DE LA RECHERCHE À L'INDUSTRIE

Binsec: a platform for binary code analysis

08/06/2016

Adel Djoudi
Robin David
Josselin Feist
Thanh Dinh Ta

www.cea.fr

Outline

Binary code analysis : Why ?

Source program

Ada
C/C++
Java
Perl
Python
...

Compiler →

Binary program

0010110101011001011011
1000101011010100101010
0101101010101100110101
0010110101011000010101
1000101011010100110101
.....

Source analysis (✓, •, ×)

Binary analysis (✓, •, ×)

- Analysis without access to source code
  - Proprietary software
  - Analysis of malware
- Alternative to source code analysis
  - Compiler independent !
  - Multi-languages programs

Binary code analysis : Why ?

Source program

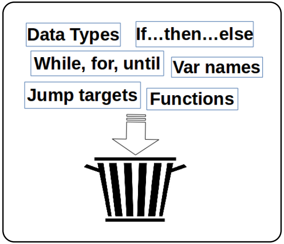
Ada
C/C++
Java
Perl
Python
...

Compiler →

Binary program

00101101010110011011
10001010110101001010
01011010101011001101
00101101010110000101
10001010110101001101
.....

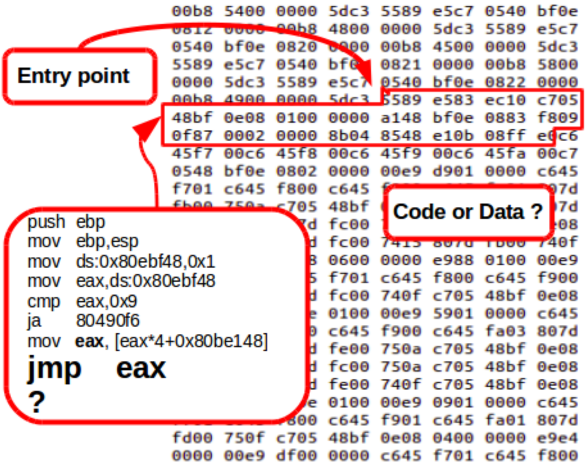
Source analysis (✓, •, ×)



Binary analysis (✓, •, ×)

- Analysis without access to source code
  - Proprietary software
  - Analysis of malware
- Alternative to source code analysis
  - Compiler independent !
  - Multi-languages programs

Challenges of binary code analysis (1)



Challenges of binary code analysis (2)

- Low-level semantics of data
  - Machine arithmetic, bit-level operations
  - Systematic usage of untyped memory [big array]

  Difficult for current formal techniques

- Low-level semantics of control
  - No clear distinction data/instructions
  - Dynamic jumps (`jump eax`)

  No easy syntactic recovery of CFG

- Diversity of architectures and instruction sets
  - Too many instructions (ex. X86, $\geq$ 900 instructions)
  - Modeling issues : side effect, addressing mode, ...

  No platform independent concise formalism

Nice progress since 2004

Intermediate languages

REIL [Zynamics]
BIL [CMU]
DBA [CEA, LaBRI]
RREIL [TUM] ...

CFG recovery

CodeSurfer/x86 [GrammaTech]
Jakstab [TU München]
CFGBuilder [CEA]
...

Tests generation

SAGE [Microsoft]
OSMOSE [CEA]
Mayhem [ForAllSecure]
...

Binary analysis in Binsec Platform

BinSec : binary analysis platform with four main services :

- Front-end [loader, decoder, disassembly, simplifications]
- Simulator [concrete interpretation]
- Generic static analyzer [fixpoint loop, CFG recovery]
- DSE [flexible : C/S (concretisation/symbolisation), path search]

Novelties :

- DBA Intermediate Representation
- Simplification engine of DBA
- Static binary analysis with (source-level) features
- Flexible DSE

DBA (IR)

9 Instructions

```
lhs := e;
goto e; < call, return >
goto bv; < call, return >
ite (e)? goto bv : goto bv
stop;
lhs := nondet();
lhs := undef;
assert (cond);
assume (cond);
```

Advantages

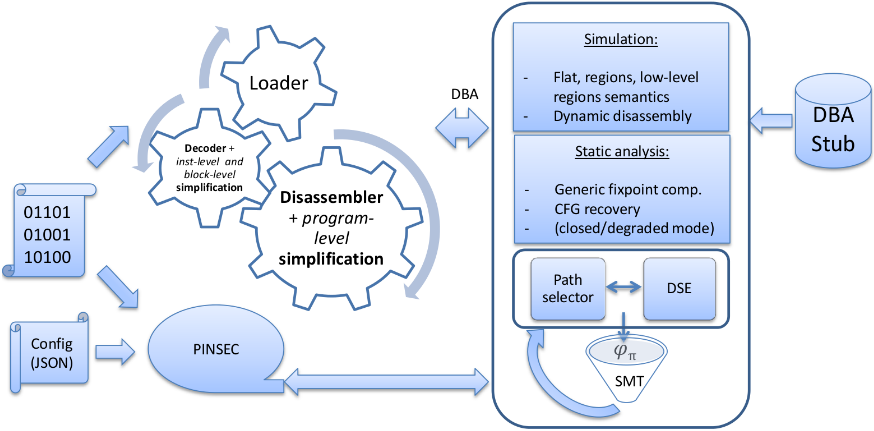
- Platform-independent
- Concise set of instructions
- Specification and abstraction mechanisms

28 Expressions

```
v < flag, temp >, (r, bv)
@[e](←k), @[e](→k)
e {i..j}, ext_{u,s}(e, n)
e { +, −, ×, /_{u,s}, %_{u,s} } e
e { ∧, ∨, ⊕, >>, <<_{u,s}, :: } e
e { <_{u,s}, ≤_{u,s}, =, ≠, ≥_{u,s}, >_{u,s} } e
```

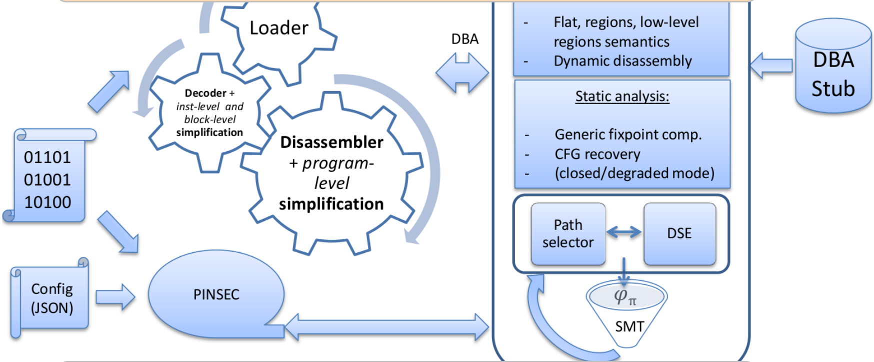
Outline

BINSEC Platform Overview



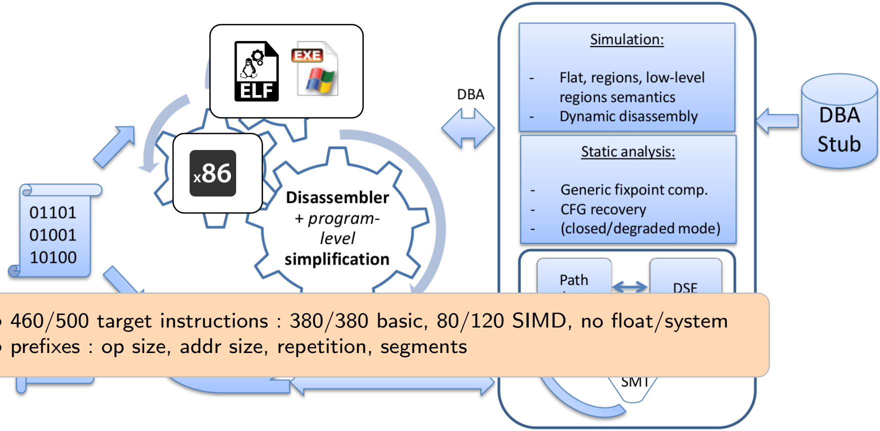
BINSEC Platform Overview

- Front-end [loader, decoder, disassembly, simplifications]
- Simulator [concrete interpretation]
- Generic static analyzer [fixpoint loop, CFG recovery]
- DSE [flexible : C/S (concretisation/symbolisation), path search]



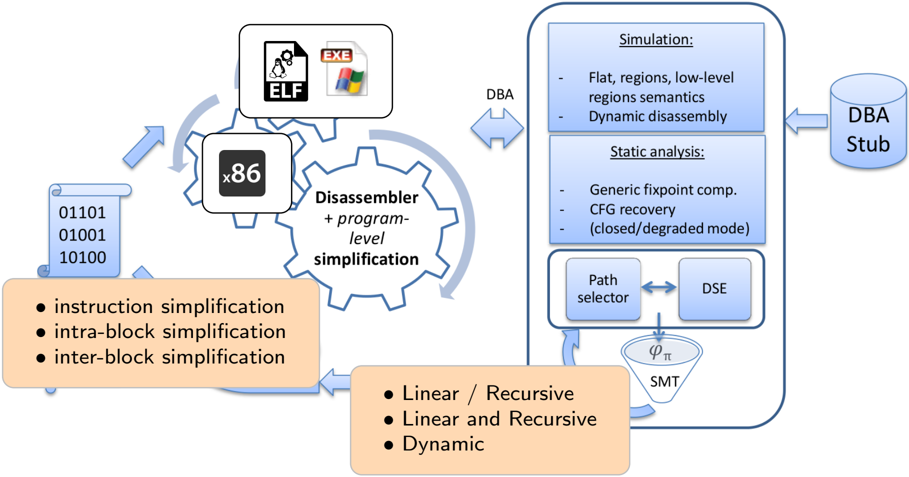
- Developed in OCaml [≈30 000 loc] [TACAS 2015]
- Within the BINSEC project [CEA, IRISA, LORIA, Univ-Genoble]

BINSEC Platform Overview



- 460/500 target instructions : 380/380 basic, 80/120 SIMD, no float/system
- prefixes : op size, addr size, repetition, segments

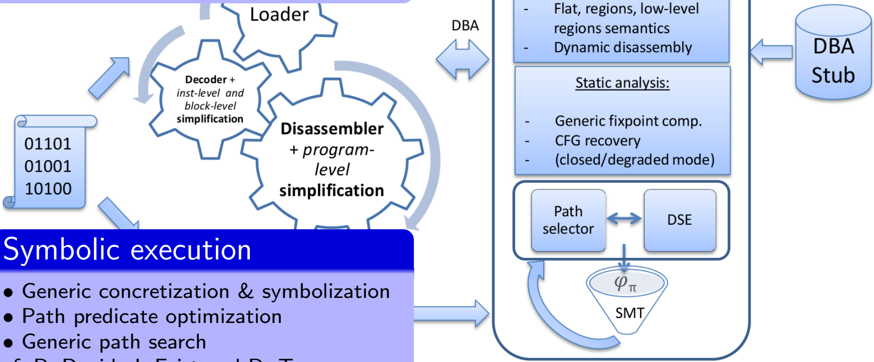
BINSEC Platform Overview



BINSEC Platform Overview

Static analysis

- Generic fixpoint computation
- Sound CFG recovery



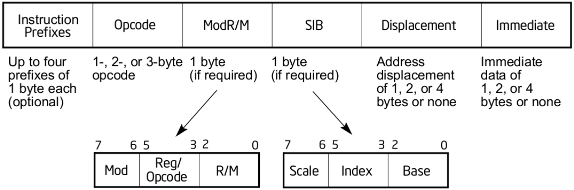
Symbolic execution

- Generic concretization & symbolization
- Path predicate optimization
- Generic path search

cf. R. David, J. Feist and D. Ta

Outline

X86 front-end



81 c3 57 1d 00 00 $\overset{X86reference}{\Longrightarrow}$ ADD EBX 1d57

```
(0x29e,0) t   := EBX + (cst, 7511 < 32 >);
(0x29e,1) OF  := (EBX{31,31}=(cst, 7511 < 32 >){31,31}) && (EBX{31,31}<>t{31,31});
(0x29e,2) SF  := t{31,31};
(0x29e,3) ZF  := t = (cst, 0<32>);
(0x28e,4) AF  := ((extu (EBX{0,7}) 9) + (cst, 7511 < 32 >){0,7}){8,8};
(0x29e,5) PF  := t{0,0}⊗t{1,1}⊗t{2,2}⊗t{3,3}⊗t{4,4}⊗t{5,5}⊗t{6,6}⊗t{7,7}⊗1<1>;
(0x29e,6) CF  := ((extu EBX 33) + (extu (cst, 7511 < 32 >) 33)){32,32};
(0x29e,7) EBX := t; goto (0x2a4,0)
```

DBA simplifications

- Instruction level simplifications
  - Idiom simplifications[local rewriting rules]
- Block level simplifications
  - Constants propagation
  - Remove redundant assigns
- Program level simplifications
  - Flag slicing (remove must-killed variables)
  - granularity : function level+automatic summary of callees

Approach

- Inspired from standard compiler optim
- Targets : flags & temp
- Sound : w.r.t. incomplete CFG
- Inter-procedural (summaries)

DBA simplifications : Experiments

program	native loc	DBA loc	opt (DBA) time	opt (DBA) loc	opt (DBA) red
bash	166K	559K	673.61s	389K	30.45%
cat	8K	23K	18.54s	18K	23.02%
echo	4K	10K	6.96s	8K	24.26%
less	23K	80K	69.99s	55K	30.96%
ls	19K	63K	65.69s	44K	30.58%
mkdir	8K	24K	19.74s	17K	29.50%
netstat	17K	50K	52.59s	40K	20.05%
ps	12K	36K	36.99s	27K	23.98%
pwd	4K	11K	7.69s	9K	23.56%
rm	10K	30K	24.93s	22K	25.24%
sed	10K	32K	28.85s	23K	26.20%
tar	64K	213K	242.96s	154K	27.48%
touch	8K	26K	24.28s	18K	27.88%
uname	3K	10K	6.99s	8K	23.62%

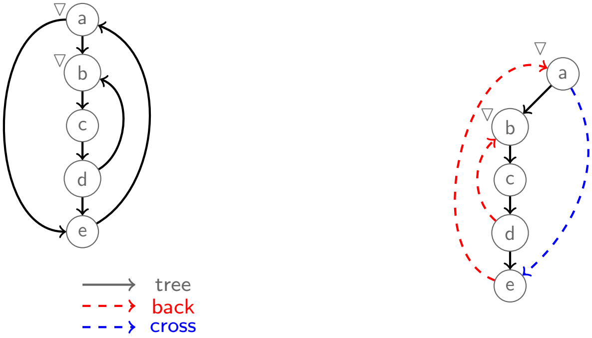
	reduction			
	time	dba instr	tmp assigns	flag assigns
BINSEC	1279.81s	28.64%	90.00%	67.04%
GDSL-Like	1077.98s	15.65%	86.21%	30.27%

Outline

Binsec/VA

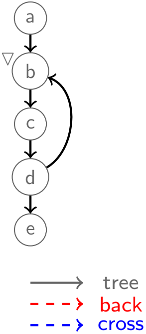
- Basic domains :
  - Dual intervals : $([0,1]_u;[0,1]_s) - 1 = ([0,255]_u;[-1,0]_s)$
  - Flags : $ZF \mapsto (eax - ebx == 0)$
  - Equality : $\{eax == ebx\} \mapsto ([0,3]_u;[0,3]_s)$
- Lifting to byte-precise memory model [Frama-C]
- Tradeoff precision *vs* scale
  - K – *callstring* context sensitivity
  - Loop unrolling
  - Variants of widening (thresholds/delayed)

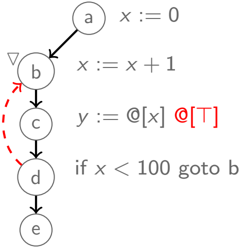
Widening points positioning



- Automatic widening point detection with DFS
- Smart positioning of widening points

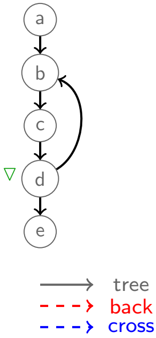
Widening points positioning

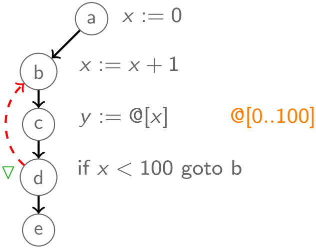




- Automatic widening point detection with DFS
- Smart positioning of widening points

Widening points positioning



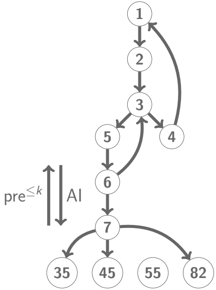


- Automatic widening point detection with DFS
- Smart positioning of widening points

CFG recovery

```
5:  ...
6:  x := @[4 * y + 100];  (y, @[100], @[104], @[108]) ↦ ([0, 2], [35, 35], [45, 45], [82, 82])
7:  goto x;               x ↦ [35, 82]   //x ↦ {35, 45, 82}
```

- Need of precise target values at **djmp**, **store, load**
- Use $\text{pre}^{\leq k}$ to check actual targets w.r.t. AI invariant [Bardin-ICST15, Brauer-EMSOFT11-ESOP11]
- If $\text{pre}^{\leq k}$ fails then switch to degraded mode (pre. iter. AI : ⑦ → {㉟, ㊺}) [Kinder-VMCAI12]
- Refine AI invariant for **load/store** ([a, b]) w.r.t $\text{pre}^{\leq k}$



High-level predicate recovery

- Store relations in flag variables
- Propagate relations (take updates into account)

```
cmp x y; //OF := (({x,31,31}≠{y,31,31}) & ({x,31,31}≠{(x−y),31,31}));
         //SF := (x−y) < 0;
         //ZF := (x−y) = 0;
jg a;    //if (¬ZF ∧ (OF = SF)) then goto a
```

- Too complex for basic non-relational domains
- Complex low-level predicate may hide simple predicate

```
    if (x > y) then goto a
```

- **Template-based** recovery (Platform independent) [sub. FM16]

Experiments

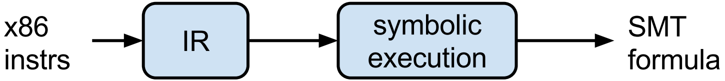
progs	#loc	#conds	#succ	#fail	time (s)	time$_{all}$ (s)
firefox	21488	150 **(137)**	**134** | 89% **(98%)**	16	1.40	55.91
cat	6490	132 **(125)**	**116** | 88% **(92%)**	16	1.08	259.24
chmod	8954	183 **(172)**	**159** | 87% **(92%)**	24	1.44	313.17
cp	67199	174 **(162)**	**152** | 87% **(94%)**	22	4.79	346.84
cut	7358	148 **(138)**	**132** | 89% **(96%)**	16	1.16	211.73
dir	9732	137 **(126)**	**118** | 86% **(94%)**	19	1.26	201.67
echo	8016	190 **(182)**	**168** | 88% **(92%)**	22	1.43	274.60
kill	6911	142 **(133)**	**125** | 88% **(94%)**	17	1.17	209.79
ln	88837	203 **(185)**	**177** | 87% **(96%)**	26	4.88	531.58
mkdir	6347	125 **(117)**	**109** | 87% **(93%)**	16	1.01	235.80
Verisec	11552	394 **(370)**	**370** | 87% **(100%)**	24	3.31	34.48
total	242884	1978 **(1847)**	**1760** | 89% (**95%**)	218	22.93	2674.81

Outline

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..)*

Originality :

- C/S meta language to modulate concretization/symbolization
- stub engine
- path predicate optimizations
  - Constant propagation
  - Variable rebasing
  - Read-Over-Write
- generic path coverage
  - DFS, BFS, random path

Solvers supported : Z3, boolector, CVC4

Results obtained

- Scale on large traces (with C/S policies) [R. David-ISSTA16]
  - benchmark on all coreutils (100 binaries)
- Provided good results for deobfuscation [sub. R. David-CCS16]
  - opaque predicates : no false negative, very low false positive
  - call stack tampering : no false positive, identify different kind of tampering
- Good results for Use-After-Free detection [sub. J. Feist-WOOT16]
  - Vulnerability found in JasPer (CVE-2015-5221)

Outline

Conclusion

Front-end

- $\approx 460/\approx 500$ targeted x86 instructions supported
- DBA simplifications
- Tested on Coreutils, Windows Malwares, Verisec/Juliet, etc.

Static analysis module

- Standard (basic domains, precision trade off) :
  Context sensitive AI, Widenings, Loop unrolling, dual intervals
- New : Natural flag recovery, automatic detection of widening points, CFG recovery with AI + $\mathrm{pre}^{\leq k}$

Dynamic symbolic execution

- Scalable/tunable approach for path coverage

Questions ?