Memory Allocation Vulnerability Analysis and Analysis Optimization for C Programs Based on Formal Methods

Deng Hui*, Liu Hui, Guo Ying, Zhang Baofeng
China Information Technology Security Evaluation Center, Beijing, 100085, China.

* Corresponding author. Email: gcdh2014@126.com
Manuscript submitted January 10, 2015; accepted April 20, 2015.
doi: 10.17706/jsw.10.9.1079-1085

Abstract: The information security problems caused by the software vulnerabilities have became more and more complex. Among these vulnerabilities, the ones existing in memory allocations appear to be difficult to diagnose due to the absence of an appropriate method. In order to solve this problem, we introduce a methodology including four novel frameworks in this paper. The formalization for a program called algebraic transition system is proposed first. It aims to transform the data exchange process and its security attribute of a program into algebraic systems which are able to be considered as objection functions and constraint conditions, respectively. Based on the systems, the behavior and structure of formalization are optimized with bisimulation to reduce the computing cost in the subsequent processes. The determination of bisimulation is implemented by numerical and symbolic computation. Finally, the specific detection of the memory allocation vulnerability in the C program can be changed into a constraints solving problem called Max function which is able to be resolved with the filled function method. The experiment results represent that our approach is feasible.

Key words: C program, memory allocation vulnerability, algebraic transition system, bisimulation, formal method.

1. Introduction

The increasing development of the computer technology brings convenience for us. Unfortunately, the instability of software always brings information security problems. A tiny vulnerability of software could lead to a great harm [1]. A memory allocation known as a design vulnerability refers to performing manual memory management in C programs for dynamic memory allocation in C programming language via a group of function in C standard library, namely malloc( ), realloc( ), calloc( ) and free( ). This vulnerability will lead to information security problems when they are exploited, just like Dos which can cause execute arbitrary commands and so on [2], [3].

Unfortunately, there doesn't exist an appropriate method to analyze memory allocation vulnerability for C programs. In order to deal with this problem, this paper proposes a novel framework to analyze this kind of vulnerabilities on the basis of the formal methods, for instance, numerical and symbolic computation [4]-[6], constraints solving [7], [8] and so on. In this framework, an algebraic transition system is applied to describe all behavior and the structure of a C program [9], [10]. Besides, bisimulation is used to optimize system to make vulnerability analysis cost less computing efforts which has already been applied to optimize behavior and structure of dynamical systems [11]-[13]. On basis of formalization, all of the data exchange processes of a program are modeled by the algebraic systems and considered as objection

functions; the formalizations of their security attributes are also called algebraic systems and considered as constraint conditions. Then, a problem called constraint solving is obtained and solved by the filled function method which is made up of an objection function and a series of constraint conditions. The results of examples in the last section verify that the memory allocation vulnerabilities of C programs are detected by the constraint solving, named Max function. In this process, if the objection function is satisfied, it means that there is no vulnerability.

The Formalization of a C Program

Each data exchange process of the source codes of a C program represents the interactive processes among variables which contain conditional statements and assignment statements. They are expressed by algebraic systems from the point view of formal method. The specific process is described by rule1.

Rule 1: The transformation between every data exchange processes in a C program and their formalizations called algebraic system M.

Step 1: For conditional statements of a data exchange process, go to **Step 1.1**.

Step 1.1: Add them directly into the algebraic system M, go to **Step 2**.

Step 2: For assignment statements of a data exchange process, go to **Step 2.1**.

Step 2.1: If there exist assignment statements that assign values to the same variable, and this variable doesn't appear on the right side of the assignment statements, then, remove these assignment statements except the last one and go to **Step 2.2**.

Step 2.2: Change the symbol of variables on the left side of the assignment statements by adding intermediate variables after **Step 2.1**; then go to **Step 2.3**.

Step 2.3: if a variable on the right side of the assignment statements is the same as the variable on the left side of the assignment statements before **Step 2.2**, then, change the symbol of the variable on the left side of the assignment statements by the symbol of this variable on the right side of statement after **Step 2.2**, and go to **Step 2.4**.

Step 2.4: Add all of the assignment statements processed by **Step 2.1**, **2.2**, **2.3** into M, end.

On the basis of the above process, all the behaviors of the C program are able to be described by algebraic systems which mean the whole structure of the program can be formalized by an algebraic transition system. The definition is established as follows:

Definition1: An algebraic transition system is showed as a tuple $<V,S,T,Q_0>$, which consists of:

- A set V of variables (possibly infinite);
- A set S of states, which is the value of V ;
- A set T of transitions: A transition is a four tuple $<s_1,s_2,AS,v_0>$; s_1 and s_2 are pre- and post- states, respectively; A transition relation AS is an algebraic system; v_0 represents the initial states of this transition;
- Q_0 is the initial global state of the transition system;

3. The Optimization of a C Program

In the existing methods, memory allocation vulnerability analysis are performed in the given programs without any optimization which could cause unnecessary computing since there always exist numbers of extra data exchange processes called nondeterminism factors. In order to solve this problem, this section applies bisimulation to optimize all of the behavior and the whole structure of algebraic transition system of the C program whose data exchange processes are modeled by polynomial systems.

Definition2: Bisimulation between C programs LS_1 and LS_2 is a binary relation $\boldsymbol{R} \subseteq S_1 \times S_2$, for $a \in AS$, and $(p,q) \in R$, there is

- If $p \xrightarrow{a} p'$, then $\exists q'$, $q \xrightarrow{a} q'$ and $(p',q') \in R$;
- If $q \xrightarrow{a} q'$, then $\exists p'$, $p \xrightarrow{a} p'$ and $(p',q') \in R$;

That is to say, two C programs are bisimular means that their behaviors are equivalent and the structures are isomorphism, namely $LS_1 =_{BE} LS_2$. For behavior equivalence detection, the numerical and symbolic computations are applied in the next two theorems. Since a non-homogeneous linear algebraic system can be transformed into a homogeneous linear algebraic system, so if its data exchange processes is able to be formalized by linear algebraic systems, the equivalence detection process can be described by theorem1 based on matrix.

Theorem1: If the reduced row echelon forms of matrix $\boldsymbol{A}$ and $\boldsymbol{B}$ are the same, then, the solutions of the homogeneous linear polynomial systems $\boldsymbol{AX} = 0$ and $\boldsymbol{BX} = 0$ are the same.

Proof: Sufficient condition: If the reduced row echelon forms of matrixes $\boldsymbol{A}$ and $\boldsymbol{B}$ are the same, then, there is an invertible matrix $\boldsymbol{P}$, and $\boldsymbol{B} = \boldsymbol{P} \times \boldsymbol{A}$. Thus, if $\boldsymbol{AX} = 0$, there is $\boldsymbol{BX} = \boldsymbol{P} \times \boldsymbol{A} \times X = 0$.

Conversely, if $\boldsymbol{BX} = 0$, then $\boldsymbol{AX} = \boldsymbol{P}^{-1} \times \boldsymbol{B} \times X = 0$, thus, the solutions of homogeneous linear polynomial systems $\boldsymbol{AX} = 0$ and $\boldsymbol{BX} = 0$ are the same.

Necessary condition: If the solutions of homogeneous linear polynomial systems $\boldsymbol{AX} = 0$ and $\boldsymbol{BX} = 0$ are the same, then, one of $\boldsymbol{A}$ and $\boldsymbol{B}$ can be linearly represented by the other. Thus, the reduced row echelon forms of matrixes $\boldsymbol{A}$ and $\boldsymbol{B}$ are the same.

If the data exchange processes are formalized by nonlinear algebraic systems, equivalence detection is described by theorem2 on the basis of Ritt-wu's method.

Theorem2: If there are irreducible characteristic sets that $M(\mathrm{x})$ corresponds to $A(x)$ and $N(x)$ corresponds to $B(x)$ are the same, then, the solutions of two nonlinear polynomial systems $A(x) = 0$ and $B(x) = 0$ are the same.

Proof: In Ritt-Wu's method, there is a solution relation between the given polynomial system and its characteristic set:

$$\boldsymbol{V}(A(x)) = \boldsymbol{V}(C(x)/I(x)); \tag{1}$$

where $C(x)$ is the characteristic set, $I(x)$ represents the set of initials of the characteristic set, and $V(x)$ is the solution of function x. In this relation, the irreducible characteristic set is unique; thus, it can be used to determine the equivalence relation for the given nonlinear algebraic systems.

4. The Formalization of the Security Attribute

Usually in the C programs, the function malloc( ) is used to apply memory, but if free( ) is not called after finishing using the memory, there will be a memory leak. It can diminish the performance of the computer by reducing the amount of available memory. Eventually, too much of the available memory may become allocated and part of the system or device stops working correctly. However, in the worst case, this flaw could be exploited by attackers. In fact, the allocation vulnerability occurs or not depends on whether security attributes are satisfied or not. Thus, before analyzing vulnerabilities, the security attributes of the program must be worked out.

Rule 2: The transformation rule between security attributes in a data exchange process of a C program and its algebraic system *N*.

Step 1: If there is malloc(x) in program, then, add an inequality $x \neq 0$ which represents a conditional

statement into the set of security attributes to determine whether the memory is released or not.

Step 2: Change the security attributes into algebraic formulas, add them into N, then end.

5. Memory Allocation Vulnerability Analysis

In the above sections, the formalizations of the data exchange process of C program and its security attribute are algebraic systems; then, if there is no memory allocation vulnerability for the given program, it means that the security attributes are satisfied. Or say differently in a mathematical level, it represents that the all the solutions of the algebraic system according to security attribute is the subset of the solutions of the algebraic system according to program. The whole process is considered as a constrained global optimization, called Max function in this section. In the process of forming this function, there are three different cases corresponding to the specific form of algebraic system.

Framework 1: Assume the algebraic systems corresponding to the data exchange process and its security attribute are M and N as follows, respectively.

$$M:\begin{cases}\cdots\\ P_i(x)=0\\ \cdots\\ P_j(x)>0\\ P_m(x)\geq 0\\ P_n(x)<0\\ P_k(x)\leq 0\end{cases};\quad N:\begin{cases}Q_a(x)=0\\ Q_b(x)\geq 0\\ \cdots\\ Q_c(x)>0\\ Q_d(x)\leq 0\\ \cdots\\ Q_e(x)<0\end{cases}; \tag{2}$$

In order to describe Max function expressing memory allocation vulnerability easier, the inequalities in a program of data exchange process are picked up, and then added into algebraic system which corresponds to the security attribute; then, formula (2) is changed into formula (3).

$$M:\begin{cases}\cdots\\ P_i(x)=0\\ \cdots\end{cases};\quad N:\begin{cases}P_j(x)>0\\ P_m(x)\geq 0\\ P_n(x)<0\\ P_k(x)\leq 0\\ Q_a(x)=0\\ Q_b(x)\geq 0\\ \cdots\\ Q_c(x)>0\\ Q_d(x)\leq 0\\ \cdots\\ Q_e(x)<0\end{cases}; \tag{3}$$

In this case, the Max function used to analysis vulnerability is (4) and (5).

$$\text{Max}\ \ (\sum_{l=1}^{s} P_l{}^2(x))=0; \tag{4}$$

$$s.t.\quad N; \tag{5}$$

Framework 2: If there exists a rule to define that how to change a semi-algebraic system into a polynomial system, then, (2) is changed into a polynomial system. In a semi-algebraic system, inequalities always have four different types: $\{>,\geq,<,\leq\}$. By adding temporary variables, inequalities are transformed into equations, respectively. Assume there are variables $x, m \in \boldsymbol{R}$:

- If $x > 0$, then, there is $(x-m^2)/x = 0$;
- If $x \geq 0$, then, there is $x - m^2 = 0$;
- If $x < 0$, then, there is $(x+m^2)/x = 0$;
- If $x \leq 0$, then, there is $x + m^2 = 0$;

With this rule, the formula (2) turns into formula (6) when temporary variables are m, n, c and d.

$$M: \begin{cases} \cdots \\ P_i(x) = 0 \\ \cdots \\ W_j(x) = 0; \\ W_m(x) = 0 \\ W_n(x) = 0 \\ W_k(x) = 0 \end{cases} \quad N: \begin{cases} Q_a(x) = 0 \\ Q_b(x) \geq 0 \\ \cdots \\ Q_c(x) > 0 \\ Q_d(x) \leq 0 \\ \cdots \\ Q_e(x) < 0 \\ m = m \\ n = n \\ c = c \\ d = d \end{cases} ; \tag{6}$$

Then, the Max function used to detect memory allocation vulnerability is:

$$\text{Max} \quad (\sum_{i=1}^{s} P_i^2(x) + W_j{}^2(x) + W_m{}^2(x) + W_n{}^2(x) + W_k{}^2(x)) = 0; \tag{7}$$

$$s.t.\ N; \tag{8}$$

In the process of solving this constraints solving problem called Max function, the filled function method is priority selected because it has less unknown variables [14], [15]. With applying filled function method to solve Max functions in the above three different frameworks, there is a memory allocation vulnerability or not can be determined. The whole process is implemented by a mathematical tool called Matlab.

6. Examples

In this section, a parallel C program existed memory allocation vulnerably in figure1 is used to verify the feasibility of our approach;

The range of variables belongs to [1], [2]. With frameworks proposed in this paper, the results of memory allocation vulnerability analysis in C programs are showed in Table 1.

Table 1. The Results

Framework	Bisimulation	Vulnerability	Founded vulnerability	Precision (%)	Time (s)
1	Yes	3	3	100	320.8
1	No	3	3	100	962.4
2	Yes	3	2	66.7	973.5
2	No	3	2	66.7	1113.7

The results represent that the optimized framework based on bisimulation costs less computing efforts. Simper data exchange processes of C programs cost less computing efforts. The precision of vulnerability analysis for the C program is decreased when its data exchange process is formalized by a semi-algebraic system. The reason is that the range of constraint conditions in this case is uncertain so that vulnerability

may be able to be founded before the end of iteration in the filled function method.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <pthread.h>
4
5   void *tfunc1(void *args)
6   {
7      int msize1;
8        ...
9      args = malloc(msize1);
10     ...
11  }
12
13  void *tfunc2(void *args)
14  {
15   int msize2;
16   void *p21 = NULL;
17     ...
18   pthread_t t21;
19   args = malloc(msize2);
20   pthread_create(&t21,p21,tfunc21,NULL);
21   pthread_join(t21,NULL);
22     ...
23  }
24
25  void *tfunc21(void *args)
26  {
27   int msize21;
28     ...
29   args = malloc(msize21);
30     ...
31  }
32
33  int main(void)
34  {
35     pthread_t t1;
36     pthread_t t2;
37     void *p1 = NULL;
38     void *p2 = NULL;
39     pthread_create(&t1,p1,tfunc1,NULL);
40     pthread_create(&t2,p2,tfunc2,NULL);
41     pthread_join(t1,NULL);
42     pthread_join(t2,NULL);
43
44     return 0;
45 }
```

Fig. 1. A parallel C program.

7. Conclusion

In this paper, a novel framework for memory allocation vulnerability analysis and analysis optimization for the C programs on basis of formal methods is established. The results of the examples prove the framework is feasible. More verification work will be researched in the future, such as the efficiency of our approach compared with other analysis methods.

Acknowledgements

The research work was supported by National Natural Science Foundation of China under Grant No. 61202493.

References

[1] Shahzad, M., Shafiq, M. Z., & Liu, A. X. (2012). A large scale exploratory analysis of software vulnerability life cycles. *Proceedings of the 2012 International Conference on Software Engineering* (pp. 771-781).

[2] Godefroid, P., Levin, M. Y., & Molnar, D. (2012). SAGE: Whitebox fuzzing for security testing.*Queue*.

[3] Xu, G., Bond, M. D., Qin, F. *et al*. (2011). Leak chaser: Helping programmers narrow down causes of memory leaks. *ACM SIGPLAN Notices*.

[4] İlarslan, K., & Yildirim, M. (2011). An application of Ritt-Wu's zero decomposition algorithm to the pseudo null Bertrand type curves in Minkowski 3-space. *Journal of Systems Science and Complexity*.

[5] Xiaoshan, G., Chunming, Y., & Guilin, Z. (2009). Ritt-Wu's characteristic set method for ordinary difference polynomial systems with arbitrary ordering. *Acta Mathematica Scientia*.

[6] Hammer, R., Hocks, M., Kulisch, U. *et al*. (2012). Numerical toolbox for verified computing I: Basic numerical problems theory, algorithms, and pascal-XSC programs.

[7] Gotlieb, A. (2012). TCAS software verification using constraint programming. *The Knowledge Engineering Review*.

[8] Taly, A., Gulwani, S., & Tiwari, A. (2011). Synthesizing switching logic using constraint solving. *International Journal on Software Tools for Technology Transfer*.

[9] Sobociński, P. (2012). Relational presheaves as labelled transition systems. *Coalgebraic Methods in Computer Science*.

[10] Ehrig, H., & Mahr, B. (2011). *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer Publishing Company.

[11] Van, G. R. J., & Weijland, W. P. (1996). Branching time and abstraction in bisimulation semantics. *Journal of the ACM*.

[12] Girard, A. (2012). Controller synthesis for safety and reachability via approximate bisimulation. *Automatica*.

[13] Girard, A, & Pappas, G. J. (2011). Approximate bisimulation: A bridge between computer science and control theory. *European Journal of Control*.

[14] Zhu, W. (2006). Globally concavized filled function method for the box constrained continuous global minimization problem. *Optimization Methods and Software*.

[15] Bai, F. S., Mammadov, M., Wu, Z. Y. *et al*. (2008). A filled function method for constrained nonlinear equations. *Pac. J. Optim*.

Hui Deng received the Ph.D. degree in computer science and technology from Beijing Jiaotong University. Her research areas include information security and formal methods.

Hui Liu received her Ph.D. degree in computer science and technology from Huazhong University of Science and Technology. Her research area is information security.

Ying Guo received the B.S. degree in computer network from Beijing University of Posts and Telecommunications. Her research area is information security.

Baofeng Zhang received his M.S. degree in information and communication technology from North China Electric Power University. His research area is information security.