ARTICLE:

LINE BASED HASH ANALYSIS OF SOURCE CODE INFRINGEMENT

By **Svein Yngvar Willassen, PhD**

Introduction

Many digital investigations deal with the theft of intellectual property. A particularly common form of intellectual property theft is when software specifications in the form of source code are re-used without permission. Unauthorized re-use can occur with a complete program, but in many cases only parts of the original program will have been copied. Even re-use of small parts of an original can constitute an infringement.

For the digital evidence specialist, investigating source code infringement is a challenging task. In many cases, the source code from both parties will be available for inspection and the two sets of source code can be compared by searching for similarities. Finding similarities between two sets of source code is, however, difficult when considering that each set may contain hundreds of thousands of lines of code, and even the smallest re-use may constitute an infringement. Moreover, the programmer may have changed much of the code when copying it, with the result that finding similarities becomes even more difficult.

Previously proposed methods for software and source code analysis involve invoking methods from linguistic forensics and other types of manual analysis.[1] While the current method for comparing source code is largely based on manual analysis, there is a clear need to make the process of source code comparison more efficient and reliable by introducing automated methods. Such methods would allow the investigator to find matching areas in the two sets of source code, or areas having an increased likelihood of constituting an infringement. This article explores how hash analysis can be used as an automated method in the analysis of source code infringement.

Background

Software is written as source code. The source code is written by the programmer, by entering instructions in an editor. The sequence of instructions defines the function of the program, such as taking input from the user, performing calculations, showing output on the screen and so on. This source code is then usually compiled into an executable program (an executable file causes a computer to perform tasks in accordance with the instructions), which is distributed to the users of the program. The source code cannot be derived completely from the executable program.

In most software development processes, the programmer will make many changes to the source code during development. In this way, the programmer gains experience with what works and what does not work, and the quality of the program is improved. As the software matures, it will reflect the experience and know-how acquired during the development of the program. Some of these experiences will be reproducible from the programmers memory, but since all of them are embodied in the source code, in most cases it is much easier for the programmer to re-use the source code itself than remaking the code from the beginning. It is therefore desirable for any programmer when making a new program, to re-use code from other projects, either projects developed by himself or by others. When this is done with the consent of the copyright holder of the original source code, such code re-use promotes efficiency and reliability and is highly desirable. When done without consent however, the re-use constitutes an infringement, establishing liability on the part of the programmer and his employer, if employed.

The most common reason for a programmer to want to re-use code from a program without consent is a

[1] *Andrew Gray, Philip Sallis and Stephen MacDonell, 'Software Forensics: Extending Authorship Analysis Techniques to Computer Programs', Proceedings of the 3rd Biannual Conference of the International Association of Forensic Linguists, Durham NC, USA, International Association of Forensic Linguists, (1997) 1-8, available at http://isis.poly.edu/kulesh/forensics/docs/gray97software.pdf, and Eugene H. Spafford and Stephen A. Weeber, 'Software Forensics: Can We Track Code to its Authors?', Technical Report CSD-TR 92-010, Purdue University, Department of Computer Sciences, 1992.*

desire to reduce the amount of work associated with software development. By using code developed by others, the programmer reduces the work required with development, debugging and testing. To do this, the programmer needs the source code of the program he wishes to copy from. This source code can be obtained in a number of ways. Perhaps the most common type of source code infringement is where an employee copies source code when he leaves a company and re-uses it in his new job or business. Another example is where the source code has been published, but the right to republish in other forms has not been granted. A typical example is the case of the re-use of open source code without adhering to the conditions set out to re-use the code. A common requirement for re-using source code from open source projects, is for all the new source code in the project to be published as well. Any re-use without adhering to this requirement is an infringement. Further, some investigations involve industrial espionage. It may, for example, be the case that source code is stolen by computer intrusion, and the code has been re-used to simplify the development of a competing product.

In cases involving the theft of source code, the investigator often needs to compare two different sets of source code, one belonging to the originator, and one belonging to the alleged infringer. Comparing two sets of source code can be an exceedingly difficult task, since each of the sets may contain hundreds of thousands of lines of code, and the infringing part may consist of only a few hundred lines. The code may also have been changed extensively from the original, so that it can be difficult to notice the infringing portions of code.

Hash analysis

A digital hash is a checksum computed from an input text using a cryptographic hashing algorithm. The hash has a fixed length regardless of the length of the input text, and is computed in such a way that even the slightest modification of the input text will result in a completely different hash. Thus for all practical purposes, if the hash of two different input texts are equal, then the input texts are equal as well.

The current use of hash analysis in digital forensics is for the most part based on computing the hash for each file in a set of files.[2] Since it is much easier for a computer to sort and compare a list of hashes than a list of files, this makes it simpler to find files with equal content among a large number of files. The hashes can then be used to identify files that are illegal to possess, such as images of child exploitation. The investigators can quickly identify images that are already known to be images of child exploitation, without having to look through each and every image. Another common use of hashes in digital forensics is to exclude known files for analysis. For example all files belonging to the Windows operating system could be excluded, thereby reducing the workload when looking for the suspect's activities.

A good hashing algorithm has an even distribution function, which means that given a random input, no resulting check sum is more likely than any other. Further, a good hashing algorithm is required to avoid collisions. Since the output of a hash function has a fixed length, and the input length is arbitrary, it is theoretically impossible to make a hash function that never produces the same output with two different input texts. But with a good hashing algorithm, it is impossible to engineer an input text that will produce the same hash as a different input text. If this objective is fulfilled, it is impossible to create a meaningful input text with the same hash as another meaningful input text.

The most common currently used hashing algorithms are MD5 and SHA1. Both algorithms have been shown to have weaknesses.[3] These weaknesses do not, however, imply that different meaningful texts can be created with the same hash. Therefore, in the context of using hashes for identification in digital forensics, these weaknesses have a limited effect.[4]

Line based hash analysis

In the comparison of two different sets of source code, it would be useful to utilize hash analysis for the identification of parts of the alleged infringing program that could be identical to parts of the original program. Such an analysis would allow the initial analysis for such comparisons to be performed automatically, something that would greatly simplify the further comparison of the two different sets of source code.

When analyzing source code, hashes can be used to identify equal portions of code. The first question to be

[2] *Eoghan Casey, Digital Evidence and Computer Crime, (Second edition, Academic Press, 2004).*

[3] *X. Wang, F. Guo, A. Lai and H. Yu, 'Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD', Rump Session of Crypto'04 and IACR Eprint Archive, August 2004 and S. Manuel, 'Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1', International Workshop on Coding and Cryptography, 10-15 May 2009, Ullensvang, Norway.*

[4] *Eric Thompson, 'MD5 collisions and the impact on computer forensics', Digital Investigation, 2:1, 2005, 36-40.*

determined when using hash analysis on source code is how to divide the source code into portions to be hashed. A clear possibility is to hash files. Any findings based on this methodology would then require that a complete file is equal in the two source code sets. Even the slightest change in the file after it has been copied to the infringing program would result in a different hash for the file, and the similarity between the files would go unnoticed. A different approach would be to do a hash analysis on line-level or even on word level in the source code file. A problem with a word-level approach is that since it would report every equal word in the two source code sets, it would produce a large number of false hits – hits where the equality of two words did not necessarily imply that the code was stolen. This is especially true for source code, where the syntax is dictated by the programming language involved.

In this work, it is proposed to use line based hash analysis for such comparison. In this approach, a hash for each and every line of source code in both source code sets is computed. The hashes can then be sorted, and hashes occurring in both sets can be isolated. The hashes occurring in both sets represent 'hits', meaning that particular lines of code can be found in both sets of source code, both the original code set and the alleged infringing code set. With a line based hash analysis approach, it is to be expected that some of the hits will represent false positives – lines of code that does not imply that copying has occurred. Typical examples would be empty source code lines, or lines containing keywords required by the programming language syntax or convention to appear on their own line. On the other hand, a line based approach provides an opportunity to discover more complex lines or source code that if found to be equal in the two sets of source code, would indicate that the two sets have a common source.

Line hash initial experiment

To perform an initial assessment of line hashing as a tool for source code comparison, an experiment was performed with two sets of source code known to have different origins and without any copied code from one set of source code to the other, but programmed in the same programming language, Java. Set one comprised of 215 files of Java code with a total of 60000 code lines. Set two comprised of 45 files of Java code with a total of 10000 code lines. The two code sets can be characterized as fairly typical Java development projects, where set one is significantly larger than set two. The source code of both projects is formatted according to common conventions among Java developers.

The amount of work required to do a manual inspection of the two code sets would be vast. For example, if every code line in code set one was to be examined for similarity with every code line in code set two, this would require 60000 x 10000 = 600,000,000 comparisons. Such an examination would surely be unfeasible.

The MD5 hash was calculated for each code line in both sets one and two, and sorted the resulting hash sums, to provide a set of check sums for each code set where each check sum appear only once. (Thus, duplicate code lines were removed within each code set.) Duplicate code lines were then found by calculating the intersection of the set of hashes from code set one and the set of hashes from code set two. The result was 114 hashes, representing code lines that exist at least once in both code sets. This number of code lines can easily be examined manually, indicating that the huge task of comparing all lines of both sets has been reduced to a much simpler task. Examination of the 114 equal code lines revealed that they were all related to words dictated by the syntax of the Java programming language, which by convention was placed on their own line.

As expected, the examination did not produce any result indicating that source had been copied from one project to another. The experiment showed how line hashing can make comparison of all source lines in two projects feasible.

Source code comparison in a real case

Based on the results in the initial experiment, a similar method was used in a real case of alleged source code infringement. It was possible to obtain two different sets of source code used in two different systems belonging to each of the parties in the alleged infringement investigation. In this case, most of the alleged infringing source code had been developed by a consultant who had previously been employed by the other party. The allegation was that the employee had copied parts of the source during his consultant assignments.

The two sets of source code were both programmed in the programming language PL/SQL for the Oracle

database system. Set one consisted of 614 files with a total of 97717 code lines. Set two (the alleged infringing code set) consisted of 598 files with a total of 62035 code lines. As with the initial experiment, a manual comparison of these large sets of code would be infeasible.

The MD5 hash was calculated for each code line in both sets one and two, and sorted the resulting hash sums, producing a set of check sums for each code set where each check sum appear only once. Duplicate code lines were then found by calculating the intersection of the set of hashes from code set one and the set of hashes from code set two. The result was 1031 hashes, representing code lines that exist at least once in both code sets. The code lines represented by these 1031 hashes were examined manually. As in the initial experiment, some of the duplicate code lines were code lines representing language syntax and convention. But in this case, many of the duplicate lines also contained named entities such as variable names, table names and column names. On closer examination of the files in which these duplicates were found, it was determined that entire sections of code in code set two was equal to sections in code set one. Some of the source lines in these sections had been changed, by changing variable names and function names. Many of the other code lines were, however, unchanged. These sections also contained comments (which are not subject to programming language syntax) with identical wording, and in some cases even identifying the name of the programmer and date the section was programmed. With these results, there could be no doubt that these sections had a common origin in the two different sets of source code. Further, since the code contained functionality that was typical for one of the source code sets, and the named programmers had worked for one of the parties, it could be concluded that the sections in question had been copied from one of the source code sets to the other.

Evaluation

The experiments described above show that line hashing can be a valuable tool when investigating alleged source code infringement, by allowing the investigator to compare all code lines in two different sets of source code simultaneously. The investigator can use this technique to find duplicate lines and can then assess the duplicates to determine if they indicate that source code was copied.

In using this technique, it is important to understand that quantitative analysis of the results has limited value. The number of duplicates does not by itself indicate if there is a source code infringement or not. As already mentioned, duplicates may arise from syntax and conventions in the programming language. Such duplicates do not indicate source code infringement, and the number of such duplicates may vary with the language used. Further, it may be the case that both projects have included source from other projects that may not constitute infringements. It is therefore necessary to manually inspect the duplicate source code lines to assess if they represent an infringement or not.

The proposed method can assist in finding duplicate source code lines. Since the method reports exact hash matches, only exact duplicates will be detected. Only a slight change in source code will result in a line not being reported as a duplicate. This will occur if, for example, the programmer has changed variable names, function names or made other small changes to source code. However, as indicated by the results set out above, if source code has been copied, there will usually be a significant number of unchanged code lines. When the investigator inspects these duplicates, he will also see the source lines between the duplicates, and can then manually determine if these are code lines from the original with only minor changes.

Conclusion

Two sets of source code can be investigated for source code infringement by calculating a hash checksum for each code line, and then comparing all hashes from both sets of source code. Line hashing is an efficient method for finding areas of possible source code infringement. In order to conclude whether source code is infringing or not, a manual analysis of the results of line hashing has to be performed.

Svein Yngvar Willassen has a PhD in Digital Investigation from the Norwegian University of Science and Technology. He previously worked for the Norwegian Police and has conducted a large number of digital investigations both in the public and private sector. He currently runs his own business focusing on digital investigation and electronic evidence.

http://www.willassen.no/
svein@willassen.no