Source Code Authorship Attribution and Verification under Adversarial Conditions

Prototype Project Report

Tashi Stirewalt

Team 18

Elang Sisson

Andrew Varkey

II.2.3. Reporting and Explainability 5

TABLE OF CONTENTS

I. Introduction 3

II. System Requirements Specification 3

II.1. Use Cases 3

II.2. Functional Requirements 4

II.2.1. Core Analysis Functions 4

II.2.2. Adversarial Toolkit 5

II.2.3. Reporting and Explainability 6

II.2.4. System Management 6

II.3. Non-Functional Requirements 7

III. System Evolution 7

IV. Glossary 8

V. References 8

I. Introduction

Stylometry, the study of linguistic style, has traditionally been applied to natural language but is being increasingly used in programming languages to identify authorship in source code. This is relevant to cybersecurity, where authorship can help in finding malicious scripts, identifying threats, and detecting plagiarism as well. In today's landscape, the rise of large language models in software development and the growing risk of attacks create significant challenges for code attribution.

Attackers can deliberately obscure or mimic stylistic traits, making it difficult to tell the difference between genuine authorship from adversarial changes which threatens how accurate traditional stylometry methods are. This project aims to explore how machine learning can be applied to verify authorship and identify when adversarial authorship occurs. By studying the impacts of these attacks and experimenting with approaches to improve robustness, our work seeks to support the development of more resilient tools for source code verification.

Our objective is to develop a tool that achieves high baseline accuracy on clean code and demonstrates resilience when adversaries attempt to disguise authorship. The system will serve cybersecurity researchers and analysts who need reliable tools for code forensics.

II. System Requirements Specification

II.1. Use Cases

UC-1: Academic Integrity Checking

Actors	Instructors, Students
Pre-condition	● The instructor has the student submissions.
Post-condition	● The system outputs a report highlighting authorship attribution for each submission ● If adversarial edits are detected, the system raises a flag and lowers confidence
Basic path	1. Instructor uploads artifacts 2. Instructor uploads student submissions 3. System process and runs authorship attribution 4. System verifies the author 5. System generates and outputs a report
Alternative path	● If a file type is unsupported, then show an error. ● If adversarial features are detected, the system adjusts attribution confidence.
Related Requirements	● [FR-01], [FR-02], [FR-03], [FR-04], [FR-05]

UC-2: Security Incident

Actors	Analyst
Pre-condition	● Analyst has a suspicious code sample ● The system has access to a code base from original author
Post-condition	● System attributes code to candidate authors ● Analyst receives explanations in report and an audit log entry
Basic path	1. Analyst uploads artifacts 2. Analyst uploads suspicious code samples 3. System performs attribution and verifies 4. System displays details 5. System records the run for the audit log entry
Alternative path	● If file size exceeds upload limits, show error ● If model lacks explanation to support findings, display "Explanation not available"
Related Requirements	● [FR-01], [FR-02], [FR-03], [FR-06], [FR-07]

UC-3: Adversarial Robustness Testing

Actors	Red team researcher
Pre-condition	● The researcher has already completed an attribution run
Post-condition	● System performed adversarial transformations and outputs metrics on robustness
Basic path	1. System applies transformations 2. System reruns attribution 3. System computes robustness score and includes in report
Alternative path	● If export fails, system logos even and displays a warning
Related Requirements	● [FR-02], [FR-04], [FR-05], [FR-08]

II.2. Functional Requirements

Each Functional Requirement is listed below with its description, source, and priority.

II.2.1. Core Analysis Functions

Functional Requirement	[FR-1] Code Artifact Upload

Description	The system shall allow an Analyst to upload one or more code artifacts in at least C/C++, Python, Java, and JavaScript, up to 10 MB per artifact.
Source	Client requirement for multi-language support
Priority	Level 0 (Essential)

Functional Requirement	[FR-2] Authorship Attribution
Description	The system shall perform authorship attribution and return a ranked list of top 5 candidate authors with associated probabilities and confidence intervals.
Source	Client need for accurate attribution capability
Priority	Level 0 (Essential)

Functional Requirement	[FR-3] Authorship Verification
Description	The system shall provide authorship verification mode: given a claimed author and code artifact, output ACCEPT/REJECT with a calibrated score in [0,1].
Source	Client need for author verification in forensic setting
Priority	Level 0 (Essential)

II.2.2. Adversarial Toolkit

Functional Requirement	[FR-4] Adversarial Transformations
Description	The system shall support an adversarial transformation toolkit (e.g., identifier renaming, comment/style perturbation, control-flow rewriting, dead-code insertion) and re-evaluate attribution post-transform.

Source	Research requirement for adversarial stylometry experiments.
Priority	Level 0 (Essential)

II.2.3. Reporting and Explainability

Functional Requirement	[FR-5] Export of Results
Description	The system shall export results as a digitally signed PDF and JSON report including inputs, configuration, metrics, and explanation artifacts.
Source	Stakeholder request for auditable reporting
Priority	Level 1 (Desirable)

Functional Requirement	[FR-6] Feature-Level Explanations
Description	The system shall compute and display feature level explanations that contributed most to the decision.
Source	Research need for interpretability
Priority	Level 1 (Desirability)

II.2.4. System Management

Functional Requirement	[FR-7] Audit Logging
Description	The system shall provide an audit log of all actions (uploads, model runs, exports, and dataset changes) with user, timestamp, and hash of artifacts.
Source	Client requirement for traceability
Priority	Level 0 (Essential)

Functional Requirement	[FR-8] Model Management

Description	The system shall provide model management: select baseline and advanced models, record hyperparameters, and log versioned runs with metrics.
Source	Stakeholder need for reproducibility and comparison
Priority	Level 1 (Desirable)

II.3. Non-Functional Requirements

Non-Functional Requirement	Description
[NFR-01] Data Security	Uploaded code shall be stored encrypted at rest and in transit.
[NFR-02] Reproducability	Every run shall be re-creatable from stored dataset snapshot, model version, and configuration hash; checksum validation must pass.
[NFR-03] End-to-End Performance	Shall be able to complete an end-to-end attribution in ≤ 10 minutes.
[NFR-04] Accuracy	On the baseline evaluation corpus, top-1 attribution accuracy shall be ≥ 70% and verification AUROC ≥ 0.85.
[NFR-05] Latency	For a 300 KB code artifact, attribution shall complete in ≤ 5 seconds.

III. System Evolution

This should describe the fundamental assumptions on which your project is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful as it may help avoid design decisions that would constrain likely future changes to your project. They are also designed to identify risk points in your design process, which need to be kept in mind as your development continues. These can be hardware, software, or client issues where your assumptions about what will work or be required end up being incorrect. For

example, if you think the computer platform you're choosing might have driver issues you'll have to either do significant extra work to fix the drivers or find another platform on a short timeline. That's something you should note in this section as an anticipated change point.

Software and Library Dependencies

- Assumption: Core ML libraries and cryptographic libraries for report signing will remain stable during development
- Risk: If APIs are deprecated or major updates occur, code refactoring may be required to maintain compatibility.

Evolving Threat Landscape

- Assumption: Current adversarial transformations represent realistic evasion strategies
- Anticipated Change: As new obfuscation or AI-assisted evasion techniques emerge, the system needs a new update to include these in the adversarial toolkit.

IV. Glossary

Stylometry: Analysis of source code style to identify authorship
Adversarial Example: Input crafted to mislead ML models
Obfuscation: Code transformation that preserves functionality but alters style
Surrogate Model: A secondary model trained to approximate another models behavior

V. References