Binsec: a platform for binary code analysis

08/06/2016

Adel Djoudi
Robin David
Josselin Feist
Thanh Dinh Ta
Introduction

The BINSEC Platform

DBA simplification

Static analysis

Symbolic execution

Conclusion
Introduction

Binary code analysis: Why?

Source program:
- Ada
- C/C++
- Java
- Perl
- Python
- ...

Compiler

Binary program:
```
00101101010110011011
10001010110101001010
01011010101001101
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
Introduction

Binary code analysis: Why?

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

Compiler

Binary program
001011010110011011
10001010110101001010
010110101011001111
001011010110000101
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
Introduction
Challenges of binary code analysis (1)

Entry point

Code or Data?

```
push ebp
mov ebp,esp
mov ds:0x80ebf48,0x1
mov eax,ds:0x80ebf48
cmp eax,0x9
ja 80490f6
mov eax,[eax*4+0x80be148]
jmp eax
```
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, ≥ 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]
...
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
9 Instructions

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

28 Expressions

\[ \text{lhs} := e; \]
\[ \text{goto } e; <\text{call, return}> \]
\[ \text{goto } bv; <\text{call, return}> \]
\[ \text{ite (e)? goto } bv : \text{goto } bv \]
\[ \text{stop;} \]
\[ \text{lhs} := \text{nondet();} \]
\[ \text{lhs} := \text{undef;} \]
\[ \text{assert (cond);} \]
\[ \text{assume (cond);} \]

\[ v <\text{flag, temp }>, (r, bv) \]
\[ @[e](k), @[e](k) \]
\[ e \{i..j}, \text{ext}_{u,s}(e, n) \]
\[ e \{+, -, \times, /_{u,s}, \%_{u,s}\} e \]
\[ e \{\&, \lor, \oplus, >>, <<<_{u,s}, ::\} e \]
\[ e \{<_{u,s}, \leq_{u,s}, =, \neq, \geq_{u,s}, >_{u,s}\} e \]
Introduction

The BINSEC Platform

DBA simplification

Static analysis

Symbolic execution

Conclusion
The BINSEC Platform

BINSEC Platform Overview

- Decoder + inst-level and block-level simplification
- Loader
- Disassembler + program-level simplification

Simulation:
- Flat, regions, low-level regions semantics
- Dynamic disassembly

Static analysis:
- Generic fixpoint comp.
- CFG recovery
- (closed/degraded mode)

PINSEC

DBA

Stub

Path selector

DSE

$\phi_{\text{SE}}$

SMT
The BINSEC Platform
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]
The BINSEC Platform

BINSEC Platform Overview

- 460/500 target instructions: 380/380 basic, 80/120 SIMD, no float/system
- Prefixes: op size, addr size, repetition, segments
• Instruction simplification
• Intra-block simplification
• Inter-block simplification

Disassembler + program-level simplification

Simulation:
- Flat, regions, low-level regions semantics
- Dynamic disassembly

Static analysis:
- Generic fixpoint comp.
- CFG recovery
- (closed/degraded mode)

Path selector

DSE

\[ \phi_{\text{TE}} \]

SMT

Linear / Recursive
Linear and Recursive
Dynamic
The BINSEC Platform
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

Simulation:
- Flat, regions, low-level regions semantics
- Dynamic disassembly

Static analysis:
- Generic fixpoint comp.
- CFG recovery
- (closed/degraded mode)

Loader
Decoder + inst-level and block-level simplification
Disassembler + program-level simplification

DBA
01101
01001
10100

Path selector
DSE

φ_{τc}
SMT
Outline

Introduction

The BINSEC Platform

DBA simplification

Static analysis

Symbolic execution

Conclusion
DBA simplification

X86 front-end

<table>
<thead>
<tr>
<th>Instruction Prefixes</th>
<th>Opcode</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to four prefixes of 1 byte each (optional)</td>
<td>1-, 2-, or 3-byte opcode</td>
<td>1 byte (if required)</td>
<td>1 byte (if required)</td>
<td>Address displacement of 1, 2, or 4 bytes or none</td>
<td>Immediate data of 1, 2, or 4 bytes or none</td>
</tr>
</tbody>
</table>

81 c3 57 1d 00 00 X86 reference ⇒ ADD EBX 1d57

1. \((0 \times 29e, 0)\) \(t := \text{EBX} + (\text{cst, } 7511 < 32 >)\);
2. \((0 \times 29e, 1)\) \(\text{OF} := (\text{EBX}_{\{31,31\}} = (\text{cst, } 7511 < 32 >)_{\{31,31\}}) \&\& (\text{EBX}_{\{31,31\}} <> t_{\{31,31\}})\);
3. \((0 \times 29e, 2)\) \(\text{SF} := t_{\{31,31\}}\);
4. \((0 \times 29e, 3)\) \(\text{ZF} := t = (\text{cst, } 0 < 32 >)\);
5. \((0 \times 28e, 4)\) \(\text{AF} := ((\text{extu}) (\text{EBX}_{\{0,7\}} 9) + (\text{cst, } 7511 < 32 >)_{\{0,7\}})\{8,8\};
6. \((0 \times 29e, 5)\) \(\text{PF} := t_{\{0,0\}} \otimes t_{\{1,1\}} \otimes t_{\{2,2\}} \otimes t_{\{3,3\}} \otimes t_{\{4,4\}} \otimes t_{\{5,5\}} \otimes t_{\{6,6\}} \otimes t_{\{7,7\}} \otimes 1 < 1\};
7. \((0 \times 29e, 6)\) \(\text{CF} := ((\text{extu}) \text{EBX} 33) + (\text{extu}) (\text{cst, } 7511 < 32 >) 33)\{32,32\};
8. \((0 \times 29e, 7)\) \(\text{EBX} := t\); goto (0 \times 2a4, 0)
DBA simplification

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

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

### Reduction

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>dba instr</th>
<th>tmp assigns</th>
<th>flag assigns</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINSEC</td>
<td>1279.81s</td>
<td>28.64%</td>
<td>90.00%</td>
<td>67.04%</td>
</tr>
<tr>
<td>GDSL-Like</td>
<td>1077.98s</td>
<td>15.65%</td>
<td>86.21%</td>
<td>30.27%</td>
</tr>
</tbody>
</table>
Outline

Introduction

The BINSEC Platform

DBA simplification

Static analysis

Symbolic execution

Conclusion
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)
Static analysis

Widening points positioning

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

Widening points positioning

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

Widening points positioning

Automatic widening point detection with DFS

Smart positioning of widening points
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 $\rightarrow \{35, 45\}$) [Kinder-VMCAI12]

- Refine AI invariant for **load/store** ([a, b]) w.r.t $\text{pre} \leq k$
Static analysis

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 (\neg 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]
## Static analysis

### Experiments

<table>
<thead>
<tr>
<th>progs</th>
<th>#loc</th>
<th>#conds</th>
<th>#succ</th>
<th>#fail</th>
<th>time (s)</th>
<th>time_{all} (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>firefox</td>
<td>21488</td>
<td>150 (137)</td>
<td>134</td>
<td>89% (98%)</td>
<td>16</td>
<td>1.40</td>
</tr>
<tr>
<td>cat</td>
<td>6490</td>
<td>132 (125)</td>
<td>116</td>
<td>88% (92%)</td>
<td>16</td>
<td>1.08</td>
</tr>
<tr>
<td>chmod</td>
<td>8954</td>
<td>183 (172)</td>
<td>159</td>
<td>87% (92%)</td>
<td>24</td>
<td>1.44</td>
</tr>
<tr>
<td>cp</td>
<td>67199</td>
<td>174 (162)</td>
<td>152</td>
<td>87% (94%)</td>
<td>22</td>
<td>4.79</td>
</tr>
<tr>
<td>cut</td>
<td>7358</td>
<td>148 (138)</td>
<td>132</td>
<td>89% (96%)</td>
<td>16</td>
<td>1.16</td>
</tr>
<tr>
<td>dir</td>
<td>9732</td>
<td>137 (126)</td>
<td>118</td>
<td>86% (94%)</td>
<td>19</td>
<td>1.26</td>
</tr>
<tr>
<td>echo</td>
<td>8016</td>
<td>190 (182)</td>
<td>168</td>
<td>88% (92%)</td>
<td>22</td>
<td>1.43</td>
</tr>
<tr>
<td>kill</td>
<td>6911</td>
<td>142 (133)</td>
<td>125</td>
<td>88% (94%)</td>
<td>17</td>
<td>1.17</td>
</tr>
<tr>
<td>ln</td>
<td>88837</td>
<td>203 (185)</td>
<td>177</td>
<td>87% (96%)</td>
<td>26</td>
<td>4.88</td>
</tr>
<tr>
<td>mkdir</td>
<td>6347</td>
<td>125 (117)</td>
<td>109</td>
<td>87% (93%)</td>
<td>16</td>
<td>1.01</td>
</tr>
<tr>
<td>Verisec</td>
<td>11552</td>
<td>394 (370)</td>
<td>370</td>
<td>87% (100%)</td>
<td>24</td>
<td>3.31</td>
</tr>
<tr>
<td>total</td>
<td>242884</td>
<td>1978 (1847)</td>
<td>1760</td>
<td>89% (95%)</td>
<td>218</td>
<td>22.93</td>
</tr>
</tbody>
</table>
Outline

Introduction

The BINSEC Platform

DBA simplification

Static analysis

Symbolic execution

Conclusion
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
Symbolic execution

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)
Conclusion

Outline

Introduction

The BINSEC Platform

DBA simplification

Static analysis

Symbolic execution

Conclusion
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 + pre$^{\leq k}$

Dynamic symbolic execution

- Scalable/tunable approach for path coverage
Questions ?