Research Journal of Applied Sciences, Engineering and Technology 6(3): 437-441, 2013
DOI:10.19026/rjaset.6.4098
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: August 17, 2012 Accepted: September 27, 2012 Published: June 15, 2013

Research Article

An Approach of System Similarity Measurement Based on Segmented-Digital-Fingerprint

Liao Gen-Wei
Faculty of Forensic Science, East China University of Political Science and Law, Shanghai, PRC

Abstract: Analysis and identification on software infringement, which is a time-consuming and complicated work, is always done in lab. However, to check whether suspect software infringes upon other's copyright quickly is the necessity in software infringement cases. An approach of copyright checking based on digital fingerprint is provided in this study, which computes system similarity through segmenting files to be compared, searching boundaries by sliding window and finding the same digital fingerprints of data blocks with simple and complex hash. The approach fits for finding preliminary evidences on the law enforcement spot of software infringement case, thus it has the attributes of efficiency and reliability.

Keywords: Compute forensics, digital fingerprint, software infringement, system similarity

INTRODUCTION

With the development of science technology, the emergence and application of a variety of new software and programs, has greatly ameliorated people's working practices and living conditions. At the same time, some enterprises and individuals have adventured to acquire illegal interests by stealing and plagiarizing digital products with legal software copyright. They sometimes use or sale the other's software as their own only by simply altering. It is important to find and identify the software infringement as such. And the key of the judgment of software copyright infringement is to compare the suspected software to the software with copyright and then judge whether they are "substantial similarity". What's more, the standard of "substantial similarity" is not the same at different times or different counties. The procedure of software infringement identification is always complex and time-consuming, for example, in USA, the way to judge the "substantial similarity" of two software product is often "the Altai test", which contains three steps of abstraction, filtration and comparison. The judicial identification or forensic technology about that is obviously a very complex process, which can not complete at a short time. However, in law enforcement, only when the preliminary evidences have found at the scene, can the suspected software or computer be copied or seized and the subsequent analysis and identification be performed.

In the law enforcement, the preliminary evidences about software infringement are often acquired according to the running interface of software, or file name, or MD5 hash value (also called the digital fingerprint). These methods have low reliability, easily lead to misjudgment, sometimes even result in illegal detention and disadvantageous social consequences. Therefore, it is necessary to study and develop the quick and comparatively accurate detecting tools of infringing software.

In the field of computer science, there are two different ways to judge the similarity of software code. One is the attribute counting method; the other is the structure measurement method. The attribute counting method measures the program operand, the program operator and others properties which are available for gathering statistics without considering the program structure. Halstead (1977) the structure measurement method measures the system similarity according to the internal structure of the program. Ottenstein (1976) and Parker and Hamblen (1989) both methods are more suitable for assisted detection of forensic computer programs, or automatically detection for network program which are often time-consuming. This study presents a new idea different from the above detection methods. The new method breaks the software into data blocks and calculates the block's digital fingerprint; at the same time, it calculates the characteristic value for each data block from its head. When comparing two program, characteristic value of each data block of them are compared and the block fingerprint will be calculated and compared only when two blocks have same characteristic values. Finally, we calculate the similarity according to the number of same block fingerprints.

New ideas for system similarity detection: In law enforcement, to check the software infringement at the scene is to compare the suspect software with the legal software. The software comparison is actually to check one file or a number of files related to the software. A file can be viewed as a string sequence composed of

bytes or bits (binary digits). Thus, to compare the legal software (software with software copyright, in the following discussion, we simplified the software to program file A) with suspect software (software may illegally infringe others' copyright. In the following discussion, we simplified the software to program file B), the simplest way is to directly compare the contents of program file A and program file B. If the program file size is very small, the direct content comparison method is fast, efficient, accurate and practical.

When the size of program file A and/or file B is large, the direct content comparison method is time-consuming, inefficient and not practical. We can also indirectly compare file A and file B by their digital fingerprints which are generated by using a one-way hash function (such as message digest algorithm MD5, Secure Hash Algorithm SHA-1 (Lu, 2003) to calculate the program file A and file B. if the file A's digital fingerprint and file B's digital fingerprint are identical, it can be concluded that the program files A and B are exactly the same. When the size of program file A and/or file B is large, the indirect comparison of digital fingerprint turns to be more practical than the direct content comparison method. It can quickly and accurately determine whether the two program files A and B are identical or not.

However, the indirect comparison method is only suitable for the content exactly same files. If the program file A and B are partly similar, the indirect comparison method can not be applied to them, because the fingerprints generated by one-way hash function are absolutely different when program files A and B are only slightly different.

In order to accurately detect the similarity of files A and B by the greatest extent, the above method of indirect comparison need to be improved. If we split up the file into several fixed-size data blocks and calculate each data block's digital fingerprint by one-way hash function, which formed the file's new fingerprint, we measure the similarity according to the comparison results of the file fingerprints of A's data blocks and file B's data blocks. This method improved by splitting up the file into data blocks can find the same content data block and detect the files with slightly different. In order to further improve the efficiency of the method, we can use two hash functions with different complexity to calculate hash values of the data blocks of file. The strong hash function can generate strong hash value which is more complex and time-consuming, whereas the weak hash function can generate weak hash value which is simper and faster. The further improved method calculated the strong hash value of a data block only when the two data blocks' weak hash values are calculated and identical. It can quickly find the data blocks with potentially same content by firstly using the weak hash value comparison and then it can be determined whether the data blocks are identical by strong hash value comparison. This improvement ensures not only the efficiency but also the accuracy of detection.

However, one of the defects of the above method to split up the file into fixed-size data is that the comparison of data blocks is based on fixed and invariable boundaries. Any operation to file, such as insertion, deletion, replacement, transposition, etc., may affect comparison accuracy of the data blocks after the operation point, because it is little probability that the size of data changes is exactly equal to one or multiple of the size of data block. To solve this problem, we need to split up the detected file (the suspect file, the program file B) into data blocks with flexible variable boundaries so as to reduce the effects of similarity detection caused by file content operation as low as possible or to zero.

In order to determine the reasonable boundary of each block, we can select a certain string (characteristic value) as the boundary of the data block and then split up the file into different data blocks according to the string. When we find the boundary of a data block we can calculate their weak and strong hash value or fingerprints. Then we can find those data blocks that have potentially same content according to the weak hash value comparison and can determine whether the data blocks are identical by strong hash value comparison.

The performance and accuracy of the above method which detects the similarity according to comparison of the weak and strong hash values of variable data blocks are subject to the reasonability of the characteristic value size selection. If the size of the characteristic value is reasonable, the different boundaries of the data blocks separated by it will be relatively appropriate and it will enhance the possibility to discover the similar data blocks. Otherwise, the data blocks separated by it may be too small or too large and the similarity detection efficiency will be low, the accuracy will be poor.

If the detection approach takes the advantages of the fixed-boundaries data block fingerprint and the variable-boundaries data block fingerprint in above detection methods, it will ensure not only the accuracy but also the efficiency of the detection approach.

In the approach, when we say two data blocks are identical, it must meet the following conditions simultaneously:

- The sizes of the two data blocks are identical
- The contents of the two data blocks at the same location are identical;
- The weak fingerprints (weak hash values) of the two data blocks are identical
- The strong fingerprints (strong hash values) of the two data blocks are identical

The content and size of the legal software (program file A) are relatively fixed and known in advance; the

suspected software (program file B) have been changed from the legal software by many file operation such as insertion, deletion, replacement, transposition, etc.

Therefore, program file A can be divided into a number of data blocks according to a fixed-size. Then several bytes or bits of the head of each data block are extracted as their characteristic value. When analyzing the file B, we use the characteristic value and a window (the data block size of file A) to determine the variable boundary of the program file B. then we compare the data block with file A's from weak to strong hash value (firstly, the size of data blocks must be identical, secondly, the weak fingerprints of the two data blocks calculated is equal; finally, to calculate the strong fingerprints of the two data blocks).

Combining with fixed-boundaries data block fingerprint detection method and variable boundary data block fingerprint detection method, selecting the most appropriate size of data block and size of characteristic value, making the best of the identical conditions of data blocks, the possible number of similar data blocks can be counted efficiently and accurately.

The detail description of the approach was provided as follows:

To divide the legal software into blocks according to the appropriate fixed-size: The size and content of the legal software (program file A) is fixed and known in advance, which need pre-process at first.

Firstly, the program file A (supposed the file size is Size A) is divided into N data blocks in accordance with an appropriate fixed size BLOCKSIZE: Block A [0], Block A [1],..., Block A [N-1]; the size of the last data block may be less than BLOCKSIZE.

The number of data blocks is:

$N = \lceil SizeA / BLOCKSIZE \rceil$

The set of all data blocks is:

SetBlockA = {BlockA[0],BlockA[1],...,BlockA[N-1]}

Secondly, extracting and saving the first WINDOWSIZE bytes of each block as its characteristic value to:

TriggerValuesA[0],TriggerValuesA[1],...,TriggerValuesA[N-1]

The set of all characteristic values is:

SetTriggerValuesA={TriggerValuesA[0]TriggerValuesA[1],...,TriggerValuesA[N-1]}

Thirdly, generating the weak fingerprint and strong fingerprint of each data with a simple hash function (such as Alder32 checksum) and a complex hash function (such as message digest algorithm MD5 or Secure Hash Algorithm SHA1) respectively to:

SimpleHashA[0],SimpleHashA[1],..., SimpleHashA[N-1];

ComplexHashA[0],ComplexHashA[1]...,ComplexHashA[N-1]

The set of weak fingerprints is:

SetSimpleHashA={SimpleHashA[0],..., SimpleHashA[N-1]}

The set of strong fingerprint is:

SetComplexHashA={ ComplexHashA[0],ComplexHashA[1],…,ComplexHashA[N-1]}

To divide the suspect software into blocks in accordance with characteristic value : In order to find the similar data block of the suspect software, the dynamic boundary of the data block shall be firstly determined. For this purpose, a rolling window with the size WINDOWSIZE (starting position of the window is the head of the program file B) is defined for file B. a data block boundary is found when the data content of the window (defined as Current Window Text) is equivalent to one value in the set of all characteristic values.

When the content of the rolling window is equivalent to any element in the set of characteristic values (Current Window Text $\in$ Set Trigger Values A), the location of the rolling window is recorded and saved to Positions B [j] and the content of rolling window may be recorded to Trigger Values B [j] (whether the data is saved or not is optional). Then rolling the window forward the WINDOWSIZE, continues to compare the content of the new window with elements in characteristic value set to find the boundary of the new data block.

When the content of the rolling window is not equivalent to any element in the set of characteristic values (Current Window Text $\notin$ Set Trigger Values A), rolls the window forward a byte and continues to compare the content of the new window with elements in characteristic value set to find the boundary of the new data block.

The detecting program will continue to do above procedure until the rolling window reaches the end of file B.

After the completion of data block boundary searching of file B, the program file B can also be viewed as a continuous data block (M is the number of data blocks of program file B after splitting). The boundary of the data blocks are separated by a series of characteristic values, which may be partly equivalent to the characteristic values of the legal software (program file A). The separated data block can be defined as follows:

Block B [0], Block B [1], …, Block B [M-1]

In addition, an array of the position of the characteristic vale and an array of data content between characteristic values can be defined as follows:
Positions B [0], Positions B [1], …, Positions B[M-1]

Trigger Values B [0], Trigger Values B [1],…, Trigger Values B[M-1]

Calculating the number of the same data blocks between the legal software and the suspect software : Pointer i and j (at the beginning, i = 0, j = 0), respectively, point to the location Positions B [i] and the location Positions B [j] of the suspect software (program file B). The data block between Positions B [i] (start position) and Positions B [j] (end position) is the current data block Current Block B, the current data block size |Current Block B| is equal to (Positions B [j] - Positions B [i]).

According to the different occasions that the size of the current data block |Current Block B| is greater than, equal to or less than the fixed size BLOCKSIZE, the main task of this stage is to do as follows:

A-1: If the current data block size (|CurrentBlockB|) is smaller than the fixed size BLOCKSIZE (|CurrentBlockB| < BLOCKSIZE), move the pointer at the end of the data block to the location of next characteristic value (j = j +1), then continue to compare the size of the data block;

A-2: If the current data block size (|CurrentBlockB|) is equal to the size BLOCKSIZE (|CurrentBlockB| = BLOCKSIZE), whether the current data block Current Block B is identical to any one data block of the legal program file A (Current Block B ∈ Set Block A) or not is determined according to the two different situations A-2-1 and A-2-2:

A-2-1: If the current data block is identical to any one data block of file A, the number of similarity data block are increased by one. Then move the pointer at the beginning of the data block to the end data block (i = j) and move the other pointer at the end of the data block to the location of the next characteristic value (j = i+1);

A-2-2: If the current data block is not equivalent to any one data block of file A, move the pointer from the beginning of the data block to the next location (i = i+1) and move the other pointer from the end of the data block to the next location of the former pointer (j = i+1) and then continue to compare the size of the data block;

A-3: If the current data block size |CurrentBlockB| is larger than BLOCKSIZE (|CurrentBlockB|> BLOCKSIZE), the BLOCKSIZE of CurrentBlockB extracted (Current Block B = Sub String (sizeof(BLOCKSIZE))), whether the new current data block Current Block B is identical to any one data block of the legal program file A (Current Block B ∈ Set Block A) or not is determined according to the two different situations as follows:

A-3-1: If the current data block is identical to any one data block of file A, the number of similarity data block will be added to one. Then move the pointer from the beginning of the data block to the end data block (i = j) and move the other pointer from the end of the data block to the location of the next characteristic value (j = i+1);

A-3-2: If the current data block is not equivalent to any one data block of file A, move the pointer from the beginning of the data block to the next location (i = i+1) and then continue to compare the size of the data block;

The above operation continues until the pointers reach the end of the program file B.

When determine whether the two data blocks are identical, procedure above firstly generates weak fingerprint with simple hash function and compares two data blocks' weak fingerprints, which can quickly find those different data blocks. And it continues to generate strong fingerprint with complex hash function and compares their strong fingerprints only when their weak fingerprint are the same.

System similarity calculation: The common method of code similarity measurement is the longest common subsequence method (Wang, 2007), which is relatively complex. The similarity between a legal software and suspect software can be defined as the percentage of the identical data blocks (Common Blocks).

Assuming the size of the suspect software is Size B, the fix-size of data block to split the legal software is BLOCKSIZE. Then in theory, the number of BLOCKSIZE data blocks that the suspect software can be divided into is: Theory Blocks B = [Size B/BLOCKSIZE].

Similarity between a legal software and suspect software is:

Similarity = (CommonBlocks/TheoryBlocksB) *100%

Pseudo code of the detecting approach:

```
    #define BLOCKSIZE 512KB;
    #define WINDOWSIZE 8KB;
    for (int i=0; i<BlocksA; i++) {
    TriggerValuesA[i] = the first WINDOWSIZE bytes of the i-th data block of file A;
    // the length of data block of file A is fixed t
    BLOCKSIZE,
// except to the last data block
CurrentBlockA = the i-th data block of file A;
SimpleHashA[i] = SimpleHash(CurrentBlockA);
ComplexHash[i] = ComplexHash(CurrentBlockA)
}
ActualBlocksB = 0;
for (int j=0; j<SizeB;) {
    HeadB = the WINDOWSIZE bytes beginning from the position j;
    if (HeadB ∈TriggerValuesA) {
    PositionB[ActualBlocksB]=j; // record the position j
    TriggerValuesB[ActualBlocksB]=HeadB; // record the characteristic value
    ActualBlocksB = ActualBlocksB + 1;
```

```
    j = j + WINDOWSIZE;
    } else {j = j + 1;}
    }
    CommonBlocks=0;
    for (int i=0; i<ActualBlocksB; )
    for (int j=i+1; j<ActualBlocksB; ) {
    CurrentBlockSize = (PositionB[j] - PositionB[i]);
    if (CurrentBlockSize == BLOCKSIZE) {
    CurrentBlockB = data block from Positions[i] to
    PositionsB[j] of file B;
    SimpleHashB = SimpleHash(CurrentBlockB);
    if (SimpleHashB ∈ SimpleHashA) {
    ComplexHashB = ComplexHash (CurrentBlockB);
    if (ComplexHashB ∈ ComplexHashA){
    CommonBlocks= CommonBlocks+1;
    i=j;
    j++;
    }
    }
    i ++; j++;
    }else if (CurrentBlockSize > BLOCKSIZE) {
    CurrentBlockB = data block from Positions[i] to
    PositionsB[i+BLOCKSIZE] of file B;
    SimpleHashB = SimpleHash(CurrentBlockB);
    if (SimpleHashB ∈ SimpleHashA) {
    ComplexHashB = ComplexHash (CurrentBlockB);
    if (ComplexHashB ∈ ComplexHashA){
    CommonBlocks= CommonBlocks+1;
    i=j;
    j++;
    }
    }
    i++;
    } else{
    // CurrentBlockSize < BLOCKSIZE
    // continue;
    j++;
    }
    } //end of internal for
    } //end of first for
    Similarity = (CommonBlocks / ActualBlocksB) *
100%
```

CONCLUSION

The approach provided in this study, by taking the advantages of the method of digital fingerprint detecting based on fix-size segment and the variable boundary splitting, can rapidly, efficiently and accurately detect the identical data blocks and calculate the similarity when the data block size and characteristic value are reasonable. The approach makes use of identical conditions of data blocks, applies the simple and complicated fingerprints based on hash functions and improves the accuracy and efficiency of similarity detecting. Thus it is particularly suitable for law enforcement.

ACKNOWLEDGMENT

This study is supported in part by Humanity and Social Science Youth Foundation of Ministry of Education of China (No. 11YJCZH175) and Scientific Research Innovation Program of Shanghai Municipal Education Commission (No. 10YS152).

REFERENCES

Halstead, M.H., 1977. Elements of Software Science. Elsevier, North-Holland, New York, USA.

Lu, K.D., 2003. Computer Cryptography: Data Privacy and Security of the Computer Network. 3rd Edn., Tsinghua University Press, Beijing, China.

Ottenstein, K.J., 1976. An algorithmic approach to the detection and prevention of plagiarism. ACM Sigcse Bull., 8(4): 30-41.

Parker, A. and J.O. Hamblen, 1989. Computer algorithms for plagiarism detection. IEEE T. Educ., 32(2): 94-99.

Wang, H., 2007. Subsequence counting as a measure of similarity for sequences. Int. J. Pattern Recogn., 21(4): 745-758.