Obfuscated Binaries Reliability Analysis Based on Qualitative Method

Wei Ding\textsuperscript{1,2}

\textsuperscript{1}School of Computer Science & Technology, Beijing Institute of Technology, Beijing 100006, China
\textsuperscript{2}College of Information Science and Engineering, Henan University of Technology, Zhengzhou 450001, Henan, China
E-mail: orangeding@163.com

Zhimi Gu
School of Computer Science & Technology, Beijing Institute of Technology, Beijing 100006, China

Abstract
Type reconstruction as essential steps in reverse engineering is gradually being taken as seriously research topic. Although there are a lot of research on binary code reliability in general, the comprehensive reliability analysis and determination of obfuscated binaries are still blind spot, especially the analysis on overflow, vulnerability monitoring, and malicious code of binaries is not perfect enough. Because of the limitations of quantitative modeling, this paper applies qualitative simulation to study the effect of different factors of binary code security confusion with in-depth analysis of binary code security confusion evolution. The proposed theoretical results show that all these factors can influence the binary code security, which approves the proposed method can obviously improve the obfuscated binaries reliability analysis, and it can improve the binaries reliability too.

Key words: Obfuscated Binaries, Reliability Analysis, Qualitative Simulation, Qualitative Analysis, Binaries Code.

1. INTRODUCTION
Code obfuscation is a new technology for software protection, because the attackers need a lot of time and energy to reverse analysis program, so this method will execute transform software for more complicated forms, increasing the complexity of the program, and it make the attacker difficult to analyze and understand the execution of the program. In order to better curb the momentum of development to the software market in pirated software and copycat software, using code obfuscation techniques to more effectively to protect the software, this paper focuses on the field of software security protection confusion code based on qualitative analysis, the source code analysis includes not only the binary code analysis involved, but also including binary code security analysis and malware behavior analysis, analysis methods are used to analyze by static analysis and dynamic analysis, or both the two(Milner,1978).

Static analysis of source code: Anti Compilation Technique: usually programming languages include C, C++ and other high-level languages, and then through the compiler to generate a computer system and it can be directly executed by the file. It is very complicated and difficult to directly analyze the executable file (Chambers and Ungar, 1990). The anti-compilation technique is a compilation of these executable files into assembly language or other high-level languages. The use of anti-compilation technology to transfer the computer virus code into a high-level language, which could be more clearly to see the internal structure of the computer virus, and the use of system call and skills can greatly enhance the work efficiency of analysts. The common anti compilation tools listed as table below:

<table>
<thead>
<tr>
<th>tool</th>
<th>platform</th>
<th>sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reverse engineering compiler (REC) provided by Caprino Giampiero</td>
<td>SunOs, Linux and Windows</td>
<td>Available in Windows, Linux, BSD, SunOs and other platforms will be X86, SPARC, 68K, PowerPC and MIPS Processor to write the program back into a powerful tool for C code</td>
</tr>
<tr>
<td>Dcc from Critina Ciftuentes</td>
<td>Run on Unix, analysis of windows executable file</td>
<td>running on the x86 processor EXE Windows program, compiled into C code</td>
</tr>
<tr>
<td>JreversePro</td>
<td>Written in Java, can be run on any platform with Java virtual machine</td>
<td>The tool can anti compile Java into Java source code in accordance with code byte</td>
</tr>
<tr>
<td>HomeBrew Decompiler</td>
<td>Unix</td>
<td>The tool can also be anti compile Java byte code</td>
</tr>
</tbody>
</table>
Step by step analysis of the anti-compilation process may be very painful, which needs a lot of time. Worm or virus attacks in the network in real time usually, in this situation it generally requires the combination of analysis technology with dynamic and static.

Binary Code Static Analysis Technology: static analysis does not execute the program under the condition of using static disassemble or other static analysis tools for program analysis (Palsberg and Schwartzbach, 1991). Disassembly code is the most important work of static analysis, the most commonly used two main algorithms include: linear scan and recursive traversal. Linear scanning algorithm starts from the first executable address of the program, simply scan the entire code section and anti-assembly instructions, assuming that each instruction is followed by another instruction. Objdump's GNU and Microsoft's DumpBin are using this approach (Callahan and Jackson, 1997; Guo, Perkins and Ernst, 2006; Balakrishnan and Reps, 2014).

In addition, the dynamic analysis technique is an analysis program in the program execution process, which based on the information of the instruction, the register content and the data value. Dynamic analysis has the advantage of being able to get the runtime data, but the dynamic analysis also has the following disadvantages: Firstly, dynamic analysis requires a lot of time, and the operation is more complex, may need to be involved in human; secondly, the dynamic analysis requires the platform environment of the target program operation; thirdly, the target program may have anti debugging techniques to prevent dynamic analysis; lastly, the dynamic analysis is difficult to traverse all the running path of the program, and there is no good solution of the dynamic analysis of the path explosion process problem (Field and Tip, 2016; Reps, 2015, 2016; Lee and Brumley, 2015). Dynamic analysis techniques are used in the commercial debugger SoftICE (Lin, Zhang and Xu, 2015) and Pro IDA (Troshina and Chernov, 2015).

The advantages of static analysis is can be able to make a more complete analysis of the program, and the disadvantage is unable to determine the dynamic path of the program, or analysis of self-modifying code function; so the dynamic analysis is just to make up for the shortcomings of static analysis, but the disadvantage is that it’s not able to complete analyze the whole code, and the requirements on time and other resources are high, so researchers usually combine these two technologies. In the field of malicious software testing, Kirda et al. proposed the use of hybrid static and dynamic analysis of the system, through the analysis of the software behavior to identify spyware. There are many other important key technologies in the static analysis and dynamic analysis, such as program slicing, control flow analysis and data flow analysis (Krishnamoorthy, Debray and Figg, 2009; Dolgova and Chernov, 2008; Troshina, Derevenets and Chernov, 2010).

The combination of static analysis and dynamic analysis based on binary analysis is more complex than the simple dynamic or static analysis, the representative of this technology is BitBlaze. BitBlaze consists of three parts, TEMU, VINE and Rudder architecture such as figure 1. TEMU is a dynamic analysis module of BitBlaze, whose essence is a virtual machine with binary program execution in it, recording the execution of the instruction operand, the formation of Trace, but also do some analysis of the dynamic stain; unlike Pin and Valgrind, TEMU achieves a full system simulation with the help of the virtual machine, not only can track the user controls the instruction stream, but also and can track the system kernel, recording the instruction and operand. (Troshina, Chernov and Derevenets, 2009).

![Figure 1. TEMU framework](image)

Rewards (Sepp, Mihaila and Simon, 2011), based on dynamic analysis, is a research prototype of Purdue University. Given a binary executable file, Rewards performs a binary, monitor execution, aggregation and analysis of operational information, and eventually recover syntax and semantics of execution observed in the data structure. Rewards prototype applies type forwarding communication technologies to expand the coverage
of program data structures. In memory image forensics, Rewards helps restore the semantic information from the memory dump binaries. In binary vulnerability mining, Rewards helps to discover the "suspects" of binary vulnerability (Linn and Debray, 2003; Drape, 2007).

This paper presents a novel method, which applies qualitative simulation to study the effect of different factors of binary code security confusion with in-depth analysis of binary code security confusion evolution in order to improve the obfuscated binaries reliability analysis, and the binaries reliability.

2. BINARY CONFUSION CODE SECURITY IMPACT FACTORS

In order to better guide the generated test cases to the code location search that may trigger the vulnerability, the test case generation process needs to consider the following factors:

1. Basic block coverage, the input file to be tested in the implementation process of the binary program to cover the basic block, the more likely to trigger a binary program potential security vulnerabilities. Here I define the basic fast coverage of this quantitative impact factor as base_module_cover, any one of the binary confusion in the process of implementing the code will generate a real base_module_cover value. When the base_module_cover value was too large, that means the more program execution module, so the basic program module execution is easier to find loopholes in the binary code obfuscation. It is worth discussing, however, that when the base_module_cover value is more likely to spend more time and overhead, this will also affect the efficiency of the entire program execution.

2. Sparse execution path information, if a test case has led to the test binary program execution path, which is not previously executed, and then the path is called sparse path. In order to distinguish the tested binary program execution path among different test cases, we calculate each path by introducing a Hash function, the Hash function uses the MD5 algorithm to calculate each test case by testing the basic block binary program execution over columns. Here I define performing quantitative path information scarce factors as rare_path_exeinfo, any obfuscated code in the implementation of the process will generate a true rare_path_exeinfo value. When the rare path exeinfo value was too large, that means the execution of the program execution path is less populated, and then more program modules will omissions, so execution path is more difficult to find sparse binary obfuscated code vulnerabilities.

3. The number of the risk function execution, a test case causes the binary program to execute more dangerous function, and then the test case is more likely to trigger potential security vulnerabilities in binary program. In this paper, I define the number of the risk function implementation as num_dan_fun, any part of the binary confusion code in the process of the implementation will generate a real value of num_dan_fun. When the num_dan_fun value was too large, the more execution of dangerous program function, and the trigger potential security vulnerabilities in binary program opportunities will be greater, and it’s easier to find the dangerous function implementation of binary obfuscated code vulnerabilities.

In the process of dynamic pollution analysis of binary program, it’s need to monitor the binary program called the risk function, the specific monitoring of the risk function as shown in table 2.

<table>
<thead>
<tr>
<th>Risk function name</th>
<th>function parameter</th>
<th>Risk function name</th>
<th>function parameter</th>
<th>Risk function name</th>
<th>function parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcopy</td>
<td>3</td>
<td>Strcat</td>
<td>3</td>
<td>Streadd</td>
<td>2</td>
</tr>
<tr>
<td>memmove</td>
<td>3</td>
<td>Gets</td>
<td>1</td>
<td>Strtns</td>
<td>4</td>
</tr>
<tr>
<td>memcpy</td>
<td>3</td>
<td>Getwd</td>
<td>1</td>
<td>Realpath</td>
<td>2</td>
</tr>
<tr>
<td>Strncmp</td>
<td>3</td>
<td>Wcsncpy</td>
<td>3</td>
<td>Wcsat</td>
<td>2</td>
</tr>
<tr>
<td>memset</td>
<td>3</td>
<td>Getopt_long</td>
<td>5</td>
<td>Wcsncpy</td>
<td>2</td>
</tr>
<tr>
<td>memccpy</td>
<td>4</td>
<td>Wmemmove</td>
<td>3</td>
<td>Gettext</td>
<td>5</td>
</tr>
<tr>
<td>Stpcpy</td>
<td>2</td>
<td>Fgetc</td>
<td>1</td>
<td>Getcwd</td>
<td>2</td>
</tr>
<tr>
<td>strpcpy</td>
<td>2</td>
<td>Getc</td>
<td>1</td>
<td>Fgets</td>
<td>3</td>
</tr>
<tr>
<td>Strcat</td>
<td>2</td>
<td>Vsscanf</td>
<td>3</td>
<td>Vswscanf</td>
<td>3</td>
</tr>
<tr>
<td>Read</td>
<td>3</td>
<td>Vsnprintf</td>
<td>4</td>
<td>Vmsmem</td>
<td>3</td>
</tr>
<tr>
<td>Wmemcpy</td>
<td>3</td>
<td>Getopt</td>
<td>2</td>
<td>Vprintf</td>
<td>3</td>
</tr>
<tr>
<td>Vsnwprintf</td>
<td>4</td>
<td>Vfwscanf</td>
<td>3</td>
<td>Getchar</td>
<td>0</td>
</tr>
<tr>
<td>Vscanf</td>
<td>3</td>
<td>Vprintf</td>
<td>3</td>
<td>Strencpy</td>
<td>2</td>
</tr>
<tr>
<td>Fread</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(4) The number of dangerous operations, a test case led to more dangerous operation of binary program implementation, the test case is more likely to trigger potential security vulnerabilities in the binary program. Here I define the number of risk operation implementation as num_dan_oper, any one of the binary confusion code in the process of executing process will generate a true num_dan_oper value. When the num_dan_oper
value was too large, that means the more dangerous operation procedures would trigger potential security vulnerabilities in binary program, and the opportunities will be greater, so the more dangerous operation execution is more easily to find binary obfuscated code vulnerabilities.

The evaluation function constructed in this paper is as follows:

$$\text{EvalFun}(x) = W_1 \times \text{base\_module\_cover}(e) + W_2 \times \text{rare\_path\_exeinfo}(e) + W_3 \times \text{num\_dan\_fun}(e) + W_4 \times \text{num\_dan\_oper}(e)$$

Among them, the variable $e$ is the test case, $\text{base\_module\_cover}$, $\text{rare\_path\_exeinfo}$, $\text{num\_dan\_fun}$ and $\text{num\_dan\_oper}$ respectively for the test case $e$ corresponding to the tested basic block coverage, minimum execution path information, risk function and risk operation execution times, binary program in which $\text{base\_module\_cover}(e)$ for the generation of test cases is covered by a binary program test more basic blocks, $\text{rare\_path\_exeinfo}(e)$ for the implementation of test case generation makes the path generated by as much as possible, to achieve maximum path coverage, $\text{num\_dan\_fun}(e)$ for the test case generation risk function is to test the binary program execution times more, $\text{num\_dan\_oper}(e)$ for the generation of test cases lead to danger the operation was to test the binary program execution times more. $W_1$, $W_2$, $W_3$ and $W_4$ respectively for the above $\text{base\_module\_cover}$, $\text{rare\_path\_exeinfo}$, $\text{num\_dan\_fun}$ and $\text{num\_dan\_oper}$, and other four factors of weight, the range of its value is $[0, 1]$.

3. QUALITATIVE ANALYSIS TO OBFUSCATED BINARIES CODE

3.1 QSIM Algorithm

QSIM algorithm is a dynamic simulation algorithm proposed by Kuipers, from University of Texas, all the algorithm functions of QSIM are reasoning (Kuipers, 2006, 1994).

Definition 1: As to $[a, b] \subseteq R^*$, the necessary and sufficient conditions for $f$ are a reasoning function on $[a, b] \to R^*$ are as follows:

1) $F$ is continuous in $[a, b]$ interval;
2) $F$ is continuously differentiable on $(a, b)$ interval;
3) There are a limited number of critical points whose first derivative equals to 0 for $f$ in any bounded interval;
4) Unilateral limits $\lim_{t \to a^+} f'(t)$ and $\lim_{t \to b^-} f'(t)$ exist on $R^*$, which can be used to defined boundary values for $f'(a)$ and $f'(b)$.

Definition 2: Signpost values set, also called qualitative values as reasoning functions vary according to system simulation time value, which is a binary packet at time $t$, $\text{qval}(f, t) = [\text{qmag}(f, t), \text{qdir}(f, t)]$, and here $\text{qmag}$ called qualitative measure and $\text{qdir}$ is qualitative change direction, expressed as

$$\text{qmag}(f, t) = \begin{cases} l_1, f(t) = l_i & \text{if } f'(t) = 0 \\
(l_1, l_{i+1}), f(t) = (l_i, l_{i+1}) & \text{if } f'(t) > 0 \\
(l_1, l_{i+1}), f(t) = (l_i, l_{i+1}) & \text{if } f'(t) < 0 \end{cases}$$

Definition 3: The system could be composed of inference functions $F = f_1, f_2, \ldots, f_m$, they all have signpost value sets and distinguished time points. Distinguished time points of $F$ are distinguished time points collections of $f$, qualitative state $\text{QS}$ of system could be formed to a $m$-tuple based on single qualitative states:

$$\text{QS}(S, t_1) = [\text{QS}(f_1, t_1), \ldots, \text{QS}(f_m, t_1)]$$
$$\text{QS}(S, t_1, t_{i+1}) = [\text{QS}(f_1, t_1, t_{i+1}), \ldots, \text{QS}(f_m, t_1, t_{i+1})]$$

3.2. QSIM Algorithm Constraints of Obfuscated Binaries Reliability Analysis

In the process of obfuscated binaries reliability analysis qualitative simulation, obfuscated binaries reliability analysis structure is usually described as a set of parameter constraints, which deduce the behavior of the obfuscated binaries reliability based on constraints. Relationship among variables of obfuscated binaries reliability analysis system can be represented by constraints (algebraic constraints and qualitative constraints) in QSIM algorithm. Common algebraic constraints include plus constraints, negated constraints, differential constraints and multiply constraints. In the function, the most common and important relationship is monotonic function, $M^+$ represents single monotonically increasing relationship between functions, and $M^-$ represents single reduction relationship between functions. The relationship among influencing factors includes both $M^+$ and $M^-$ in this paper to the predicted results of the study of the obfuscated binaries reliability analysis modeling.
3.3. QSIM-based Obfuscated Binaries Reliability Analysis Modeling

Simulation modeling methods and algorithms can comprehensively analyze the influence of various parameters on obfuscated binaries reliability analysis, exploring the influence of different factors as base_module_cover, rare_path_exeinfo, num_dan_fun and num_dan_oper on obfuscated binaries reliability analysis prediction based on quantitative forecasting results, which can be represented as \( R = \text{Fun}(W1 \times \text{base}_\text{module}_\text{cover}(e) + W2 \times \text{rare}_\text{path}_\text{exeinfo}(e) + W3 \times \text{num}_\text{dan}_\text{fun}(e) + W4 \times \text{num}_\text{dan}_\text{oper}(e)) \), here \( R \) representing the final prediction result, and \( \text{Fun} = (W1 \times \text{base}_\text{module}_\text{cover}(e) + W2 \times \text{rare}_\text{path}_\text{exeinfo}(e) + W3 \times \text{num}_\text{dan}_\text{fun}(e) + W4 \times \text{num}_\text{dan}_\text{oper}(e)) \) representing the functional equation formed by different influence factors, so the data of this study mainly consider the relationship between the reliability issues occurred and fixed times and the time that program crashed.

3.4. Reasoning and Analysis of Obfuscated Binaries Reliability by Qualitative Modeling

According to the qualitative modeling knowledge described above, we assume that there is a corresponding qualitative state \( QS(S, t) \) to the obfuscated binaries reliability at each time point, and the qualitative state value is \( qval(v, t) = [qmag(v, t), qdir(v, t)] \) at the same time point. Here we mainly apply the means of qualitative spatial reasoning to analyze the obfuscated binaries reliability changing direction and the relationship with program crash numbers.

Here the main means of reasoning method is qualitative spatial, in order to better analyze the relationship between the direction of change in each state and the reliability of binaries program errors point, two qualitative state transitions are introduced in this paper: one is qualitative state transition from one significant time point to other significant time point, which is called P-transfer; the other is qualitative state transition between significant time points to other significant time point, which is called I-transfer, the detailed description are as follows:

\[
\begin{align*}
\text{P-transfer:} & \quad QS(f, t) \rightarrow QS(f, t+1) \\
\text{I-transfer:} & \quad QS(f, t, t+1) \rightarrow QS(f, t)
\end{align*}
\]

P-transfer: first assume that \( f \) is a parameter of a physical system, and qualitative state of \( f \) at time point \( t \) is \( <\text{qval}(f, t), \text{qdir}(f, t)> <\text{lj}, \text{std}> \), here \( \text{std} \) means qualitative state remains unchanged, \( P1 \) transfer indicates that \( f \) in the next step from time point \( t \) to time interval \( (t, t+1) \), \( \text{qval} \) and \( \text{qdir} \) will remain unchanged, ; in other words, qualitative state of \( f \) in time interval \( (t, t+1) \) is \( \text{qval}(f, (t, t+1)), \text{qdir}(f, (t, t+1)) <\text{lj}, \text{std}> \), in fact, \( P1 \) transfer stands for system qualitative state remained unchanged all the time. Qualitative state \( <\text{lj}, \text{std}> \) at time point \( t \) from time point \( t \) to time interval \( (t, t+1) \) transferred to \( P2 \), \text{qval} \) increases from \( \text{lj} \) to \( (\text{lj}, \text{lj}+1) \), and \text{qdir} value changes from the original stable state to increased state.

I-transfer: qualitative state of \( f \) in the time interval \( (t, t+1) \) is \( <\text{qval}(f, (t, t+1)), \text{qdir}(f, (t, t+1)) >=<\text{lj}, \text{std}> \), after \( I1 \) transfer, qualitative state of \( f \) at significant time point \( t+1 \) is \( <\text{qval}(f, (t+1)), \text{qdir}(f, (t+1)) >=<\text{lj}, \text{std}> \), namely \( \text{qdir} \) and \( \text{qval} \) keep constant, similar to \( P1 \) transfer, qualitative state also remains constant for \( I1 \) transfer. As to \( <\text{qval}(f, (t, t+1)), \text{qdir}(f, (t, t+1)) >=<\text{lj}, \text{inc}> \), here \( \text{inc} \) means increase, after \( I2 \) transfer \( <\text{qval}(f, (t+1)), \text{qdir}(f, (t+1)) >=<\text{lj}+1, \text{std}> \), namely after transformation \( \text{qval} \) of \( f \) increase from \( (\text{lj}, \text{lj}+1) \) to \( (\text{lj}+1, \text{lj}+1) \), \text{qdir} changes from the increased state to unchanged state.

The obfuscated binaries reliability qualitatively states division is applied to take advantage of continuously differentiable of reasoning function; generally speaking, obfuscated binaries reliability system time would be represented as time series by significant time points and time intervals, and each qualitative system state parameter could be reasonable transformation analyzed through \( P- \) and \( I- \) transfer, achieving characteristics qualitative analysis to obfuscated binaries reliability system.

According to \( P- \) and \( I- \) transfer, the obfuscated binaries reliability system change process according to the binaries program crash issues could be qualitatively handled combined with the definition of qualitative phase space. From the initial state, applying \( P4 \) transfer could infer that the binaries program of obfuscated binaries reliability system error occurred time points also stays at inc, after then using \( I3 \) transfer can achieve the boundary position of adjacent interval, the error occurred time interval also increases as the number of program crash increases. Repeatedly using \( P- \) and \( I- \) transfer to analyze the change trend to each point, which could obtain the stream qualitative phase space of reliability system error occurred time points with binaries program crash issues.

3.5 Qualitative Experiments Results Analysis to Obfuscated Binaries Reliability Analysis

\[
W1 \times \text{base}_\text{module}_\text{cover}(e) + W2 \times \text{rare}_\text{path}_\text{exeinfo}(e) + W3 \times \text{num}_\text{dan}_\text{fun}(e) + W4 \times \text{num}_\text{dan}_\text{oper}(e)
\]

According to the above introduction, base_module_cover and rare_path_exeinfo are a couple a contrary evaluation index, if it’s need to get a higher performance of rare_path_exeinfo and then we need to run and do more experiments, which would cause base_module_cover be greater and reduce the efficiency of obfuscated
binaries reliability analysis. And as to another index of num\_dan\_fun and num\_dan\_oper, we also get the same results, if we need to get a good performance of obfuscated binaries reliability analysis with lower cost, and then we may meet the problem is that num\_dan\_fun and num\_dan\_oper may not be good. So according to this kind of contrast situation, we need to apply qualitative analysis method to get a satisfied result with good performance of obfuscated binaries reliability analysis result and low cost.

The main content of this paper is to study the relationship between the number of obfuscated binaries reliability issues and obfuscated binaries reliability system error occurred time interval, which reflects an increase in cost and performance and they are obtained by qualitatively analysis and reasoning with the help of P-transfer and I-transfer. As to the qualitative prediction formula Fun (base\_module\_cover, rare\_path\_exeinfo, num\_dan\_fun, num\_dan\_oper), not only the number of obfuscated binaries reliability issues, but also many obfuscated binaries reliability parameters, such as code length, load operation pressure, functional inline depth and function complexity, etc. could be modeled and analyzed through the same method, and here code length, load operation pressure, functional inline depth and function complexity correspond to base\_module\_cover, rare\_path\_exeinfo, num\_dan\_fun and num\_dan\_oper in Fun(W1 * base\_module\_cover (e) + W2 * rare\_path\_exeinfo(e) + W3 * num\_dan\_fun (e)+ W4 *num\_dan\_oper(e)). Therefore, this research could be also applied to the qualitative analysis of simulation modeling on other factors that affecting obfuscated binaries reliability.

Here we suppose Pa has positive effect to obfuscated binaries reliability and Pb has negative effect to obfuscated binaries reliability, while if increases the strength of Pa would cause Pb decrease, so thus could be described as below:

\[ qval(Fun(P_a, P_b), t) = [qmag(Fun(Pa, Pb), t), qdir(Fun(Pa, Pb), t)] \]

In a word, here Pa, Pb are a couple of contrast parameters and has different effect to obfuscated binaries reliability, according to the introduction in the above section, Pa, and Pb not only have quantitative effect to Fun(Pa, Pb) but also have qualitative effect, which could be displayed as below Figure 2. From Figure 2 we can see that if we want to get a good performance of obfuscated binaries reliability then we should adjust all of these parameters’ strength, not only limited to Pa, Pb but all parameters such as Pa, Pb, Pc…Pn.

![Figure 2. relationship between Obfuscated binaries reliability and parameter strength](image)

In order to get an accurate result of obfuscated binaries reliability analysis based on quantitative and qualitative methods, we need to duplicate numerous experiments to decide the strengths of different parameters, and then we can set different values for each parameter’s strength.

4. CONCLUSIONS

Qualitative analysis method can help to understand the whole binary code reliability system change trend when any influence factor changes, in order to better study and analyze binary obfuscated code identification and reliability, more quantitative indicators should refined into a multi-level index system, combined with the qualitative simulation of various factors on different binary confusion generated code effect, at the same time it can improve the reliability and accuracy of a number of comparative tests. Through qualitative analysis it can be seen, when a plurality of factors can confuse binary code reliability analysis time of impact based on the qualitative impact with acceptable cost on an appropriate increase in the positive role of other factors, not only can improve obfuscated binary reliability analysis accuracy and reliability, but also to the depth of analysis the role of different factors.
REFERENCES


