Using Logic Programming to Recover C++ Classes and Methods from Compiled Executables

Edward J. Schwartz, Cory F. Cohen, Michael Duggan, Jeffrey Gennari, Jeffrey S. Havrilla, Charles Hines
Using Logic Programming to Recover C++ Classes and Methods from Compiled Executables

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM18-0828
Goal

C++ Binary

C++ Abstractions

Shape

Square

SquareHelper

Inherits From

Is Composed Of
Goal

C++ Abstractions

Inherits From

Shape
- virtual draw()
- virtual getArea()
- print()

Square
- virtual draw()
- virtual getArea()
- getLength()

Is Composed Of

SquareHelper
- areaLogic()
### Goal

#### C++ Abstractions

**Shape**
- virtual draw()
- virtual getArea()
- print()

**Square**
- virtual draw()
- virtual getArea()
- getLength()
- int side_len

**SquareHelper**
- areaLogic()

**Inherits From**
- Shape

**Is Composed Of**
- Square
- SquareHelper

#### Binary

- 101011

---

**C++ Abstractions**

- Inherits From Shape
- Is Composed Of Square and SquareHelper

---

**Square**
- virtual draw()
- virtual getArea()
- getLength()
- int side_len

**SquareHelper**
- areaLogic()

---

**Goal**

- Inherit From SquareHelper

---

**C++ Abstractions**

- Shape
- Square
- SquareHelper
Goals: How?

• **Static**
  - Analyze program *without* executing it
  - Can be applied to unknown software
  - No need for test cases

• **Recover all classes and all methods**

• **Do not rely on RTTI metadata**
  - Only available for polymorphic classes
  - Often removed or corrupted in malware
## Existing Work

<table>
<thead>
<tr>
<th>System</th>
<th>Static</th>
<th>All classes</th>
<th>All methods</th>
<th>Works w/o RTTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SmartDec Fokin 2010</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SecondWrite Yoo 2014</td>
<td>✔</td>
<td>✘</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Various CFI systems</td>
<td>✔</td>
<td>✘</td>
<td>❌</td>
<td>✔</td>
</tr>
<tr>
<td>Pawlowski 2017, etc.</td>
<td>✔</td>
<td>✘</td>
<td>❌</td>
<td>✔</td>
</tr>
</tbody>
</table>

These systems primarily recover information from virtual function tables.
Virtual Function Tables

Shape's VFTable
0: Shape::draw
1: Shape::getArea

Square's VFTable
0: Square::draw
1: Square::getArea

VFTables enumerate the virtual functions of each class... but only for polymorphic classes.
## Existing Work

<table>
<thead>
<tr>
<th>System</th>
<th>Static</th>
<th>All classes</th>
<th>All methods</th>
<th>Works w/o RTTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SmartDec Fokin 2010</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SecondWrite Yoo 2014</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>Various CFI systems</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td>Lego Srinivasan 2014</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>ObjDigger Jin 2014</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
ObjDigger

ObjDigger uses **object pointer tracking** to recover non-polymorphic classes too

```c++
void foobar () {
    Obj *x = new Obj();
    x->foo();
    func1(x);
}
```

- **foo** is either defined on *Obj* class or inherited
- **bar** is either defined on *Obj* class or inherited

If the *Obj* class does not inherit, it defines **foo** and **bar**
Ambiguous Scenarios

```cpp
int main() {
    Square sq;
    sq.draw();
}
```

**Shape**
- **draw()**

**Square**
- Inherits From **Shape**
- **draw()**

**Inherits From**
ObjDigger's Shortcomings

• ObjDigger makes mistakes on ambiguous scenarios
  - We tuned it so it was "usually" correct
  - But mistakes propagated and caused more serious problems

• ObjDigger was written procedurally in C++
  - \texttt{if (ambiguous\_scenario) \{ classes[foo].add\_method(bar); \} // guess!}
  - There was no mechanism to detect mistakes and repair them

• Improving accuracy became more and more difficult
OOAnalyzer

OOAnalyzer was designed to overcome ObjDigger's shortcomings

• Reasoning implemented in Prolog

• Hypothetical reasoning
  - Detect ambiguous scenarios, and make best guess first
  - New: detect when guesses introduce inconsistency and undo them
  - Implemented using Prolog backtracking

• Reasoning is defined using declarative rules
  - Much easier to understand and maintain
OOAnalyzer System Design
C++

Binary

Pharos Framework
Fact Exporter

Prolog Reasoning

Forward Reasoning

Hypothetical Reasoning

Consistency Checking

C++ Abstractions

Inherits From

Shape

Square

virtual draw() virtual getArea() getLength() int cached_area

Int side_len SquareHelper sqh

Is Composed Of

SquareHelper areaLogic()
Fact Exporter

- Uses conventional binary analysis to produce initial facts about the program
  - Initial facts describe low-level program behaviors

- Simple symbolic analysis
  - Intentionally favors scalability over accuracy
  - Does not use constraint solvers
  - Symbolic memory aliases if memory addresses are equal after simplification
  - Path sensitive up to a threshold

- Sufficient because reasoning system can cope with mistakes
## Initial Facts

Initial facts describe low-level program behaviors and form the basis upon which OOAnalyzer's reasoning system operates.

<table>
<thead>
<tr>
<th>Fact Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjPtrAllocation(I, F, P, S)</td>
<td>Instruction I in function F allocates S bytes of memory for the object pointed to by P.</td>
</tr>
<tr>
<td>ObjPtrInvoke(I, F, P, M)</td>
<td>Instruction I in function F calls method M on the object pointed to by P.</td>
</tr>
<tr>
<td>ObjPtrOffset(P₁, O, P₂)</td>
<td>Object pointer P₂ points to P₁ + O.</td>
</tr>
<tr>
<td>MemberAccess(I, M, O, S)</td>
<td>Instruction I in method M accesses S bytes of memory at offset O from the current object's pointer.</td>
</tr>
<tr>
<td>ThisCallMethod(M, P)</td>
<td>Method M receives the object pointed to by P in the ecx register.</td>
</tr>
<tr>
<td>NoCallsBefore(M)</td>
<td>No methods are called on any object pointer before method M.</td>
</tr>
<tr>
<td>ReturnsSelf(M)</td>
<td>Method M returns the object pointer that was passed as a parameter.</td>
</tr>
<tr>
<td>UninitializedReads(M)</td>
<td>Method M reads memory that was not written to by M.</td>
</tr>
<tr>
<td>PossibleVFTableEntry(VFT, O, M)</td>
<td>Method M may be at offset O in vftable VFT.</td>
</tr>
</tbody>
</table>
### Entity Facts

Entity facts are produced during the reasoning process and describe the high-level model of the program being analyzed.

<table>
<thead>
<tr>
<th>Fact Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method(M)</td>
<td>Method M is an OO method on a class or struct.</td>
</tr>
<tr>
<td>Constructor(M)</td>
<td>Method M is an object constructor.</td>
</tr>
<tr>
<td>Destructor(M)</td>
<td>Method M is an object destructor.</td>
</tr>
<tr>
<td>( Cl_a = Cl_b )</td>
<td>The sets of methods ( Cl_a ) and ( Cl_b ) both represent methods from the same class. These sets should be combined into a single class.</td>
</tr>
<tr>
<td>( Cl_a \leq Cl_b )</td>
<td>The sets of methods ( Cl_a ) and ( Cl_b ) either both represent methods from the same class, or, the methods in ( Cl_b ) are inherited from ( Cl_a ).</td>
</tr>
<tr>
<td>M ( \in ) Cl</td>
<td>Method M is defined directly on class Cl.</td>
</tr>
<tr>
<td>ClassCallsMethod(Cl, M)</td>
<td>An instance of class Cl calls method M.</td>
</tr>
</tbody>
</table>

Other categories include virtual functions, class relationships, and sizes of classes and tables.
Reasoning

\[ P_1 \quad P_2 \quad \ldots \quad P_n \]

\[ \underline{C} \]

- **Forward reasoning**
  - Unambiguous scenarios
  - Interpretation: If \( P_1 \), \( P_2 \), \ldots and \( P_n \) are satisfied, then \( C \) is true
  - If inconsistency is reached, \( P_1 \), \( P_2 \), \ldots or \( P_n \) must not be true

- **Hypothetical reasoning**
  - Ambiguous scenarios
  - Interpretation: If \( P_1 \), \( P_2 \), \ldots and \( P_n \) are satisfied, then guess \( C \) is true
  - If inconsistency is reached, then retract \( C \) and assume \( \neg C \)
  - If inconsistency is still reached, \( P_1 \), \( P_2 \), \ldots or \( P_n \) must not be true
Forward Reasoning

If a method is directly called on a base class, it cannot be directly defined on the derived class.

\[
\begin{align*}
\text{Constructor}(M_d) & \quad M_d \in Cl_d \\
\text{Constructor}(M_b) & \quad M_b \in Cl_b \\
\text{ClassCallsMethod}(Cl_d, M) & \\
\text{ClassCallsMethod}(Cl_b, M) & \quad M_d \neq M_b \\
M \in Cl_m & \quad Cl_d \neq Cl_b & \text{DerivedClass}(Cl_d, Cl_b, _) \\
\hline
Cl_m & \neq Cl_d
\end{align*}
\]
Hypothetical Reasoning

If a method is called on a derived class but not a base class, (first) assume the method is defined on the derived class.

\[
\text{ClassCallsMethod}(\text{Cl}_d, M) \quad \rightarrow \quad \text{ClassCallsMethod}(\text{Cl}_b, M)
\]

\[
M \in \text{Cl} \quad \text{DerivedClass}(\text{Cl}_d, \text{Cl}_b, _) \quad \therefore \quad \text{Cl}_d = \text{Cl}
\]
Hypothetical Reasoning

Square

| draw() |

Inherits From

Shape

\textit{Reasoning}

\textit{Inconsistency}

\textit{Backtracking}

\texttt{int main() \{ \\
\hspace{1em} Square sq; \\
\hspace{1em} sq.draw(); \\
\}}
Hypothetical Reasoning

Square

Inherits From

Shape
draw()
Research Questions & Evaluation
Research Questions

• How well does OOAnalyzer identify C++ classes and their constituent methods?
  - OOAnalyzer classes may not naturally correspond to ground truth classes
  - New metric: edit distance
Edit Distance Example

OOAnalyzer

Class ?
&Class1::A
&Class2::B

Class ?
&Class1::B
&Class2::A

Edit 1

Class ?
&Class1::A

Class ?
&Class2::B

Edit 2

Class ?
&Class1::A

&Class1::B

Class ?
&Class2::A

&Class2::B

Ground Truth

Class 1
&Class1::A
&Class1::B

Class 2
&Class2::A
&Class2::B

Move

Move
## ObjDigger vs OOAnalyzer Edit Distances on ObjDigger Programs

ObjDigger made 121 mistakes

ObjDigger recovered about 45% of methods

OOAnalyzer recovered about 90% of methods

<table>
<thead>
<tr>
<th>Program</th>
<th># Class</th>
<th># Method</th>
<th>ObjDigger Edits</th>
<th>ObjDigger Edits (%)</th>
<th>OOAnalyzer Edits</th>
<th>OOAnalyzer Edits (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clmg</td>
<td>29</td>
<td>220</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light-pop3-smtp</td>
<td>44</td>
<td>295</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optionparser</td>
<td>11</td>
<td>56</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PicoHttpD</td>
<td>95</td>
<td>656</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X3c</td>
<td>6</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### ObjDigger vs OOAAnalyzer Edit Distances on Cleanware

<table>
<thead>
<tr>
<th>Program</th>
<th># Class</th>
<th># Method</th>
<th>ObjDigger Edits</th>
<th>ObjDigger Edits (%)</th>
<th>OOAAnalyzer Edits</th>
<th>OOAAnalyzer Edits (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox.exe</td>
<td>141</td>
<td>638</td>
<td>507</td>
<td>79.5%</td>
<td>212</td>
<td>33.2%</td>
</tr>
<tr>
<td>Log4cpp Debug</td>
<td>139</td>
<td>893</td>
<td>829</td>
<td>92.8%</td>
<td>239</td>
<td>26.8%</td>
</tr>
<tr>
<td>Log4cpp Release</td>
<td>76</td>
<td>378</td>
<td>272</td>
<td>72.0%</td>
<td>75</td>
<td>19.8%</td>
</tr>
<tr>
<td>muParser Debug</td>
<td>180</td>
<td>1437</td>
<td>1361</td>
<td>94.7%</td>
<td>483</td>
<td>33.6%</td>
</tr>
<tr>
<td>muParser Release</td>
<td>94</td>
<td>598</td>
<td>369</td>
<td>61.7%</td>
<td>183</td>
<td>30.6%</td>
</tr>
<tr>
<td>MySQL cfg_editor.dll</td>
<td>190</td>
<td>1266</td>
<td>∞</td>
<td>∞</td>
<td>391</td>
<td>30.9%</td>
</tr>
<tr>
<td>MySQL mysql.exe</td>
<td>202</td>
<td>1395</td>
<td>∞</td>
<td>∞</td>
<td>439</td>
<td>31.5%</td>
</tr>
<tr>
<td>TinyXML Debug</td>
<td>35</td>
<td>415</td>
<td>268</td>
<td>64.6%</td>
<td>69</td>
<td>16.6%</td>
</tr>
<tr>
<td>TinyXML Release</td>
<td>33</td>
<td>283</td>
<td>174</td>
<td>61.5%</td>
<td>55</td>
<td>19.4%</td>
</tr>
<tr>
<td>Program</td>
<td># Class</td>
<td># Method</td>
<td>ObjDigger Edits</td>
<td>ObjDigger Edits (%)</td>
<td>OOAnalyzer Edits</td>
<td>OOAnalyzer Edits (%)</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------</td>
<td>----------</td>
<td>-----------------</td>
<td>---------------------</td>
<td>-----------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Firefox.exe</td>
<td>141</td>
<td>638</td>
<td>507</td>
<td>79.5%</td>
<td>212</td>
<td>33.2%</td>
</tr>
<tr>
<td>Log4cpp Debug</td>
<td>139</td>
<td>893</td>
<td>829</td>
<td>92.8%</td>
<td>239</td>
<td>26.8%</td>
</tr>
<tr>
<td>Log4cpp Release</td>
<td>76</td>
<td>378</td>
<td>272</td>
<td>72.0%</td>
<td>75</td>
<td>19.8%</td>
</tr>
<tr>
<td>muParser Debug</td>
<td>180</td>
<td>1437</td>
<td>1361</td>
<td>94.7%</td>
<td>483</td>
<td>33.6%</td>
</tr>
<tr>
<td>muParser Release</td>
<td>94</td>
<td>598</td>
<td>369</td>
<td>61.7%</td>
<td>183</td>
<td>30.6%</td>
</tr>
<tr>
<td>MySQL cfg_editor.dll</td>
<td>190</td>
<td>1266</td>
<td>∞</td>
<td>∞</td>
<td>391</td>
<td>30.9%</td>
</tr>
<tr>
<td>MySQL mysql.exe</td>
<td>202</td>
<td>1395</td>
<td>∞</td>
<td>∞</td>
<td>439</td>
<td>31.5%</td>
</tr>
<tr>
<td>TinyXML Debug</td>
<td>35</td>
<td>415</td>
<td>268</td>
<td>64.6%</td>
<td>69</td>
<td>16.6%</td>
</tr>
<tr>
<td>TinyXML Release</td>
<td>33</td>
<td>283</td>
<td>174</td>
<td>61.5%</td>
<td>55</td>
<td>19.4%</td>
</tr>
</tbody>
</table>

OOAnalyzer recovers between 67 and 84% of methods on medium to large programs.
<table>
<thead>
<tr>
<th>Program</th>
<th># Class</th>
<th># Method</th>
<th>ObjDigger Edits</th>
<th>ObjDigger Edits (%)</th>
<th>OODigger Edits</th>
<th>OODigger Edits (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malware 0faaa3d3</td>
<td>21</td>
<td>135</td>
<td>121</td>
<td>89.6%</td>
<td>21</td>
<td>15.6%</td>
</tr>
<tr>
<td>Malware 29be5a33</td>
<td>19</td>
<td>130</td>
<td>91</td>
<td>70.0%</td>
<td>15</td>
<td>11.5%</td>
</tr>
<tr>
<td>Malware 6098cb7c</td>
<td>55</td>
<td>339</td>
<td>131</td>
<td>38.6%</td>
<td>29</td>
<td>8.6%</td>
</tr>
<tr>
<td>Malware 628053dc</td>
<td>207</td>
<td>1920</td>
<td>1245</td>
<td>64.8%</td>
<td>378</td>
<td>19.7%</td>
</tr>
<tr>
<td>Malware 67b9be3c</td>
<td>400</td>
<td>2072</td>
<td>1299</td>
<td>62.7%</td>
<td>670</td>
<td>32.3%</td>
</tr>
<tr>
<td>Malware cfa69fff</td>
<td>39</td>
<td>184</td>
<td>125</td>
<td>67.9%</td>
<td>37</td>
<td>20.1%</td>
</tr>
<tr>
<td>Malware d597bee8</td>
<td>19</td>
<td>133</td>
<td>68</td>
<td>51.1%</td>
<td>17</td>
<td>12.8%</td>
</tr>
<tr>
<td>Malware deb6a7a1</td>
<td>283</td>
<td>2712</td>
<td>1900</td>
<td>70.1%</td>
<td>639</td>
<td>23.6%</td>
</tr>
<tr>
<td>Malware f101c05e</td>
<td>169</td>
<td>1601</td>
<td>987</td>
<td>61.6%</td>
<td>329</td>
<td>20.5%</td>
</tr>
</tbody>
</table>
Research Questions

• How well does OOAnalyzer identify C++ classes and their constituent methods?
  - OOAnalyzer classes may not naturally correspond to ground truth classes
  - New metric: edit distance

• How well does OOAnalyzer classify methods as constructors, destructors, and virtual methods?
## OOAnalyzer Method Classification on ObjDigger Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Constructor</th>
<th>Recall</th>
<th>Prec.</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clmg</td>
<td></td>
<td>44/51</td>
<td>44/53</td>
<td>0.85</td>
</tr>
<tr>
<td>Light-pop3-smtp</td>
<td></td>
<td>41/52</td>
<td>41/44</td>
<td>0.85</td>
</tr>
<tr>
<td>Optionparser</td>
<td></td>
<td>10/11</td>
<td>10/10</td>
<td>0.95</td>
</tr>
<tr>
<td>PicoHttpD</td>
<td></td>
<td>117/142</td>
<td>117/126</td>
<td>0.87</td>
</tr>
<tr>
<td>X3c</td>
<td></td>
<td>6/7</td>
<td>6/6</td>
<td>0.92</td>
</tr>
</tbody>
</table>

6 of the 7 constructors were identified

Of the 6 constructors identified, 6 were correct

\[
2 \times \frac{\text{recall} \times \text{precision}}{\text{recall} + \text{precision}}
\]
<table>
<thead>
<tr>
<th>Program</th>
<th>Constructor</th>
<th></th>
<th></th>
<th>Destructor</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox.exe</td>
<td>40/51</td>
<td>40/54</td>
<td>0.76</td>
<td>1/39</td>
<td>1/1</td>
<td>0.05</td>
<td>18/33</td>
<td>18/18</td>
<td>0.71</td>
<td>85/101</td>
<td>85/98</td>
</tr>
<tr>
<td>Log4cpp Debug</td>
<td>192/209</td>
<td>192/197</td>
<td>0.95</td>
<td>40/118</td>
<td>40/40</td>
<td>0.51</td>
<td>18/18</td>
<td>18/18</td>
<td>1.00</td>
<td>84/101</td>
<td>84/86</td>
</tr>
<tr>
<td>Log4cpp Release</td>
<td>135/165</td>
<td>135/170</td>
<td>0.81</td>
<td>24/73</td>
<td>24/36</td>
<td>0.44</td>
<td>18/21</td>
<td>18/18</td>
<td>0.92</td>
<td>84/101</td>
<td>84/86</td>
</tr>
<tr>
<td>muParser Debug</td>
<td>293/325</td>
<td>293/314</td>
<td>0.92</td>
<td>28/156</td>
<td>28/30</td>
<td>0.30</td>
<td>12/12</td>
<td>12/13</td>
<td>0.96</td>
<td>35/47</td>
<td>35/43</td>
</tr>
<tr>
<td>muParser Release</td>
<td>197/252</td>
<td>197/269</td>
<td>0.76</td>
<td>15/91</td>
<td>15/21</td>
<td>0.27</td>
<td>12/14</td>
<td>12/13</td>
<td>0.89</td>
<td>35/47</td>
<td>35/37</td>
</tr>
<tr>
<td>MySQL cfg_editor.dll</td>
<td>260/290</td>
<td>260/311</td>
<td>0.87</td>
<td>107/281</td>
<td>107/111</td>
<td>0.55</td>
<td>69/69</td>
<td>69/69</td>
<td>1.00</td>
<td>321/427</td>
<td>321/325</td>
</tr>
<tr>
<td>MySQL mysql.exe</td>
<td>282/314</td>
<td>282/341</td>
<td>0.86</td>
<td>115/300</td>
<td>115/121</td>
<td>0.55</td>
<td>75/75</td>
<td>75/75</td>
<td>1.00</td>
<td>341/453</td>
<td>341/345</td>
</tr>
<tr>
<td>TinyXML Debug</td>
<td>53/60</td>
<td>53/57</td>
<td>0.91</td>
<td>0/39</td>
<td>0/3</td>
<td>0.00</td>
<td>24/24</td>
<td>24/24</td>
<td>1.00</td>
<td>101/119</td>
<td>101/102</td>
</tr>
<tr>
<td>TinyXML Release</td>
<td>49/60</td>
<td>49/53</td>
<td>0.87</td>
<td>27/39</td>
<td>27/36</td>
<td>0.72</td>
<td>24/24</td>
<td>24/24</td>
<td>1.00</td>
<td>101/119</td>
<td>101/103</td>
</tr>
</tbody>
</table>
Conclusion: OOAnalyzer

• Can scalably and statically recover abstractions about all classes and all methods from C++ executables

• Combines a light-weight symbolic analysis with a Prolog-based reasoning system that enables hypothetical reasoning about ambiguous scenarios

• Recovers 70% of classes/methods or more on most cleanware and malware

• Classifies constructors, virtual methods, and virtual function tables with high accuracy

Open source tool in the Pharos binary analysis framework:
https://github.com/cmu-sei/pharos
Questions?

**Presenter**
Edward J. Schwartz  
Research Scientist  
Carnegie Mellon University  
SEI CERT/CC

**Contact**
Email: eschwartz@cert.org

Open source tool in the Pharos binary analysis framework:  
[https://github.com/cmu-sei/pharos](https://github.com/cmu-sei/pharos)
finalClass(ClassID, VFTable, MinSize, MaxSize, RealDestructor, MethodList).

finalClass(0x412190, 0, 0x5, 0x5, 0, [0x412190, 0x4122f0, 0x4123c0, 0x4124c0]).

finalClass(0x412400, 0, 0x4, 0x4, 0, [0x412400]).

finalClass(0x417824, 0x417824, 0x54, 0x54, 0, [0x411860, 0x411970, 0x4119f0]).

finalClass(0x417860, 0x417860, 0xc, 0xc, 0, [0x4117e0, 0x411a70, 0x411ac0]).

finalClass(0x41787c, 0x41787c, 0x4, 0x4, 0, [0x4118f0, 0x411b40, 0x411b90]).

finalClass(0x41789c, 0x41789c, 0x4, 0x4, 0, [0x412800, 0x412830]).

finalClass(0x41b2f8, 0, 0, 0, 0, [0x41b2f8, 0x41b2fc, 0x41b324, 0x41b330]).

finalClass(0x41b304, 0, 0, 0, 0, [0x41b304, 0x41b310, 0x41b348]).

finalClass(0x41b308, 0, 0, 0, 0, [0x41b308, 0x41b320, 0x41b33c, 0x41b340]).

finalClass(0x41b30c, 0, 0, 0, 0, [0x41b30c, 0x41b31c, 0x41b334, 0x41b338]).

finalClass(0x41b328, 0, 0, 0, 0, [0x41b328, 0x41b32c, 0x41b344]).
finalInheritance(DerivedClassID, BaseClassID, ObjectOffset, VTable, Virtual).
finalInheritance(0x412190, 0x412400, 0, 0, false).
finalInheritance(0x417824, 0x417860, 0, 0x417824, false).
finalInheritance(0x417824, 0x41787c, 0xc, 0x417814, false).
finalMember(Class, Offset, Sizes, Certainty).
finalMember(0x412190, 0x4, [0x1], certain).
finalMember(0x412400, 0, [0x4], certain).
finalMember(0x417824, 0x50, [0x4], certain).
finalMember(0x417860, 0, [0x4], certain).
finalMember(0x41787c, 0, [0x4], certain).
finalMember(0x41789c, 0, [0x4], certain).
finalMethodProperty(Method, Type, Certainty).
finalMethodProperty(0x411860, constructor, certain).
finalMethodProperty(0x4118f0, virtual, certain).
finalMethodProperty(0x411970, virtual, certain).
finalMethodProperty(0x4119f0, virtual, certain).
finalMethodProperty(0x411ac0, virtual, certain).
finalMethodProperty(0x411b40, constructor, certain).
finalMethodProperty(0x411b90, virtual, certain).
finalMethodProperty(0x412190, constructor, certain).
finalMethodProperty(0x412400, constructor, certain).
finalMethodProperty(0x412800, constructor, certain).
finalMethodProperty(0x412830, virtual, certain).