T. Sağlam, S. Hahner, L. Schmid, and E. Burger, “Obfuscation-Resilient Software Plagiarism Detection with JPlag,” in 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion ’24), IEEE/ACM, 2024. doi: 10.1145/3639478.3643074.

Obfuscation-Resilient Software Plagiarism Detection with JPlag

Timur Sağlam, Sebastian Hahner, Larissa Schmid, Erik Burger
firstname.lastname@kit.edu
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany

ABSTRACT

The rise of automated obfuscation techniques challenges the widespread assumption that evading a software plagiarism detector requires more effort than completing programming and modeling assignments in computer science education. This threatens plagiarism detectors without comprehensive obfuscation resilience and, ultimately, academic integrity. This paper summarizes recent enhancements of JPlag, a widely-used software plagiarism detector, enabling it to achieve broad resilience against automated obfuscation. The findings demonstrate that JPlag significantly outperforms the state-of-the-art in terms of obfuscation resilience.

CCS CONCEPTS

• **Information systems → Near-duplicate and plagiarism detection**; • **Social and professional topics →** *Computer science education*; *Software engineering education.*

KEYWORDS

Plagiarism Detection, Obfuscation Attacks, CS Education

ACM Reference Format:
Timur Sağlam, Sebastian Hahner, Larissa Schmid, Erik Burger. 2024. Obfuscation-Resilient Software Plagiarism Detection with JPlag. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion ’24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3639478.3643074

1 INTRODUCTION

Plagiarism is widespread in computer science education, and students often try to *obfuscate* their plagiarism, e.g., by renaming, reordering, or insertion [12]. Manual inspection, however, is impractical due to larger class sizes. Software plagiarism detectors have been employed to address this issue [2, 8]. JPlag [8], in use and maintained since 1996, is one of the most used detectors worldwide. These detectors ensure that it requires more effort and skill to evade detection than completing the actual assignment [4]. Yet, this paradigm is challenged by the emergence of automated *obfuscation attacks* [9]. *Mossad* [4], e.g., is a plagiarism generator that repeatedly inserts statements into a plagiarized program without changing its behavior. It is unclear what other automated obfuscation attacks exist [3], requiring reevaluating existing detection methods. This paper presents enhancements of JPlag with the recently released version 5.0, enabling it to achieve broad resilience against automated obfuscation attacks. Thus, JPlag is not only a reliable, open-source[1], and GDPR-compliant software plagiarism detector with an intuitive result visualization, but it also outperforms the state-of-the-art regarding obfuscation resilience (see Figure 1). In the following, we first introduce JPlag and its capabilities. Then, we discuss the latest research on obfuscation resilience.

ICSE-Companion ’24, April 14–20, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0502-1/24/04.
https://doi.org/10.1145/3639478.3643074

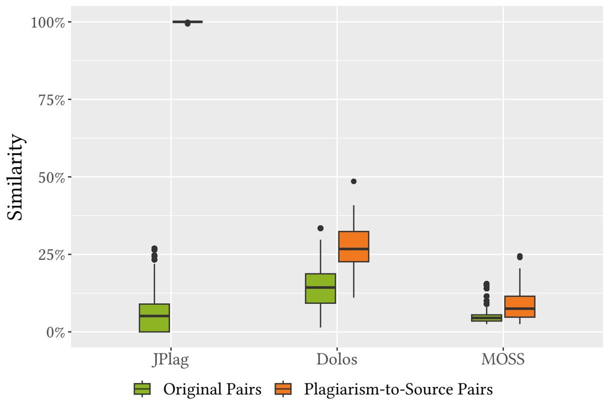

Figure 1: Results of JPlag, Dolos, and MOSS for PROGpedia-19 dataset with plagiarism instances automatically obfuscated through statement reordering and insertion.

2 PLAGIARISM DETECTION WITH JPLAG

JPlag compares the structure of programs to identify suspicious candidates [8]. While it ranks these candidates and highlights outliers, instructors make the final decision. The result visualization of JPlag includes features such as histograms, clustering, code comparison, and anonymization, enhancing its ease of use and report visualization. It also provides a side-by-side view to explain *how* similarity scores are calculated. Thus, JPlag allows for ethical academic misconduct investigations by combining automated analysis with human judgment. While JPlag was first developed in 1996, it is actively maintained and a subject of active research. It currently supports 17 languages. Recent additions include the support of modeling languages [11], thus addressing plagiarism in modeling assignments [12]. JPlag scales well, facilitating plagiarism detection even in large courses.

Internally, JPlag works as depicted in Figure 2. First, it parses the program, extracting a subset of the parse tree nodes as tokens, thus linearizing the tree [9]. Second, it uses newly introduced normalization techniques [9, 10] to achieve obfuscation resilience. Third, it conducts pairwise comparisons (using GST with KR matching [13]) to identify matching sections. After that, it applies post-processing for both obfuscation resilience and clustering calculation. Fifth, JPlag calculates a similarity score for each pair based on the matches. Finally, the results are visualized in a human-readable way.

[1] JPlag Github Repository: https://github.com/jplag/JPlag

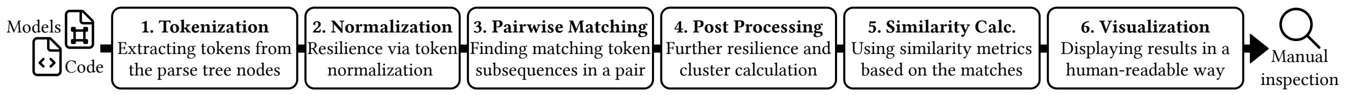
Models
Code
1. Tokenization
Extracting tokens from the parse tree nodes
2. Normalization
Resilience via token normalization
3. Pairwise Matching
Finding matching token subsequences in a pair
4. Post Processing
Further resilience and cluster calculation
5. Similarity Calc.
Using similarity metrics based on the matches
6. Visualization
Displaying results in a human-readable way
Manual inspection

Figure 2: Token-based plagiarism detection process of JPlag.

3 ACHIEVING OBFUSCATION RESILIENCE

As JPlag is token-based, it abstracts from the underlying program and thus inherits resilience against some obfuscation attacks, including renaming, retyping, and obscuring values [8]. Furthermore, this includes all *lexical* [8] attacks. However, for broad obfuscation resilience, we introduce three defense mechanisms. Token sequence normalization via *token normalization graphs* (TNGs) [9] is specifically tailored to counter insertion- and reordering-based obfuscation attacks on programming assignments. After the tokenization, the mechanism constructs a TNG for each program, a language-independent version of a PDG for token sequences. The TNG enables the removal of dead tokens and establishes a deterministic token order. Thus, it generates a normalized token sequence invariant to insertion-based and reordering-based obfuscation. *Model subtree reordering* [10] is tailored for assignments involving tree-based modeling artifacts, such as EMF models. This mechanism counters reordering attacks, common for modeling assignments [12]. Reordering attacks are more straightforward for them since, for code, the statement sequence strongly impacts the behavior. To counteract this, we normalize the order of the nodes in the model tree. This is done through a multi-step algorithm, including a token distribution of a given subtree, its interpretation as a coordinate space, and a nearest neighbor path calculation. For any obfuscation attack to be effective against token-based plagiarism detectors, it must alter the token sequence to interfere with the subsequence matching. *Subsequence merging* [6] heuristically combines pairs of neighboring matches, thus reverting the effects of obfuscation attacks. Subsequence merging is language-independent and attack-type-independent and thus achieves broad resilience. As this defense mechanism solely operates on the match subsequences, it is independent of the language of the programs and can be applied for both modeling and programming assignments.

4 EVALUATION

To demonstrate the obfuscation resilience of JPlag, we evaluate it with a Java dataset from PROGpedia [7] and generate plagiarism based on reordering and insertion [9]. We compare the results with MOSS [1] and Dolos [5]. For all three detectors, we use default parameters. Dolos only supports single file submissions, thus limiting the choice of datasets. MOSS does not return all comparisons, only the ones with the highest scores. JPlag has neither of these limitations. Figure 1 shows the evaluation results. For each tool, we show the similarity distribution of the plagiarism instances (higher is better) and the unrelated originals (lower is better). For JPlag, note that the plagiarism instances have a median similarity of 100%. Thus, JPlag significantly outperforms Dolos with a median of 26.7% and MOSS with a median of 7.5%. Moreover, JPlag achieves low similarity scores for unrelated originals with a median similarity of 5.8%. Dolos achieves 14.3%, while MOSS achieves 4.5%. A high separation between the plagiarism instances and the unrelated originals allows straightforward detection and is thus preferable. For JPlag, we observe no overlap, with a median delta of 94.2 percentage points (pp). We observe an overlap and a median delta of only 12.4pp for Dolos. We observe the most substantial overlap and the lowest median delta with 3.0pp for MOSS. Thus, reliably differentiating plagiarism from the original is impaired for Dolos and impossible for MOSS. These results can be explained by Dolos and MOSS not providing any defense against automated obfuscation attacks. In practice, reordering and inserting statements is a viable attack for Dolos and MOSS but no longer for JPlag.

5 CONCLUSION

In summary, we introduced enhancements to JPlag, raising its resilience against automated obfuscation attacks. Our evaluation results demonstrate their effectiveness, positioning JPlag as a robust and reliable tool compared to other detectors. With these advances, JPlag addresses the evolving challenges of academic plagiarism, thus enabling educators to uphold academic integrity.

ACKNOWLEDGMENTS

We thank our student developers, Alexander Milster and Alexander Vogt, for their assistance. This work is based on the project *SofDCar* (19S21002) funded by the German Federal Ministry for Economic Affairs and Climate Action, by the Ministry of Science, Research and the Arts Baden-Württemberg (Az: 7712.14-0821-2), the pilot program Core Informatics of the Helmholtz Association (HGF), and by the topic Engineering Secure Systems of the HGF and by KASTEL Security Research Labs.

REFERENCES

[1] Alex Aiken. 2022. *MOSS Software Plagiarism Detector Website.* Stanford University. http://theory.stanford.edu/~aiken/moss/
[2] Rodrigo C Aniceto, Maristela Holanda, Carla Castanho, and Dilma Da Silva. 2021. Source Code Plagiarism Detection in an Educational Context: A Literature Mapping. In *FIE'21*. IEEE. https://doi.org/10.1109/FIE49875.2021.9637155
[3] Stella Biderman and Edward Raff. 2022. Fooling MOSS Detection with Pretrained Language Models. In *CIKM'22*. ACM. https://doi.org/10.1145/3511808.3557079
[4] Breanna Devore-McDonald and Emery D. Berger. 2020. Mossad: Defeating Software Plagiarism Detection. OOPSLA (2020). https://doi.org/10.1145/3428206
[5] Rien Maertens, Charlotte Van Petegem, Niko Strijbol, Toon Baeyens, Arne Carla Jacobs, et al. 2022. Dolos: Language-agnostic plagiarism detection in source code. *JCAL* 38, 4 (2022), 1046–1061. https://doi.org/10.1111/jcal.12662
[6] Nils Niehues. 2023. *Intelligent Match Merging to Prevent Obfuscation Attacks on Software Plagiarism Detectors.* master's thesis. Karlsruher Institut für Technologie (KIT). https://doi.org/10.5445/IR/1000167446
[7] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2023. PROGpedia. *Data in Brief* 46 (2023), 108887. https://doi.org/10.1016/j.dib.2023.108887
[8] Lutz Prechelt, Guido Malpohl, Michael Philippsen, et al. 2002. Finding plagiarisms among a set of programs with JPlag. *J.UCS* 8, 11 (2002), 1016.
[9] Timur Sağlam, Moritz Brödel, Larissa Schmid, and Sebastian Hahner. 2024. Detecting Automatic Software Plagiarism via Token Sequence Normalization. In *ICSE'24* (Lisbon, Portugal). IEEE. https://doi.org/10.1145/3597503.3639192
[10] Timur Sağlam, Sebastian Hahner, Larissa Schmid, and Erik Burger. 2024. Automated Detection of AI-Obfuscated Plagiarism in Modeling Assignments. In *ICSE-SEET'24* (Lisbon, Portugal). IEEE. https://doi.org/10.1145/3639474.3640084
[11] Timur Sağlam, Sebastian Hahner, Jan Willem Wittler, and Thomas Kühn. 2022. Token-Based Plagiarism Detection for Metamodels. In *MODELS-C* (Montreal, Quebec, Canada) *(MODELS '22)*. https://doi.org/10.1145/3550356.3556508
[12] Timur Sağlam, Larissa Schmid, Sebastian Hahner, and Erik Burger. 2023. How Students Plagiarize Modeling Assignments. In *MODELS-C* (Västerås, Sweden). IEEE, 98–101. https://doi.org/10.1109/MODELS-C59198.2023.00032
[13] Michael J. Wise. 1995. Neweyes: a system for comparing biological sequences using the running Karp-Rabin Greedy String-Tiling algorithm. *ISMB* 3 (1995).