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
Outline

Introduction

Dynamic Symbolic Execution

BINSEC/SE

Demo
The need to reverse engineer an executable: malware, bug discovery, safety, testing...
The need to reverse engineer an executable: **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.
The need to reverse engineer an executable: **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
The need to reverse engineer an executable: **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 worlds

- only doable paths
- can recover new paths [*regardless of path rarity*]
Various problems occur when trying to cover program paths:

**Dynamic jumps**

```plaintext
mov eax, var_x
shl eax, 2
add eax, off_y
mov eax, [eax]
jmp eax
```
Various problems occurs when trying to cover program paths:

**Dynamic jumps**

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

```assembly
mov edx, eax
mov eax, edx
jmp eax
```

**Heuristics limitations**

IDA Pro 6.9 fooled by such trick.
Various problems occur 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.
Various problems occur 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..**
Various problems occur 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.

- Common disassemblers do not disassemble after unknown byte and ret instructions.
Various problems occur 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...

And many others...

**Heuristics limitations**

Common disassemblers do not disassemble after unknown byte and ret instructions
Outline

Introduction

Dynamic Symbolic Execution

BINSEC/SE

Demo
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..)
Introduction

Dynamic Symbolic Execution

**BINSEC/SE**

Demo
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, ..
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  //push back the value
5 100400d c3 ret  //return on 0x1004008
6 1004008 eb 04 jmp 0x100400e

→ Fool the disassembler (which we are trying to find the violations)

(Goal : Trying to find the violations)
Thank you!

ありがとうございます