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

Aylin Caliskan

@aylin_cim

Princeton University

CITP Fellow and Postdoctoral Research Associate

PRINCETON
UNIVERSITY

Aylin Caliskan

PRINCETON UNIVERSITY

Fabian Yamaguchi

ShiftLeft

Edwin Dauber

Drexel UNIVERSITY

Richard Harang

SOPHOS

Konrad Rieck

CAROLO-WILHELMINA BRAUNSCHWEIG

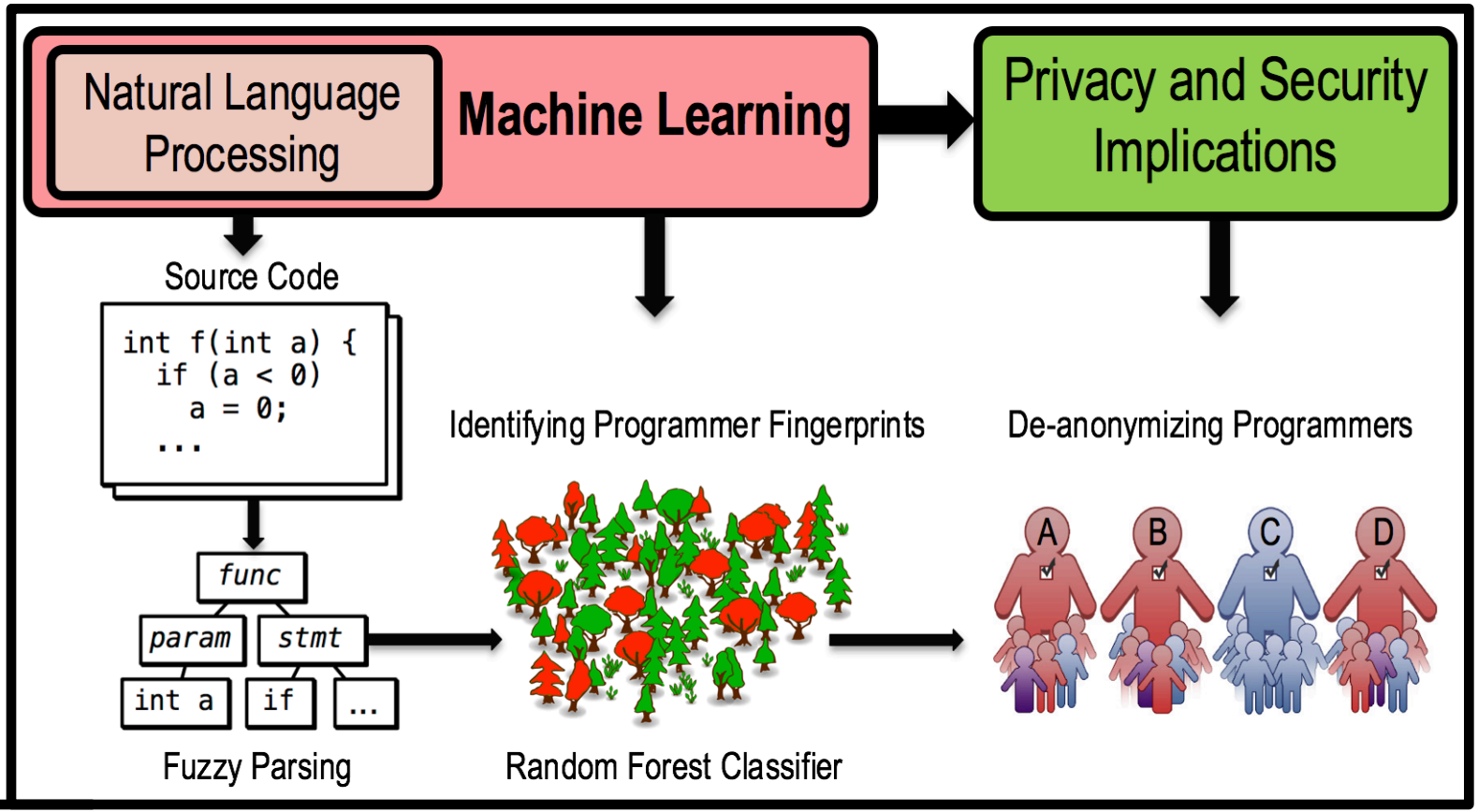
Rachel Greenstadt

Drexel UNIVERSITY

Arvind Narayanan

PRINCETON UNIVERSITY

“Style expressed in code can be quantified and characterized.”



Usenix 2015

De-anonymizing Programmers via Code Stylometry. 24th Usenix Security Symposium.
Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt.

What about executable binaries?

Source Code

```
#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 00101000 00000000
00000000 00000000 00110100 00000000 00000000 00000000
00000100 00001000 00000000 00000001 00000000 00000000
00000000 00000001 00000000 00000000 00000101 00000000
00000000 00000000 00000100 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 00000001 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 10000000
00000100 00001000 00000000 10000000 00000100 00001000
11001000 00010111 00000000 00000000 11001000 00010111
00000000 00000000 00000101 00000000 00000000 00000000
00000000 00010000 00000000 00000000 00000001 00000000
00000000 00000000 11001000 00010111 00000000 00000000
11001000 10100111 00000100 00001000 11001000 10100111
00000100 00001000 00101100 00000001 00000000 00000000
00000000 00000000 00000000 00010000 00000000 00000000
00000010 00000000 00000000 00000000 11011100 00010111
```

. . .

Why de-anonymize programmers?

x0rz Follow
Security Researcher
Sep 13, 2016 · 4 min read

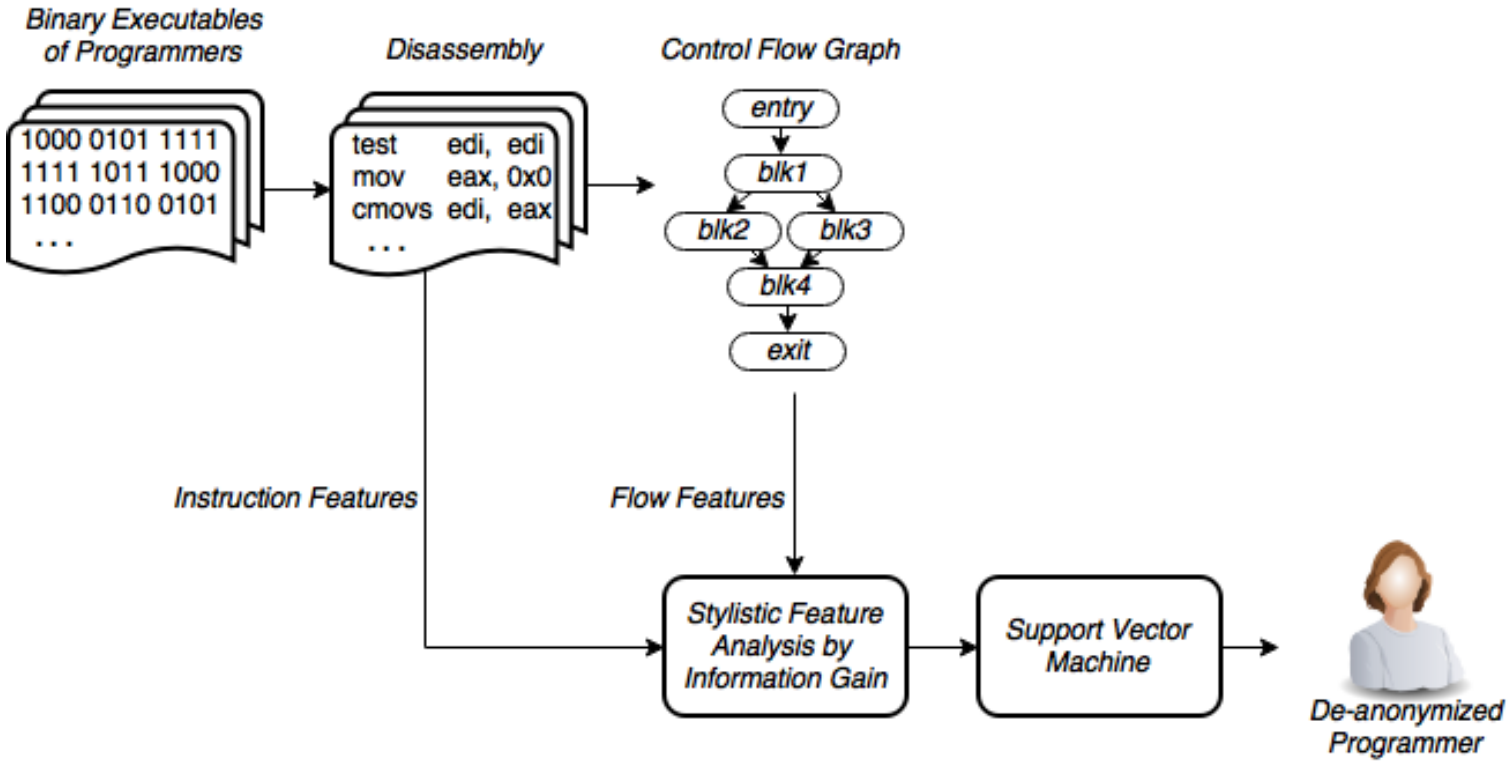
Interview with the LuaBot malware author

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

Related Work	Number of Programmers	Number of Training Samples	Classifier	Accuracy
Rosenblum et al.	**20**	**8-16**	**SVM**	**77%**
This work	20	8	SVM	90%
This work	20	8	Random forest	99%
Rosenblum et al.	191	8-16	SVM	51%
This work	191	8	Random forest	92%
This work	**600**	**8**	**Random forest**	**83%**

Comparison to related work

Hex Rays
IDA Pro v6.3

600 contestants – C++

code jam

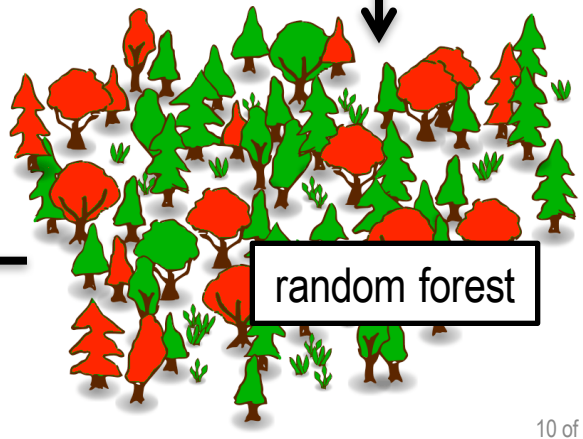
```
System.out.println("hello, world!");
```

preprocessing

joern

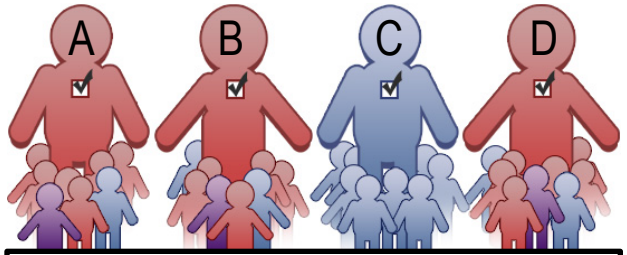
fuzzy AST parser

extract features

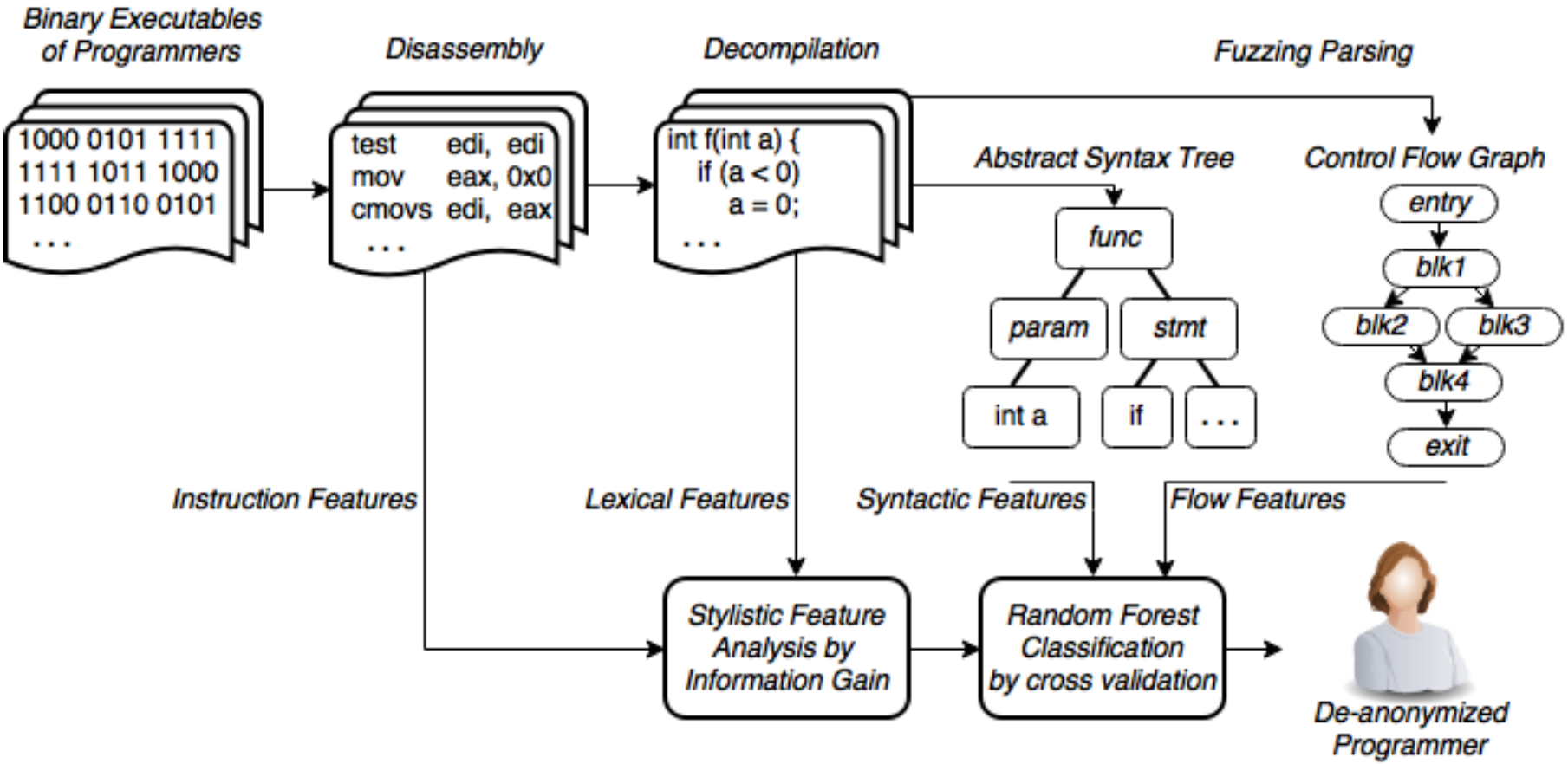


random forest

majority vote

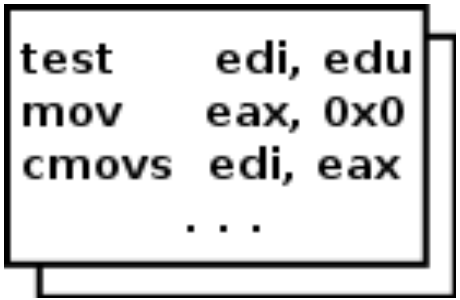


de-anonymized programmers


Binary Executables of Programmers
Disassembly
Decompilation
Fuzzing Parsing
1000 0101 1111
1111 1011 1000
1100 0110 0101
...
test edi, edi
mov eax, 0x0
cmovs edi, eax
...
int f(int a) {
if (a < 0)
a = 0;
...
Abstract Syntax Tree
func
param
stmt
int a
if
...
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



Assembly Features

Assembly unigrams

```
test
```

Assembly bigrams

```
eax, 0x0
```

Assembly trigrams

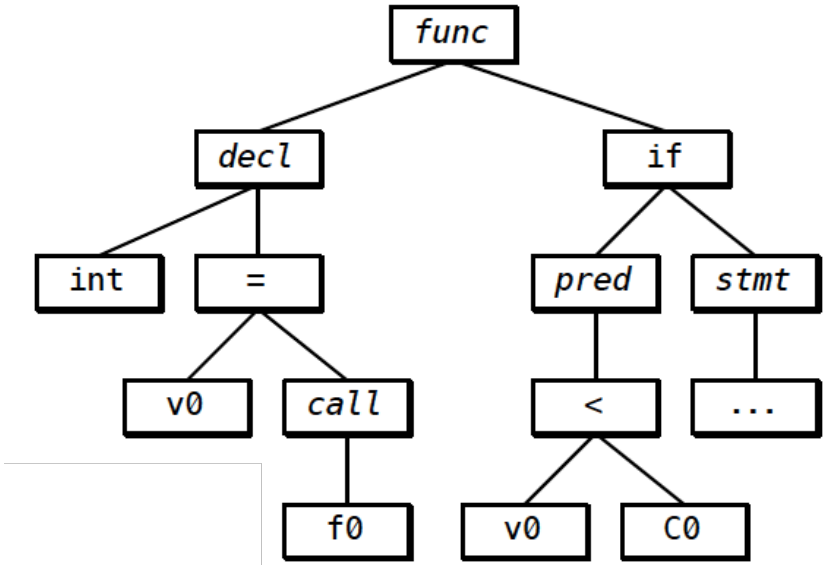
```
cmovs edi, eax
```

Two consecutive assembly lines

```
mov   eax, 0x0
cmovs edi, eax
```

Features: Syntactic

Abstract syntax tree (AST)



Syntactic features

AST unigrams:

func	decl	if	int
=	pred	stmt	...

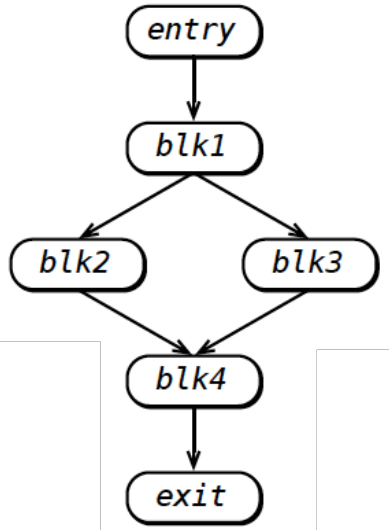
AST bigrams:

func	func	decl	...
decl	if	int	

AST depth: 5

Features: Control flow

Control-flow graph (CFG)

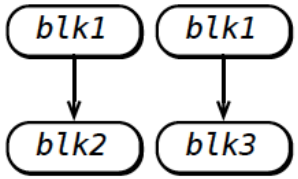


Control-flow features

CFG unigrams:

blk1 blk2 blk3 blk4 ...

CFG bigrams:

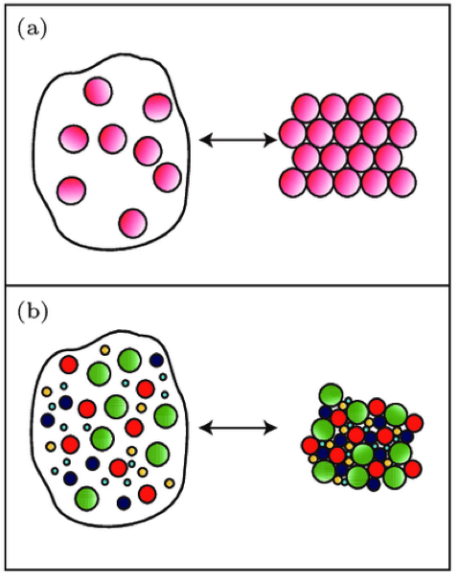


...

Dimensionality Reduction

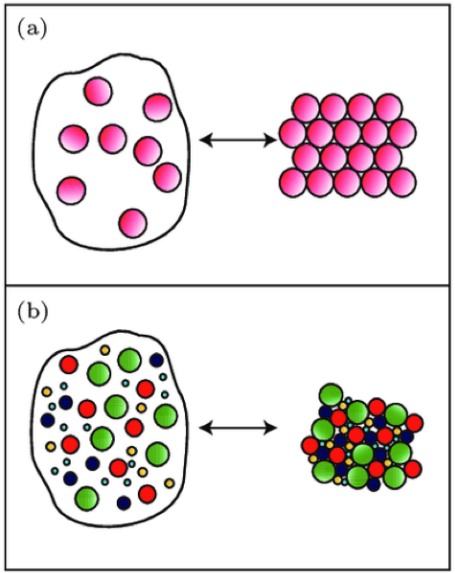
– Information gain criterion

- Keep features with low entropy – see (a)
- Reduce dimension from ~700,000 to ~2,000.

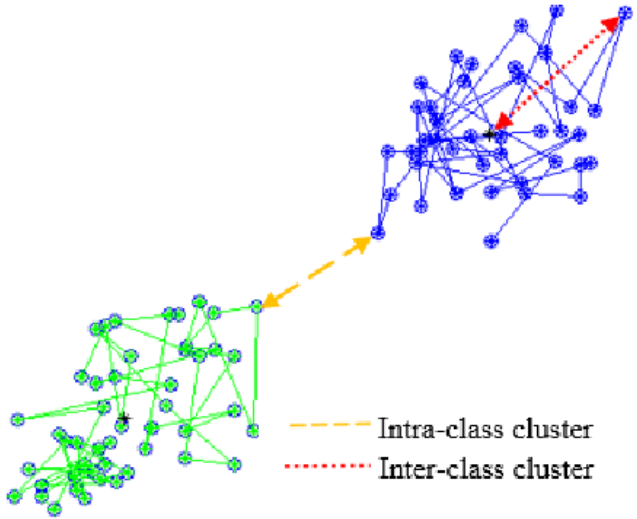


Dimensionality Reduction

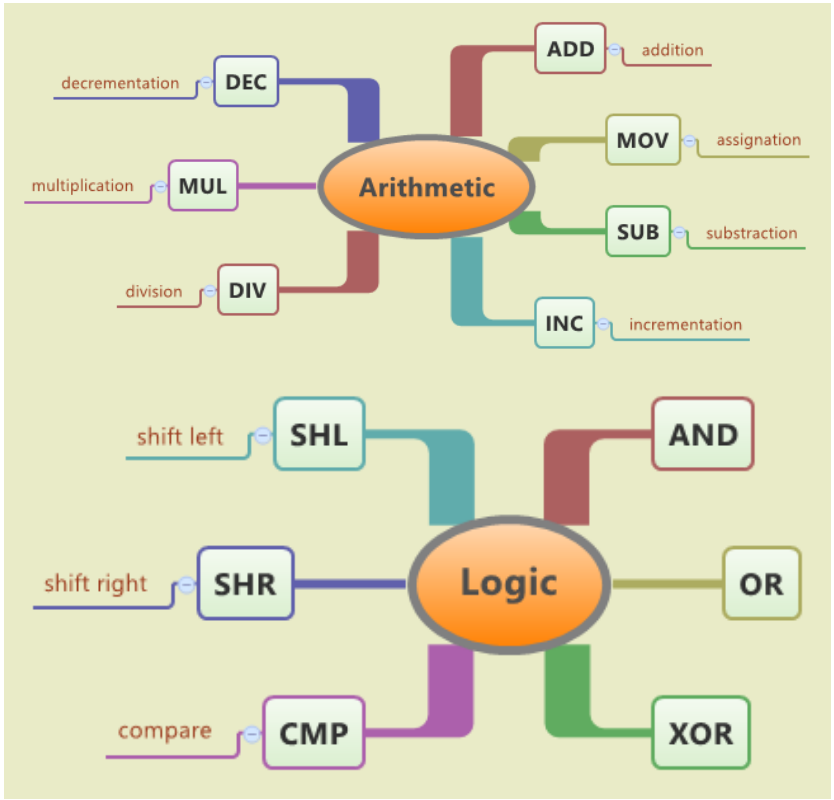
– Information gain criterion
  - Keep features with low entropy – see (a)
  - Reduce dimension from ~700,000 to ~2,000.

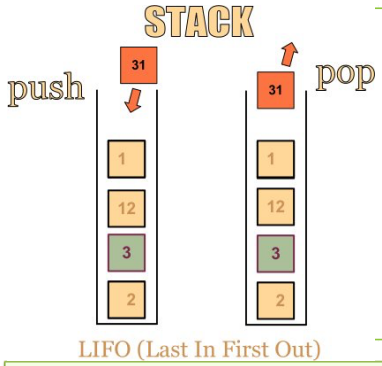


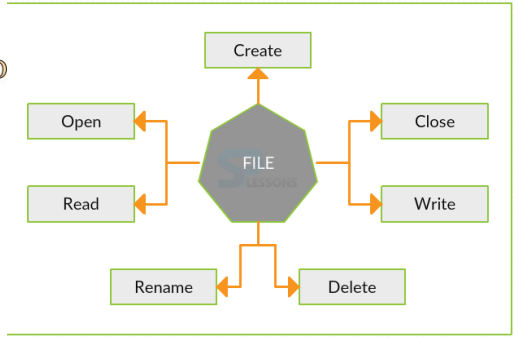
– Correlation based feature selection
  - Keep features with low inter-class correlation
  - Reduce dimension from ~2,000 to 53.

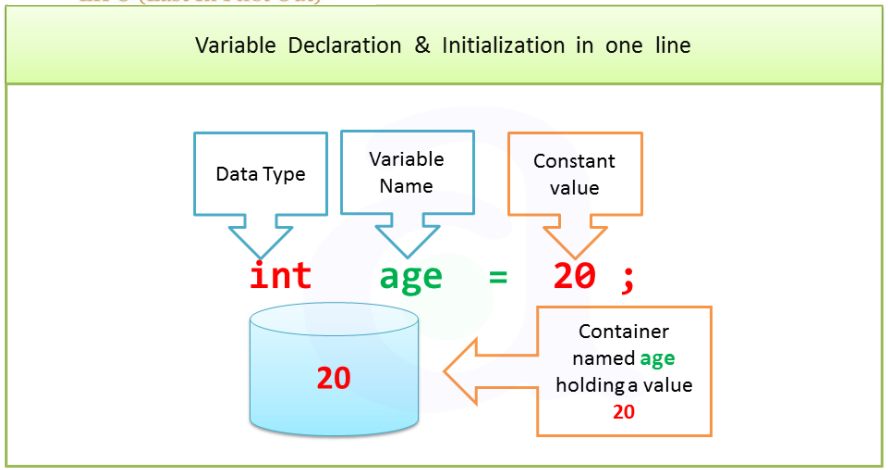


Predictive features





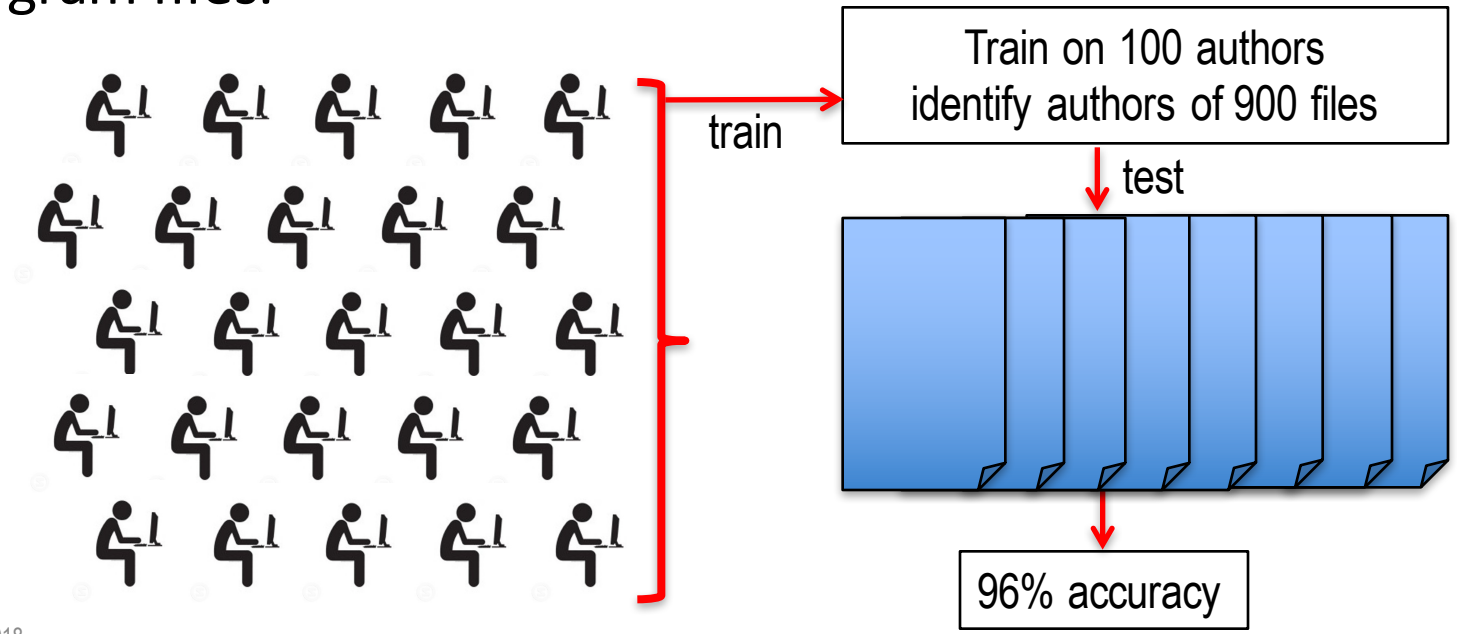




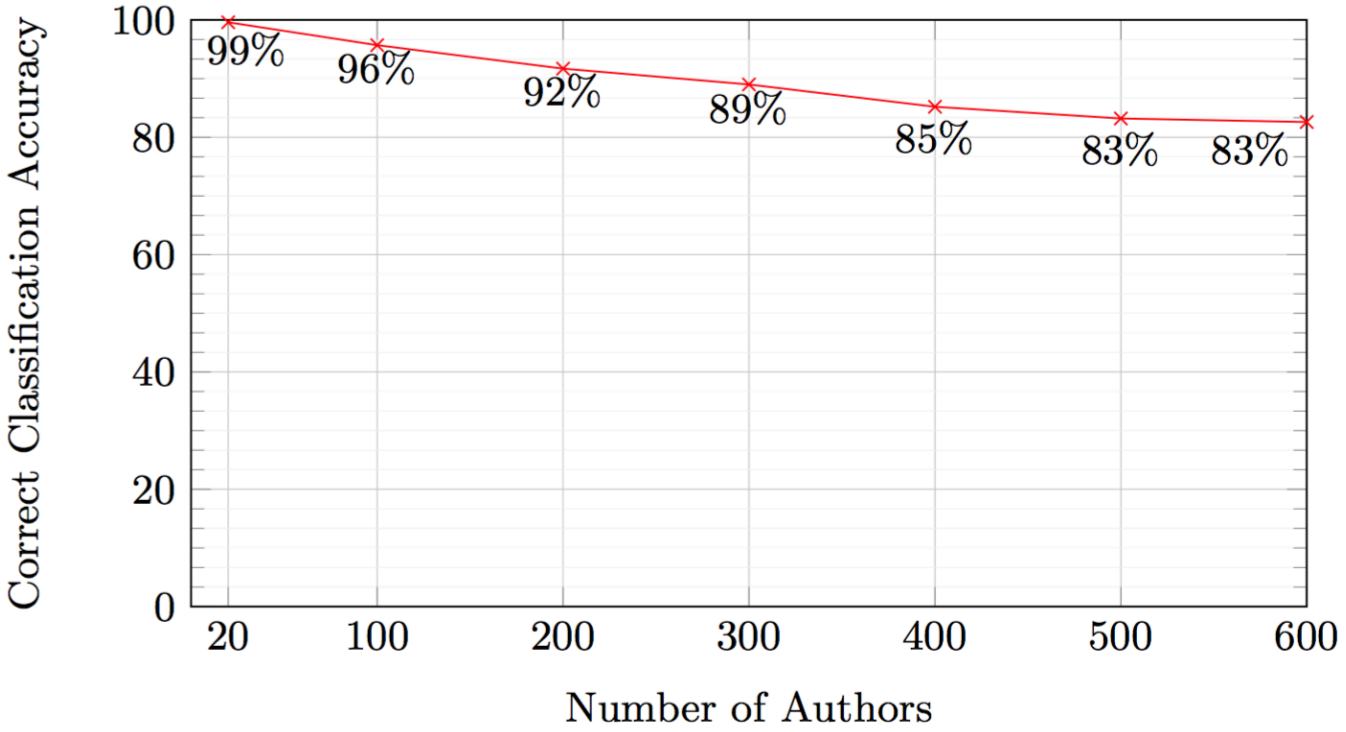
int variable Declaration and Initialization

Authorship attribution

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



Large scale programmer de-anonymization



Real world applications

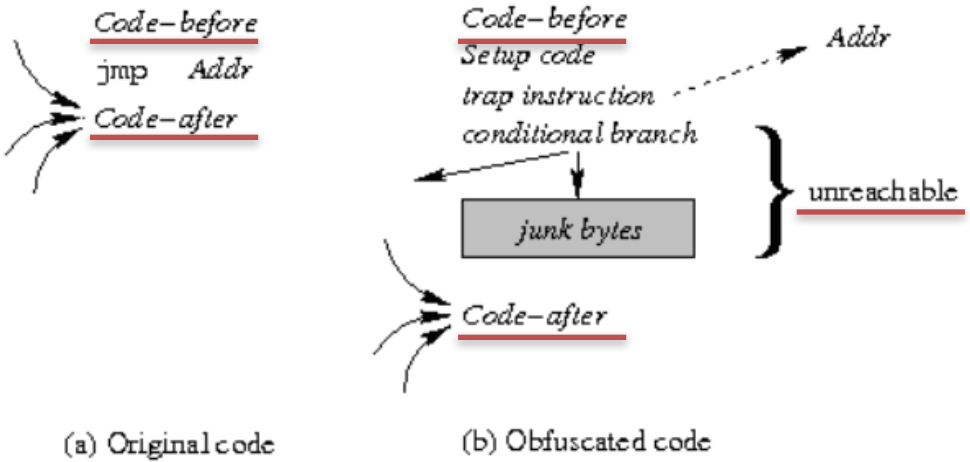
1) Optimized binaries
2) Obfuscated binaries
3) GitHub binaries
4) Nulled.IO and malware binaries

Optimizations and stripping symbols

Number of programmers	Number of training samples	Compiler optimization level	Accuracy
100	8	None	96%
100	8	1	93%
100	8	2	89%
100	8	3	89%
100	8	Stripped symbols	72%

Obfuscation

1. Bogus control flow insertion

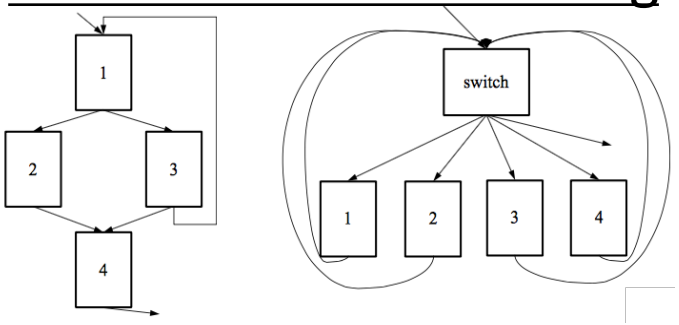


2. Instruction substitution

```
mov     eax, [rbp+var_14]
add     eax, 61h
mov     cl, al
movsxd  rdx, [rbp+var_14]
mov     [rbp+rdx+var_F], cl
jmp     loc_40059B
```

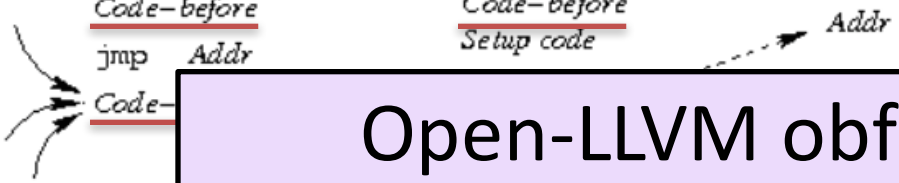
```
mov     eax, 61h
mov     ecx, [rbp+var_14]
sub     eax, 5EC7EBEEh
add     eax, ecx
add     eax, 5EC7EBEEh
mov     dl, al
movsxd  rsi, [rbp+var_14]
mov     [rbp+rsi+var_F], dl
jmp     loc_4005B3
```

3. Control flow flattening



Obfuscation

1. Bogus control flow insertion

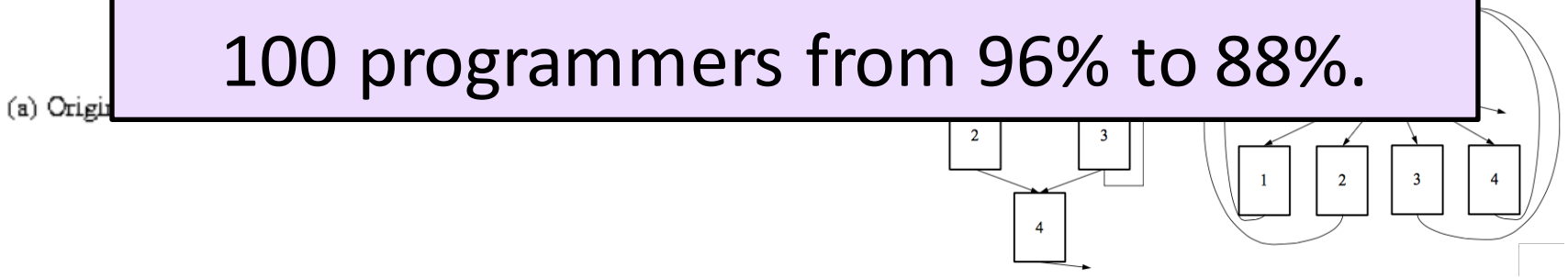


2. Instruction substitution

```
mov     eax, [rbp+var_14]
add     eax, 61h
mov     cl, al
movsxd  rdx, [rbp+var_14]
mov     [rbp+rdx+var_F], cl
```

```
mov     eax, 61h
mov     ecx, [rbp+var_14]
sub     eax, 5EC7EBEEh
add     eax, ecx
add     eax, 5EC7EBEEh
```

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

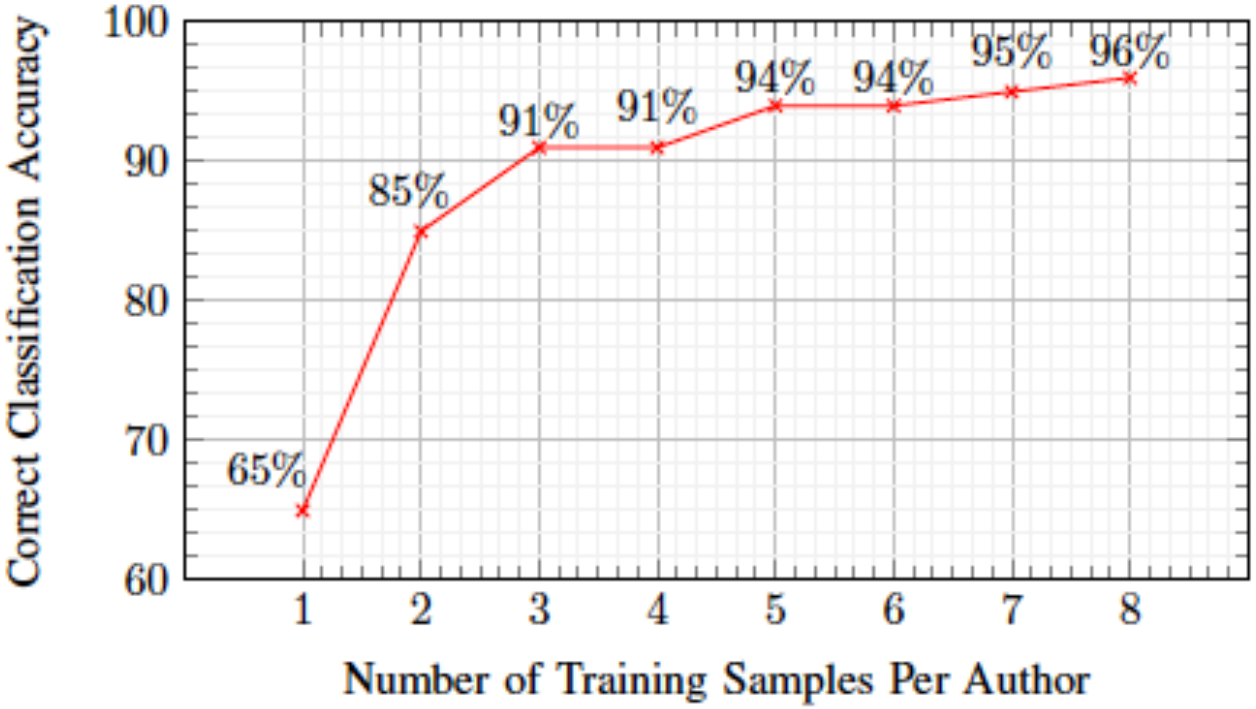
- ✓ Single authored GitHub repositories
- ✓ The repository has at least 500 lines of code

Type	Amount
Authors	161
Repositories	439
Files	3,438
Repositories / Author	2 - 8
Files / Author	2 - 344

Compile repositories

Dataset	Authors	Total Files	Accuracy
GitHub	50	542	65%
GCJ	50	450	97%

Amount of Training Data Required for De-anonymizing 100 Programmers



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

Aylin Caliskan

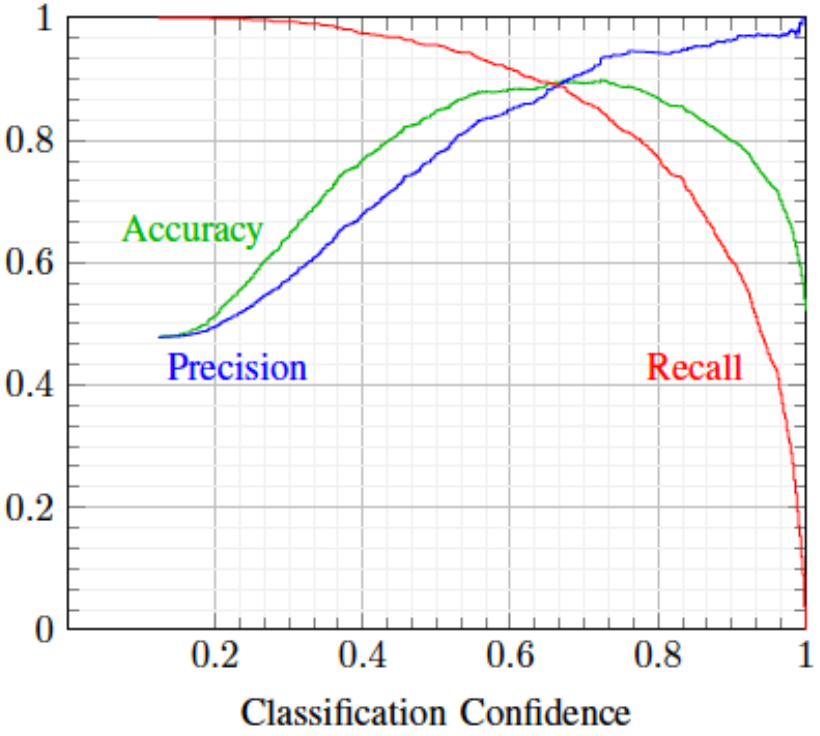
@aylin_cim

aylinc@princeton.edu

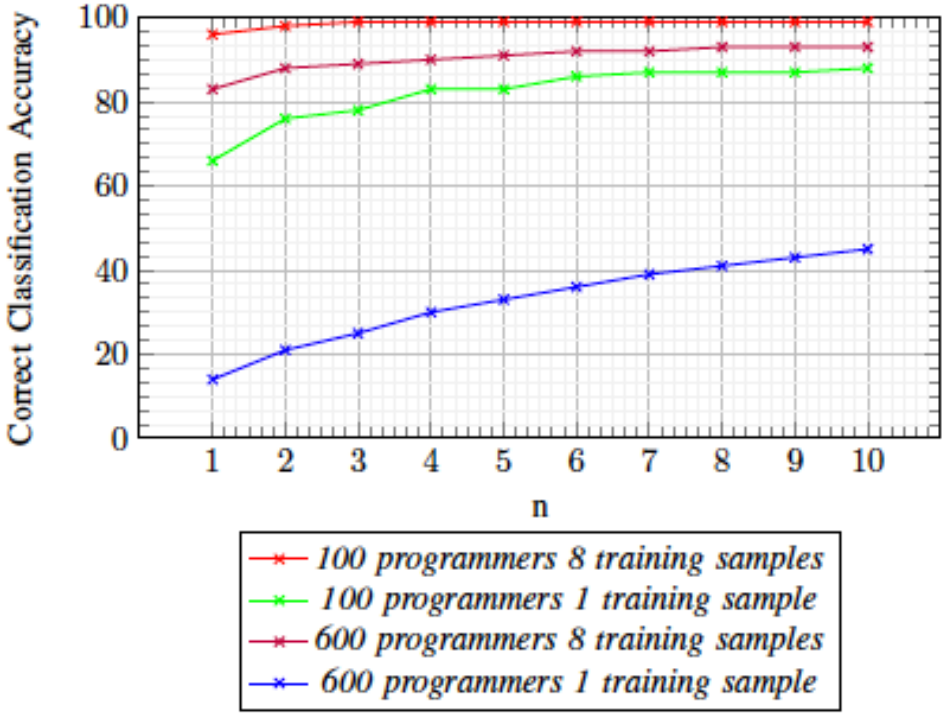
www.princeton.edu/~aylinc

www.github.com/calaylin

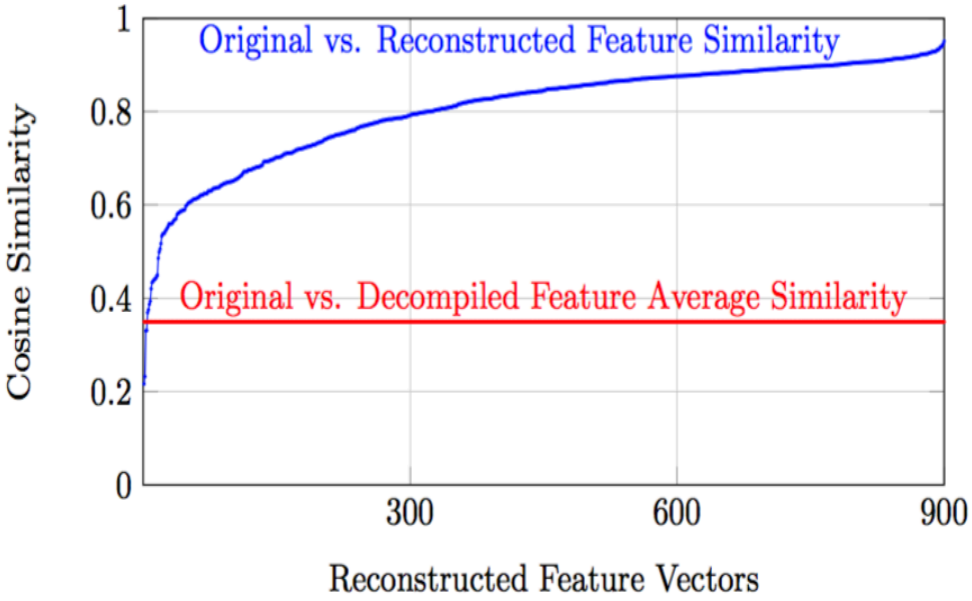
Open world:
Classification thresholds for verification



Reducing Suspect Set Size: Top-n Classification

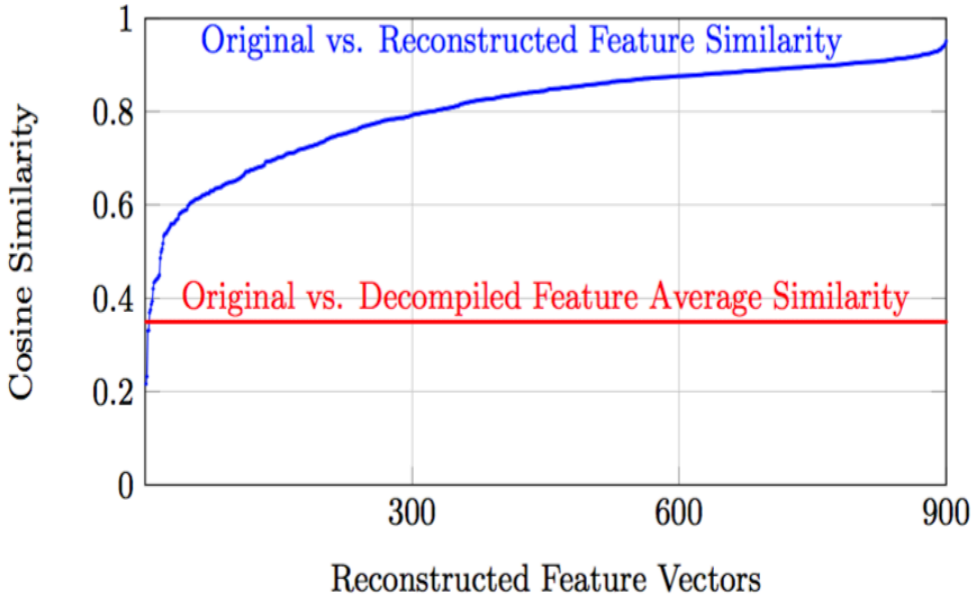


Reconstructing original features



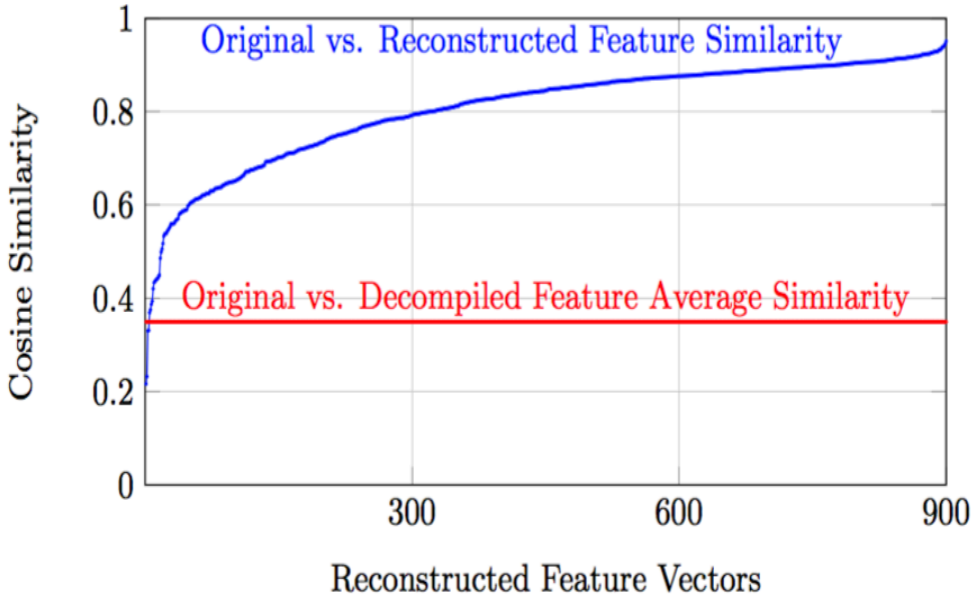
- Original vs decompiled features
  - Average cos similarity: 0.35

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

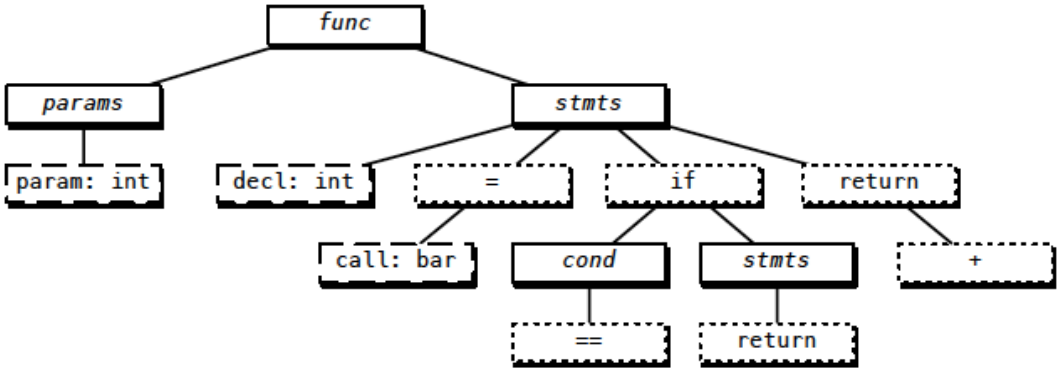
Source code

```
int foo(int y)
{
    int n = bar(y);

    if (n == 0)
        return 1;

    return (n + y);
}
```

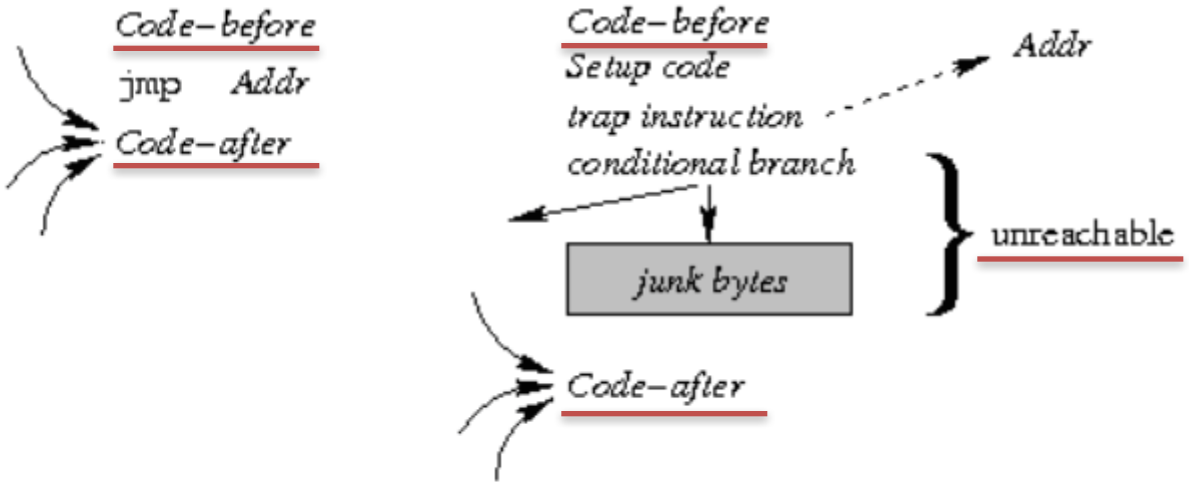
Abstract Syntax Tree



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

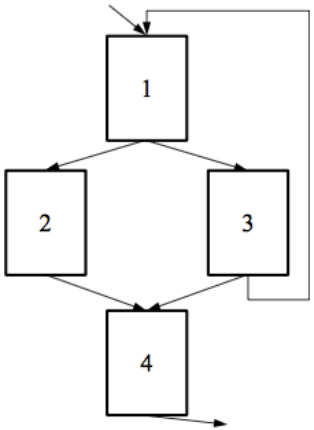


(a) Original code

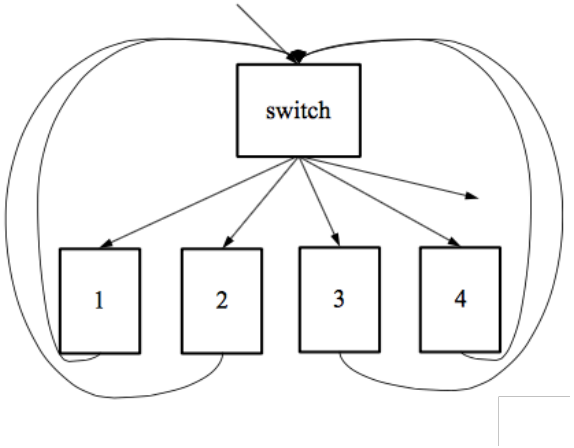
(b) Obfuscated code

Obfuscation 3: Control Flow Flattening

Original CF



Flattened CFG



Obfuscation 3: Control Flow Flattening

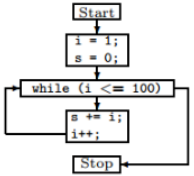
Original CFG

```
i = 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: {
      s += i;
      i++;
      swVar = 2;
      break;
    }
  }
}
```