TypeMiner: Recovering Types in Binary Code using Machine Learning

Alwin Maier    Hugo Gascon    Christian Wressnegger    Konrad Rieck, DIMVA 2019

Institute of System Security, TU Braunschweig
Decompilation

- decompilers profit from type information
- manual analysis of *usage patterns*
- What about *automatization*?
Motivation

Decompilation
- decompilers profit from type information
- manual analysis of *usage patterns*
- What about *automatization*?

W/o Type Information

```c
ulong idx = 0;
int *pt1, *pt2;
if (0 < (int) len)
    do {
        pt1 = *(int **) (pts1 + idx * 8);
        pt2 = *(int **) (pts2 + idx * 8);
        *pt1 = *pt1 + *pt2;
        *(double *) (pt1 + 2) =
            *(double *) (pt1 + 2) +
            *(double *) (pt2 + 2);
        idx = idx + 1;
    } while (len != idx);
```
Motivation

Decompilation
- decompilers profit from type information
- manual analysis of usage patterns
- What about automatization?

W/ Type Information

```c
ulong idx = 0;
struct point *pt1, *pt2;
if (0 < len)
    do {
        pt1 = pts1[idx];
        pt2 = pts2[idx];
        pt1->x = pt1->x + pt2->x;
        pt1->y = pt2->y + pt1->y;
        idx = idx + 1;
    } while (len != idx);
```
Manual Rules vs. Machine Learning

**Manual Rules**
- requires human expertise
- process rules manually
- requires profound knowledge of the respective ISA

**Machine Learning**
- learn type recovery rules automatically
- process rules automatically
- training data can be generated for the respective ISA
TypeMiner Overview

Binary Code Analysis

- data dependence analysis
- *data object graph*; modeling (indirect) data dependencies
- extraction of *data object traces* by traversing the data object graph
TypeMiner Overview

Machine Learning

- type recovery of data objects (i.e. variables and parameters)
- classification model based on embedded data object traces
- prediction of data types in multiple classification steps

```
TypeMiner: Recovering Types in Binary Code using Machine Learning

Institute of System Security

DIMVA 2019 | Alwin Maier, Hugo Gascon, Christian Wressnegger, Konrad Rieck | Page 5
TypeMiner: Recovering Types in Binary Code using Machine Learning
```
Trace Extraction and Normalization

Trace Extraction
- represent usage patterns
- start at access locations of data objects
- traverse the data object graph

Trace Normalization
- strip irrelevant information
- normalize each instruction in trace
- consider previous instructions

```
movsd | loc_w8 | obj(0)_w8
addsd | obj(0)_w8 | loc_w8
movsd | loc_w8 | obj(0)_w8
```
Training Data
- training data is based on real software projects written in C
- traces are extracted for each data object in the compiled program
- debugging information is used to label training data

Training Process
- embedding of traces in a vector space using an n-gram model
- classifiers (SVMs, random forests) are trained for each classification step
- cross-validation; grouped by compiled programs
Prediction Phase
- extract, normalize, and embed all traces of an unknown data object
- merge all traces into a single vector representation
- run through all classification steps to recover the data type
Evaluation

Dataset
- 14 popular open-source software projects
- optimized release configuration
- compiled for X86-64 architecture
- ground truth data types from debugging information

Experimental Setup
- 13 binary programs for training, one for testing
- evaluation of each classification step
- evaluation of different trace lengths
- comparison with manually created rules
Results: Overview

Accuracy
- total amount of recovered types
- average over all software projects
- each sample contributes equally to the score

Sensitivity
- amount of correctly classified data objects per type
- average over all software projects
- each type contributes equally to the score
Results: Pointer Types

**Pointer Types**

- Performance increases with length of traces.
- Very good performance for pointer to char and pointer to structures.
- TypeMiner fails to detect function pointers.
- Moderate performance for array types and other pointer types.

```
<table>
<thead>
<tr>
<th>max_t</th>
<th>array types</th>
<th>ptr2struct</th>
<th>ptr2char</th>
<th>other ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥1</td>
<td>0.0</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>≥2</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>≥3</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1.0</td>
</tr>
<tr>
<td>≥4</td>
<td>0.6</td>
<td>0.8</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>≥5</td>
<td>0.8</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

---

TypeMiner: Recovering Types in Binary Code using Machine Learning

 Institute of System Security

DIMVA 2019 | Alwin Maier, Hugo Gascon, Christian Wressnegger, Konrad Rieck | Page 11
Results: Arithmetic Types

Arithmetic Types
- performance increases with length of traces
- very good performance for int, long int, and double
- short int incorrectly predicted as int or long int
- good performance for char and _Bool

<table>
<thead>
<tr>
<th>Type</th>
<th>min_t</th>
<th>max_t</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Bool</td>
<td></td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>char</td>
<td></td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>short int</td>
<td></td>
<td>1</td>
<td>0.6</td>
</tr>
<tr>
<td>int</td>
<td></td>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>long int</td>
<td></td>
<td>1</td>
<td>0.6</td>
</tr>
<tr>
<td>double</td>
<td></td>
<td>1</td>
<td>0.4</td>
</tr>
</tbody>
</table>

DIMVA 2019 | Alwin Maier, Hugo Gascon, Christian Wressnegger, Konrad Rieck | Page 12
TypeMiner: Recovering Types in Binary Code using Machine Learning
Signed vs. Unsigned
- \( \approx 76\% \) accuracy

Pointer vs. Arithmetic Types
- \( \approx 92\% \) accuracy
- detection of pointer types without being dereferenced

Omitted Types
- union, enumeration, “void *”

Encountered Dilemmas
- different types, same semantic
- array of type \( T \) vs. pointer to type \( T \)
- structured data types
Summary

Recovery of Data Types using Machine Learning

- extraction of traces (characteristic traits) in compiled C code
- automatic identification of data types using machine learning
- recovery of data types in multiple classification steps

Results

- evaluation with 14 real world software projects
- evaluation on X86-64 architecture
- correct recovery of data types in 76 % – 93 %
Thanks for your attention. Questions?
<table>
<thead>
<tr>
<th>Program</th>
<th># Data Obj.</th>
<th># Instr.</th>
<th>Program</th>
<th># Data Obj.</th>
<th># Instr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bash</td>
<td>6496</td>
<td>157 K</td>
<td>gzip</td>
<td>424</td>
<td>10 K</td>
</tr>
<tr>
<td>bc</td>
<td>422</td>
<td>10 K</td>
<td>indent</td>
<td>174</td>
<td>10 K</td>
</tr>
<tr>
<td>bison</td>
<td>2470</td>
<td>58 K</td>
<td>less</td>
<td>961</td>
<td>20 K</td>
</tr>
<tr>
<td>cflow</td>
<td>768</td>
<td>18 K</td>
<td>libpng</td>
<td>1968</td>
<td>33 K</td>
</tr>
<tr>
<td>gawk</td>
<td>3472</td>
<td>98 K</td>
<td>nano</td>
<td>1526</td>
<td>34 K</td>
</tr>
<tr>
<td>grep</td>
<td>1227</td>
<td>24 K</td>
<td>sed</td>
<td>709</td>
<td>15 K</td>
</tr>
<tr>
<td>gtypist</td>
<td>145</td>
<td>5 K</td>
<td>wget</td>
<td>2720</td>
<td>58 K</td>
</tr>
</tbody>
</table>
TypeMiner: Recovering Types in Binary Code using Machine Learning

DIMVA 2019 | Alwin Maier, Hugo Gascon, Christian Wressnegger, Konrad Rieck | Page 17

Institute of System Security
TypeMiner: Recovering Types in Binary Code using Machine Learning

Step 1
Step 2a
Step 2b
Step 3

Accuracy

Sensitivity

DIMVA 2019 | Alwin Maier, Hugo Gascon, Christian Wressnegger, Konrad Rieck | Page 18
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Prob-based</th>
<th></th>
<th>Rule-based</th>
<th></th>
<th>TypeMiner</th>
<th></th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prec.</td>
<td>Recall</td>
<td>Prec.</td>
<td>Recall</td>
<td>Prec.</td>
<td>Recall</td>
<td></td>
</tr>
<tr>
<td>pointer types</td>
<td>0.45</td>
<td>0.45</td>
<td>0.88</td>
<td>0.90</td>
<td>0.93</td>
<td>0.91</td>
<td>3296</td>
</tr>
<tr>
<td>arithmetic types</td>
<td>0.55</td>
<td>0.55</td>
<td>0.91</td>
<td>0.90</td>
<td>0.93</td>
<td>0.95</td>
<td>3990</td>
</tr>
<tr>
<td>micro avg.</td>
<td>0.50</td>
<td>0.50</td>
<td>0.90</td>
<td>0.90</td>
<td>0.93</td>
<td>0.93</td>
<td>7286</td>
</tr>
<tr>
<td>macro avg.</td>
<td>0.50</td>
<td>0.50</td>
<td>0.90</td>
<td>0.90</td>
<td>0.93</td>
<td>0.93</td>
<td>7286</td>
</tr>
<tr>
<td>1-byte integer</td>
<td>0.01</td>
<td>0.01</td>
<td>0.06</td>
<td>0.27</td>
<td>0.48</td>
<td>0.73</td>
<td>22</td>
</tr>
<tr>
<td>2-byte integer</td>
<td>0.00</td>
<td>0.00</td>
<td>0.33</td>
<td>0.33</td>
<td>–</td>
<td>0.00</td>
<td>3</td>
</tr>
<tr>
<td>4-byte integer</td>
<td>0.70</td>
<td>0.70</td>
<td>0.95</td>
<td>0.90</td>
<td>0.96</td>
<td>0.93</td>
<td>2752</td>
</tr>
<tr>
<td>8-byte integer</td>
<td>0.29</td>
<td>0.29</td>
<td>0.84</td>
<td>0.88</td>
<td>0.86</td>
<td>0.92</td>
<td>1157</td>
</tr>
<tr>
<td>micro avg.</td>
<td>0.57</td>
<td>0.57</td>
<td>0.89</td>
<td>0.89</td>
<td>0.93</td>
<td>0.93</td>
<td>3934</td>
</tr>
<tr>
<td>macro avg.</td>
<td>0.25</td>
<td>0.25</td>
<td>0.54</td>
<td>0.60</td>
<td>0.77</td>
<td>0.64</td>
<td>3934</td>
</tr>
</tbody>
</table>