De-anonymizing Programmers from Source Code and Binaries

Rachel Greenstadt
@ragreens
Associate Professor
Drexel University

Aylin Caliskan
@aylin_cim
Assistant Professor
George Washington University
Why de-anonymize programmers?
Source code stylometry

Iran confirms death sentence for 'porn site' web programmer.

No technical difference between security-enhancing and privacy-infringing
Source code stylometry

A machine learning classification task

<table>
<thead>
<tr>
<th>Application</th>
<th>Learner</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software forensics</td>
<td>Multiclass</td>
<td>Open world</td>
</tr>
<tr>
<td>Stylometric plagiarism detection</td>
<td>Multiclass</td>
<td>Closed world</td>
</tr>
<tr>
<td>Copyright investigation</td>
<td>Two-class</td>
<td>Closed world</td>
</tr>
<tr>
<td>Authorship verification</td>
<td>Two-class/One-class</td>
<td>One-class open world</td>
</tr>
</tbody>
</table>
Random Forest Classifier

De-anonymizing Programmers

Identifying Programmer Fingerprints

Privacy and Security Implications

Language Processing

Supervised Machine Learning

Source Code

Fuzzy Parsing

int f(int a) {
    if (a < 0)
        a = 0;
    ...

func

param stmt

int a if ...

Random Forest Classifier

A

B

C

D

Supervised

Machine Learning

Privacy and Security

Implications

August 10, 2018
De-anonymizing programmers

Principled method & robust syntactic feature set

<table>
<thead>
<tr>
<th>Application</th>
<th>Classes</th>
<th>Instances</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stylometric plagiarism detection</td>
<td>250 class</td>
<td>2,250</td>
<td>98%</td>
</tr>
<tr>
<td>Large scale de-anonymization</td>
<td>1,600 class</td>
<td>14,400</td>
<td>94%</td>
</tr>
<tr>
<td>Copyright investigation</td>
<td>Two-class</td>
<td>540</td>
<td>100%</td>
</tr>
<tr>
<td>Authorship verification</td>
<td>Two-class/One-class</td>
<td>2,240</td>
<td>91%</td>
</tr>
<tr>
<td>Open world problem</td>
<td>Multi-class</td>
<td>420</td>
<td>96%</td>
</tr>
</tbody>
</table>
Source code stylometry

1,600 contestants – C++

de code jam

preprocessing

fuzzy AST parser

extract features

majority vote

random forest

De-anonymized Programmers
Features

Source code

```c
int foo(int y)
{
    int n = bar(y);
    if (n == 0)
        return 1;
    return (n + y);
}
```

Abstract syntax tree
Case 1: Authorship attribution

• Who is this anonymous programmer?
• Who is Satoshi?
Case 1: Authorship attribution

• If only we had a suspect set for Satoshi…

Train on the suspect set to de-anonymize the initial Bitcoin author

Satoshi = git contributor
Case 1: Authorship attribution

- 94% accuracy in identifying 1,600 authors of 14,400 anonymous program files.

Train on 1,600 authors to identify the authors of 14,400 files

94% accuracy
Case 2: C++ Obfuscation - STUNNIX

```cpp
#ifndef __STL_USE_EXCEPTIONS /* this is conditional preprocessing */
extern void __out_of_range (const char *);
define OUTOF range (cond, msg) \n    do { if (cond) __out_of_range (#cond); } while (0)
#define OUTOF range (cond, msg) assert (! (cond))
#endif

template <class charT, class traits, class Allocator>
basic_string <charT, traits, Allocator> &
basic_string <charT, traits, Allocator>::
replace (size_type pos1, size_type n1,
    const basic_string& str, size_type pos2, size_type n2)
{
    // rather complex body follows
    const size_t len2 = str.length () + 2;
    if (pos1 == 0 && n1 >= length () && pos2 == 0 && n2 >= len2)
```
Case 2: C++ Obfuscation - STUNNIX

Same set of 25 authors with 225 program files

<table>
<thead>
<tr>
<th>Type of Source Code</th>
<th>Classification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original source code</td>
<td>97%</td>
</tr>
<tr>
<td>STUNNIX-Obfuscated source code</td>
<td>97%</td>
</tr>
</tbody>
</table>
Case 2: C Obfuscation - TIGRESS

```c
#include<stdio.h>
int main()
{
    int T,test=1;
    double C,F,X,rate,time;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%lf %lf %lf",&C,&F,&X);
        rate=2.0;
        time=0;
        while(X/rate>C/rate+X/(rate+F))
        {
            time+=C/rate;
            rate+=F;
        }
        time+=X/rate;
        printf("Case %d: %lf\n",test++,time);
    }
    return 0;
}
```
#include<stdio.h>

int main()
{
    int T,test=1;
    double C,F,X,rate,time;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%lf %lf %lf",&C,&F,&X);
        rate=2.0;
        time=0;
        while(X/rate>C/rate+X/(rate+F))
        {
            time+=C/rate;
            rate+=F;
        }
        time+=X/rate;
        printf("Case #\d: %lf\n",test++,time);
    }
    return 0;
}
Case 2: C Obfuscation - TIGRESS

```
#include<stdio.h>

int main()
{
    int T, test=1;
    double C,F,X,r4t, time;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%lf %lf %lf",&C,&F,&X);
        rate=2.0;
        time=0;
        while(X/rate>C/r4t+X/(rate+F))
        {
            time+=C/r4t;
            rate++;
        }
        time+=X/rate;
        printf("Case #\d: %lf\n",test++,time);
    }
    return 0;
}
```

Same set of 20 authors with 180 program files

<table>
<thead>
<tr>
<th></th>
<th>Classification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original C source code</td>
<td>96%</td>
</tr>
<tr>
<td>TIGRESS-Obfuscated source code</td>
<td>67%</td>
</tr>
</tbody>
</table>

August 10, 2018
Case 3: Authorship verification

• Is this source code really written by this programmer?

**Train on 8 files from Mallory and one file from authors A, B, C, D, E, F, G, and H.**

**Test on 6 files that belong to Mallory and 6 files that belong to 6 random authors.**

93% accuracy in 80 sets of experiments
What about executable binaries?

Source Code

```c
#include <cstdio>
#include <algorithm>
using namespace std;
#define For(i,a,b) for(int i = a; i < b; i++)
#define FOR(i,a,b) for(int i = b-1; i >= a; i--)
double nextDouble() {
    double x;
    scanf("%lf", &x);
    return x;
}
int nextInt() {
    int x;
    scanf("%d", &x);
    return x;
}
int n;
double a1[1001], a2[1001];
int main() {
    freopen("D-small-attempt0.in", "r", stdin);
    freopen("D-small.out", "w", stdout);
    int tt = nextInt();
    For(t,1,tt+1) {
        int n = nextInt();
        . . .
    }
    . . .
}
```

Compiled code looks cryptic

```
01000000 00000000 00010000 00000000 00101000 00000000
00000000 00000000 00110100 00000000 00000000 00000000
00000010 00001000 00000000 00000000 00000001 00000000
00000000 00000001 00000000 00000000 00000101 00000000
00000000 00000000 00000100 00000000 00000000 00000000
00000011 00000000 00000000 00000000 00110100 00000001
00000000 00000000 00110100 00000000 00000100 00001000
00000000 00000000 00010011 00000000 00000000 00000000
00000100 00000000 00000000 00000001 00000000 00000000
00000000 00000000 00000001 00000000 00000000 00000000
00000010 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000010 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000010 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
```

. . .
Interview with the LuaBot malware author

Creating a botnet of thousands of routers for DDoS activities

Who are you?

Just some guy who likes programming. I’m not known security researcher/programmer or member of any hack group, so probably best answer for this would be—nobody
Features: Assembly

Disassembly

Assembly unigrams
- test

Assembly bigrams
- eax, 0x0
- cmovs edi, eax

Assembly trigrams
- cmovs edi, eax

Two consecutive assembly lines
- mov eax, 0x0
- cmovs edi, eax
Features: Syntactic

Abstract syntax tree (AST)

```
  func
   decl
      int =
      v0  call
      f0
  if
      pred stmt
      < ...
```

Syntactic features

```
AST unigrams:
  func  decl  if  int
  =   pred  stmt ...

AST bigrams:
  func  func  decl ...
  decl  if  int ...

AST depth: 5
```
Features: Control flow

Control-flow graph (CFG)

Control-flow features

CFG unigrams:

\[ \text{blk1} \quad \text{blk2} \quad \text{blk3} \quad \text{blk4} \ldots \]

CFG bigrams:

\[ \text{blk1} \quad \text{blk1} \quad \text{blk2} \quad \text{blk3} \ldots \]
Dimensionality Reduction

• Information gain criterion
  • Keep features that reduce entropy – see (a)
  • Reduce dimension from ~700,000 to ~2,000
Dimensionality Reduction

- Information gain criterion
  - Keep features that reduce entropy – see (a)
  - Reduce dimension from ~700,000 to ~2,000

- Correlation based feature selection
  - Keep features with low inter-class correlation
  - Reduce dimension from ~2,000 to 53

(a)

(b)
Predictive features

Variable Declaration & Initialization in one line

```
int age = 20;
```

int variable Declaration and Initialization
Optimizations and stripping symbols

<table>
<thead>
<tr>
<th>Number of programmers</th>
<th>Number of training samples</th>
<th>Compiler optimization level</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>8</td>
<td>None</td>
<td>96%</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>1</td>
<td>93%</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>2</td>
<td>89%</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>3</td>
<td>89%</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>Stripped symbols</td>
<td>72%</td>
</tr>
</tbody>
</table>
Obfuscation

1. Bogus control flow insertion

(a) Original code

(b) Obfuscated code

2. Instruction substitution

3. Control flow flattening
Obfuscation

1. Bogus control flow insertion

2. Instruction substitution

Open-LLVM obfuscations reduce de-anonymization accuracy of 100 programmers from 96% to 88%.
Large scale programmer de-anonymization

![Graph showing Correct Classification Accuracy vs Number of Authors]

- 99%
- 96%
- 92%
- 89%
- 85%
- 83%
- 83%
GitHub and Nulled.IO

• De-anonymizing 50 GitHub programmers
  • with 65% accuracy.

• De-anonymizing 6 malicious programmers
  • Nulled.IO hackers and malware authors
  • with 100% accuracy.
Programmer De-anonymization on GitHub

✓ Single authored GitHub repositories
✓ The repository has at least 500 lines of code

<table>
<thead>
<tr>
<th>Type</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>161</td>
</tr>
<tr>
<td>Repositories</td>
<td>439</td>
</tr>
<tr>
<td>Files</td>
<td>3,438</td>
</tr>
<tr>
<td>Repositories / Author</td>
<td>2 - 8</td>
</tr>
<tr>
<td>Files / Author</td>
<td>2 - 344</td>
</tr>
</tbody>
</table>

Compile repositories

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Authors</th>
<th>Total Files</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub</td>
<td>50</td>
<td>542</td>
<td>65%</td>
</tr>
<tr>
<td>GCJ</td>
<td>50</td>
<td>450</td>
<td>97%</td>
</tr>
</tbody>
</table>
Collaborative Code

I will believe that code stylometry works when it can be shown to work on big github commit histories instead of GCJ dataset.

I just heard from an intern at Apple that they disallow her from contributing to open source on her own time. That's illegal, right?
Segment and Account Attribution

● Sometimes we only care who wrote a small piece of code
● Sometimes we want to deanonymize a pseudonymous account
  ○ Without whole files belonging to it, only small pieces
● In these cases, we can only attribute small segments, or “snippets”

● Using the manual feature set
  ○ Large, sparse features (3,407 nonzero out of 369,097 total)
Segment attribution results

73% accuracy
(average sample 4.9 lines of code)
Accuracy vs LOC

<table>
<thead>
<tr>
<th>Samples</th>
<th>min LOC</th>
<th>Programmers</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>38</td>
<td>90</td>
<td>54%</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>90</td>
<td>63%</td>
</tr>
<tr>
<td>10</td>
<td>18</td>
<td>90</td>
<td>76%</td>
</tr>
<tr>
<td>23</td>
<td>8</td>
<td>90</td>
<td>75%</td>
</tr>
<tr>
<td>90</td>
<td>3</td>
<td>90</td>
<td>77%</td>
</tr>
<tr>
<td>150</td>
<td>1</td>
<td>90</td>
<td>75%</td>
</tr>
</tbody>
</table>
Attribute accounts not individual commits?

Works much much better!
- close to 100% after 4 snippets
Deep Learning AST Representations

Using AST features allowed us to get good results.

But....

A Tree is not a feature!
We manually chose features of the ASTs

Abstract syntax tree (AST)

```
  func
   decl
     int
     =
     v0
     call
       f0
     C0
   if
     pred
     stmt
       <
       ...
```

Syntactic features

- **AST unigrams:**
  - `func`, `decl`, `if`, `int`
  - `=`
  - `pred`
  - `stmt`

- **AST bigrams:**
  - `func` `func`
  - `decl`
  - `if`
  - `int`

- **AST depth:** 5
Can a deep neural net do better?

- **Embedding Layer**
  - Map AST nodes to feature vectors

- **Subtree Layers**
  - Learn the structure of the AST
    - Subtree LSTM
    - Subtree BiLSTM (bidirectional)

- **Softmax Layer**
  - Generate a probability distribution of the programmers
Long Short-Term Memory Networks

Recurrent neural networks (RNNs)
- Handle sequential input
- Add feedback loops to remember information

LSTMs add memory cells
- Sequential long-term dependencies
- Use gates to control flow of information

What should I remember?
What should I ignore?
What should I forget?
# Results

Using only AST features (No lexical or layout features)

<table>
<thead>
<tr>
<th></th>
<th>Python (25 programmers)</th>
<th>Python (70 programmers)</th>
<th>C++ (10 programmers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Forest</td>
<td>86.00</td>
<td>72.90</td>
<td>75.90</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>77.20</td>
<td>61.28</td>
<td>73.50</td>
</tr>
<tr>
<td>LSTM</td>
<td>92.00</td>
<td>86.36</td>
<td>80.00</td>
</tr>
<tr>
<td>BiLSTM</td>
<td>96.00</td>
<td>88.86</td>
<td>85.00</td>
</tr>
</tbody>
</table>
So what?

- Learn better AST representations without feature engineering
- Language independent - any programming language that supports ASTs

Future work

- Combine with Random Forests and fuller feature sets
  - Better results or just overlap with other features?
What about other languages?

Porting requires AST parser and lexical/layout features

Similar accuracies so far (on GCJ dataset)
Results with just AST vary
Train on one language test on another?

- This is something we’d like to try
- Need either universal intermediate AST representation or pairwise
- Babblefish project (doesn’t appear to be ready yet)
Interesting Software Engineering Insights
What about attributing groups?

Looked at team programming competition

Teams compete on sets of problems

Preliminary results:

118 Codeforces teams, at least 20 submissions each

- 10-fold cross-validation: 67.2% accuracy
- 20-fold cross-validation: 67.8% accuracy

Difficult because they are likely splitting up the problems completely

Future work: code repositories
Difficult vs. Easy Tasks

Implementing harder functionality makes programming style more unique.

<table>
<thead>
<tr>
<th>Same set of 62 programmers</th>
<th>Classification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solving 7 Easy Problems</td>
<td>90%</td>
</tr>
<tr>
<td>Solving 7 Harder Problems</td>
<td>95%</td>
</tr>
</tbody>
</table>
Effect of Programming Skill?

Programmers who got further in the GCJ Contest were easier to attribute.

<table>
<thead>
<tr>
<th>Same set of 62 programmers</th>
<th>Classification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less Advanced Coders</td>
<td>80%</td>
</tr>
<tr>
<td>More Advanced Coders</td>
<td>95%</td>
</tr>
</tbody>
</table>
How does coding style change over time?

- 92% accuracy, train and test on 2012
- 88% accuracy, train on 2012, test on 2014
Coding style by country?

GCJ files (in javascript) written by programmers in Canada and China

- 84 files
- 91.9% classification accuracy
Future Applications

● Find malicious code authors
  ○ anonymous contributors
● Write better obfuscators
  ○ target AST directly
● Find authors who write vulnerable code
  ○ open source code
● Find who to recruit directly
  ○ from git commits
Thanks to collaborators

Bander Alsulami, Edwin Dauber, Richard Harang, Andrew Liu, Spiros Mancoridis, Arvind Narayanan, Frederica Nelson, Mosfiquur Rahman, Dennis Rollke, Konrad Rieck, Gregory G. Shearer, Clare Voss, Michael J. Weisman, Fabian Yamaguchi
Contact information and Q & A

Aylin Caliskan
@aylin_cim
aylin@gwu.edu

Rachel Greenstadt
@ragreens
greenstadt@gmail.com

Source code authorship attribution: https://github.com/calaylin/bda
Binary authorship attribution: https://github.com/calaylin/bda
## Comparison to related work

<table>
<thead>
<tr>
<th>Related Work</th>
<th>Author Size</th>
<th>Instances</th>
<th>Average LOC</th>
<th>Language</th>
<th>Fetaures</th>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MacDonell et al.</td>
<td>7</td>
<td>351</td>
<td>148</td>
<td>C++</td>
<td>lexical &amp; layout</td>
<td>Case-based reasoning</td>
<td>88.0%</td>
</tr>
<tr>
<td>Frantzeskou et al.</td>
<td>8</td>
<td>107</td>
<td>145</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>100.0%</td>
</tr>
<tr>
<td>Elenbogen and Seliya</td>
<td>12</td>
<td>83</td>
<td>100</td>
<td>C++</td>
<td>lexical &amp; layout</td>
<td>C4.5 decision tree</td>
<td>74.7%</td>
</tr>
<tr>
<td>Shevertalov et. al.</td>
<td>20</td>
<td>N/A</td>
<td>N/A</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Genetic algorithm</td>
<td>80%</td>
</tr>
<tr>
<td>Frantzeskou et al.</td>
<td>30</td>
<td>333</td>
<td>172</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>96.9%</td>
</tr>
<tr>
<td>Ding and Samadzadeh</td>
<td>46</td>
<td>225</td>
<td>N/A</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>75.2%</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td><strong>35</strong></td>
<td><strong>315</strong></td>
<td><strong>68</strong></td>
<td><strong>C++</strong></td>
<td><strong>lexical &amp; layout</strong></td>
<td><strong>Random forest</strong></td>
<td><strong>100.0%</strong></td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td><strong>250</strong></td>
<td><strong>2,250</strong></td>
<td><strong>77</strong></td>
<td><strong>C++</strong></td>
<td></td>
<td></td>
<td><strong>98.0%</strong></td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td><strong>1,600</strong></td>
<td><strong>14,400</strong></td>
<td><strong>70</strong></td>
<td><strong>C++</strong></td>
<td><strong>lexical &amp; layout &amp; syntactic</strong></td>
<td></td>
<td><strong>93.6%</strong></td>
</tr>
</tbody>
</table>
## Comparison to related work

<table>
<thead>
<tr>
<th>Related Work</th>
<th>Author Size</th>
<th>Instances</th>
<th>Average LOC</th>
<th>Language</th>
<th>Fetaures</th>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frantzeskou et al.</td>
<td>30</td>
<td>333</td>
<td>172</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>96.9%</td>
</tr>
<tr>
<td>Ding and Samadzadeh</td>
<td>46</td>
<td>225</td>
<td>N/A</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>75.2%</td>
</tr>
<tr>
<td>Ours</td>
<td>35</td>
<td>315</td>
<td>68</td>
<td>C++</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>100.0%</td>
</tr>
<tr>
<td>Ours</td>
<td>250</td>
<td>2,250</td>
<td>77</td>
<td>C++</td>
<td>lexical &amp; layout &amp; syntactic</td>
<td>Random forest</td>
<td>98.0%</td>
</tr>
<tr>
<td>Ours</td>
<td>1,600</td>
<td>14,400</td>
<td>70</td>
<td>C++</td>
<td>lexical &amp; layout &amp; syntactic</td>
<td></td>
<td>93.6%</td>
</tr>
</tbody>
</table>
Comparison to related work

<table>
<thead>
<tr>
<th>Related Work</th>
<th>Author Size</th>
<th>Instance Size</th>
<th>Average LOC</th>
<th>Language</th>
<th>Fetaures</th>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frantzeskou et al.</td>
<td>30</td>
<td>333</td>
<td>172</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>96.9%</td>
</tr>
<tr>
<td>Ding and Samadzadeh</td>
<td>46</td>
<td>225</td>
<td>N/A</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>75.2%</td>
</tr>
<tr>
<td>Ours</td>
<td>35</td>
<td>315</td>
<td>68</td>
<td>C++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>250</td>
<td>2,250</td>
<td>77</td>
<td>C++</td>
<td></td>
<td></td>
<td>100.0%</td>
</tr>
<tr>
<td>Ours</td>
<td>1,600</td>
<td>14,400</td>
<td>70</td>
<td>C++</td>
<td>lexical &amp; layout &amp; syntactic</td>
<td>Random forest</td>
<td>98.0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>93.6%</td>
</tr>
</tbody>
</table>

August 10, 2018
## Comparison to related work

<table>
<thead>
<tr>
<th>Related Work</th>
<th>Author Size</th>
<th>Instances</th>
<th>Average LOC</th>
<th>Language</th>
<th>Features</th>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MacDonell et al.</td>
<td>7</td>
<td>351</td>
<td>148</td>
<td>C++</td>
<td>lexical &amp; layout</td>
<td>Case-based reasoning</td>
<td>88.0%</td>
</tr>
<tr>
<td>Frantzeskou et al.</td>
<td>8</td>
<td>107</td>
<td>145</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>100.0%</td>
</tr>
<tr>
<td>Elenbogen and Seliya</td>
<td>12</td>
<td>83</td>
<td>100</td>
<td>C++</td>
<td>lexical &amp; layout</td>
<td>C4.5 decision tree</td>
<td>74.7%</td>
</tr>
<tr>
<td>Shevertalov et al.</td>
<td>20</td>
<td>N/A</td>
<td>N/A</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Genetic algorithm</td>
<td>80%</td>
</tr>
<tr>
<td>Frantzeskou et al.</td>
<td>30</td>
<td>333</td>
<td>172</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>96.9%</td>
</tr>
<tr>
<td>Ding and Samadzadeh</td>
<td>46</td>
<td>225</td>
<td>N/A</td>
<td>Java</td>
<td>lexical &amp; layout</td>
<td>Nearest neighbor</td>
<td>55.2%</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td>35</td>
<td>315</td>
<td>68</td>
<td>C++</td>
<td>lexical &amp; layout &amp; syntactic</td>
<td>Random forest</td>
<td>100.0%</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td>250</td>
<td>2,250</td>
<td>77</td>
<td>C++</td>
<td>Random forest</td>
<td>98.0%</td>
<td></td>
</tr>
</tbody>
</table>
Publications

Usenix 2015:
Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt.
Publications & Code on github

**NDSS 2018:**

**Usenix 2015:**
Source code stylometry

• Everyone learns coding on an individual basis, as a result code in a unique way,
  which makes de-anonymization possible.

• Software engineering insights
  • programmer style changes while implementing sophisticated functionality
  • differences in coding styles of programmers with different skill sets

• Identify malicious programmers.
Case 2: Obfuscation

• Who is the programmer of this obfuscated source code?
• Code is obfuscated to become unrecognizable.
• Our authorship attribution technique is impervious to common off-the-shelf source code obfuscators.
Case 3: Copyright investigation

• Copyleft programs are free but licensed
• Did this programmer take a copyleft code and distribute it commercially?
  • Jacobsen vs Katzer (Java Model Railroad Interface)
• Two-class machine learning classification task
  • Class 1: the copyleft code
  • Class 2: the copyright code
Case 3: Copyright investigation

30 pairs of authors each with 9 program files

Classification Accuracy

Two-class task 100%
Case 5: Coding style throughout years

• Is programming style consistent?
• If yes, we can use code from different years for authorship attribution.

```c
int main()
{
    freopen("a.in", "r", stdin);
    freopen("a.out", "w", stdout);
    int tt;
    scanf("%d", &tt);
    for(int t = 0; t < tt; t++)
    {
        int n;
        scanf("%d", &n);
        ...
    }
    ...
}

int main()
{
    freopen("a.in", "r", stdin);
    freopen("a.out", "w", stdout);
    int TT;
    scanf("%d", &TT);
    for(int T = 0; T < TT; T++)
    {
        printf("Case #\%d: ", T+1);
        ...
    }
}
```
Case 5: Coding style throughout years

• Is programming style consistent?
• If yes, we can use code from different years for authorship attribution.

2012

```c
int main()
{
    freopen("a.in", "r", stdin);
    freopen("a.out", "w", stdout);

    int tt;
    scanf("%d", &tt);

    for(int t = 0; t < tt; t++)
    {
        int n;
        scanf("%d", &n);
        ...
    }
}
```

2014

```c
int main()
{
    freopen("a.in", "r", stdin);
    freopen("a.out", "w", stdout);

    int TT;
    scanf("%d", &TT);

    for(int T = 0; T < TT; T++)
    {
        printf("Case #%d: ", T+1);
    }
    ...
}
```
Case 5: Coding style throughout years

• Coding style is preserved up to some degree throughout years.

Train on 25 authors from 2012 to identify the author of 25 files in 2014

96% accuracy
Case 5: Coding style throughout years

• 98% accuracy, train and test in 2014
• 96% accuracy, train on 2012, test on 2014

Train on 25 authors from 2012 to identify the author of 25 files in 2014

96% accuracy
Case 6: Difficult tasks & advanced coders

- Insights about programmers and coding style:
  - Implementing harder functionality makes programming style more unique

<table>
<thead>
<tr>
<th>Same set of 62 authors</th>
<th>Classification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solving 7 easy problems</td>
<td>98%</td>
</tr>
<tr>
<td>Solving 7 more difficult problems</td>
<td>99%</td>
</tr>
</tbody>
</table>
Case 6: Difficult tasks & advanced coders

• Insights about programmers and coding style.
  • Better programmers have more distinct coding style

<table>
<thead>
<tr>
<th>Two sets of 62 authors</th>
<th>Classification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less advanced programmers</td>
<td>97%</td>
</tr>
<tr>
<td>More advanced programmers</td>
<td>98%</td>
</tr>
</tbody>
</table>
Case 7: Generalizing the approach - python

Feature set: Using ‘only’ the Python equivalents of syntactic features

<table>
<thead>
<tr>
<th>Application</th>
<th>Programmers</th>
<th>Instances</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python programmer de-anonymization</td>
<td>229</td>
<td>2,061</td>
<td>53.9%</td>
</tr>
<tr>
<td>Top-5 relaxed classification</td>
<td>229</td>
<td>2,061</td>
<td>75.7%</td>
</tr>
<tr>
<td>Python programmer de-anonymization</td>
<td>23</td>
<td>207</td>
<td>87.9%</td>
</tr>
<tr>
<td>Top-5 relaxed classification</td>
<td>23</td>
<td>207</td>
<td>99.5%</td>
</tr>
</tbody>
</table>
Comparison to related work

600 contestants – C++

de-anonymized programmers

preprocessing

extract features

majority vote

random forest

fuzzy AST parser
### Comparison to related work

<table>
<thead>
<tr>
<th>Related Work</th>
<th>Number of Programmers</th>
<th>Number of Training Samples</th>
<th>Classifier</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rosenblum et al.</td>
<td>20</td>
<td>8-16</td>
<td>SVM</td>
<td>77%</td>
</tr>
<tr>
<td>This work</td>
<td>20</td>
<td>8</td>
<td>SVM</td>
<td>90%</td>
</tr>
<tr>
<td>This work</td>
<td>20</td>
<td>8</td>
<td>Random forest</td>
<td>99%</td>
</tr>
<tr>
<td>Rosenblum et al.</td>
<td>191</td>
<td>8-16</td>
<td>SVM</td>
<td>51%</td>
</tr>
<tr>
<td>This work</td>
<td>191</td>
<td>8</td>
<td>Random forest</td>
<td>92%</td>
</tr>
<tr>
<td>This work</td>
<td>600</td>
<td>8</td>
<td>Random forest</td>
<td>83%</td>
</tr>
</tbody>
</table>
Amount of Training Data Required for De-anonymizing 100 Programmers
Reducing Suspect Set Size: Top-n Classification
Open world:
Classification thresholds for verification
Reconstructing original features

- Original vs predicted features
  - Average cos similarity: 0.81

- Original vs decompiled features
  - Average cos similarity: 0.35

This suggests that original features are transformed but not entirely lost in compilation.
Ongoing work - DARPA

• Malware author attribution

• Dataset with ground truth

• Automated malware analysis
Future work

• De-anonymizing collaborative code
  • Group fingerprint vs individual fingerprint

• Anonymizing source code
  • Obfuscation is not designed for anonymization
Underground forums

- Micro-text
- L33t sp34k
- Multi-lingual
- Products
- Doppelgänger Finder
  - Carders

No technical difference between security-enhancing and privacy-infringing