Identifying Authorship by Byte-Level N-Grams: The Source Code Author Profile (SCAP) Method

Georgia Frantzeskou
Efstathios Stamatatos
Stefanos Gritzalis
Laboratory of Information and Communication Systems Security
Department of Information and Communication Systems Engineering
University of the Aegean

Carole E. Chaski, PhD
Blake Stephen Howald, JD
Institute for Linguistic Evidence, Inc

Abstract

Source code author identification deals with identifying the most likely author of a computer program, given a set of predefined author candidates. There are several scenarios where digital evidence of this kind plays a role in investigation and adjudication, such as code authorship disputes, intellectual property infringement, tracing the source of code left in the system after a cyber attack, and so forth. As in any identification task, the disputed program is compared to undisputed, known programming samples by the predefined author candidates. We present a new approach, called the SCAP (Source Code Author Profiles) approach, based on byte-level n-gram profiles representing the source code author's style. The SCAP method extends a method originally applied to natural language text authorship attribution; we show that an n-gram approach also suits the characteristics of source code analysis. The methodological extension includes a simplified profile and a less complicated, but more effective, similarity measure. Experiments on data sets of different programming-language (Java or C++) and commented/commentless code demonstrate the effectiveness of these extensions. The SCAP approach is programming-language independent. Moreover, the SCAP approach deals surprisingly well with cases where only a limited amount of very short programs per programmer is available for training. Finally, it is also demonstrated that SCAP effectiveness persists even in the absence of comments in the source code, a condition usually met in cyber-crime cases.

1. The Forensic Significance of Source Code

Nowadays, in a wide variety of legal cases it is important to identify the author of a usually limited piece of programming code. Such situations include cyber attacks in the form of viruses, Trojan horses, logic bombs, fraud, and credit card cloning, code authorship disputes, and intellectual property infringement. Identifying the authorship of malicious or stolen source code in a reliable way has become a primary goal for digital investigators (Spafford and Weeber 1993). Please see Appendix 1 for a legal analysis of the forensic significance of source code.

In this paper we present a new approach, which we call the SCAP (Source Code Author Profiles) method, based on byte-level n-gram –or sequential slicing– profiles representing the source code author's style. The SCAP method extends an approach originally applied to natural language text authorship attribution by Keselj et. al. (2003). We show that the n-gram approach also suits the characteristics of source code analysis. Our methodological extension includes a simplified profile and a less complicated but more effective similarity measure. Although Frantzeskou's doctoral research includes numerous experiments which test SCAP under multiple forensically-significant conditions, in this article we present only two experiments. These experiments show that the SCAP method functions well on different programming languages, deals surprisingly well with cases where only a limited amount of very short programs per programmer is available for training, and performs well even in the absence of comments in the source code, a condition usually met in cyber-crime cases.

The rest of this paper is organized as follows. Section 2 contains a brief review of relevant research in the area of authorship attribution, focusing on Keselj et. al.'s (2003) method. Section 3 describes our approach. Section 4 presents the results two experiments using SCAP. Finally, section 5 discusses the forensic application of SCAP and our research agenda for future work.

2. Related Work in Computing and Natural Language Authorship

The general methodology of authorship attribution applies to both natural and computing languages. Although source code is much more grammatically restrictive than natural languages, there is still a large degree of flexibility when writing a program (Krsul and Spafford 1996).

Computational authorship attribution methodology for both natural and computing languages requires two main steps (Krsul and Spafford 1995; Chaski 1997, 2005; MacDonell and Gray 2001, Ding and Samadzadeh 2004). The first step is the extraction of variables representing the author's style. Ideally, authorial features should have low within-author variability, and high between-author variability (Krsul and Spafford 1996, Kilgour, Gray, Sallis and MacDonell 1997, Chaski 1997). The second step is applying a statistical or machine learning algorithm to these variables in order to develop models that are capable of discriminating between several authors. Defining the variables and discovering the best classification algorithm for the defined variables is a difficult, empirical task, but it is feasible and prevents subjective pronouncements which are no longer considered by courts to be acceptable scientific forensic evidence (Chaski 1997, 2005).

Authorship Attribution Methods for Computing Languages

In general, when authorship attribution methods have been developed for computing languages, the suggested software features are programming language-dependent and require either computational cost or hand-coding for their calculation. The main focus of the previous work was the definition of the most appropriate features for representing the style of an author (Oman and Cook 1989; Longstaff and Shultz 1993; Spafford and Weeber 1993;

Sallis, et. al. 1996). For author identification in computing languages, proposed metrics have included, for example, indentation, placement of comments, placement of braces, character preferences, construct preferences, statistical distribution of variable lengths and function name lengths, statistical distribution of lines of code per function, ratio of keywords per lines of code, spelling errors, the degree to which code and comments match, and whether identifiers used are meaningful. This list shows that many of the previously proposed features either cannot be measured objectively in any source code program (a condition which also plagued natural language authorship identification methods, until very recently) or require hand-coding.

Krsul and Spafford (1995) developed a software analyzer program to automate the coding of software metrics. The software analyzer extracted layout, style and structure features from 88 C programs belonging to 29 known 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. Discriminant function analysis was applied on the chosen subset of metrics to classify the programs by author. The experiment achieved 73% overall accuracy.

MacDonell and his colleagues (Kilgour, Gray, Sallis and MacDonell 1997; Gray, Sallis and MacDonell 1998; MacDonell and Gray 2001) have automated authorship identification of computer programs written in C++. Gray, Sallis and MacDonell 1998 developed a dictionary-based system called IDENTIFIED (Integrated Dictionary-based Extraction of Non-language-dependent Token Information for Forensic Identification, Examination, and Discrimination) to extract source code metrics for authorship analysis. In MacDonell and Gray's 2001 work, satisfactory results were obtained for C++ programs using case-based reasoning, feed-forward neural network, and multiple discriminant analysis. The best prediction accuracy – at 88% for 7 different authors-- was achieved using Case-Based Reasoning.

Focusing on Java source code, Ding and Samadzadeh (2004) investigated the extraction of a set of software metrics that could be used to identify the author. A set of 56 metrics of Java programs was proposed for authorship analysis. The contributions of the selected metrics to authorship identification were measured by canonical discriminant analysis. Forty-six groups of programs were diversely collected. They achieved a classification accuracy of 87.0% with the use of canonical variates.

This brief review of previous work reveals four criteria for our own research agenda. First, metrics selection is not a trivial process and usually involves setting thresholds to eliminate those metrics that contribute little to the classification model. Second, some of the metrics are not readily extracted automatically because they involve subjective judgments. Third, many software metrics are programming-language dependent. For example, metrics useful for Java programs cannot be used for examining C or Pascal programs. Fourth, even with automated feature extraction and analysis, the classification accuracy rates do not reach 90%.

In sum, the previous work in author identification of programming code has suffered from language-dependence, manual coding of subjective features and accuracy rates below 90%. In this context, our goal is to provide a fully-automated, language-independent method with high reliability for distinguishing authors and assigning programs to programmers. Can recent

computational approaches in natural language author identification provide assistance in reaching our goal?

Authorship Attribution Methods for Natural Language

For natural language texts, authorship puzzles have engaged scholars in a wide range of disciplines, from literature (Elliot and Valenza 1991), linguistics (Chaski 1997, 2004, 2005) and computer science (Keselj et. al. 2003, Peng et. al. 2004, Stamatatos et. al. 2000). Recently, a number of computational authorship attribution approaches have been presented (Peng et. al. 2004, Stamatatos et. al. 2000, O'Brien and Vogel 2003, Chaski 2005) proving that the author of a natural language text can be reliably identified with methods which focus on syntactic, lexical and character-level variables using classification procedures such as naïve Bayes, chi by degrees of freedom, and discriminant function analysis.

The SCAP method extends Keselj et. al.'s 2003 work, so it is important to describe this particular method. In Keselj et. al.'s 2003 work, the text is decomposed into character-level n-grams (using a Perl text processing program by Keselj 2003). 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 long been used in a variety of applications such as speech recognition, language identification, context sensitive spelling correction, optical character recognition etc. For the Roman alphabet's 26 graphemes, 676 character-level bi-grams are thus possible, although not all of these possible bi-grams will be instantiated in any given text due to the phonotactic constraints of any particular natural of programming language; for instance, English permits [xa] as in [Xavier] but not [xb], although the bi-gram [xb] may occur in a mathematical equation or programming variable name.

Keselj et. al. (2003) defines an author profile as "a set of length L of the most frequent n-grams with their normalized frequencies." The profile of an author is, then, the ordered set of pairs $\{(x_1; f_1); (x_2; f_2),\ldots,(x_L; f_L)\}$ of the L most frequent n-grams x_i and their normalized frequencies f_i. Keselj et. al. (2003) determine authorship based on the dissimilarity between two profiles, comparing the most frequent n-grams. Identical texts will obviously have an identical set of L most frequent bi-grams, and thus have zero dissimilarity. Different texts will be more or less similar to each other, based on the amount of most-frequent bi-grams which they share. It is important to note that the normalized frequencies constitute the author profile in Keselj et. al.'s 2003 approach.

The original dissimilarity measure used by Keselj et. al. (2003) in text authorship attribution is a form of relative distance:

$$\sum_{n\in profile}\left(\frac{f_1(n)-f_2(n)}{\frac{f_1(n)+f_2(n)}{2}}\right)^2=\sum_{n\in profile}\left(\frac{2(f_1(n)-f_2(n))}{f_1(n)+f_2(n)}\right)^2 \quad (1)$$

where $f_1(n)$ and $f_2(n)$ are either the normalized frequencies of an n-gram n in the two compared texts or 0 if the n-gram does not exist in the text(s). A text is classified to the author whose

profile has the minimal distance from the text profile, using this measure. Hereafter, this distance measure will be called Relative Distance (RD).

3. The Scap Method

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 by Peng et. al. (2004). As in Keselj et. al.'s 2003 work, a profile for each author is developed from the frequency rank of n-grams and a similarity measure is used to classify a program to an author. But the SCAP method differs from the Keselj approach in two ways: first, we use the raw frequencies of the byte-level n-grams, and second, we use a simple, overlap measure for classification. The procedure is explained in the following steps.

- Step 1. Divide the known source code programs into training and testing data.
- Step 2. Concatenate all the programs in the training set into one large file. Leave the testing data programs in their own files.
- Step 3. For each author training and testing file, get the author profile:
    - Step 3.1. Extract the n-grams at the byte-level, including all non-printing characters. All characters, even the non-printable, such as spaces, tabs, new line characters are included in the extraction of the n-grams.
    - Step 3.2. Get the frequency for each n-gram type.

In our analyses, Keselj's (2003) Perl package Text::N-grams has been used to produce n-gram tables for each file or set of files that is required. An example of such a table is given in Table 1A. The first column contains the n-grams found in a source code file and the second column the corresponding frequency of occurrence.

Table 1A. N-gram Frequencies Extracted From a Source Code File

3-gram	Frequency
sio	28
_th	28
f_(	20
_=_	17
usi	16
_ms	16
out	15
ine	15
\n/*	15
on_	14
_in	14
fp_	14
the	14
sg_	14
_i_	14
in_	14

The n-grams are ranked by frequency, in descending order, so that the most frequently-occurring n-grams are listed first.

- Step 3.3. List the n-gram types in descending frequency, so the most frequent are listed first.

This is the author profile. The author profile will have varying lengths depending on the length (in terms of characters) of the programming data and the length of the n-gram. Keep a record of each author's profile length for each n-gram length, such as shown in Table 1B.

Table 1B. Example of Varying Length of Author Profile for Varying N-Grams

	bigram	trigram	4-gram	5-gram	6-gram	7-gram
author 1	250	400	650	890	1000	1300
author 2	475	680	980	1200	1700	1982
test document	100	223	447	589	793	874

Unlike Keselj et. al.'s approach, SCAP does *not* use the normalized frequencies of the n-grams. Hence the profile we propose is a Simplified Profile (SP). The SP is the set of the L most frequent n-grams $\{x_1, x_2,\ldots,x_L\}$, when they are ranked by descending frequency. The actual frequency is not used mathematically except for ranking the n-grams. The length L of the profile is discussed below.

- Step 4. For each test file, compare its profile to each author using the SPI measure:
  - Step 4.1. Select a specific n-gram length, such as trigram (or 6-gram or whatever).
  - Step 4.2. Select a specific profile length at which to cut off the author profile.

For instance, given the data in Table 1B, if we select trigram and profile length of 500, when we compare author 1 to author 2, we will use all of author 1's trigrams but we will use only the first 500 most frequent trigrams from author 2. Author 1's profile only contains 400 trigrams, so all of author 1's profile is included in the comparison process. Author 2's profile contains 680 trigrams, so only the first 500 most frequent trigrams are included in the comparison process. Setting the Profile Length at 500 means that we cut off the profile at that specified length, regardless of how many trigrams occur in a particular author's profile beyond that specified length.

- Step 4.3. For each pair of test and known author profiles, create the Simplified Profile Intersection.

Letting SP_A and SP_P be the simplified profiles of one known author and test or disputed program, respectively, then the similarity distance is given by the size of the intersection of the two profiles:

$$\left|SP_A \cap SP_P\right| \quad (2)$$

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.

- Step 4.4. Classify the test document to the author whose profile at the specified length has the highest number of common n-grams with the test document profile at the specified length. The test document is classified to the author with whom we achieved the largest amount of intersection. 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 (i.e., greatest overlap).

By shifting the n-gram length and the profile length (or cut-off, or number of n-gram types included in the SPI), we can test how accurate the method is under different conditions, as shown in the following experimental results.

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 longer the profile lengths can be selected, and, as will be seen in the following experiments, the more reliable the author profile.

4. The Data Sets and Experimental Results

We present two experiments from Frantzeskou's doctoral research. Each experiment uses a different programming language in order to test SCAP's language independence. In each experiment, some authors are represented by very few samples and very short programs. The second experiment uses data which includes no comments.

Table 2 shows the data sets used in this study. The first dataset "MacDonell C++" contains C++ programs which have previously been used in authorship analysis by MacDonell and his colleagues. The second dataset "NoCom Java" contains Java programs which were downloaded from the website freshmeat.net as open source programs. These files were stripped of any comments.

In Table 2, "Programs per author" is expressed by the minimum and maximum number of program samples per author in the data set. Program sample length is expressed by the average Lines Of Code (LOC).

Table 2. Data Sets

	MacDonell C++	**NoCom Java**
Number of Authors	6	8
Samples per Author	5-114	4-29
Total Samples	268	107
Training Set Samples	134	56
Testing Set Samples	133	51
Size of smallest sample (LOC)	19	10
Size of biggest sample (LOC)	1449	639
Mean LOC in Training Set	206.4	122.28
Mean LOC in Test Set	213	95.92
Mean LOC/sample	210	109.1
Used in Experiment	1	2

Comparison of Relative Distance and Simplified Profile Intersection on MacDonell Data

Our purpose in this experiment was to check that the SCAP works at least equally as well as the previous methodologies for source code author identification. As mentioned earlier, MacDonell et. al. (2001) reported the best result, using the case-based reasoning (that is, a memory-based learning) algorithm, for classification accuracy at 88%.

The MacDonell data set was split (as equally as possible) into the training set (134 programs) and the test set (133 programs). We ran the aforementioned perl programs to extract n-grams from two to eight consecutive byte-level characters. For each of the six authors in the MacDonell dataset, we calculated the possible profile lengths; these are shown in Table 3.

Table 3. Profile Lengths of Six Authors in MacDonell Dataset

Author	1	2	3	4	5	6
Profile Length for bigram	1949	2391	1580	2219	767	1522
Profile Length for trigram	8487	12687	5778	7815	1893	6060
Profile Length for 4-gram	20080	21224	10666	14353	2915	13543
Profile Length for 5-gram	34407	31732	15268	20533	3710	22492
Profile Length for 6-gram	48462	41733	19338	26304	4411	31757
Profile Length for 7-gram	61362	51561	22992	31697	5008	41190
Profile Length for 8-gram	72791	61050	26122	36776	5533	50471

Table 3 shows that, for example, author 1's data allows for a profile length of 1949, while author 5's data allows for a profile length of 767, when bigrams are extracted.

Next, we created profile lengths for each author, for each n-gram length, at L equalling 200, 500, 1000 and so forth as shown in Table 4. From these author profiles at different L lengths, we calculated both Relative Distance (RD) and Simplified Profile Intersection (SPI) between each known author profile and the test source code profile. Table 4 allows us to compare the accuracy of RD and SPI, when we have different profile lengths and different n-gram lengths. Table 4 presents the results, demonstrating clearly that the Relative Distance method and the SCAP method are both capable of highly reliable results, with most authorial assignments being 100% accurate.

Table 4. Classification accuracy (%) on the MacDonell C++ data set for different values of n-gram size and profile size using two similarity measures: Relative Distance and Simplified Profile Intersection.

Profile Size L	**n-gram Size**													
	2		3		4		5		6		7		8	
	RD	**SPI**	RD	**SPI**	RD	**SPI**	RD	**SPI**	RD	**SPI**	RD	**SPI**	RD	**SPI**
200	98.4	**98.4**	97.7	**97.7**	97	**97**	95.5	**95.5**	94.7	**95.5**	92.5	**92.5**	92.5	**94.7**
500	100	**100**	100	**100**	100	**100**	99.2	**100**	98.4	**98.4**	97.7	**97.7**	97.7	**97.7**
1000	51	**99.2**	100	**100**	100	**100**	100	**100**	100	**100**	100	**100**	99.2	**99.2**
1500	5.3	**98.4**	100	**100**	100	**100**	100	**100**	100	**100**	99.2	**99.2**	99.2	**100**
2000	1.5	**97.7**	98.4	**100**	100	**100**	100	**100**	100	**100**	100	**100**	100	**100**
2500	1.5	**95.5**	99.2	**100**	100	**100**	100	**100**	100	**100**	100	**100**	100	**100**
3000	1.5	**95.5**	55.6	**100**	100	**100**	100	**100**	100	**100**	100	**100**	100	**100**

Further, the results in Table 4 show that the SCAP method outperforms the RD method especially with bi-grams and profile lengths of 1000 or less. The RD and SPI results equalize with tri-grams and larger n-grams at the 1000 profile length. But, in most cases, for n<4 and L>1000 accuracy drops for the RD method.

RD performs much worse than SPI in all cases where the compared author profile is shorter than the selected L profile length. For L=1000 and n=2, L is greater than the size of the profile of Author Number 5 (the maximum L of the profile of Author No 5 for n=2 is 769). The accuracy rate declines to 51% using the RD similarity measure. This occurs because the RD similarity measure (1) is affected by the size of the author profile. When the size of an author profile is lower than L, some programs are wrongly classified to that author. In summary, we can conclude that the RD similarity measure is not as accurate for those n, L combinations where L exceeds the size of even one author profile in the dataset. In all cases, the accuracy using the SPI similarity measure is better than (or equal to) that of RD. This indicates that this new and simpler similarity measure included in SCAP approach is not affected by cases where L is greater than the smaller author profile.

But the results of this experiment also demonstrate that Keselj et. al.'s (2003) RD method is indeed a reliable method for authorship identification of source code when the dataset allows for profile lengths greater than 1000 and n-grams greater than 4.

Performance of RD and SPI on A Different Programming Language without Comments

Since the source code used in malicious cyberattacks typically do not contain comments, the second experiment reported here examines the performance of SCAP on comment-free code and on a different programming language. We used the NoCom Java dataset as described earlier.

Table 5 shows that the SPI method consistently outperforms the RD method when the n-grams are less than seven characters long and the selected profile lengths are 500 n-grams or greater. Further, the best accuracy rates for SPI occur when the profile length is set at 2000.

Table 5. Classification accuracy (%) on the NoComJava data set for different n-gram size, profile size and two similarity measures (Relative Distance or Simplified Profile Intersection)

Profile Size	n-gram Size											
	3		4		5		6		7		8	
	RD	**SPI**	RD	**SPI**	RD	**SPI**	RD	**SPI**	RD	**SPI**	RD	**SPI**
500	94	**94**	94	**94**	94	**94**	94	**94**	92	**94**	92	**92**
1500	35	**98**	47	**90**	80	**98**	96	**98**	98	**98**	98	**98**
2000	33	**92**	14	**98**	20	**100**	31	**100**	61	**100**	78	**100**

In more detail, for L=500, when the n-gram ranges from three to eight consecutive characters, RD and SPI have (almost) identical performance. When L increases to 1500, the accuracy of RD drops for shorter n-grams, i.e., low values of n (n<6). When L increases to 2000, the accuracy of RD drops for all values of n. This happens because at least one author has an author profile shorter than the predefined value of L. Just as we saw in the first experiment, RD is not able to handle effectively cases in which an author's profile is shorter than the predefined length of the profile for comparison. Note that the accuracy of SPI increases with L. This is a strong indication that the SPI similarity measure in SCAP suits the source code author identification problem well.

5. Discussion and Future Work

These experiments (as well as Frantzeskou's other experiments) demonstrate that the n-gram approach is indeed a reliable method for authorship identification in computing languages, even though the approach was originally developed for natural languages. Our version of this approach, the SCAP method, performed consistently well on different programming languages and commented/commentless code. Compared to Keselj et. al.'s (2003) Relative Distance n-gram method, the SCAP method includes a new simplified profile and a less-complicated similarity measure which better suit the characteristics of the source code authorship analysis problem. In particular the SCAP method 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) and where the programs are free of comments (two conditions usually met in source code authorship analysis problems) with no significant compromise in performance. The experimental results presented here indicate that the best classification models are acquired for n-gram size 6 or 7 and profile size 1500 or 2000.

Critics of the SCAP method, and any n-gram approach, can argue that the n-gram approach allows for a subjective, and potentially biased, selection of the n-gram size and the profile length. Critics, for example, might suggest that a biased forensic examiner could select a

particular n-gram size and profile length in order to obtain the authorship decision which is desired. Our response to this criticism is thus: the SCAP method is currently semi-automated and therefore open to subjective manipulations. The extraction of n-grams and ranking is fully automated, but the choice of n-gram size and profile length is not and therefore open to subjective manipulations. The SCAP (or any n-gram) method can only be protected from unscrupulous and dishonest examiners by continued validation research and full automation which conceals these choices from examiners. 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. When this validation work is completed, a fully-automated system which cannot be manipulated will be available for forensic use. Meanwhile, digital forensic investigators who are independent of case advocacy and whose record of integrity supports their independence should certainly consider using the SCAP method given the current state of research. In fact, we think that a digital forensic investigator applying SCAP method should use a range of n-gram lengths and program lengths (such as shown in Tables 4 and 5) and then relate his/her results to the validation results presented herein, until litigation-independent validation results allow us to decide the best combination of n-gram size and profile length for particular forensic problems.

In future work we will present additional experimental results dealing with multiple candidate authors, the role of comments, coding tasks and collaborative programming. Further, 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. Finally, another line of research is the development of a statistical likelihood which we can attach to the yes/no classification results, since courts are not only interested in the accuracy rates of methods such as SCAP, but also the likelihood of a particular classification for a particular set of programs in a particular case.

Acknowledgments

We would like to thank Dr Steve MacDonell, Dr Panayotis Adamidis and Mr Efthimios Kotsialos.for supplying the student programs we used in this paper.

About the Authors

Georgia Frantzeskou is currently pursuing a Ph.D. in Software Forensics at the Department of Information and Communication Systems Engineering, University of the Aegean, Greece. She holds a B.Sc. in Mathematics from the University of Athens and a M.Sc. in Computer Science from Aston University, Birmingham UK. During her 10 year long career in the IT industry in Greece and UK, she has been involved in a number of different roles and projects. Some of the projects she has worked on include, the London AirTraffic Control System, Office Automation Systems, Customer Care Systems etc. Her research interests are in the fields of Software Forensics, Software Metrics, and Machine Learning Techniques. gfran@aegean.gr

Dr. Efstathios Stamatatos is a lecturer of the Information and Communication Systems Engineering Department at the University of the Aegean, Greece. He received the Diploma degree in Electrical Engineering and the Ph.D. in Electrical and Computer Engineering, both from the University of Patras, Greece. He was a research associate of the Wire Communications Lab. of the University of Patras from 1995 to 2003. He also joined the Polytechnic University of Madrid as a Visiting Researcher, the Austrian Research Institute for Artificial Intelligence as a Post-doc researcher, and he was an Adjunct Professor of the Dept. of Audio and Musical Instruments Technology, TEI of Ionian Islands. stamatatos@aegean.gr

Prof. Stefanos Gritzalis is an Associate Professor, the Head of the Department of Information and Communication Systems Engineering, at the University of the Aegean, Greece, and the Director of the Laboratory of Information and Communication Systems Security. He has been involved in several national and EU funded R&D projects in the areas of Information and Communication Systems Security. His published scientific work includes several books on Information and Communication Technologies topics, and more than 130 journal and national and international conference papers. The focus of these publications is on Information and Communication Systems Security. He is a member of the ACM and the IEEE. Since 2006 he is a member of the "IEEE Communications and Information Security Technical Committee" of the IEEE Communications Society, and of the "IFIP WG 11.6 Identity Management." sgritz@aegean.gr

Dr. Carole E. Chaski is the President of ALIAS Technology LLC, a provider of forensic computational linguistics services to the legal, law enforcement, security and defense communities and the Executive Director of the Institute for Linguistic Evidence, a non-profit research organization dedicated to validating methods in forensic linguistics. She is also an adjunct professor of linguistics at the University of Delaware and previously was on the faculties at North Carolina State University and the University of South Carolina, after earning her Ph.D. in Linguistics at Brown University. Her publications focus on empirical results of validating forensic linguistic methods. She is an active member of the American Academy of Forensic Sciences (Engineering Sciences), the International Association of Forensic Linguists, Law and Society Association and the "IFIP WG 11.9 Digital Forensics."

Blake S. Howald is currently pursuing a Ph.D. in Linguistics at Georgetown University, having previously earned his JD at Detroit Mercy School of Law and B.A. in Linguistics at the University of Pittsburgh. He is a Research Associate of the Institute for Linguistic Evidence. He authored the appendix on the legal significance of source code for this article, and generally writes on legal strategies for using linguistic evidence.

REFERENCES

1. Bennett, W. R. (1976). *Scientific and engineering problem-solving with the computer*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc..

2. Chaski, C.E. (1997). "Who wrote it? Steps toward a Science of Authorship Identification." *National Institute of Justice Journal,* pp. 15-22. September. Also available through www.ncjrs.org.

3. Chaski, C.E. (2004). "The syntactic analysis method of author identification." National Institute of Justice Research Committee, American Academy of Forensic Sciences Annual Meeting. Dallas, Texas,USA.

4. Chaski, C.E. (2005). "Who's At the Keyboard? Recent results in authorship attribution." *International Journal of Digital Evidence*, Volume 4:1, Spring. Available at www.ijde.org.

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

6. Elliot, W., and. Valenza, R.(1991). "Was the Earl of Oxford The True Shakespeare?" *Notes and Queries*, 38:501-506.

7. Gray, A., Sallis, P., and MacDonell, S. (1998). "IDENTIFIED (integrated dictionary-based extraction of non-language-dependent token information for forensic identification, examination, and discrimination): A dictionary-based system for extracting source code metrics for software forensics." In *Proceedings of SE:E&P'98* (Software Engineering: Education and Practice Conference), IEEE Computer Society Press, pages 252–259.

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

9. Frantzeskou, G., Gritzalis, S., MacDonell, S. (2004). "Source Code Authorship Analysis for supporting the cybercrime investigation process." In *Proc. 1st International Conference on e-business and Telecommunications Networks (ICETE04),* Vol 2, pages (85-92), 2004.

10. Keselj, V., Peng, F., Cercone, N., Thomas, C. (2003). "N-gram based author profiles for authorship attribution." In *Proc. Pacific Association for Computational Linguistics*.

11. Keselj, V. (2003). Perl package Text::N-grams http://www.cs.dal.ca/~vlado/srcperl/N-grams or http://search.cpan.org/author/VLADO/Text-N-grams-0.03/N-grams.pm.

12. Kilgour, R. I., Gray, A.R., Sallis, P. J., and MacDonell, S. G. (1997). "A Fuzzy Logic Approach to Computer Software Source Code Authorship Analysis." Accepted for The Fourth International Conference on Neural Information Processing -- The Annual Conference of the Asian Pacific Neural Network Assembly (ICONIP'97). Dunedin. New Zealand.

13. Krsul, I., and Spafford, E. H. (1995). "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.

14. Krsul, I., and Spafford, E. H. (1996). "Authorship analysis: Identifying the author of a program." Technical Report TR-96-052.

15. Longstaff, T. A., and Schultz, E. E. (1993). "Beyond Preliminary Analysis of the WANK and OILZ Worms: A Case Study of Malicious Code." *Computers and Security*, 12:61-77.

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

17. O'Brian, C. and Vogel, C. (2003). "A Forensic Examination of "A Funerall Elegy." Technical Report.

18. Oman, P., and Cook, C., (1989). "Programming style authorship analysis". In *Seventeenth Annual ACM Science Conference Proceedings*, pages 320–326. ACM.

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

20. Sallis P., Aakjaer, A., and MacDonell, S. (1996). "Software Forensics: Old Methods for a New Science." *Proceedings of SE:E&P'96 (Software Engineering: Education and Practice).* Dunedin, New Zealand, IEEE Computer Society Press, 367-371.

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

22. Spafford, E. H., and Weeber, S. A. (1993). "Software forensics: tracking code to its authors." *Computers and Security*, 12:585-595.

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

Appendix 1: Copyright and Source Code

Approaching these authorship problems through source code analysis is forensically feasible (as pointed out by Spafford and Weeber [18]), but also legally justifiable because, at least in the United States, source code is copyrightable. Copyright law in the United States, as it currently exists, was enacted by the United States Congress in 1976. Act of October 19, 1976 Pub. L. No. 94-553 90 Stat. 2541, codified at Title 17 United States Code §§ 101 et. seq. A creative work is copyrightable if it falls within the purview of 17 U.S.C. § 102(a) which states that, "[c]opyright protection subsists, in accordance with this title, in original works of authorship fixed in any tangible medium of expression, now known or later developed, from which they can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device." Computer programs are not explicitly enumerated in the list of examples that follow this definition. 17 U.S.C. 102(a)(1-8) Congress contemplated the authorship of computer programs as falling under the guise of "literary works" before adopting 17 U.S.C. §§ 101 et. seq. "The term 'literary works' does not connote any criterion of literary merit or qualitative value: it includes catalogs, directories, and similar factual, reference, or instructional works and compilations of data. It also includes computer data bases, and computer programs to the extent that they incorporate authorship in the programmer's expression of original ideas, as distinguished from the ideas themselves." H.R. Rep. No. 1476, 94th Cong. 2d Sess. 54. In a subsequent 1980 Amendment to 17 U.S.C § 101, "computer program" became defined as a "set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result." However, despite the accounting of computer programs by Congress in 17 U.S.C. § 101 and later amendments, there was nothing specific mentioned in regard to source codes or other component building blocks of computer programs.

Copyright law is a matter of federal law and jurisdiction in the United States. 18 U.S.C. § 1338 As such, each federal circuit is free to establish precedent in interpreting copyright law so long as the interpretation is in accord with the language of the statute and the United States Constitution. The most cited example of the extension of copyright protection to source codes comes from the Third Circuit. Building off of its decision in Williams Electronics, Inc. v. Artic International, Inc., 685 F.2d 870 (3d Cir. 1982), which first established that the copyrightability of computer programs existed after the 1980 amendment to 17 U.S.C. § 101 in the Third Circuit, the court turned to evaluating the copyrightability of source codes in the seminal case of Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240 (3d Cir. 1983), cert. dismissed 464 U.S. 1033 (1984). Franklin Computer Corporation ("Franklin") manufactured the ACE 100 computer which was designed to be compatible with the programs of the Apple II. Franklin copied Apple Computer Incorporated's ("Apple") operating system to achieve this compatibility. Franklin did not contest that it copied the operating system; rather, it asserted that operating systems are not copyrightable. In rejecting this argument, the court maintained that "[c]omputer programs can be categorized by function into either application programs or operating systems programs." Id. at 1243. Further, these programs originate from "three levels of computer language:"

> [1] High level language, such as the commonly used BASIC or FORTRAN, uses English words and symbols, and is relatively easy to learn and understand (e.g., "GO TO 40" tells the computer to skip intervening steps and go to the step at line 40).

[2] A somewhat lower level language is assembly language, which consists of alphanumeric labels (e.g., "ADC" means "add with carry"). . .

[3] [the] lowest level computer language, is machine language, a binary language using two symbols, 0 and 1, to indicate an open or closed switch (e.g., "01101001" means, to the Apple, add two numbers and save the result). Id. "The statements in high level language, and apparently also statements in assembly language, are referred to as written in 'source code'. Statements in machine language are referred to as written in 'object code'." Id.

The determination that source codes, in addition to computer programs generally, are copyrightable, was essential for Apple's success. Franklin argued that operating system programs were distinguishable from application programs because the operating system constituted a "process, method, or system of operation," which is considered to be too broad of a subject matter to be copyrightable. Baker v. Selden, 101 U.S. 99 (1879). Additionally, Franklin relied on Mazer v. Stein, 347 U.S. 201 (1954), which draws a line between the copyright of expression and ideas. Copyright protection "is given only to the expression of the idea--not the idea itself." Id. at 217. This dichotomy is now expressed in 17 U.S.C. § 102(b). Franklin wanted to argue that the operating system of a computer is not copyrightable as it is necessary function that can only be expressed in a particular way. The Third Circuit answered the threshold question of "[i]f other programs can be written or created which perform the same function as an Apple's operating system program, then that program is an expression of the idea and hence copyrightable," in the affirmative. Apple, 714 F.2d at 1253. Consequently, despite the different function of operating systems and application programs, the computer programs are still based on source and object codes which properly fall within the contemplation of "literary works."[1]

Once a subject matter is deemed copyrightable, the author of the subject matter retains exclusive rights to that subject matter. 17 U.S.C. § 106. Copyright infringement occurs when "[a]nyone who violates any of the exclusive rights of the copyright owner as provided by sections 106 through 122…[is] an infringer of the copyright or right of the author, as the case may be." 17 U.S.C. § 501(a). If "anyone" is found to be an infringer, monetary damages or injunctions from using the copyrighted material may follow. Cases, such as the one discussed, as well as cases in each of the other federal circuits, which establish the copyrightability of the source and object codes of a computer program through an interpretation of 17 U.S.C. § 102(a), are referred to as "first generation" cases. Each federal circuit to date has held that a computer programs and source codes are "literary works" for purposes of copyright.[2]

However, what remains divergent among the federal circuits are the number of tangential issues currently being litigated which constitute "second generation" cases. Although too extensive for purposes of the present discussion, it is worth mentioning the "second generation" cases. "Second generation" cases focus on: 1) which elements of computer

[1] In addition to the subject matter being an "original work of authorship" for purposes of determining copyrightability, the subject matter must be "fixed in [a] tangible medium of expression." 17 U.S.C. § 102(a). In *Apple*, the court firmly established that "a computer program in object code embedded in a ROM chip is an appropriate subject of copyright." *Apple*, 714 F.2d at 1249.

[2] It is also largely accepted on the international level that computer programs are copyrightable, *see generally*, Article 4 of the World Intellectual Property Organization (WIPO) "[c]omputer programs are protected as literary works within the meaning of Article 2 of the Berne Convention. (1971) Such protection applies to computer programs, whatever may be the mode or form of their expression."

programs are copyrightable; 2) what should the scope of protection be in infringement actions; 3) is there copyright protection of computer screen display formats (see generally, Computer Associates International, Inc. v. Altai Inc., 982 F.2d 698 (2d Cir. 1992) and Lotus Development Corp. v. Borland International, Inc., 49 F.3d 807 (1st Cir. 1995)); 4) reverse engineering (see generally, Sega Enterprises Ltd. v. Accolade, Inc., 977 F.2d 1510 (9th Cir. 1993); Atari Games Inc. v. Nintendo of America, Inc., 975 F.2d 832 (Fed. Cir. 1992); and Sony Computer Entertainment Inc. v. Connectix Corp., 203 F.3d 596 (9th Cir. 2000); and 5) microcodes (see generally, Syntek Semiconductor Co. v. Microship Technology, Inc., 307 F.3d 775 (9th Cir. 2002). These cases will take time to resolve.[3] For the time being, it is certainly clear that source codes are copyrightable, source codes are subject to infringement actions for improper use, and that identification of source code authorship can and should be presented as reliable and admissible digital evidence.

[3] It should be noted that copyright is but one protection that could potentially be extended to source codes. However, alternative protections of patent, trade secret, and licensing agreements, tend to focus more on the finished product rather than the determination of source code.