Where Did This Code Come From?
Discovering the provenance of program binaries

Nathan Rosenblum
Computer Sciences Department
University of Wisconsin
nater@cs.wisc.edu
http://pages.cs.wisc.edu/~nater/

Joint work with
Barton Miller and Xiaojin Zhu
terroir
vineyard
aging
storage
discovery
restoration
possession
focus of this talk

terroir
vineyard
aging
storage

discovery
restoration
possession
Program provenance
Program provenance

Compiler
- family
- version
- optimization level
- source language
Program provenance

Compiler
- family
- version
- optimization level
- source language

System
- glibc static code
- library imports
Program provenance

Compiler
- family
- version
- optimization level
- source language

System
- glibc static code
- library imports

Link & post-link
- whole-program optimization
- rewriting tools
- obfuscation tools
Program provenance

Compiler
- family
- version
- optimization level
- source language

System
- glibc static code
- library imports

Link & post-link
- whole-program optimization
- rewriting tools
- obfuscation tools

Authorship
Program provenance

Compiler
- family
- version
- optimization level
- source language

System
- glibc static code
- library imports

Link & post-link
- whole-program optimization
- rewriting tools
- obfuscation tools

Authorship
Applications

Debugging remote deployments
Applications

Debugging remote deployments
Applications

Debugging remote deployments

compiler bug?
subtle incompatibility?
Applications

Debugging remote deployments

compiler bug?

subtle incompatibility?

libasound.so.2
libdl.so.2
libstdc++.so.6
libm.so.6
libgcc_s.so.1
libc.so.6
/lib64/ld-linux-x86-64.so.2
librt.so.1
Applications

Debugging remote deployments

- compiler bug?
- subtle incompatibility?

Forensics

- `linux-vdso.so.1`
- `libpthread.so.0`
- `libasound.so.2`
- `libdl.so.2`
- `libstdc++.so.6`
- `libm.so.6`
- `libgcc_s.so.1`
- `libc.so.6`
- `/lib64/ld-linux-x86-64.so.2`
- `librt.so.1`
Applications

Debugging remote deployments

- compiler bug?
- subtle incompatibility?

Forensics

- reverse engineering
- decompiling
- tools, obfuscations?
Applications

Debugging remote deployments

- compiler bug?
- subtle incompatibility?

Forensics

- reverse engineering
- decompiling
- tools, obfuscations?
Outline

Program Modeling

Compiler Toolchain

Authorship Attribution

Future Directions
Outline

Program Modeling

Compiler Toolchain

Authorship Attribution

Future Directions
Digression: finding code

program binary
Digression: finding code
Digression: finding code

program binary
Digression: finding code

… 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 …
Digression: finding code

```
push ebp  mov esp, ebp  sub 0x2c, esp  ...
```

… 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 …

program binary
Digression: finding code

```
… 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 …
```

```
in 0x83,eax in [dx],al sub 0x57,al ...
push ebp mov esp, ebp sub 0x2c, esp ...
```
Digression: finding code

Model compiler-specific “function entry points”

Compute max-likelihood labels \( P(\text{addr is FEP}|\ldots, \lambda) \propto \ldots \)

\( F_1 \) from .86 - .99 *depending on compiler*
Instruction-level features

IDIOMS

<push EBP ; * ; mov ESP, EBP>
<mov [IMM], RAX ; sub [IMM], RAX>
Instruction-level features

IDIOMS

single-instruction wildcard

\[ \text{push EBP ; * ; mov ESP, EBP} \]

\[ \text{mov [IMM], RAX ; sub [IMM], RAX} \]

opcode class
abstraction

hidden immediates
Instruction-level features

IDIOMS

single-instruction wildcard

<push EBP ; * ; mov ESP, EBP>

<mov [IMM], RAX ; sub [IMM], RAX>

opcode class abstraction

hidden immediates

N-GRAMS

<4889c2be>  <018b45f8>  <8d45f8>

4-grams

3-grams
Binary code model

... 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 ...

<push EBP ; * ; mov ESP, EBP>  <mov [IMM], RAX ; sub [IMM], RAX>
Binary code model

… 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 …

\[
\text{\textless push EBP ; * ; mov ESP, EBP\textgreater } \quad \text{\textless mov [IMM], RAX ; sub [IMM], RAX\textgreater }
\]
Binary code model

... 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 ...

\[\text{push EBP} ; * ; \text{mov ESP, EBP} \] \quad \langle \text{mov [IMM], RAX ; sub [IMM], RAX} \rangle

Control Flow Graph

layout, block contents
Binary code model

... 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 ...

\langle \text{push EBP} ; * ; \text{mov ESP, EBP} \rangle \quad \langle \text{mov [IMM], RAX} ; \text{sub [IMM], RAX} \rangle
Binary code model

... 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 ...

〈push EBP ; * ; mov ESP, EBP〉 〈mov [IMM], RAX ; sub [IMM], RAX〉

Control Flow Graph
Call Graph
External Libraries
Graphlets
Graphlets

code element nodes (e.g. basic blocks)

typed edges (branch, call, etc.)
Graphlets

code element nodes (e.g. basic blocks)

typed edges (branch, call, etc.)

node colors
Graphlets

code element nodes (e.g. basic blocks)

typed edges (branch, call, etc.)

node colors

Ex: instruction summary graphlets

Color bit field

14 instruction categories

2^{14} possible colors

sparse in practice
Modeling approach
Modeling approach

- some amount of code
- feature vector
Modeling approach

some amount of code

"decompiles to <push ebp,..."

"contains 27 occurrences of"

feature vector
Modeling approach

some amount of code

"decompiles to <push ebp, ..."

"contains 27 occurrences of "

feature vector
Feature selection

training programs
Feature selection

training programs
Feature selection

(training programs)

(condor)
Feature selection

(training programs)

(conda)

n-grams
idioms
graphlets
libcalls
combined
supergr.
call gr.

Acc. 1.0
500 2500 4500
Features
Outline

Program Modeling

Compiler Toolchain

Authorship Attribution

Future Directions
Compiler toolchain

- C
- C++
- F77

- GCC
- Visual Studio
- Intel

- Optimized
- Not optimized

- 3.4
- 4.2
- 4.4
- 2003
- 2005
- 2008

- 10
- 11
Compiler toolchain

- GCC (C)
- C++
- F77

Platforms and versions:
- 3.4, 4.2, 4.4
- 10, 11

Optimized vs. not optimized
Compiler toolchain

- C
- C++
- F77

GCC
Microsoft Visual Studio
Intel

- Optimized
- Not optimized

- 3.4
- 4.2
- 4.4
- 2003
- 2005
- 2008
- 10
- 11
Compiler toolchain

- C
- C++
- F77

GCC

Visual Studio

Intel

3.4 4.2 4.4 2003 2005 2008

optimized  not optimized

10 11
Byte-sequence model

program binary
Byte-sequence model

GCC

ICC

ICC

GCC

program binary
Byte-sequence model

program binary
Byte-sequence model

… 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 …
Byte-sequence model

\[\text{program binary}\]

\[\ldots\ 55\ 89\ e5\ 83\ ec\ 2c\ 57\ 56\ 53\ 8b\ 45\ 0c\ 8b\ 00\ a3\ 90\ a3\ 05\ 08\ 85\ c0\ 74\ 2b\ 83\ c4\ \ldots\]
Byte-sequence model

... 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 ...

program binary
Byte-sequence model

sequence labels $y_{i-1}$ $y_i$ $y_j$ $y_{j+1}$ $\in \{icc,gcc,\ldots,\text{data}\}$

program binary

... 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 ...
Byte-sequence model

program binary

... 55 89 e5 83 ec 2c 57 56 53 8b 45 0c 8b 00 a3 90 a3 05 08 85 c0 74 2b 83 c4 ...

sequence labels

$y_{i-1}$ $y_i$ ...

$y_j$ $y_{j+1}$ $\in \{icc, gcc, ..., data\}$

Compiler labels modeled as CRF...
Digression: Conditional Random Fields

\[ P(Y|X) \propto \exp \left( \sum_{i=1}^{K} \sum_{u \in \mathcal{U}} \lambda_{y_i,u} \cdot f_u(x_i) \right) \]
Digression: Conditional Random Fields

\[
P(Y|X) \propto \exp \left( \sum_{i=1}^{K} \sum_{u \in U} \lambda_{y_i,u} \cdot f_u(x_i) \right)
\]
Digression: Conditional Random Fields

\[ P(Y|X) \propto \exp \left( \sum_{i=1}^{K} \sum_{u \in U} \lambda_{y_i,u} \cdot f_u(x_i) \right) \]

labels
weights (learned)
evidence
feature functions

Idiom feature function

\[ f_u = \begin{cases} 
1 & \text{if } x_i \text{ decompiles to idiom } u \\ 
0 & \text{otherwise} 
\end{cases} \]
Digression: Conditional Random Fields

\[ P(Y|X) \propto \exp \left( \sum_{i=1}^{K} \sum_{u \in U} \lambda_{y_i,u} \cdot f_u(x_i) \right) \]

weights (learned)

labels

evidence

weights (learned)

evidence

labels

feature functions

Linear chain CRF

Idiom feature function

\[ f_u = \begin{cases} 
1 & \text{if } x_i \text{ decompiles to idiom } u \\
0 & \text{otherwise} 
\end{cases} \]
Byte-sequence model

sequence labels $y_{i-1}$ $y_i$ $y_j$ $y_{j+1} \in \{icc,gcc,\ldots,data\}$ sequence labeling

94% accuracy labeling mixed-compiler sequences
+18% accuracy increase in function finding
Toolchain details

compiler family
GNU, Intel,
Microsoft
### Toolchain details

<table>
<thead>
<tr>
<th>source language</th>
<th>compiler family</th>
<th>version</th>
<th>optimization level</th>
</tr>
</thead>
<tbody>
<tr>
<td>C, C++, Fortran</td>
<td>GNU, Intel, Microsoft</td>
<td>[several]</td>
<td>low, high</td>
</tr>
<tr>
<td>source language</td>
<td>compiler family</td>
<td>version</td>
<td>optimization level</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>---------</td>
<td>--------------------</td>
</tr>
<tr>
<td>C, C++, Fortran</td>
<td>GNU, Intel, Microsoft</td>
<td>[several]</td>
<td>low, high</td>
</tr>
<tr>
<td>source language</td>
<td>compiler family</td>
<td>version</td>
<td>optimization level</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>---------</td>
<td>--------------------</td>
</tr>
<tr>
<td>C, C++, Fortran</td>
<td>GNU, Intel, Microsoft</td>
<td>[several]</td>
<td>low, high</td>
</tr>
</tbody>
</table>
## Toolchain details

<table>
<thead>
<tr>
<th>source language</th>
<th>compiler family</th>
<th>version</th>
<th>optimization level</th>
<th>functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>C, C++, Fortran</td>
<td>GNU, Intel, Microsoft</td>
<td>[several]</td>
<td>low, high</td>
<td>idioms + graphlets</td>
</tr>
</tbody>
</table>
## Toolchain details

<table>
<thead>
<tr>
<th></th>
<th>Individually (SVM)</th>
<th>Linear CRF</th>
<th>Joint CRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>.910</td>
<td>.999</td>
<td>-</td>
</tr>
<tr>
<td>Compiler</td>
<td>.987</td>
<td>.998</td>
<td>.992</td>
</tr>
<tr>
<td>Optimization</td>
<td>.971</td>
<td>.993</td>
<td>.982</td>
</tr>
<tr>
<td>Version</td>
<td>.616</td>
<td>.910</td>
<td>.845</td>
</tr>
</tbody>
</table>
## Toolchain details

<table>
<thead>
<tr>
<th></th>
<th>Individually (SVM)</th>
<th>Linear CRF</th>
<th>Joint CRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>.910</td>
<td>.999</td>
<td>-</td>
</tr>
<tr>
<td>Compiler</td>
<td>.987</td>
<td>.998</td>
<td>.992</td>
</tr>
<tr>
<td>Optimization</td>
<td>.971</td>
<td>.993</td>
<td>.982</td>
</tr>
<tr>
<td>Version</td>
<td>.616</td>
<td>.910</td>
<td>.845</td>
</tr>
</tbody>
</table>

*slow, approximate inference*
## Toolchain details

<table>
<thead>
<tr>
<th></th>
<th>Individually (SVM)</th>
<th>Linear CRF</th>
<th>Joint CRF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
<td>.910</td>
<td>.999</td>
<td>-</td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
<td>.987</td>
<td>.998</td>
<td>.992</td>
</tr>
<tr>
<td><strong>Optimization</strong></td>
<td>.971</td>
<td>.993</td>
<td>.982</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td>.616</td>
<td>.910</td>
<td>.845</td>
</tr>
</tbody>
</table>

*slow, approximate inference possibly inconsistent labels*

C GCC 2005 LO
F77 MSVS 2008 HI

[ISSTA 2011]
## Toolchain details

<table>
<thead>
<tr>
<th></th>
<th>Individually (SVM)</th>
<th>Linear CRF</th>
<th>Joint CRF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
<td>.910</td>
<td>.999</td>
<td>-</td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
<td>.987</td>
<td>.998</td>
<td>.992</td>
</tr>
<tr>
<td><strong>Optimization</strong></td>
<td>.971</td>
<td>.993</td>
<td>.982</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td>.616</td>
<td>.910</td>
<td>.845</td>
</tr>
</tbody>
</table>

*slow, approximate inference possibly inconsistent labels*

*allows partial labels*

C GCC 2005 LO F77 MSVS 2008 HI
# Toolchain details

<table>
<thead>
<tr>
<th></th>
<th>Individually (SVM)</th>
<th>Linear CRF</th>
<th>Joint CRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>.910</td>
<td>.999</td>
<td>-</td>
</tr>
<tr>
<td>Compiler</td>
<td>.987</td>
<td>.998</td>
<td>.992</td>
</tr>
<tr>
<td>Optimization</td>
<td>.971</td>
<td>.993</td>
<td>.982</td>
</tr>
<tr>
<td>Version</td>
<td>616</td>
<td>.910</td>
<td>.845</td>
</tr>
</tbody>
</table>

* slow, approximate inference possibly inconsistent labels

allows partial labels

C GCC 2005 LO
F77 MSVS 2008 HI
Outline

Program Modeling

Compiler Toolchain

Authorship Attribution

Future Directions
Program authorship
Program authorship

for(int i=0; i<sz;++i) {
    // etc
}

std::vector<int>::iterator it = foo.begin();
while(it != foo.end()) {
    // etc
Program authorship

```c++
for(int i=0; i<sz;++i) {
    // etc
}
```

```c++
std::vector<int>::iterator it = foo.begin();
while(it != foo.end()) {
    // etc
}
```
Long-range control flow
Long-range control flow

basic blocks
Long-range control flow

insn graphlets

basic blocks
Long-range control flow

Insn graphlets

Basic blocks

Merged instruction summaries

Supergraphlets
Interprocedural graphlets
Interprocedural graphlets

color ⇒ target

anonymous internal methods

FPRINTF

[local]

FOPEN
Program-author dataset

Ideal:

1. Author labels
2. Parallel corpus
3. Linguistic homogeneity
Program-author dataset

Ideal:

1. Author labels  
2. Parallel corpus  
3. Linguistic homogeneity

code jam
hello, world!

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON
COMPUTER SCIENCES
(CS 537)
Program-author dataset

Ideal:

1. Author labels
2. Parallel corpus
3. Linguistic homogeneity

code jam

several contest years
C and C++ programs
8-16 programs per contestant

C programs
some provided/template code

4 16

24
Author attribution

- 391,056 N-grams
- 54,705 idioms
- 152 library calls
- 37,358 graphlets
- 117,997 supergraphlets
- 8,062 call graphlets

1,900 features
Author attribution

- 391,056 N-grams
- 54,705 idioms
- 152 library calls
- 37,358 graphlets
- 117,997 supergraphlets
- 8,062 call graphlets

1,900 features

20 random programmers
Author attribution

391,056 N-grams 37,358 graphlets
54,705 idioms 117,997 supergraphlets
152 library calls 8,062 call graphlets

1,900 features

CJ 2009  CJ 2010  CS 537
Exact 77.8% 76.8% 38.4%

20 random programmers
x 20 experiments

25
Author attribution

- 391,056 N-grams
- 54,705 idioms
- 152 library calls
- 37,358 graphlets
- 117,997 supergraphlets
- 8,062 call graphlets
- 1,900 features

20 random programmers x 20 experiments

Top-5

- CJ 2009: 94.7%
- CJ 2010: 93.7%
- CS 537: 84.3%

Exact

- CJ 2009: 77.8%
- CJ 2010: 76.8%
- CS 537: 38.4%

Students have less distinctive styles?
Author attribution

391,056 N-grams  37,358 graphlets
54,705 idioms      117,997 supergraphlets
152 library calls  8,062 call graphlets

1,900 features

CJ 2009    CJ 2010    CS 537
Exact  77.8%  76.8%  38.4%
Top-5  94.7%  93.7%  84.3%

Students have less distinctive styles?
1. CS537 has much less data
2. Template code + instructor guidance confound results

Students have less distinctive styles?
20 random programmers
x 20 experiments
Style clustering

Programs, no training data
Style clustering

Programs, no training data

Conclude:
Style clustering

Programs, no training data

Conclude:
Distance metrics
Distance metrics
Distance metrics
Distance metrics

Mahalanobis distance

\[ D_A(x_a, x_b) = \sqrt{(x_a - x_b)^T A (x_a - x_b)} \]
Distance metrics

Mahalanobis distance

\[ D_A(x_a, x_b) = \sqrt{(x_a - x_b)^T A(x_a - x_b)} \]

equivalently:

\[ x^T \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \]

distance metric
Distance metrics

Mahalanobis distance

\[ D_A(x_a, x_b) = \sqrt{(x_a - x_b)^T A (x_a - x_b)} \]

Equivalently:

\[ x^T \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \]

How do we get \( A \)?
Transfer learning
Transfer learning
Transfer learning
Transfer learning

Large-margin Nearest Neighbors (LMNN)
Weinberger, Saul 2009

semi-definite program 😞
one-time cost 😊
Clustering (Code Jam 2010)

Compare Euclidean vs. Mahalanobis distance

![Graphs showing cluster improvement for training and test set authors.]

Cluster improvement (%)

0  10  20  30

2  30  60

Training set authors

Cluster improvement (%)

0  10  20  30

2  30  60

Test set authors
Clustering (Code Jam 2010)

Compare Euclidean vs. Mahalanobis distance

better metric
Clustering (Code Jam 2010)

Compare Euclidean vs. Mahalanobis distance

![Graph showing cluster improvement vs. training set authors](image1.png)

![Graph showing cluster improvement vs. test set authors](image2.png)

- Better metric
- Stable relative improvement
Clustering (Code Jam 2010)

Compare Euclidean vs. Mahalanobis distance

Average .723 NMI clustering 10 authors’ programs

[0,1] → 1 is better
Outline

Program Modeling

Compiler Toolchain

Authorship Attribution

Future Directions
Component models

semi-open world provenance

component sharing (e.g. command and control)
programmer movement between groups
Component models

semi-open world provenance

component sharing (e.g. command and control)
programmer movement between groups

mixture of styles
Component models

semi-open world provenance

component sharing (e.g. command and control)
programmer movement between groups

mixture of styles

style vs. functionality?
infinite mixture models
interpreting style clusters
Social code networks
Social code networks

program binaries
Social code networks

program binaries

32
Social code networks
Summing up
Summing up

Use many simple features
Summing up

Use many simple features

Take advantage of structure
Summing up

Use many simple features

Take advantage of structure
questions