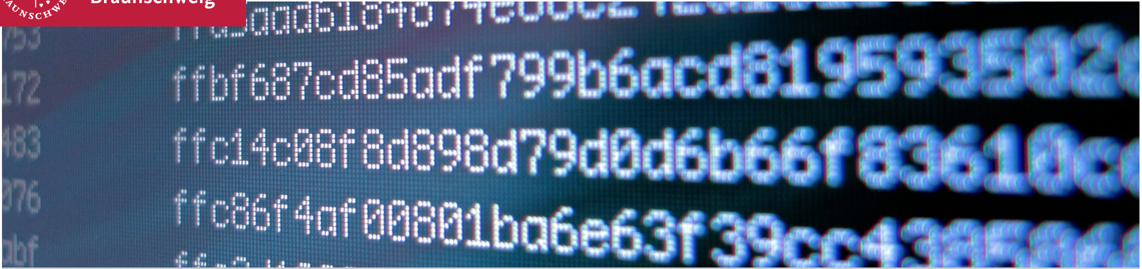
Technische Universität Braunschweig

Institute of System Security



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

Motivation

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

```
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

```
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

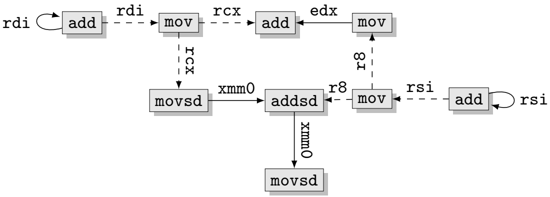
Machine Learning

- 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

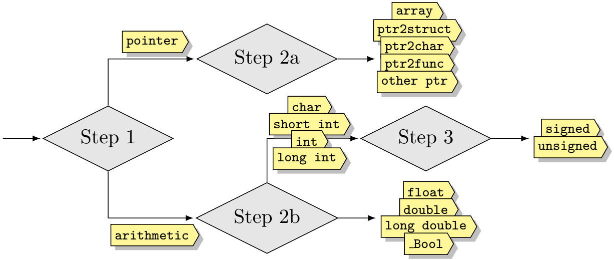


JanusGraph

TypeMiner Overview

Machine Learning

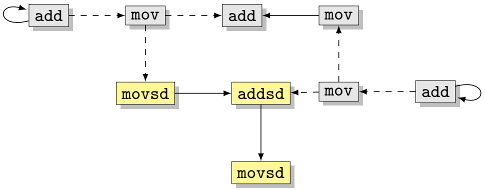
- type recovery of *data objects* (i.e. variables and parameters)
- classification model based on embedded *data object traces*
- prediction of data types in multiple classification steps



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 Phase

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

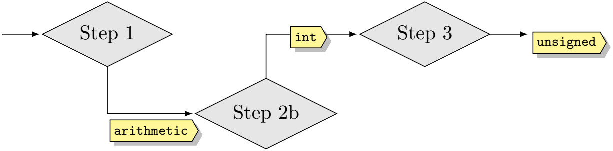
Training Process

- embedding of traces in a vector space using a *n-gram model*
- classifiers (SVMs, random forests) are trained for each classification step
- *cross-validation*; grouped by compiled programs

Type Recovery

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

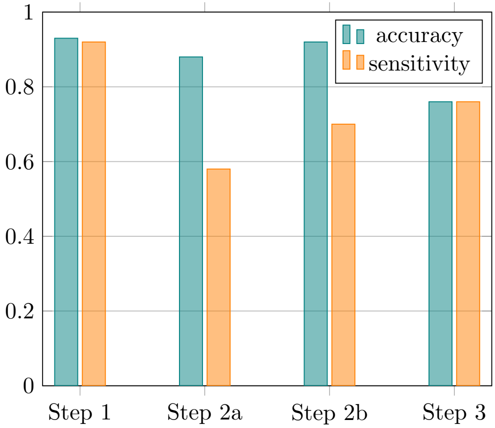
Results: Overview

Accuracy

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

Sensitivity

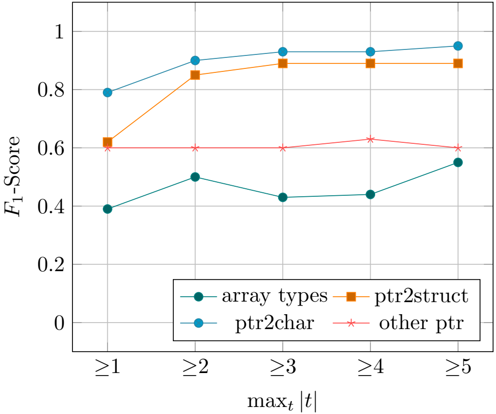
- amount of correctly classified data objects per type
- average over all software projects
- each type contributes equally to the score



Results: Pointer Types

Pointer Types

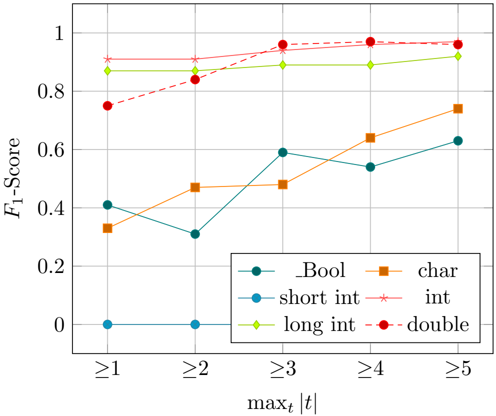
- performance increases with length of traces
- very good performance for pointer to char and pointer to structures
- TypeMiner fails to detect function pointers
- moderate performance for array types and other pointer types



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`



Other Results

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

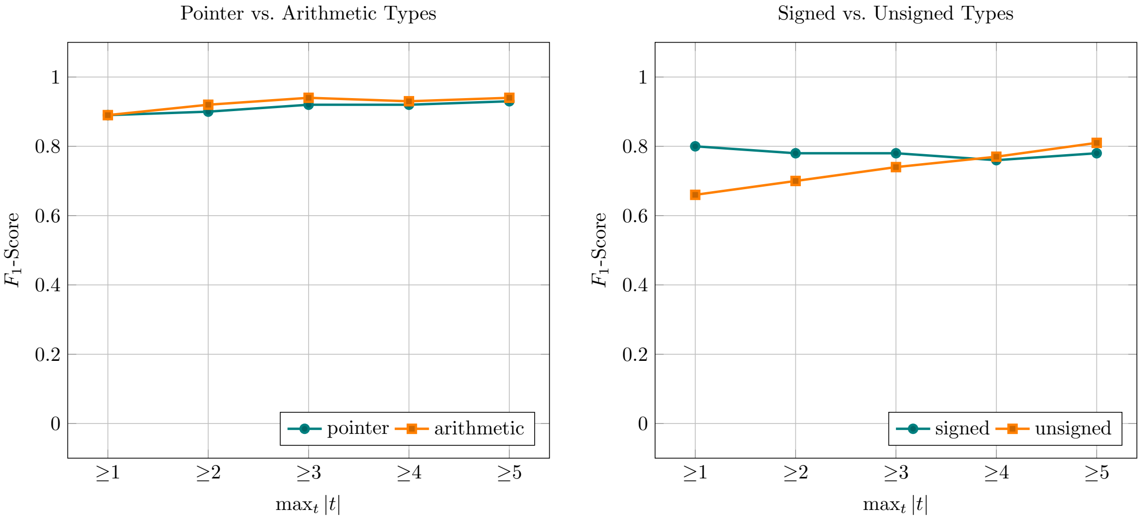
- extraction of traces (characteristic traits) in compiled C code
- automatic identification of data types using machine learning
- recovery of data types in multiple classification steps

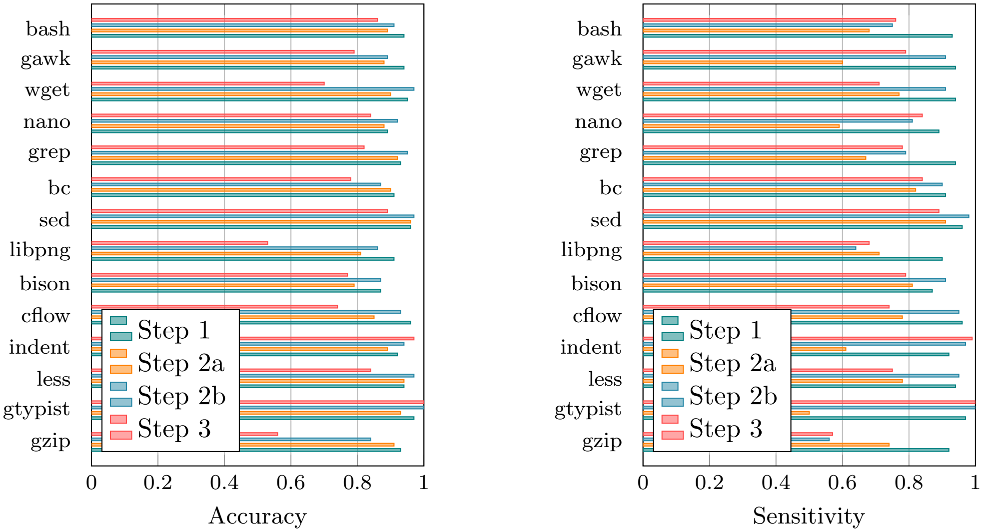
Results

- 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?

Program	# Data Obj.	# Instr.	Program	# Data Obj.	# Instr.
bash	6496	157 K	gzip	424	10 K
bc	422	10 K	indent	174	10 K
bison	2470	58 K	less	961	20 K
cflow	768	18 K	libpng	1968	33 K
gawk	3472	98 K	nano	1526	34 K
grep	1227	24 K	sed	709	15 K
gtypist	145	5 K	wget	2720	58 K


Pointer vs. Arithmetic Types
Signed vs. Unsigned Types
F_1-Score
1
0.8
0.6
0.4
0.2
0
≥1
≥2
≥3
≥4
≥5
$\max_t |t|$
pointer
arithmetic
signed
unsigned


bash
gawk
wget
nano
grep
bc
sed
libpng
bison
cflow
indent
less
gtypist
gzip
Step 1
Step 2a
Step 2b
Step 3
0
0.2
0.4
0.6
0.8
1
Accuracy
Sensitivity

Data Type	Prob-based		Rule-based		TypeMiner		Support
	Prec.	Recall	Prec.	Recall	Prec.	Recall	
pointer types	0.45	0.45	0.88	0.90	0.93	0.91	3296
arithmetic types	0.55	0.55	0.91	0.90	0.93	0.95	3990
micro avg.	0.50	0.50	0.90	0.90	0.93	0.93	7286
macro avg.	0.50	0.50	0.90	0.90	0.93	0.93	7286
1-byte integer	0.01	0.01	0.06	0.27	0.48	0.73	22
2-byte integer	0.00	0.00	0.33	0.33	–	0.00	3
4-byte integer	0.70	0.70	0.95	0.90	0.96	0.93	2752
8-byte integer	0.29	0.29	0.84	0.88	0.86	0.92	1157
micro avg.	0.57	0.57	0.89	0.89	0.93	0.93	3934
macro avg.	0.25	0.25	0.54	0.60	0.77	0.64	3934