PerfectPlaggie: Source Code Plagiarising Tool

Juraj PETRÍK*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
`petrik@fiit.stuba.sk`

Abstract. This paper presents new tool for creating plagiarism copies of existing source codes, which supports multiple levels of plagiarism. The tool is especially helpful for creating large datasets of source codes, which will be used for benchmarking different methods for measuring source code similarity. The hypothesis of creation of such a tool is supported by perfect plagiarism experiment – which shows, that it is pretty easy to trick existing source code plagiarism detection systems.

1 Introduction

According to Merriam-Webster dictionary is plagiarism defined as:

- to steal and pass off (the ideas or words of another) as one's own
- use (another's production) without crediting the source
- to commit literary theft
- present as new and original an idea or product derived from an existing source

Plagiarism is serious problem in academic field – in 2002 a survey was performed, where students of Swinburne and Monash University were asked if they were ever engaged in academic dishonesty – 85.4% of Swinburne University students and 69.3% of Monash University students admitted it. [1]

Another study done at Faculty of Informatics and Information Technologies STU in Bratislava states that 33% of students admitted that they have ever created plagiarism during their study and 63% of them have ever given their work to someone to plagiarize it. As you can see software plagiarism is a big problem in academic sphere, but in commercial is too. [4]

For instance, events such as Google vs Oracle shows that software plagiarism is even worse problem than we thought. Jury did not find Google guilty of violating any Oracle's patents, but it was clear, that Google had plagiarized parts of Oracle's source codes. [5]

Software plagiarism is commonly detected by automated tools, but the problem with these tools is, that they are mostly not designed to resist sophisticated obfuscation attacks. [7]

* Doctoral degree study programme in field: Software Engineering
Supervisor Assoc. Professor Daniela Chudá, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

Another essential problem in this area is lack of large enough datasets with obfuscated source codes with different types of obfuscation. These datasets are crucial for benchmarking new methods and tools to fight plagiarism.

Therefore, idea of tool which will be able to automatically produce this dataset is described in this paper.

2 Related work

We can divide obfuscation attacks to two main categories [6]:

- Lexical changes
- Structural changes

Lexical changes can be done in text editor and do not require any special knowledge of programming language. Examples of such changes are adding/removing/modification of comments, source code formatting, changing identifier names.

Structural changes need some sort of special knowledge of programming language (understanding of the language) and are very language dependent. Loop changes (while<->for<->do while), condition changes (if<->case<->ternary operator) or statement order replacement are examples of structural changes.

2.1 Source code obfuscators

2.1.1 ARTIFICE

This tool performs transformations directly on source codes. Obfuscation types are as follows – renaming of variables, if else statements are transformed to ternary operators and vice versa, while statements are transformed to while and vice versa, expanding variable definitions, variable assignments.

2.1.2 ProGuard

ProGuard is a free Java class file shrinker, optimizer, obfuscator and preverifier. It detects and removes unused classes, fields, methods and attributes. It optimizes bytecode and removes unused instructions. It renames the remaining classes, fields, and methods using short meaningless names. [8]

It is able to create more compact code, make software harder to reverse-engineer or detect dead code. However, ProGuard works on byte code level – so there is need of additional steps to be done – compiling and decompiling of the bytecode to get obfuscated source code.

2.1.3 yGuard

yGuard is a free Java bytecode obfuscator and shrinker that improves your software deployment by prohibiting unwanted access to your source code and drastically shrinking the processed Jar files at the same time. [12]

This tool could do name obfuscation – replacing package, class, method, field names with random strings. This tools can also do code shrinking – code shrinking engine detects which parts of codes could not be reached from a set of given entry points. These not needed parts are then removed.

Similar to ProGuard this tools works on byte code level, so compiling to bytecode and decompiling from bytecode is required to get source code.

3 Perfect plagiarism experiment

To support the hypothesis, an experiment of manual plagiarism creation was done. The aim of this experiment was to show that it is possible to create "perfect" plagiarism source code – the modified version of original file will not be marked as suspicious (similarity percentage is below threshold value) by any of chosen source code similarity checkers.

For this experiment MOSS [3][10], JPlag [9] and SIM were selected as representants of plagiarism detection systems – because these tools are commonly used in academic area for evaluating student's exams and are de facto used as standard benchmarking tools. Additionally, Simian was chosen, to see if there is any difference between special plagiarism detection systems and system used for software refactoring.

3.1 Source code sample

Red-black tree java implementation downloaded from the internet website providing source code samples was used in this experiment. This data structure implementation was chosen because it is typical student programming assignment at universities. Java as programming language was selected because it is most used programming language worldwide, also it is the most popular language among students.

3.2 Plagiarising

The goal of this experiment is too show, that it is possible to create perfect plagiarism in relatively short time. The person who was doing these obfuscations to achieve perfect plagiarism is simulating multiple levels of programming skills – to divide these copies by difficulty. Additionally, to simulate automatic obfuscation, the programmer was doing this changes without knowledge, what is this programming actually doing. In next three chapters these levels with results are described – first means that only beginner level is required to do this changes, second level requires advanced knowledge of language and third required semi-expert skill in this language.

3.2.1 First level

Just basic changes were done to the source code:

- Code formatting
- Comments removal
- Renaming of classes, methods and variables

Overall length of these changes was 10 minutes.

3.2.2 Second level

First level changes were done, plus some advanced changes:

- Loop changes (for->while, while->for…)
- Added new constants
- Negation of conditions
- Variable types
- Line ordering

Overall length of these changes was 30 minutes.

3.2.3 Third level

First level changes and second level changes were done, plus some more advanced changes:

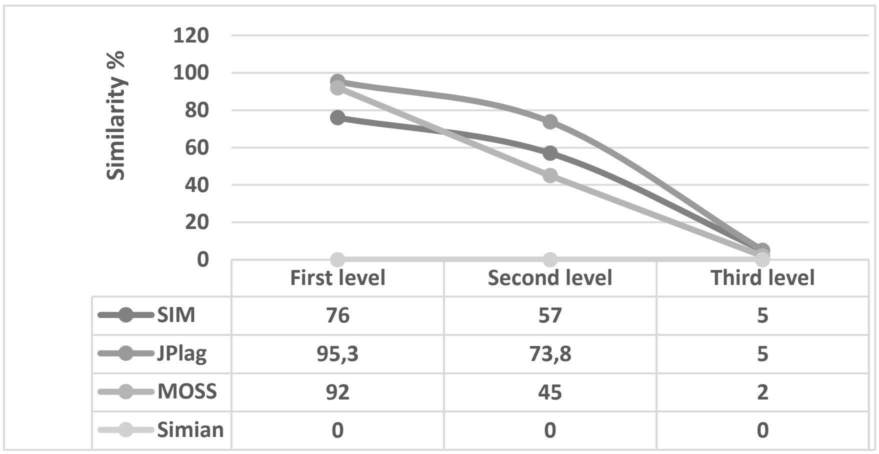
- Parameters order
- Variable modifiers
- Wrapping of return values
- Merging of some methods
- Splitting of some methods

Overall length of these changes was 30 minutes.

You can see example of this obfuscation level in Figure 2 (left side is original source code and on right side is obfuscated copy).

3.3 Discussion

Figure 1. Results of experiment represents results of completed experiment. First and second level obfuscations are not problem for plagiarism detection systems, however for Simian even first level changes are real problem. But third level changes are big deal even for specialized plagiarism detection tools – mainly it is because of excessive amount of unnecessary code added (wrapping).



	First level	Second level	Third level
SIM	76	57	5
JPlag	95,3	73,8	5
MOSS	92	45	2
Simian	0	0	0

Figure 1. Results of experiment.

When we are adding a lot of unneeded code, these tools are unable to detect this kind of situation, so the similarity percentage is naturally declining.

As we can see, it is easy to confuse standard plagiarism detection tools and it only took about one hour – it is surely faster than to do assignment on your own. Another alarming finding is that to do these changes it is not needed to be an expert in programming language, even the programmer does not need to know what the program is doing in real. This problem is not presented only in tested tools, but in other tools too. [11]

Based on these results I realized, that it is possible to create automated tool for creating obfuscated source codes. These obfuscated source codes (produced by the tool) will not be detected as suspicious (similarity value will be bellow threshold value) – to create perfect plagiarism by machine.

```
for (;;) {
 if (x.compareTo(current.element) < 0)
  current = current.left;
 else if (x.compareTo(current.element) > 0)
  current = current.right;
 else if (current != nullNode)
  return current.element;
 else
  return null;
}
```

```
while (true) {
 if (x.compareTo(getCurrentNode().getNodeElement()) > ZERO) {
  setCurrentNode(getCurrentNode().getRightNode());
 } else if (x.compareTo(getCurrentNode().getNodeElement()) < ZERO) {
  setCurrentNode(getCurrentNode().getLeftNode());
 } else if (getCurrentNode() != getNillLeaf()) {
  return getCurrentNode().getNodeElement();
 } else {
  return null;
 }
}
```

Figure 2. Source code obfuscation example.

4 PerfectPlaggie

Name of this tool is derived from “Perfect plagiarism” and plagiarism detection system Plaggie [2]. In next few chapters proposed design of this tool and planned supported obfuscation types are described.

4.1 Design

Figure 3 displays important architectural parts of PerfectPlaggie tool. These parts are GitHub crowler, file picker, obfuscator, tester.

GitHub crowler part will be searching for suitable projects on GitHub and downloading them. Suitable project means, that the project will have unit and integration tests with enough code coverage. Also it will ensure, that the downloaded project will be unique.

File picker part will pick only files, that will have 100% integration and unit test coverage – this is crucial for tester part of the PerfectPlaggie. It will also select files, that are interesting for dataset creation. The selection will be based on metrics such as LOC, NOM, MCC etc.

Obfuscator part will obfuscate source code files from file picker part. There will be multiple options of obfuscations – they are described in next chapter.

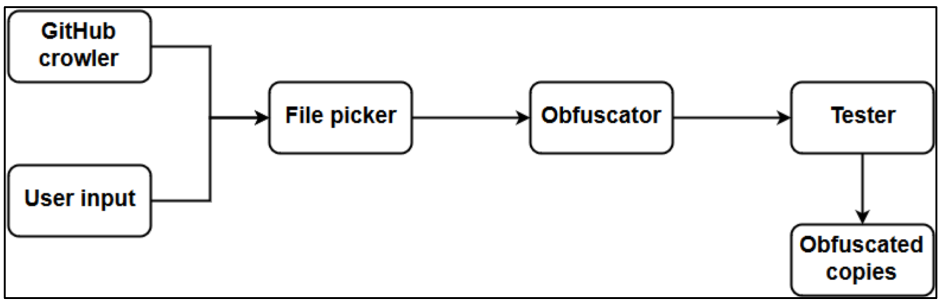


Figure 3. Design of PerfectPlaggie.

Testing of these obfuscated copies is very important – in theory if everything is done right, obfuscating will not change program behaviour, but we need to be sure. Thus, tester part will make sure, that all unit and integration tests pass.

4.2 Supported obfuscation types

Most important is to hide from computer, that source code is plagiarized. But also important thing is to think about that there is also possibility that some human expert will be also reviewing this obfuscated source code. For example, totally random variable names are very suspicious and therefore could trigger deep control of this code. Most important planned supported obfuscation types are described below.

Comments – there are multiple possibilities what to do with comments in source code. But we need to be very careful – because writing of comments is not so strict as writing source code, so any similarity in comments is considered as very suspicious. The safest option is too delete all comments.

Source code formatting – source code will be formatted according to programming language conventions – so even if two codes have exactly same formatting it is not suspicious – because it is convention.

Renaming of variables, methods, classes – random variable names or slight modification of original names is too suspicious for human experts. So there need to be some kind of synonyms dictionary for variables.

Conditions – the simplest method is to negate all conditions, but it is too easy to detect. Reordering of parts in composition conditions seems like a headache for plagiarism detection systems.

Line reordering – one of most effective methods for obfuscating, but also one of most difficult to implement – need to be implemented with help of PDG (Program dependence graph).

Splitting/merging of methods - another relatively simple and effective obfuscation. But we need to be careful with too much splitting or merging – there needs to be balance to be safe from human experts detection.

Wrapping – effective and simple. Plus, wrapping also adds a lot of extra lines of codes – so it is lowering similarity percentage by this way too.

Adding of redundant code – this can get very tricky. Because this added code must look and must do similar tasks like original, otherwise it could get really suspicious.

5 Conclusions

This paper proposes design of source code obfuscating system called PerfectPlaggie. This tool is designed to construct autonomously big datasets of obfuscated (plagiarized) copies from freely available source codes.

The hypothesis that such a tool can be constructed is supported by perfect plagiarism experiment. This experiment shows that it is not too complicated to create “perfect plagiarism” for human, also it is not much time consuming.

This tool is unique – because it works directly on transformation of source code (not bytecode), it is autonomous and user can choose what obfuscations he wants to be applied to original source code.

Because this is proposal of such a tool, there needs implementation to be done in future, to fully confirm or reject the hypothesis. Also I see potential in research for new obfuscation types, but we need to create them that way, that even human experts will not get suspicious – and this is really challenging task.

Acknowledgement: This contribution is the partial result of the project Research of methods for acquisition, analysis and personalized conveying of information and knowledge, ITMS 26240220039, co-funded by the ERDF.

References

[1] Arwin, C., Tahaghoghi, S.M.M.: Plagiarism Detection across Programming Languages. In *Twenty-Ninth Australasian Computer Science Conference* (ACSC2006). (2003). pp. 10.

[2] Ahtiainen, A., Surakka, S., Rahikainen, M.: Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. In *Proceedings of the 6th Baltic Sea conference on Computing education research*: Koli Calling. (2006). , pp. 141–142.

[3] Bowyer, K.W., Hall, L.O.: Experience using MOSS to detect cheating on programming assignments. In *Frontiers in Education Conference 1999 FIE99 29th Annual*. (1999), vol. 3, Piscataway, NJ, United States, pp. 13–18.

[4] Chudá, D. Návrat, P., Kováčová, B., Humay, P.: The issue of (software) plagiarism: A student view. In *IEEE Transactions on Education*. (2012), vol. 55, no. 1, pp. 22–28.

[5] Fiducia, N.: *When Two Worlds Collide: The Oracle And Google Dispute*. Mondaq.com. (2013). Available at: http://www.mondaq.com/unitedstates/x/271942/ [Accessed: 14 Feb 2016].

[6] Joy M., Luck, M.: Plagiarism in Programming Assignments. In *IEEE Transactions of Education*. (1999), vol 42, no. 2, pp. 129-133.

[7] Juan, A.C.: Studying the Impact of Obfuscation on Source Code Plagiarism Detection. January, (2014), pp. 1–39.

[8] Lafortune, E.: 'ProGuard' Proguard.sourceforge.net. (2013), [online] Available at: http://proguard.sourceforge.net/ [Accessed: 18 Feb 2016].

[9] Prechelt, L. Malpohl, G., Philippsen, M.: Finding Plagiarisms among a Set of Programs with JPlag. In *Journal Of Universal Computer Science*. (2002). vol. 8, no. 11, pp. 1016–1038.

[10] Schleimer, S., Wilkerson, D.S., Aiken, A.: Winnowing: Local Algorithms for Document Fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data* - SIGMOD '03. (2003), pp. 76–85.

[11] Tahir Ali, A.M. EL, Abdulla H.M.D., Snášel, V.: Overview and comparison of plagiarism detection tools. In *CEUR Workshop Proceedings*. (2011). vol. 706, pp. 161–172.

[12] yWorks GmbH. yGuard - Java™ Bytecode Obfuscator and Shrinker. (2016). Available at: https://www.yworks.com/products/yguard [Accessed: 18 Feb 2016].