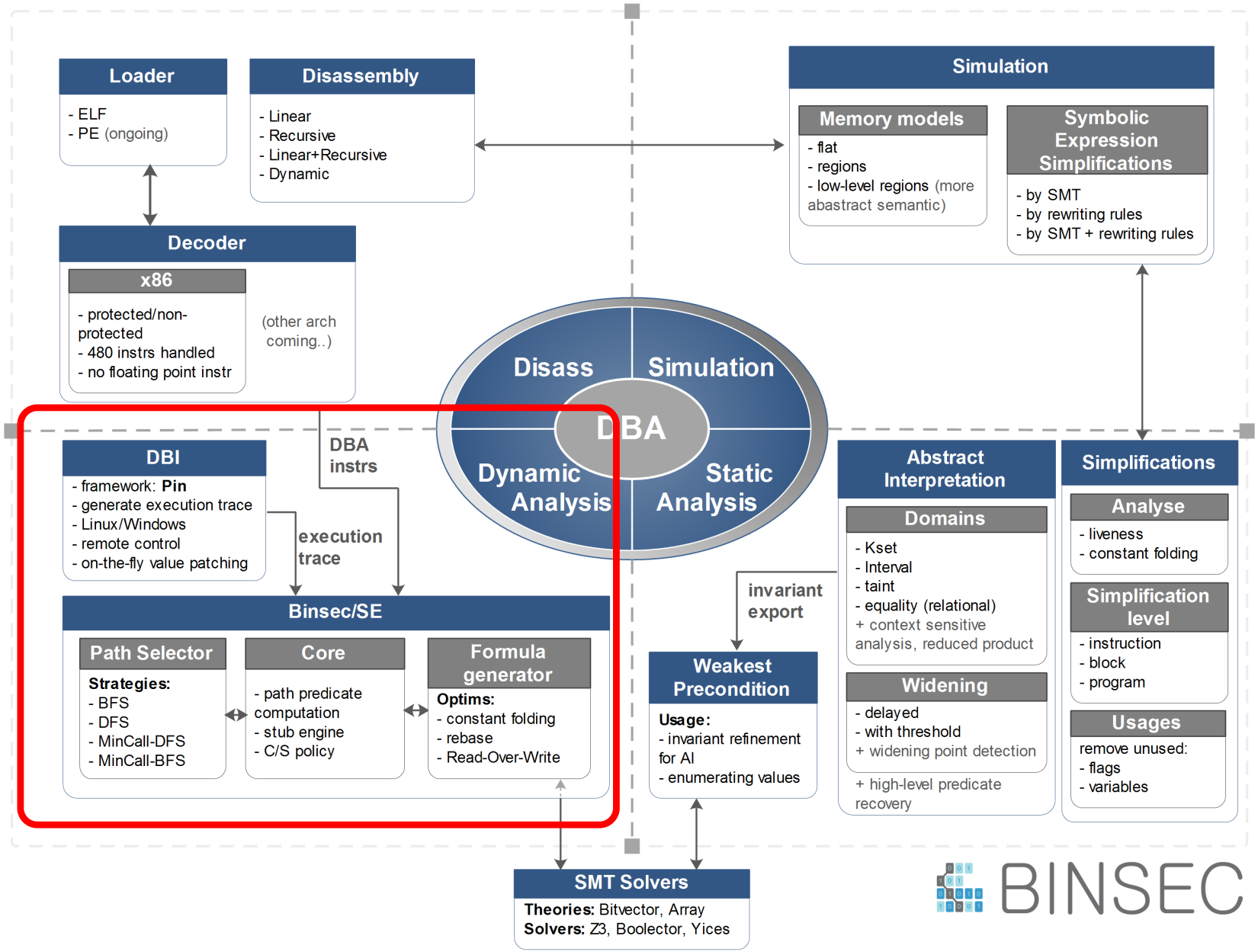
BINSEC/SE

Demo

Binsec : Global overview



Binsec : Global overview



Binsec/SE : In depth

Tracing (Pin)

- gather certain library calls concrete infos
- **arbitrary value retrieval** *(registers/memory)*
- **On-the-fly value patching**
- Linux/Windows
- **Remote control**

Core (10K OCaml loc)

- **stub engine for library calls**
- **generic path selection**
- **path predicate optimization :**
- handle JSON conf. files
- Solvers : Z3, boolector, ..

Outline

Introduction

Dynamic Symbolic Execution

BINSEC/SE

Demo

Demo : Call/Ret violation

Example code obfuscated by the ASPack packer :

```
1 1004002 e8 03 00 00 00 call 0x100400a //push 0x1004007 as return
2 100400a 5d             pop ebp        //pop return address in ebp
3 100400b 45             inc ebp        //increment ebp
4 100400c 55             push ebp       //push back the value
5 100400d c3             ret            //return on 0x1004008
6 1004008 eb 04          jmp 0x100400e
```

→ **Fool the disassembler (which works here).**

(Goal : Trying to find the violations with DSE)

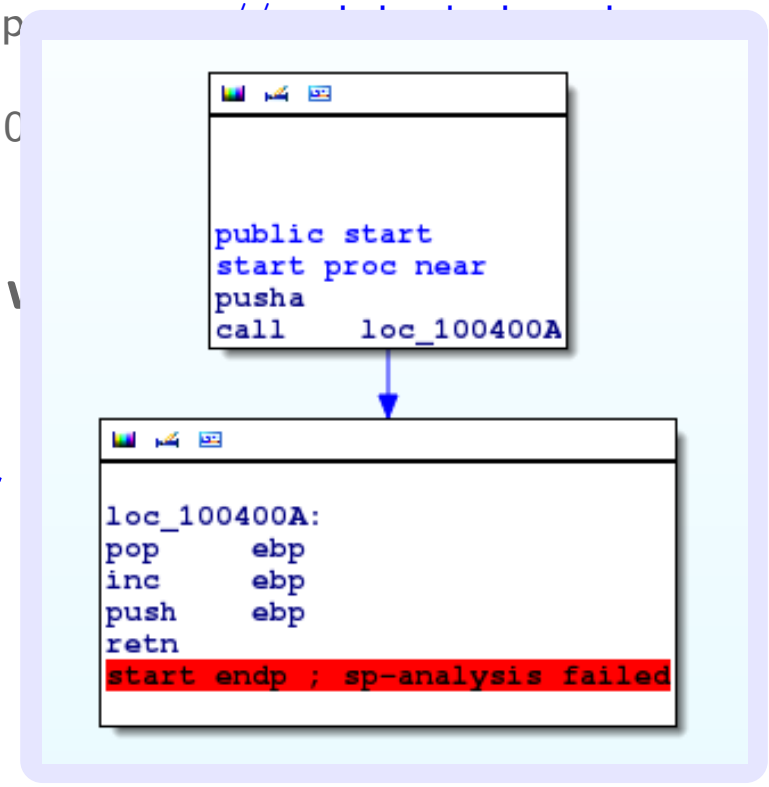
Demo : Call/Ret violation

Example code obfuscated by the ASPack packer :

```
1 1004002 e8 03 00 00 00 call 0x100400a //push 0x1004007 as return
2 100400a 5d             pop ebp        //pop return address in ebp
3 100400b 45             inc ebp        //increment ebp
4 100400c 55             push ebp
5 100400d c3             ret
6 1004008 eb 04          jmp 0x10
```

→ **Fool the disassembler (which v**

(Goal : Trying to find the violations



Thank you !

ありがとうございます

Direction de la Recherche Technologique
Département d'Ingénierie des Logiciels et des Systèmes
Laboratoire de Sûreté des Logiciels

Commissariat à l'énergie atomique et aux énergies alternatives
Institut Carnot CEA LIST
Centre de Saclay — 91191 Gif-sur-Yvette Cedex
Etablissement public à caractère industriel et commercial — RCS Paris B 775 685 019