Malware Devirtualization with Predicate Synthesis

Fabrizio Biondi
CentraleSupélec / IRISA Rennes
# A False History of Binary Obfuscation

<table>
<thead>
<tr>
<th>Reverse Engineering</th>
<th>Obfuscation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static analysis</td>
<td>Packing</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>Evasion</td>
</tr>
<tr>
<td>Concolic analysis</td>
<td>Virtualization</td>
</tr>
<tr>
<td>Devirtualization</td>
<td>Opaque predicates</td>
</tr>
<tr>
<td>Predicate synthesis</td>
<td>...</td>
</tr>
</tbody>
</table>

= ™ severely hindered by ™
A False History of Binary Obfuscation

Reverse Engineering  Obfuscation

Static analysis → Packing
Dynamic analysis → Evasion
Concolic analysis → Virtualization
Devirtualization → Opaque predicates
Predicate synthesis → …

= “severely hindered by”
A False History of Binary Obfuscation

Reverse Engineering          Obfuscation

Static analysis           Packing

Dynamic analysis           Evasion

Concolic analysis           Virtualization

Devirtualization          Opaque predicates

Predicate synthesis

= ¨severely hindered by¨
Concolic (CONCrete-symbOLIC) Analysis

- Static symbolic analysis
- To increase code coverage:
  - SMT solver to determine conditional satisfiability
  - Concretize traces as necessary (less static)

Angr: Concolic Binary Analysis Engine

http://angr.io/
A False History of Binary Obfuscation

Reverse Engineering

Static analysis
Dynamic analysis
Concolic analysis
Devirtualization
Predicate synthesis

Obfuscation

Packing
Evasion
Virtualization
Opaque predicates

= "severely hindered by"
Virtualization

- Control Flow Flattening
  - Embeds an interpreter in the code and uses it as dispatcher
  - Hinders concolic analysis because the dispatcher is too complex for SMT solvers
- Tigress: source-to-source virtualizer

http://tigress.cs.arizona.edu/
Virtualization

Original trace

Virtualized trace

The dispatcher:
- computes the next operation’s address
- Is too complex to be analyzed symbolically
A False History of Binary Obfuscation

Reverse Engineering  Obfuscation

Static analysis  Packing
Dynamic analysis  Evasion
Concolic analysis  Virtualization
Devirtualization  Opaque predicates
Predicate synthesis  ...

= "severely hindered by"
Devirtualization

Virtualized trace

Reconstructed trace

Goal: identify and remove dispatcher
Devirtualization: Data Dependency Graph

Example code:
1. addl eax, ebx  // ebx = ebx + eax
2. pushl ebx  // mem[rsp] = ebx
3. subl 8, eax  // eax = eax - 8
4. jmp _fun  // jump to _fun
_back:
5. popl eax  // eax = mem[rsp]
6. addl eax, ebx  // esi = esi + eax
_fun:
7. movl 1, eax  // eax = 1
8. jmp _back  // jump back to _back

**Bolded** instructions/nodes/arrows:
Keep in simplified program

**Underlined** instructions:
Implicit dependencies, also kept

(Not shown): control flow dependencies (implicit)
Devirtualization: Traces to Programs

devirtualize

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic

Devirtualization: Traces to Programs

concolic
Devirtualization: Traces to Programs

Problem: we are assuming that the non-dispatcher predicates are actually solvable...

devirtualize

concolic

devirtualize

concolic

...
A False History of Binary Obfuscation

Reverse Engineering
- Static analysis
- Dynamic analysis
- Concolic analysis
- Devirtualization
- Predicate synthesis

Obfuscation
- Packing
- Evasion
- Virtualization
- Opaque predicates
- ...

= “severely hindered by”
Opaque Predicates

These predicates are equivalent:

• if (x==4)...

• if (x^3 - 4x^2 + 6x - 24 = 0)...

• if(sha3(x)==37f558134baa535903c6a88931c8122e334368bf951f2cada569b11774ef9795ef6d2ac961d13ee44a0c837db3817bb9db68ac3bdfb8b19a1308618484a9da8f)...

Note that they are point functions:
• true for one input, false for any other
• SMT solvers often report point functions as constant false
Mixed Boolean Arithmetic Obfuscation

1) Wrap conditional $c$ in polynomial function $F$ of degree $d$
2) Add linear identities $e_1...e_n$ multiplied by $a_1...a_n$
3) Wrap all in $F^{-1}$

original

\[ c = F^{-1} \left( \sum_i a_i e_i + F(c) \right) \]

obfuscated

---

Y. Zhou, A. Main, Y. X. Gu, H. Johnson, *Information hiding in software with mixed Boolean-arithmetic transforms*, WISA'07
Mixed Boolean Arithmetic Obfuscation

original

if (IN == 0x87654321)
  ..
else
  ..

obfuscated (d = 2)

```c
if (IN == f )
  ..
else
  ..
```

Y. Zhou, A. Main, Y. X. Gu, H. Johnson, *Information hiding in software with mixed Boolean-arithmetic transforms*, WISA'07
Cost of MBA Obfuscation

![Graph showing the relationship between the degree of the polynomial and compilation time and obfuscated binary file size. The compilation time and obfuscated file size increase exponentially with the degree of the polynomial.](image)
Attacking MBA Obfuscation with SMT Solvers

The graph shows the time (in seconds) taken by different solvers to handle polynomials of increasing degree. The solvers include STP (KLEE), STP (MetaSMT), Z3, and Boolector. The x-axis represents the degree of the polynomial, and the y-axis represents the time taken in seconds. The solvers' performance varies significantly with degree, with Z3 showing much higher time for higher degrees compared to the others.
A False History of Binary Obfuscation

Reverse Engineering

Static analysis
Dynamic analysis
Concolic analysis
Devirtualization
Predicate synthesis

Obfuscation

Packing
Evasion
Virtualization
Opaque predicates

= “severely hindered by”
Predicate Synthesis

- Synthesis method: drill-and-join (efficient Reed-Muller)
- Obfuscated conditional interrogated as **black-box** oracle
- Conditional function reconstructed
  - Not only satisfiability
  - Smaller constraints
  - Can be reinjected in the code
- Depends on input size

---

R. Balaniuk, *Drill and join: A method for exact inductive program synthesis*, LOPSTR’14
Attacking MBA Obfuscation with Synthesis

![3D Graph]

- **Time (s)**
- **Degree**
- **Input size (bits)**

- Time increases with increasing degree and input size.
Insight Gained from Predicate Synthesis

• Synthesis is more expensive than satisfiability
• ... so why is synthesis faster in this case?
  − based on a "smart brute force approach"
  − black box: does not start by analyzing the actual formula
  − better for simple formulas written in a complex manner?
• Probably not better than SMT in general
• Seems SMT with targeted brute forcing could improve concolic analysis
A False History of Binary Obfuscation

Reverse Engineering  Obfuscation

Static analysis  →  Packing

Dynamic analysis  →  Evasion

Concolic analysis  →  Virtualization

Devirtualization  →  Opaque predicates

Predicate synthesis  → ...

\[= \text{"severely hindered by"} \]
Conclusions

- Virtualization can be countered by cycles of data dependency analysis and concolic analysis
  - Note: the dispatcher is too complex to hope to solve/simplify/synthesize it directly
- Opaque predicates can be synthesized to analyze multiple traces
  - Synthesis provides simplified predicate, can be reinjected
  - Effectiveness of synthesis suggests that SMT solving equipped with brute force would be more effective

Thank you for your attention!