Effective Identification of Source Code Authors Using Byte-Level Information

Georgia Frantzeskou Efstathios Stamatatos Stefanos Gritzalis Sokratis Katsikas
Laboratory of Information and Communication Systems Security
Department of Information and Communication Systems Engineering
University of the Aegean, Karlovasi, Samos, 83200, Greece
(gfran, stamatatos, sgritz, ska)@aegean.gr

ABSTRACT

Source code author identification deals with the task of identifying the most likely author of a computer program, given a set of predefined author candidates. This is usually .based on the analysis of other program samples of undisputed authorship by the same programmer. There are several cases where the application of such a method could be of a major benefit, such as authorship disputes, proof of authorship in court, tracing the source of code left in the system after a cyber attack, etc. We present a new approach, called the SCAP (Source Code Author Profiles) approach, based on byte-level n-gram profiles in order to represent a source code author's style. Experiments on data sets of different programming-language (Java or C++) and varying difficulty (6 to 30 candidate authors) demonstrate the effectiveness of the proposed approach.

A comparison with a previous source code authorship identification study based on more complicated information shows that the SCAP approach is language independent and that n-gram author profiles are better able to capture the idiosyncrasies of the source code authors. Moreover, the SCAP approach is able to deal surprisingly well with cases where only a limited amount of very short programs per programmer is available for training. It is also demonstrated that the effectiveness of the proposed model is not affected by the absence of comments in the source code, a condition usually met in cyber-crime cases.

Categories and Subject Descriptors

K.4.1 [**Public Policy Issues**] *Abuse and crime involving computers*

General Terms

Experimentation, Security.

Keywords

Source Code Authorship Identification, Software Forensics.

1. OVERVIEW AND MOTIVATION

Nowadays, in a wide variety of cases source code authorship identification has become an issue of major concern. Such situations include authorship disputes, proof of authorship in court, cyber attacks in the form of viruses, trojan horses, logic bombs, fraud, and credit card cloning etc.

The most extensive and comprehensive application of authorship analysis is in literature. One famous authorship analysis study is related to Shakespeare's works and is dating back over several centuries. Recently, a number of authorship attribution approaches have been presented ([15, 6, 11]) proving that the author of a natural language text can be reliably identified.

Although source code is much more formal and restrictive than spoken or written languages, there is still a large degree of flexibility when writing a program [2]. Spafford and Weeber [14] suggested that it might be feasible to analyze the remnants of software after a computer attack, such as viruses, worms or trojan horses, and identify its author.

On the evening of 2 November 1988, someone infected the Internet with a *worm* program. Spafford [13] conducted a manual analysis of the program using three reversed-engineered versions and conclusions were drawn about the author's abilities and intent. Longstaff and Shultz [9] studied the WANK and OILZ worms which in 1989 attacked NASA and DOE systems. They have manually analyzed code structures and features and have reached a conclusion that three distinct authors worked on the worms. In addition, they were able to infer certain characteristics of the authors, such as their educational backgrounds and programming levels. Sallis et al [12] expanded the work of Spafford and Weeber by suggesting some additional features, such as cyclomatic complexity of the control flow and the use of layout conventions.

An automated approach was taken by Krsul and Spafford [8] to identify the author of a program written in C. The study relied on the use of software metrics, collected from a variety of sources. They were divided into three categories: layout, style and structure metrics. These features were extracted using a software analyzer program from 88 programs belonging to 29 authors. A tool was developed to visualize the metrics collected and help select those metrics that exhibited little within-author variation, but large between-author variation. A statistical approach called discriminant analysis (SAS) was applied on the chosen subset of metrics to classify the programs by author. The experiment achieved 73% overall accuracy.

Other research groups have examined the authorship of computer programs written in C++ [7]; [10], a dictionary based system called IDENTIFIED was developed to extract source code metrics for authorship analysis [3]. Satisfactory results were obtained for C++ programs using case-based reasoning, feed-forward neural network, and multiple discriminant analysis [10]. The best prediction accuracy has been achieved by Case-Based Reasoning and it was 88% for 6 different authors.

ICSE'06, May 20–28, 2006, Shanghai, China.

Ding [1], investigated the extraction of a set of software metrics of a given Java source code, that could be used as a fingerprint to identify the author of the Java code. The contributions of the selected metrics to authorship identification were measured by a statistical process, namely canonical discriminant analysis, using the statistical software package SAS. A set of 56 metrics of Java programs was proposed for authorship analysis. Forty-six groups of programs were diversely collected. Classification accuracies were 62.7% and 67.2% when the metrics were selected manually while those values were 62.6% and 66.6% when the metrics were chosen by SDA (stepwise discriminant analysis).

The traditional methodology that has been followed in this area of research is divided into two main steps. The first step is the extraction of software metrics representing the author's style and the second step is using these metrics to develop models that are capable of discriminating between several authors, using a classification algorithm.

However, there are some disadvantages in this traditional approach. The first is that software metrics used are programming - language dependant. For example metrics used in Java cannot be used in C or Pascal. The second is that metrics selection is not a trivial process and usually involves setting thresholds to eliminate those metrics that contribute little to the classification model. As a result, the focus in a lot of the previous research efforts, such as [1] and [4] was into the metrics selection process rather than into improving the effectiveness and the efficiency of the proposed models.

In this paper we present an approach to source code author identification we call the SCAP (Source Code Author Profiles) approach, which is an extension of a method that has been applied to natural language text authorship identification [3]. The assumption used is, that programs are written by a single author and that it is possible in cases where programs are the result of team effort to distinguish the pieces of code written by a certain author. In the SCAP method, byte-level n-grams are utilised together with author profiles. We propose a new simplified profile and similarity measure that proved to be quite effective even in cases where only limited training set is available for each author. Our methodology is programming - language independent since it is based on low-level information and has been tested to data sets from two different programming languages Java and C++. Special attention is paid to the evaluation methodology. Disjoint training and test sets of equal size were used in all the experiments in order to ensure the reliability of the presented results. Moreover, the significance of the comments in the source code is examined. It is demonstrated that the effectiveness of the SCAP model is not affected by the absence of comments, a condition usually met in cyber-crime cases.

2. DESCRIPTION OF PRELIMINARY RESULTS

2.1 THE SCAP Approach

In this paper, we present the SCAP (Source Code Author Profiles) approach, which is an extension of a method that has been successfully applied to text authorship identification [5]. It is based on byte level n-grams and the utilization of a similarity measure used to classify a program to an author. Therefore, this method does not use any language-dependent information.

An n-gram is an n-contiguous sequence and can be defined on the byte, character, or word level. Byte, character and word n-grams have been used in a variety of applications such as text authorship attribution, speech recognition, language modelling, context sensitive spelling correction, optical character recognition etc. In our approach, the Perl package Text::N-grams [6] has been used to produce n-gram tables for each file or set of files that is required. The n-gram table contains the n-grams found in a source code file in descending frequency order.

The algorithm we propose, computes n-gram based profiles that represent each of the author category. First, for each author the available training source code samples are concatenated to form a big file. Then, the set of the L most frequent n-grams of this file is extracted. Hence the profile of an author we propose is a Simplified Profile (SP) and is the set {x_1; x_2;,…,x_L} of the L most frequent n-grams x_i Similarly, a profile is constructed for each test case (a simple source code file). In order to classify a test case in to an author, the profile of the test file SP_T is compared with the profiles of all the candidate authors SP_A based on a similarity measure. The similarity distance is given by the size of the intersection of the two profiles:

$$\left| SP_A \cap SP_T \right| \quad (1)$$

where |X| is the size of X. In other words, the similarity measure we propose is the amount of common n-grams in the profiles of the test case and the author. The program is classified to the author with whom we achieved the biggest size of intersection. Hereafter, this similarity measure will be called Simplified Profile Intersection (SPI). We have developed a number of perl scripts in order to create the sets of n-gram tables for the different values of n (i.e., n-gram length), L (i.e., profile length) and for the classification of the program file to the author with the smallest distance.

One of the inherent advantages of this approach is that it is language independent since it is based on low-level information. As a result, it can be applied with no additional cost to data sets where programs are written in C++, Java, perl etc. Moreover, it does not require multiple training examples from each author, since it is based on one profile per author. The more source code programs available for each author, the more reliable the author profile. Also the new similarity measure SPI is suitable for cases where only a limited training set is available for each author. Note that this is especially the case in many source code author identification problems, where only a few short source code samples are available for each author.

2.2 Experiments & Results

2.2.1 Comparison with a previous method

We have performed a number of experiments in order to demonstrate the effectiveness of our approach in a number of data sets with different characteristics. The data set of the first experiment has been initially used by Mac Donell [10] for evaluating a system for automatic discrimination of source code author based on more complicated, programming language-dependent measures. All the source code samples were written in C++. The data set was split (as equally as possible) into the training set 50% (134 programs) and the test set 50% (133 programs) and we had to classify programs from 6 different authors. The best reported result by Mac Donell [10] on the test set was 88% using the case-based reasoning (that is, a memory-based learning) algorithm. We used byte-level n-grams extracted from the sample programs in order to create the author and program profiles as well as the author and program simplified profiles. Classification accuracy reached 100% (see Table 1) for various combinations of n (n-gram size) and L (profile size), much better than the best reported ([10]) accuracy for this data set (88% on the test set). This proves that the presented methodology can cope effectively with the source code author identification problem based on low-level information.

Table 1. Classification accuracy (%) on the MacDonellC++ data set for different values of n-gram size and profile size

Profile Size L	n-gram Size					
	3	4	5	6	7	8
500	100	100	100	98	98	98
1000	100	100	100	100	100	99
1500	100	100	100	100	99	100
2000	100	100	100	100	100	100
2500	100	100	100	100	100	100
3000	100	100	100	100	100	100

2.2.2 Evaluation on another programming language.

The next experiment was performed on student programs written in Java. Henceforth, this data set will be called StudentJava. There are 8 different programmers in total and 6-8 programs per author. In particular, the source code samples of this data set include assignments from an introductory programming course. Hence the programs on this data set were on the same subject written by different programmers. The size of the programs was between 36 and 258 lines of code. The data set was split into quasi equally-sized training and test sets. This data set contains limited data per programmer (6-8 per programmer) and, moreover, the available source code samples are short (mean LOC per program 129). In addition, the programs written by students usually have no comments while their programming style is influenced by the guidelines of the instructor. More significantly, the source code samples are plagiarised. All these facts introduce some extra difficulties in the source code authorship analysis. As a consequence, the classification results for the StudentJava data set are expected to be lower than that of McDonellC++ data set. The best result achieved was 88.5% for a number of n, L combinations (see Table 2). These results are quite satisfactory given the difficulties of this data set. This indicates that the SCAP method can reliably handle difficult cases. Finally, it is demonstrated that the proposed method can be applied to any programming language equally well. Note, that no modification or adjustment is required in order to apply our method to this data set.

2.2.3 The role of comments

The experiments described in this section are based on a data set of open source programs written in Java. In more detail, source code samples by 8 different authors were downloaded from freshmeat.net. The amount of programs per programmer is highly unbalanced, ranging from 4 to 30 programs per author. The source code sample size was between 23-760 lines of code. In many cases, source code samples by the same programmer have common comment lines at the beginning of the program. Such comment lines were manually removed since they could (positively) influence the classification accuracy. The total number of programs was 107 and they were split into equally-training and test sets. Hereafter, this data set will be called OSJava1. This data set provides a more realistic case of source code author identification than student programs. Open source code is similar to commercial programs which usually have comments and they are usually well structured. Most of the open source programs were longer than the student programs. More importantly, this data set enables us to examine the role comments play in the classification model. We have decided to perform three different experiments on this data set. For this reason, we first filtered out any comments from the OSJava1 data set, resulting a new data set (hereafter, called NoComJava). Then, another data set was constructed using only the comments from each source code sample (hereafter, called OnlyComJava). Note that in the latter case, the resulting data set includes fewer programs than the original because any source code files with no comments were removed. The OnlyComJava data set includes samples by 6 different authors with 9 – 25 files per author.

On this set of experiments we used two different profile sizes 1500 and 2000, since they provide the best results (as has been demonstrated in the previous experiments). The classification results for the OSJava1 data set are perfect for any n-gram size, 100% in all n, L combinations (see Table 2). This is mainly because the source code samples of this data set are relatively long. Moreover, for many candidate authors there is a sufficient amount of training samples. Interestingly, the accuracy remains at the top level, between 94%-100%, when removing the comments lines of these samples (NoComJava data set). This is a strong indication that the proposed SPI similarity measure suits the source code author identification problem. On the other hand, when examining only the comments of each source code sample (OnlyComJava dataset), the results remain high between 95 – 100%. This result is an indication that it is possible to identify the author of a program by analysing its comments only.

Table 2. Best classification results

Data Sets	No of Authors	Best Classification accuracy	Profile size(L) & n-gram size(n) used for the best accuracy
MacDonellC++	6	100%	As shown in Table1
StudentJava	8	88.5%	L=2000,2500 & n=6
OSJava1	8	100%	L=1500,2000 & n=3,4,5,6,7,8
NoComJava	8	100%	L=2000 & n=5,6,7,8
OnlyComJavas	6	100%	L=1500 & n=5 and L=2000 & n=4,5,6
OSJava2	30	96.9%	L=1500 & n=7

2.2.4 Dealing with many authors.

The previous experiments have shown that our approach is quite reliable when dealing with a limited number of candidate authors (6 to 8). In this section we present an experiment that demonstrates the effectiveness of the proposed method when dealing with dozens of candidate authors. For that purpose a data set was created by downloading open-source code samples by 30 different authors from freshmeat.net. Hereafter, this data set will be called OSJava2. This data set includes programs on the same application domain written by different authors. In addition the samples of many authors are written over a long time period and therefore there might be programming style changes of certain authors. The samples were split into equally-sized training and test set. Note that the training set was highly unbalanced (as OSJava1). The best accuracy result was 96.9% and in most cases, accuracy exceeds 95%, indicating that the SCAP approach can reliably identify the author of a source code sample even when there are multiple candidate authors.

3. CONCLUSIONS

In this paper, the SCAP approach to source code authorship analysis has been presented. It is based on byte-level n-gram profiles. The current version of the SCAP approach does not claim that authorship identification is possible when programs have been written with the purpose to disguise the author. Our method was applied to data sets of different programming languages and varying difficulty demonstrating surprising effectiveness. The SCAP approach includes a new simplified profile and a similarity measure that better suit the characteristics of the source code authorship analysis problem. In particular the SCAP approach can deal with cases where very limited training data per author is available (especially, when at least one author profile is shorter than the predefined profile size) or there are multiple candidate authors, conditions usually met in source code authorship analysis problems (e.g. source code authorship disputes, etc.) with no significant compromise in performance.

More significantly, the role of comments in the source code is examined. The SCAP method can reliably identify the most likely author when there are no comments in the available source code samples, a condition usually met in cyber-attacks. However, it is demonstrated that the comments provide quite useful information and can significantly assist the classification model to achieve quasi-perfect results. Actually, the comments alone can be used to identify the most likely author in open-source code samples where there are detailed comments in each program sample.

The presented experiments indicate that the best classification models are acquired for n-gram size 6 or 7 and profile size 1500 or 2000. However, more experiments have to be performed on various data sets in order to be able to define the most appropriate combination of n-gram size and profile size for a given problem. The high accuracy achieved with the data sets, OSJava1 and OSJava2 could be contributed (to some extent) to the fact that the programs used from each author belong to a different application domain, although some initial experiments have shown that variable, class and method names do not affect classification accuracy. It is under investigation the factors that contribute to authorship identification using the SCAP approach (i.e. style, variable naming etc). Finally, the visualization of the stylistic properties of each author could be of major benefit in order to explain the differences between candidate source code authors.

4. REFERENCES

[1] Ding, H., Samadzadeh, M., H., *Extraction of Java program fingerprints for software authorship identification,* The Journal of Systems and Software, Volume 72, Issue 1, Pages 49-57 June 2004.

[2] Frantzeskou, G., Gritzalis, S., Mac Donell, S., *Source Code Authorship Analysis for supporting the cybercrime investigation process*, (ICETE04), Vol 2, pages (85-92), 2004.

[3] Gray, A., Sallis, P., and MacDonell, S.,, Identified: A dictionary-based system for extracting source code metrics for software forensics. In Proceedings of SE:E&P'98, IEEE Computer Society Press, pages 252–259., 1998.

[4] Gray, A., Sallis, P., and MacDonell, S., *Software forensics: Extending authorship analysis techniques to computer programs*, in Proc. 3rd Biannual Conf. Int. Assoc. of Forensic Linguists (IAFL'97), pages 1-8, 1997.

[5] Keselj, V., Peng, F., Cercone, N., Thomas, C., *N-gram based author profiles for authorship attribution*, In Proc. Pacific Association for Computational Linguistics 2003.

[6] Keselj, V.,. Perl package Text::N-grams http://www.cs.dal.ca/~vlado/srcperl/N-grams , 2003.

[7] Kilgour, R. I., Gray, A.R., Sallis, P. J., and MacDonell, S. G., *A Fuzzy Logic Approach to Computer Software Source Code Authorship Analysis,* Accepted In Proc. Of (ICONIP'97). Dunedin. New Zealand, 1997.

[8] Krsul, I., and Spafford, E. H, *Authorship analysis: Identifying the author of a program,* In Proc. 8th National Information Systems Security Conference, pages 514-524, National Institute of Standards and Technology., 1995.

[9] Longstaff, T. A., and Schultz, E. E., Beyond Preliminary Analysis of the WANK and OILZ Worms: A Case Study of Malicious Code, Computers and Security, 12:61-77, 1993.

[10] MacDonell, S.G**,** and Gray, A.R. Software forensics applied to the task of discriminating between program authors. *Journal of Systems Research and Information Systems* 10: 113-127 (2001).

[11] Peng, F., D., Shuurmans, and S., Wang., *Augmenting naive bayes classifiers with statistical language models,* Information Retrieval Journal, 7(1): 317-345, 2004.

[12] Sallis P., Aakjaer, A., and MacDonell, S., *Software Forensics: Old Methods for a New Science*. Proceedings of SE:E&P'96. Dunedin, New Zealand, IEEE Computer Society Press, 367-371, 1996.

[13] Spafford, E. H., *The Internet Worm Program: An Analysis*," Computer Communications Review, 19(1): 17-49, 1989.

[14] Spafford, E. H., and Weeber, S. A., *Software forensics: tracking code to its authors*, Computers and Security, 12:585-595, 1993.

[15] Stamatatos, E., N., Fakotakis, and G. Kokkinakis. *Automatic text categorisation in terms of genre and author.* Computational Linguistics, 26(4): 471-495, 2000