Code Stylometry

CS463/ECE424

University of Illinois

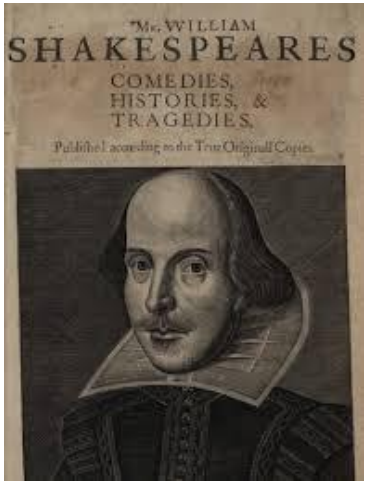
Stylometry and authorship attribution background

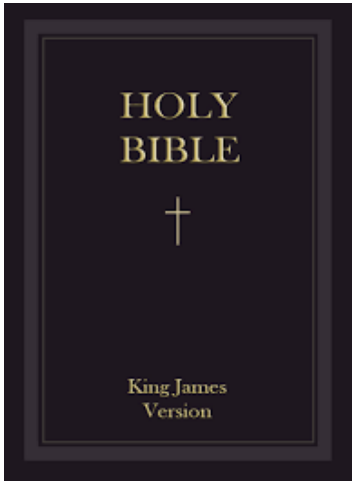
Code stylometry methods

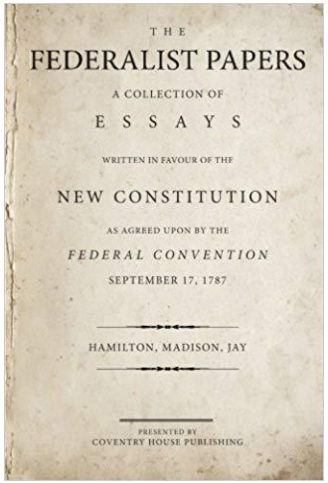
- Source code stylometry
- Executable binary stylometry

Motivating Examples

- There has been debate over who wrote:
  - Shakespeare's works
  - Bible passages
  - The Federalist Papers

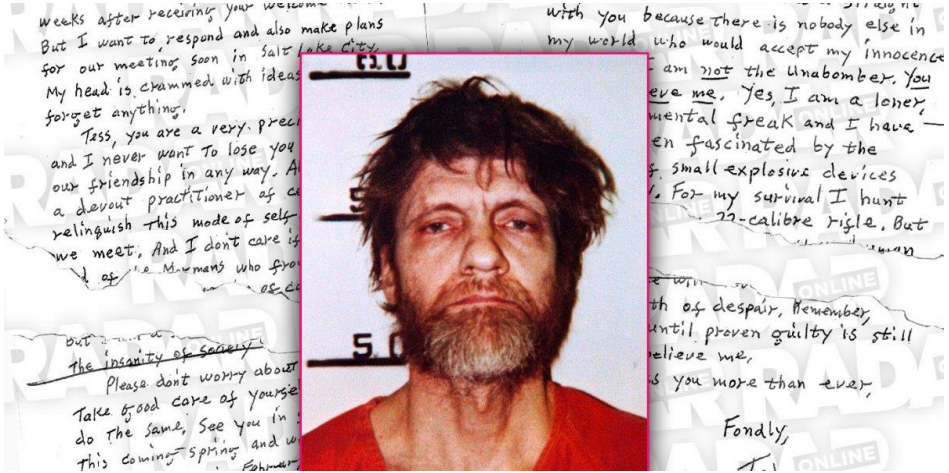






Motivating Examples

- Linguistic work was pivotal in capture of Unabomber (Ted Kaczynski)
- The Unabomber's Manifesto



Authorship Attribution

- Authorship attribution aims to infer the identity of an author of a document by examining it

- Stylometry: inferring properties of the author by examination
  - This idea is over a century old
  - Stylome/fingerprint: differences in how individuals write

Linguistic Stylometry

- Use different features of written text to fingerprint authors
  - Vocabulary
  - Average word length
  - Frequency of specific words
  - Many others

- Machine learning is generally used to classify works based on these features

Examples of Linguistic Stylometry

- [Narayanan12] used stylometry to identify anonymous bloggers in large datasets
  - This is a privacy issue

- Adversarial stylometry [Brennan12]
  - Authorship attribution based on linguistics can be evaded
  - Defenses:
    - Obfuscate writing style
    - Imitate someone else's writing style

Code Stylometry

- We want to determine who wrote some code
- Goal: programmer de-anonymization
- Can you think of reasons why we would want to determine code authorship?

Code Stylometry

- We want to determine who wrote some code
- Goal: programmer de-anonymization
- Can you think of reasons why we would want to determine code authorship?
  - Company wants to determine which employee wrote harmful code
  - Government wants to determine who is engaging in cyber warfare
  - A professor wants to determine if students are plagiarizing assignments
  - Identify Satoshi Nakomoto
  - Identify cyber criminals
  - Determine source of malware
  - Reveal creators of anti-censorship tools

Types of Code Stylometry

- Source code stylometry
- Executable binary stylometry
- Malware attribution

Source Code Stylometry

- We can study source code for authorship attribution
- Examples of features used for source code stylometry:
  - Simple byte-level and word-level n-grams
  - Abstract syntax trees
  - Lexical markers such as line length
  - Layout

- Techniques usually include classification by ML

Executable Binary Stylometry

- We want to study executable binaries for authorship attribution
- Binaries are typically produced by compiling or assembling source code
- Goal: perform stylometry on executable binaries

Executable Binary Stylometry

- Harder than source code stylometry
- During compilation,
  - Variable names, function names, and other symbols and metadata about the source code can be removed
  - The structure of the code can be changed through optimization
- This removes information that may suggest authorship

Executable Binary Stylometry

- What information can we use about binary code to reveal authorship information?
  - Use tools to parse executable binaries
  - Reconstruct instruction sequences and control flow graphs
  - Use this information as features to determine a code author's stylometric fingerprint

When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries

- Goal: **executable binary** stylometry using automatic tools

- Main idea:
  - Use **machine learning** to classify sample executable binaries from a set of known authors
  - Determine a good set of features for executable binary stylometry

Attack Model

- Consider an analyst interested in determining the author of an executable binary purely based on its style (not content)
- Assume that the analyst only has access to executable binary samples each assigned to one of a set of candidate programmers
- The analyst:
  - Obtains labeled executable binaries from each candidate programmer (training set)
  - Converts each labeled sample into numerical feature vector, using low-level features from disassemblers and decompilers
  - Derives a classifier from these vectors using machine learning
  - Uses this classifier to attribute the anonymous executable binary (test set) to the most likely programmer

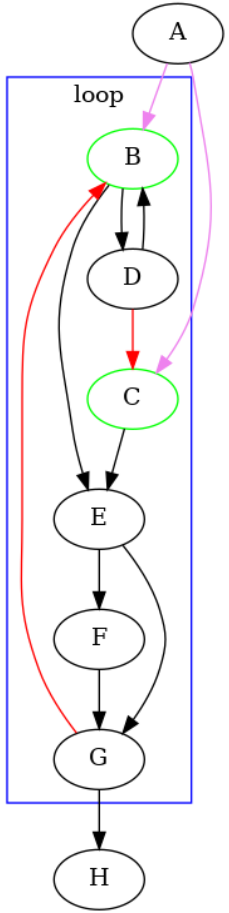
Background: Disassemblers and Decompilers

- Disassemblers
  - Programs that translate executable binary code into assembly code
  - The inverse of an assembler
- Decompilers
  - Programs that translate executable binary into high level source code
  - The inverse of a compiler

- These tools do not perfectly reconstruct the original source or assembly code

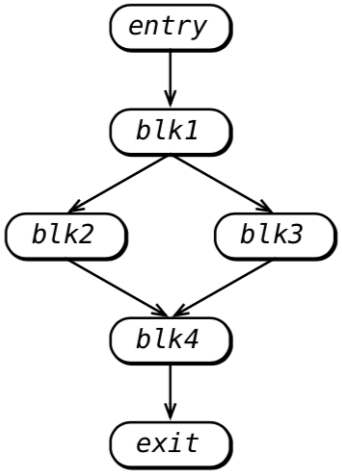
Background: Control Flow Graphs

- A control flow graph is a graph of all paths that might be traversed through a program during execution
- Each node represents a basic block in the code
  - A basic block is a piece of code with no jumps
- Directed edges represent jumps in the control flow



Background: CFG examples

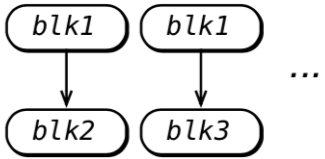
Control-flow graph (CFG)



Control-flow features

CFG unigrams:

CFG bigrams:



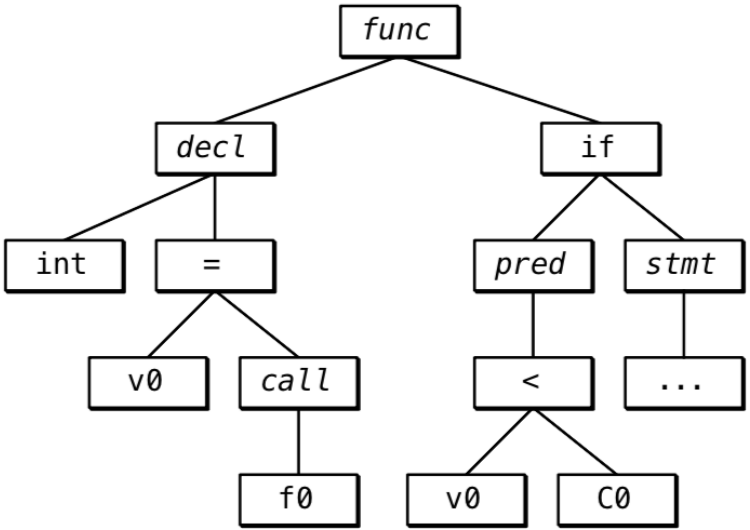
Background: Abstract Syntax Trees

- Tree representation of the abstract syntactic structure of source code written in a programming language
  - A structure containing only the meaning of a program, but no language details (ex. semicolons, spaces, formatting)
- Each node of the tree denotes a construct that occurs in the source code

- These trees abstract away certain parts of the high-level language such as: parentheses, if statements, etc

Background: AST examples

Abstract syntax tree (AST)



Syntactic features

AST unigrams:

func	decl	if	int
=	pred	stmt	...

AST bigrams:

func	func	decl	...
decl	if	int	

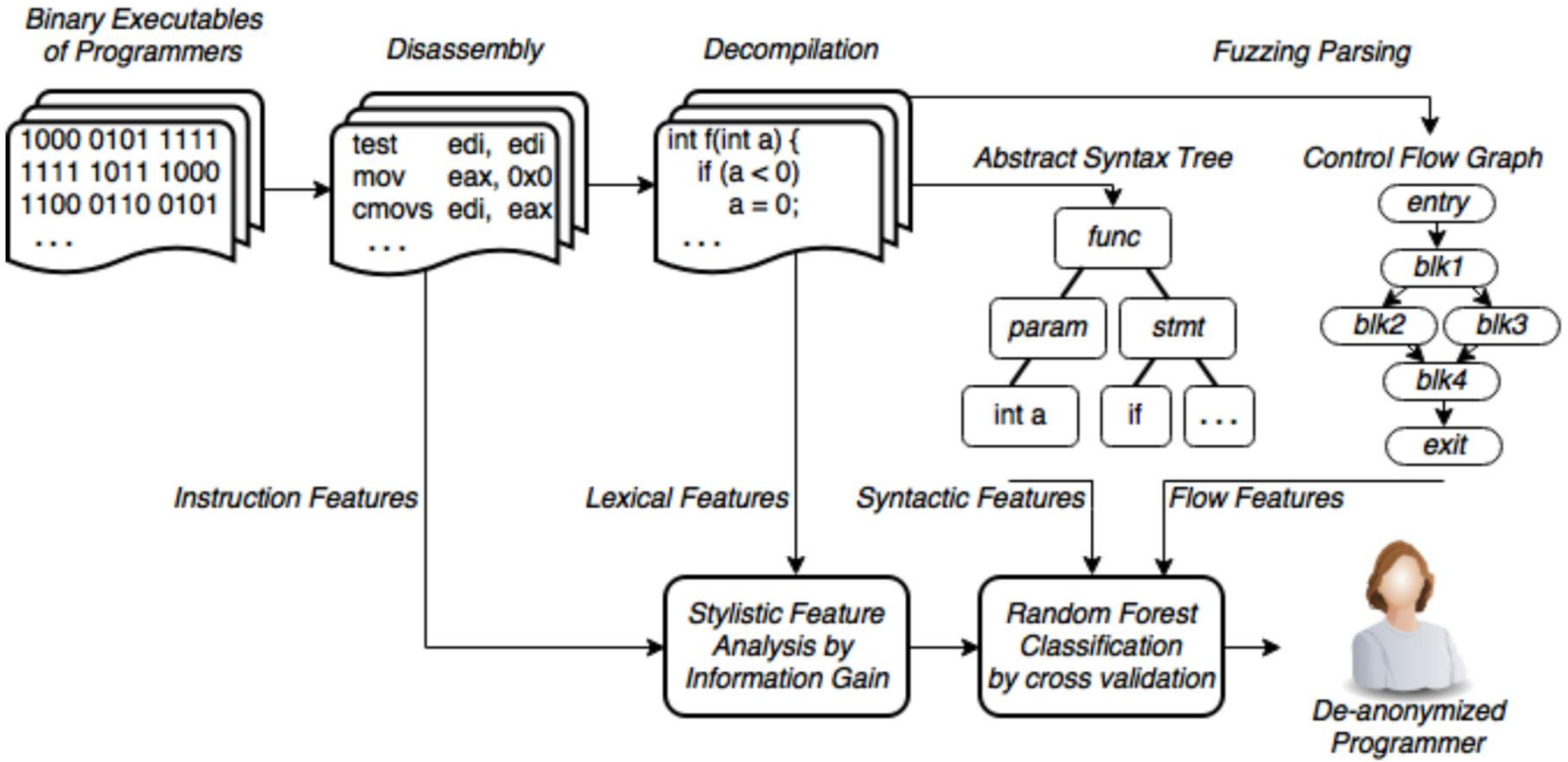
AST depth: 5

When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries, Continued

- Executable Binary Stylometry [Caliskan18]
- Extract features of executable binary code for stylometry:
  - Use automated decompilation of binaries
  - Generate abstract syntax trees of decompiled source code
  - Use multiple tools for disassembly and decompilation in parallel

- ML framework
  - Feature reduction
  - Predict code authorship using a random forest classifier

[Caliskan18] Overview



Stylistic Features

- Representations of the program from binary code
  - Disassembler
    - Obtain low level features in assembly code
    - Based on machine code instructions, referenced strings, symbol information, etc.
  - Decompiler
    - Translate the program into C-like pseudo code
    - Pass this code to a fuzzy parser for C
    - Generate control flow graph to capture the flow of the program
    - Convert the low-level instructions to high level decompiled source code in order to obtain abstract syntax trees
- Use these three data formats to numerically represent the stylistic properties embedded in binary code

Dimensionality Reduction

- Analyst determines the set of stylistic features through dimensionality reduction

- Two steps of feature selection:
  - Information gain based dimensionality reduction
  - Correlation based feature selection
- Select features particularly useful for classification
- 53 features are identified to represent programmer style
  - Out of 705,000 representations of code properties

Machine Learning Task

- Closed world problem
  - The set of potential code authors is known
- Supervised learning task
  - The training data is labeled
- Multi-class problem
  - Classifier calculates the most likely author for the anonymous executable binary sample among multiple code authors

Experimental Setup

- [Caliskan18] performs experiments with data from the Google Code Jam
  - GCJ is an annual programming competition
  - Contestants implement solutions for the **same tasks**
- Focused on C++ code
- Compiled with gcc or g++
  - Experimented with no optimizations, and optimization levels-1,2,3

Results

Related Work	Number of Programmers	Number of Training Samples	Accuracy	Classifier
Rosenblum [39]	20	**8-16**	77%	SVM
This work	20	**8**	90%	SVM
This work	20	**8**	99%	RF
Rosenblum [39]	100	8-16	**61%**	SVM
This work	100	8	**84%**	SVM
This work	100	8	**96%**	RF
Rosenblum [39]	191	8-16	**51%**	SVM
This work	191	8	**81%**	SVM
This work	191	8	**92%**	RF
This work	**600**	8	**71%**	SVM
This work	**600**	8	**83%**	RF

TABLE II: Comparison to Previous Results

Results

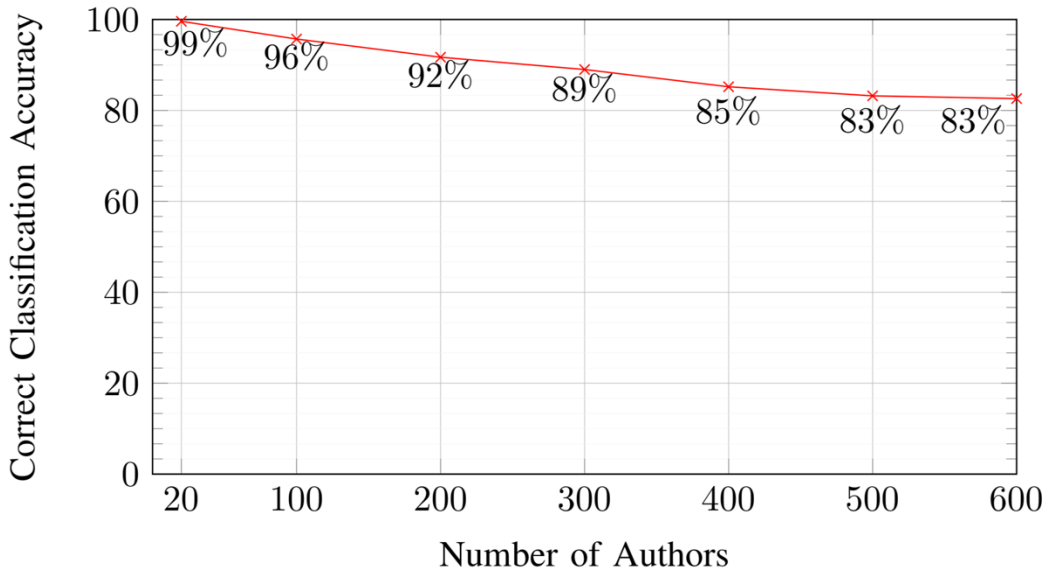


Fig. 5: Large Scale Programmer De-anonymization

Findings

- Even a single training sample per programmer is sufficient for de-anonymization
- Accuracy can be improved by finding the top-n most likely authors
- This work can de-anonymize 600 programmers from their executable binaries
- Removing symbol information does not anonymize binaries
- Programmers can be de-anonymized from obfuscated binaries*

*This experiment is quite brief, not very conclusive

Practical Implications of this Work

- Coding style survives compilation!
- Why?
  - Decompiled source code is not necessarily similar to the original source code in terms of the features used in this work
  - The feature vector obtained from disassembly and decompilation can be used to predict the features in the original source code
- More skilled programmers are more fingerprintable
  - Programmers gradually acquire their own unique style as they gain experience

References

- [Narayanan12] On the Feasibility of Internet-Scale Author Identification. Arivind Narayanan, Hristo Paskov, Neil Zhenqiang Gong, John Bethencourt, Emil Stefanov, Eui Chul Richard Shin, Dawn Song. IEEE Symposium on Security and Privacy 2012.
- [Brennan12] Adversarial Stylometry: Circumventing Authorship Recognition to Preserve Privacy and Anonymity. Michael Brennan, Sadia Afroz, Rachel Greenstadt. ACM Transactions on Information and System Security (TISSEC) 2012.
- [Caliskan18] When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries. Aylin Caliskan, Fabian Yamaguchi, Edwin Dauber, Richard Harang, Konrad Rieck, Rachel Greenstadt, Arvind Narayanan. NDSS 2018.

Discussion

- Can you think of some countermeasures that might be possible to preserve privacy against code stylometry analysis?

- What are the pros and cons of authorship attribution?
  - Natural language
  - Code