# A Malware Behavior Analysis Method based on Coupling Degree

GUO Gang[1], Wei Sheng-jun[2]

[1]School of Computer Science and Technology, Beijing Key Laboratory of Software Security Engineering Technology,Beijing Institute of Technology; Beijing, China

[2]Beijing Key Laboratory of Software Security Engineering Technology

Beijing Institute of Technology

Beijing, China

guo_gang@126.com, shj_w@163.com

**Keywords:** obfuscation technique; Data fusion; coupling degree

**Abstract.** Aiming at the malware obfuscation technique, a new software behavior analysis method is proposed in the paper. The instruction coupling degree is calculated through mapping and associating the code analysis and log analysis to judge whether the instructions belong to the same behavior and then obtain the instruction information and operation process of different behaviors. The experiment proves that the method can effectively avoid the interference caused by the obfuscation techniques with the characteristics of good fault tolerance and high analysis accuracy.

## 1. Introduction

With the development of the malware detection technique, malwares begin to improve their self-protection ability through different techniques, in which obfuscation technique is a common one. Thus, how to confront obfuscation technique is one of the key problems to be solved in malware detection. Obfuscation technique includes code obfuscation and behavior obfuscation[1,2,3]. Code obfuscation is mainly aimed at the static analysis with the purpose of obstructing the decompiling of software and analyzing the codes, including code change and name rewriting. The former upsets the normal format of codes on the premise of not changing the code function and replaces the normal codes with those coded in special ways to make the analysis results of the decompiling tool contain a large amount of error data or confused coding structure, or even make it fail to complete the decompiling process. The latter rewrites a large number of function names, variable names and constant names to be the insignificant characters. Then, even decompiling be completed, the results are difficult to be understood and analyzed. Behavior obfuscation is mainly aimed at dynamic analysis, including behavior change and behavior mixing. The former changes the key process of malicious behaviors during operation to make them fail to operate as normal, or makes the key process changeable in operation. The latter inserts other irrelevant behaviors in the operation process of malicious behaviors or disperses malicious behaviors to other behaviors to reduce the probability of being detected.

The research on malware behavior and obfuscation technique has achieved much progress, including analyzing the source of information flow to avoid the information loss[4], obtaining malicious behaviors by the use of control flow diagram and statistics[5], detecting malicious behaviors through monitoring the code structure and semantics[6-7], coping with the decompiling technique with the middle code and expert system[8], solving the obfuscation problem through the abstract interpretation method[9-11], etc. Those methods contain many continuous reasoning and tracing steps, so the analysis errors caused by obfuscation may be spread and expanded in the follow-up analysis process and even make the analysis results fail to reach the minimum expectation or fail to complete the analysis. To solve those problems, this paper proposes a method which obtains the software behavior data through analyzing the coupling degree. It acquires the key instruction code information and software operation information in malwares through the log and source code analysis, maps the coupling degree among the calculation instructions and then obtains the composition of the behaviors and actual operation process after analysis, which can effectively reduce the interference of obfuscation technique on the analysis.

## 2. Method Design

### 2.1 Design Idea

The use of obfuscation technique increases the difficulty of software analysis, but the following characteristics of malwares after obfuscation can be used:

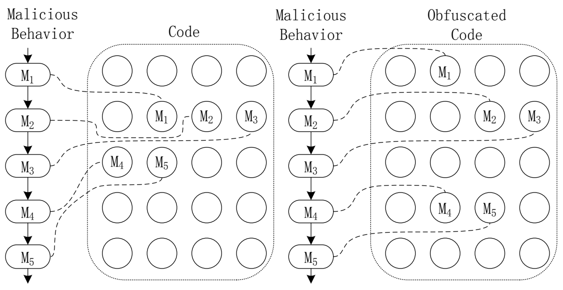

Figure 1. code obfuscation diagram

(1)Although code obfuscation increases the difficulty of static analysis, the key process of malicious behaviors will not change, as shown in Figure 1.

(2) After the behavior obfuscation, the operation process of malware is dispersed and upset. In addition, the execution sequence of the instructions is changed through circulating, waiting and jumping, leading to different processes of malicious behaviors in several times of operation. However, since malicious behaviors must be realized through executing malicious codes and the codes are fixed, no matter how malicious behaviors change in operation, the position of the corresponding malicious codes in the software code is unchanged, as shown in Figure 2.

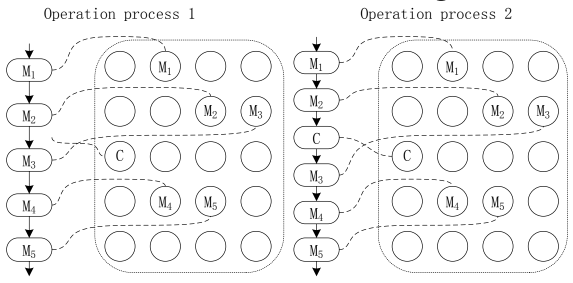

Figure 2. Behavior obfuscation diagram

(3) The instruction codes used by the behaviors with different purposes in the software are usually unrelated in the separating state while the instruction codes in the same behavior are interrelated. This phenomenon is particularly obvious between malicious behaviors and common behaviors due to the significant differences in the type of instructions and the calling frequency. In addition, even if malicious behaviors and common behaviors use the same instruction, they are present separately in the codes or called through their own instructions.

(4) Malicious codes can be spread by being bundled to the normal software and operate with them to cover the malicious behaviors. Althought the malicious codes and normal ones are bundled, they are usually separate in operation , as shown in Figure 3.

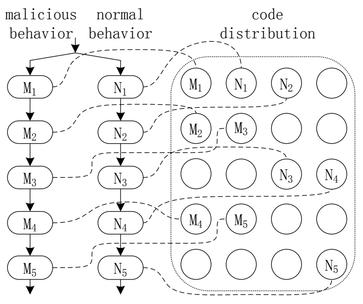

Figure 3. Bundled malware behavior diagram

According to the above analysis, the design idea of the method is as follows: to analyze independently the local data first and then make the overall analysis in order to avoid the error spreading,; to associate the results of dynamic analysis and static analysis with the help of the fixed correspondence between the instructions and the codes, due to the complementary property of the data obtained by the dynamic analysis and static; one,; to calculate the coupling degree among instructions and divide the codes on this basis to separate the codes of different behaviors, taking advantage of the separating characteristic of different behaviors in operation,; to obtain the behavior process data by connecting the behavior segments with the associated instruction information.

## 2.2 Method Design

The main steps of the method include log analysis, code analysis, data fusion, coupling degree analysis of instruction codes and behavior segment connection, as shown in Figure 4.

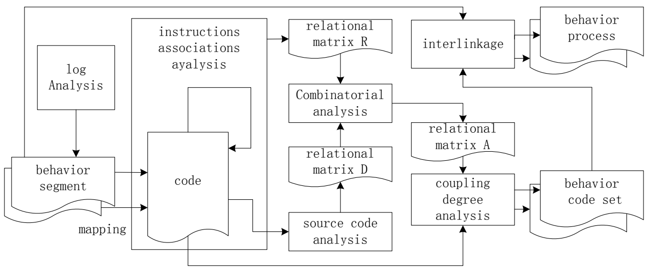

Figure 4. Architecture for obfuscated malware analysis

The steps are as follows:

(1) Log Analysis

In this step, the operation log of the software is obtained through dynamic analysis. Then, the log is analyzed to obtain the relationship among instructions in the behaviors and the relative address information of the instructions in codes. The fields to be recorded in the log include: instruction name instr, recording time time, process ID pid, user ID uid, application name appid, etc. A special field rva is added in the log to represent the relative address of the recorded instruction in codes. Instrumentation technique can be used in the dynamic analysis. When instrumenting the instructions, the document names where the instructions are located and their location information in documents are obtained and the relative address value rva is calculated. Analyzing the field information in the log can obtain the association relationship among the log records. This step just needs to analyze the local associated record information, including the instruction information executed in sequential order in the single process, end information transmitted by the data and end information of jumping instructions. There is no requirement for the association length of the analysis result. The data which can not be associated accurately are not processed elsewhere. After completion, a group of behavior segment information Bf will be obtained, in which each behavior segment is an instruction sequence Bfi which is executed in a sequential order and has no comple association with the outside. Analyzing and calculating Bfi can get the association relationship among the corresponding instruction codes and calculating the connection relationship of a group of Bf can get the relational matrix R of an instruction code.

(2) Code Analysis

In this step, the association relationship among the instructions in codes Cf is obtained through the static analysis to make up the path loss caused by the incomplete cover of the actual executing path. This step just analyzes the local association of the codes. It neither analyzes multiple paths nor requires the completeness of the analysis result. For the instructions which can be associated, the relative address value rva in the codes should be recorded. After the step, a group of local instruction association set Cf will be got with the expression form of Cfi ={Ins1，Ins2，Ins3...}，Ins={instr，rva}. Similar to calculating the relational matrix R, an relational matrix D will be got after calculation.

(3) Data Association Analysis

In this step, the data association analysis is made on the obtained relational matrix R and D to calculate the complete association relationship among the instruction codes and express it with the relational matrix A. A group of data R can be obtained from each dynamic analysis and a group of data D can be obtained from each static analysis. All the information is independent, in which the instruction name instr may be repeated, but the relative address value rva of each instruction code is unique and can be taken as the media connecting all groups of data in the data fusion. Since the association relationship of the relative address value rva of the instruction code can be re-connected, all the association information R and D can be fused to generate a complete relational matrix A.

(4) Coupling Degree Analysis on Instruction Codes

This step uses the relational matrix A to calculate the coupling degree of instruction codes, and analyze whether each instruction code belongs to the same behavior and then process the instruction codes. The corresponding instruction codes to different behaviors are separated and the corresponding instruction codes to the same behavior are gathered. In calculation, the association condition among all instructions and the possible error association in the previous calculations should be considered, so as to reduce the influence caused by the above error data. After calculation, a group of sets B can be obtained and the elements in each set Bi represents the instruction code used by the same behavior

(5) Behavior Analysis

The purpose of this step is to take advantage of the corresponding instruction code segments to all behaviors to extract all segments of the same behavior and interlink them to obtain the behavior process data in the actual operation. The instruction code information set B of each behavior has been obtained in the previous calculation. Although the behavior segments obtained in the log are separate, when an instruction belongs to the same set Bi in two behavior segments, these two behavior segments can be inferred as the same behavior. Then, these two segments can be connected according to the operation time and other information. The operation process of each behavior can be obtained after analyzing and connecting the behavior segments.

## 3.Key Algorithm Design and Realization

### 3.1 Mapping Relationship Calculation between Behavior Segments and Instruction Codes

Behavior segment refers to the instruction sequence which is executed in a sequential order and has no complicated relationship with the outside. It’s expressed as below:

$Bf=\{Ins_1，Ins_2，Ins_3...\}，Ins=\{instr，rva，time，pid，uid，appid...\}$

Bf represents the behavior segment; Ins represents the corresponding information of an instruction; instr is the instruction name; rva is the relative address value of the instruction in the source code; time is the operation time of the instruction; pid refers to the process ID; uid refers to the user ID; and appid refers to the application ID.

Several behavior segments can be obtained through analyzing the operation log:

$Bf_1=\{Ins_{1,1}，Ins_{1,2}，Ins_{1,3}... Ins_{1,p}\}$

$Bf_2=\{Ins_{2,1}，Ins_{2,1}，Ins_{2,3}... Ins_{2,q}\}$

...

$Bf_n =\{Ins_{n,1}，Ins_{n,1}，Ins_{n,3}... Ins_{n,r}\}$

The first is to map the instruction-code addresses and establish the mapping relationship *Ins-rva* between Ins and rva:

$Ins\text{-}rva_1: \{Ins_{m1,n1} , Ins_{m2,n2} , Ins_{m3,n3} ... \} \text{->} rva_1$

*Ins-rva*$_2$*: {Ins*$_{p1,q1}$ *, Ins*$_{p2,q2}$ *, Ins*$_{p3,q3}$ *... } -> rva*$_2$

。。。

*Ins-rva*$_r$*: {Ins*$_{s1,t1}$ *, Ins*$_{s2,t2}$ *, Ins*$_{s3,t3}$ *... } -> rva*$_r$

The mapping algorithm of the instruction-code addresses is as below:

```
Input  Bf
For each Bfi  in Bf
  For each $Ins_{i,j}$ in $Bf_i$
    If $Ins_{i,j}$ .rva exist in $Ins\text{-}rva_r$
      Add $Ins_{i,j}$ -> rva in $Ins\text{-}rva_r$
    Else
      Create $Ins\text{-}rva_r$
      Add $Ins_{i,j}$ -> rva in $Ins\text{-}rva_r$
Output  Ins-rva
```

After obtaining the mapping relationship between the code and the code address, the mapping relationship Bf-rva between behavior segments and code addresses is established:

$Bf\text{-}rva_1$: $Bf_1$ -> {$rva_{1,1}$， $rva_{1,2}$， $rva_{1,3}$... $rva_{1,p}$}

$Bf\text{-}rva_2$: $Bf_2$ -> {$rva_{2,1}$， $rva_{2,1}$， $rva_{2,3}$... $rva_{2,q}$}

... ...

$Bf\text{-}rva_n$: $Bf_n$ -> {$rva_{n,1}$， $rva_{n,1}$， $rva_{n,3}$... $rva_{n,r}$}

The algorithm of establishing the mapping relationship between behavior segments and code addresses is as below:

```
Input  Bf and Ins-rva
For each $Bf_i$ in Bf
  For each $Ins\text{-}rva_j$ in Ins-rva
    If $Bf_i$ .Ins == Ins- $rva_j$.Ins
      If exist $Bf\text{-}rva_i$
        Add $rva_j$ In $Bf\text{-}rva_i$
      Else
        Create $Bf\text{-}rva_i$
        Add $rva_j$ In $Bf\text{-}rva_i$
Output  Bf-rva
```

### 3.2 Instruction Code Association Calculation

Instruction code association calculation is to calculate the direct correlation degree of each instruction with others. It's expressed by the relational matrix R:

$$R = \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1n} \\ R_{21} & R_{21} & \cdots & R_{2n} \\ & & \cdots & \\ R_{m1} & R_{m1} & \cdots & R_{m1} \end{bmatrix}$$

$R_{ij}$ is used to represent the correlation between the corresponding instruction to $rva_i$ and $rva_i$; $R_{ij}$ is the number of correlation times in the operation process.

The algorithm of establishing the relational matrix R is as below:

```
Input  Bf-rva
Create matrix R
For each $R_{m,n}$
  $R_{m,n} = 0$
For each $Bf\text{-}rva_i$
  For each $rva_{i,j}$
    If $rva_{i,j}$ is related to $rva_{i,k}$
      $R_{j,k} = R_{j,k} + 1$
Output  R
```

### 3.3 Combination of Relational Matrix

Several relational matrixes R can be obtained through repeatedly analyzing the log records;

relational matrix D can be obtained through the source code analysis; Combining those matrix data can obtain the more accurate relational n matrix A. The algorithm is as below:

```
    Input  R，D，
Create matrix $A$
For each $A_{mn}$ in $A$
  $A_{mn} = 0$
For each $R'$ , $R''$ in $R$
  If $R'_{ij}.rva_i == R''_{pq}.rva_p$ && $R'_{ij}.rva_j == R''_{pq}.rva_q$
    If exist $A_{mn}$ &&
      $A_{mn}.rva_m == R'_{ij}.rva_i$ && $A_{mn}.rva_n == R'_{ij}.rva_j$
      $A_{mn} == R'_{ij} + R''_{pq}$
    else
      Add $A_{mn}$
      $A_{mn}.rva_m = R'_{ij}.rva_i$
      $A_{mn}.rva_n = R'_{ij}.rva_j$
      $A_{mn} == R'_{ij} + R''_{pq}$
For each $D_{st}$ in $D$
  If $A_{mn}.rva_m == D_{st}.rva_s$ && $A_{mn}.rva_n == D_{st}.rva_t$
      $A_{mn} == A_{mn} + D_{st} * ratio$
Output  $A$
```

**3.4 Coupling Degree Analysis of Instruction Codes**

The coupling degree of the instruction sets of different behaviors is often significantly lower than thatof the same behavior. Thus, the differences in the coupling degrees of instructions can be used to distinguish the corresponding instruction code set to different behaviors. The input is the instruction code relational matrix A and the output is a group of sets B. The element in each set Bi represents the corresponding instruction code to the same behavior. The algorithm is as below:

```
    Input  A
Create set $B$
For each $A_{ij}$ in $A$
  If $A_{ij} > thershold$
    If $A_{ij}.rva_i$ exist in $B_m$
      Add $A_{ij}.rva_j$ in $B_m$
    Else Create set $B_m$
      Add $A_{ij}.rva_j$ in $B_m$
    If $A_{ij}.rva_i$ exist in $B_n$
      Combine $B_m$ and $B_n$
Output  $B$
```

**3.5 Interlinkage of Behavior Segments**

Using the corresponding instruction code set B to the same behavior can recombine the discrete behavior segments Bf obtained from the log analysis to generate a more complete behavior process. After calculation, a group of sets C can be obtained. The element in each C represents the operation record of the same behavior. The algorithm is as below:

```
    Input  Bf , B
    Create set C
For each $Bf$
  For each $B_p$ in B
    If $B_p.rva_m$ exist in $Bf_i$
      && $B_p.rva_n$ exist in $Bf_j$
      If $Bf_i \in C_s$ && $Bf_i \in C_t$
        $C_s = C_s + C_t$
        delete $C_t$
      Else Create $C_s$ Or $C_t$
```

```
        $C_s$ += $Bf_i$ Or $C_t$ += $Bf_j$
        $C_s = C_s + C_t$
        delete $C_t$
Output  $C$
```

## 4.Experiment

### 4.1 Experimental Setup

The prototype system was realized with this method in Android platform. The environment of the experiment was 2.4G Xeon X3430，16GB Rom，Ubuntu13.04，Android version 4.0.2，Kernel version 2.6.29，ADT version 22.01. Malicious samples selected in the paper were from several secure websites (including bbs.kafan.cn, www.52pojie.cn, etc.), a total of 137. The code obfuscation technique (including the built-in obfuscation function of android) was applied to all samples, in which 32 were confirmed to apply the behavior obfuscation technique. The instrumentation tool APImonitor and automatic test tool Monkeyrunner were used in the dynamic analysis and IDA Hex-Rays ARM Decompiler was used in the static analysis.

### 4.2 Experimental Data and Analysis

10 samples were selected first to do the threshold, dynamic analysis time and frequency experiments. Dynamic analysis was carried out for 10 or 30 times, each 5 min or 15 min. 4 groups of experiments were done. The thresholds in the coupling degree analysis were 2, 5, 10 and 25, respectively. The results were shown in Figure 5.

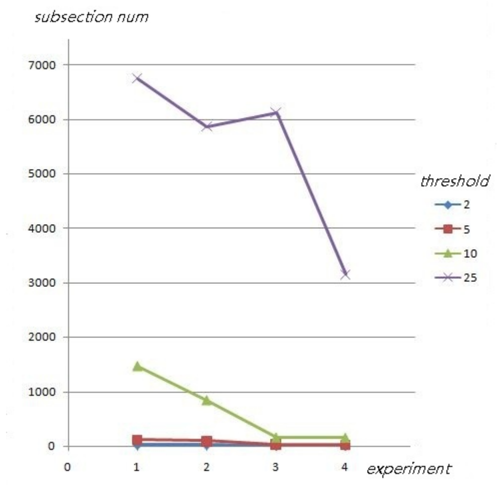

Figure 5. Experiment result

It can be found after analyzing the data in Fig.4 that the selection of thresholds directly affects the analysis results. In order to improve the accuracy and efficiency of the analysis, the single analysis time of the sample can be reduced and the number of tests can be increased. The number of tests decides the number of the matrix R and restricts the selection scope of thresholds in the analysis and calculation. Low threshold may make different behaviors mixed without accurate distinction while high threshold may divide the same behavior into several behaviors by mistake. According to the experimental results, the effect is the best when the threshold is 0.3 times more than the number of times.

The follow-up experiment selected the number of dynamic test times to be 30 and the test time to be 5 min for each and corrected the threshold to be 9. The statistical results after testing 137 samples were shown in Figure 6. The code and behavior log of each sample could be divided into several parts.

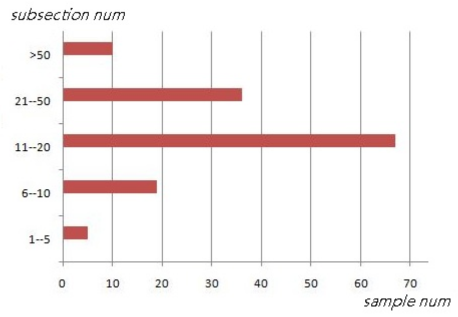

Figure 6. Experiment result statistics

An experimental sample was selected for the specific analysis. The sample was a little game software bundled with malicious codes. The malicious behaviors included downloading other softwares automatically, popping up a large number of advertisements and stealing the IMEI and contact information of the mobile phone. In the experiment, the corresponding original software was downloaded for test and contrast. After analysis, the code of the malicious sample was divided into 17 subsets while that of the original software was divided into 8 subsets. It could be found by comparing the subsets that a large number of sensitive instructions constituting the malicious behavior was concentrated in 5 subsets. After associating the behavior segments and obtaining the behavior process data, it was found that the data basically covered the key process that the malicious behavior actually executed.At the same time, it was found after analyzing the original software that the IMEI of the mobile phone would also be obtained in the operation of the software. Thus, there were 2 obtaining operations each time the malicious sample operated. In the analysis results, those two instruction codes can be accurately classified to different subsets rather than being judged by mistake due to the same instruction name. The similar situation occurred for several times. Despite for three kinds of malicious behaviors of the malicious sample and 5 resulting subsets, malicious codes have been separated from normal codes and the key process has been extracted from the log. Thus, the use of obfuscation technique will exert no influence on the follow-up analysis.

## 5. Conclusion

To solve the difficulty in analyzing the obfuscated malware, this paper proposes a malware behavior analysis method based on coupling degree which realizes the data association by the use of the fixed mapping relationship between instructions and code addresses and distinguishes different behaviors according to the separation characteristics to obtain the code information of the behaviors and the actual operation process information. The experiment proves that the method has good effect in analyzing the obfuscated malware. Next, how to improve the accuracy of judging the belongings of instructions and behaviors will be studied to get the more precise analysis results.

## References

[1] Schrittwieser Sebastian Covert computation: Hiding code in code for obfuscation purposes Source: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, p529-534, 2013

[2] Linn Cullen; Debray Saumya Obfuscation of executable code to improve resistance to static disassembly Source: Proceedings of the ACM Conference on Computer and Communications Security, p290-299, 2003

[3] Schrittwieser S.; Katzenbeisser S. Code Obfuscation against Static and Dynamic Reverse Engineering Source: Information Hiding 13th International Conference, IH 2011. p270-84, 2011

[4] Kinder J. Towards static analysis of virtualization-obfuscated binaries Source: 2012 19th Working Conference on Reverse Engineering (WCRE), p61-70, 2012

[5] Kruegel, C.; Robertson, W.; Valeur, F.; Vigna, G. Static disassembly of obfuscated binaries Source: Proceedings of the 13th USENIX Security Symposium, p255-70, 2004

[6] Udupa, Sharath K.; Debray, Saumya K.; Madou, Matias Deobfuscation reverse engineering obfuscated code Source: Proceedings - Working Conference on Reverse Engineering, WCRE, v2005, p45-56, 2005

[7] Joonsoo Jeon; Taisook Han Dynamic Analysis of Virtualization-obfuscated Binary Executables Source: Journal of KIISE: Software and Applications, v40, n1, p61-71, Jan. 2013

[8] Smith, A.J.; Mills, R.F.; Bryant, A.R.; Peterson, G.L.; Grimaila, M.R. REDIR: Automated static detection of obfuscated anti-debugging techniques Source: 2014 International Conference on Collaboration Technologies and Systems (CTS), p 173-80, 2014

[9] Visaggio, Corrado Aaron; Pagin, Giuseppe Antonio; Canfora, Gerardo An empirical study of metric-based methods to detect obfuscated code Source: International Journal of Security and its Applications, v7, n2, p59-74, 2013

[10] Madou, M.; van Put, L.; de Bosschere, K. Understanding obfuscated code Source: 14th IEEE International Conference on Program Comprehension, p4 pp., 2006

[11] Lakhotia, A.; Kumar, E.U. Abstracting stack to detect obfuscated calls in binaries Source: Fourth IEEE International Workshop on Source Code Analysis and Manipulation, p17-26, 2004