WHEN CODING STYLE SURVIVES COMPILATION: DE-ANONYMIZING PROGRAMMERS FROM EXECUTABLE BINARIES

Aylin Caliskan
@aylin_cim

Princeton University
CITP Fellow and Postdoctoral Research Associate
“Style expressed in code can be quantified and characterized.”
Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt.
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

```
00100000 00000000 00001000 00000000 00001000 00000000
00000000 00000000 00110100 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000011 00000000 00000000 00000000 00110100 00000001
00000000 00000000 00110100 10000001 00000100 00001000
00000000 00000000 00010011 00000000 00000000 00000000
00000100 00000000 00000000 00000000 00000001 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 10000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
```

Why de-anonymize programmers?
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
Related work
## 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>
Comparison to related work

600 contestants – C++

code jam

preprocessing

majority vote

random forest

fuzzy AST parser

de-anonymized programmers

System.out.println("hello, world!");
Binary Executables of Programmers

Disassembly

Decomposition

Fuzzing Parsing

Abstract Syntax Tree

Control Flow Graph

entry

blk1

blk2

blk3

blk4

exit

Instruction Features

Lexical Features

Syntactic Features

Flow Features

Stylistic Feature Analysis by Information Gain

Random Forest Classification by cross validation

De-anonymized Programmer
Features: Assembly

Disassembly

test edi, edu
mov eax, 0x0
cmovs edi, eax

Assembly Features

Assembly unigrams

test

Assembly bigrams

eax, 0x0

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
  if
     pred
     stmt
     <
     ...  
  v0
  f0
  C0
```

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:

```
blk1  blk2  blk3  blk4  ...
```

CFG bigrams:

```
blk1  blk1  blk2  blk3  ...
```
Dimensionality Reduction

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

- Information gain criterion
  - Keep features with low 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.
Predictive features

Variable Declaration & Initialization in one line

```
int age = 20;
```

int variable Declaration and Initialization
Authorship attribution

• 96% accuracy in identifying 100 authors of 900 anonymous program files.

Train on 100 authors identify authors of 900 files

96% accuracy
Large scale programmer de-anonymization
Real world applications

1) Optimized binaries
2) Obfuscated binaries
3) GitHub binaries
4) Nulled.IO and malware binaries
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
Open-LLVM obfuscations reduce de-anonymization accuracy of 100 programmers from 96% to 88%.
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 in the wild

- 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>

<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>

Compile repositories
Amount of Training Data Required for De-anonymizing 100 Programmers

Correct Classification Accuracy

Number of Training Samples Per Author
Future work

• Anonymizing executable binaries
  – optimizations do not anonymize
• De-anonymizing collaborative binaries
  – Group vs individual fingerprint
• Malware actor attribution
  – If you have a malware dataset with known authors:
Future work

• Anonymizing executable binaries
  – optimizations do not anonymize
• De-anonymizing collaborative binaries
  – Group vs individual fingerprint
• Malware actor attribution
  – If you have a malware dataset with known authors:
    GET IN TOUCH WITH ME: aylinc@princeton.edu
Open world:
Classification thresholds for verification
Reducing Suspect Set Size: Top-n Classification

![Graph showing correct classification accuracy for different values of n (1 to 10) and different sample sizes (100 and 600 programmers) with varying training samples (8 and 1)].
Reconstructing original features

- Original vs decompiled features
  - Average cos similarity: 0.35
Reconstructing original features

- Original vs predicted features
  - Average cos similarity: 0.81

- Original vs decompiled features
  - Average cos similarity: 0.35
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.
Features

Source code

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

Abstract Syntax Tree
Dataset: Development and validation sets

• Obtain a dataset in CPP
  – Ground truth in authorship
  – Scraped Google Code Jam to build a corpus
  – Compile code with the same settings

• Take two disjoint sets of 100 programmers
  – Develop method on first set – controlled setting
  – Validate method on second set

• Google Code Jam:
  – Everyone implements the same algorithmic functionality
  – Complete task in a limited time
  – Problems get harder
Obfuscation 2: Bogus Flow Insertion

(a) Original code
(b) Obfuscated code
Obfuscation 3: Control Flow Flattening

Original CF

Flattened CFG
Obfuscation 3: Control Flow Flattening

Original CFG

```
1 = 1;
s = 0;

while (i <= 100) {
    s += i;
i++;
}
```

Flattened CFG

```
int swVar = 1;
while (swVar != 0) {
    switch (swVar) {
    case 1: {
        i = 1;
s = 0;
        swVar = 2;
        break;
    }
    case 2: {
        if (i <= 100)
            swVar = 3;
        else
            swVar = 0;
        break;
    }
    case 3: {
        i = 1;
i++;
s = 1;
        swVar = 2;
        break;
    }
    }
}
```