ATLANTIS PRESS

Advances in Computer Science Research, volume 63

International Conference on Artificial Intelligence and Engineering Applications (AIEA 2016)

Static memory leak detection based on a new memory model

Hangyuan Liu [a], Hua Zhang [b]

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China.

[a]460660596@qq.com, [b]47797730@qq.com

Abstract. Memory leaks of static analysis need to be more accurate to determine memory state of different times. The current static memory model has a high false positive rate and false negative rate, mainly because the current memory model cannot accurately represent memory state and memory hierarchy. In this paper, a domain-sensitive memory model that can represent the memory hierarchy is proposed. On this basis, a static memory leak detection method is implemented. The experimental results show that the model has high accuracy and detection efficiency.

Keywords: Memory leaks; Static detection; Memory model.

1. Introduction

In a program written in a language that supports explicit memory allocation, such as C / C ++, memory leak is a serious problem. Memory leaks vulnerability will run out of memory system, reduces the system performance and even lead to system crash [1]. Memory leak problem is difficult to use conventional method to detect, because often need to run for a period of time to have an impact, so often because the system downtime was finally discovered by operation and maintenance personnel, and code positioning is also difficult after problem discovery. Memory leaks have a significant impact on long-running, server-side applications and embedded applications [1], so studying how to make effective detection and elimination of memory leaks in the program is very important work.

The corresponding memory leak detection can be used in a dynamic and static testing. Because of dynamic testing [2-4] need for code instrumentation and dynamic link library, restricting the scope of the code test, static detection is currently the mainstream of mobile Internet security vulnerability detection [5-8]. In the static test, the memory model has the greatest influence on the accuracy of detection. At present, there are many researches on memory model.Typical job has SATURN [9], LCLint [10], etc., can find all the possible memory leak error, but there is a higher rate of false positives. Therefore, how to ensure precision under the premise of making static analysis tool can analyze large program is a hotspot and difficulty of research on the static analysis. Because of the complexity of C language syntax design, it is very difficult to design static test and memory model. The symbolic execution memory model must be able to accurately represent the memory running state of the program.

In recent years, many researches have been done on symbolic memory model, mainly focused on name binding model [15], array simulation model [11], and tree memory model [12]. Name binding model is the most basic memory model. This model treats computer memory as a key-value pair (name, value). This memory model can't simulate pointer aliases, and there is no memory address concept. In order to solve this problem, people put forward the concept of array simulation model [11]. We can think of memory as a large array, all of the memory operations can be converted to operations on the array, and the index number of the array can be seen as abstract representations of the memory address. Array model to a certain extent, solve the problem of the name binding model [13]. But his shortcomings are obvious: 1) Array model needs to ensure that each variable is a fixed size; 2) The array model has no way to represent hierarchical relationships of compound data types, such as multi-level arrays, structures, and so on. Therefore, a tree model was proposed [12].The tree memory model unifies the variable value and the pointer value into the SValue type, expressed as a form of a four-tuple <tp, val, ptr, data>, which is very convenient to support various operations of the data

Because the use of a unified SValue type to represent the memory location and memory data, and there is no clear hierarchical relationship, resulting in a lot of useless data and empty data, resulting in the symbol execution engine consumption of memory is too large, and run time is too long. In a word, the current memeory models have the problem that they can not represent memory hierarchy and complex data structure.

To solve this problem, this paper propose a new static domain-sensitive memory detection method based on the tree memory model [12]. And on the basis of the memory model, this paper implements a static detection system for detecting memory leak. At the end of this paper, through the experimental results to verify the validity and accuracy of the memory model.The rest of this paper is organized as follows. The second section details the new memory model. The third section introduce how to use the memory model to detect memory leaks security vulnerabilities. Finally, through the test to show the effect of the new memory model.

2. Domain - Sensitive Memory Model

This paper follows the concept of the memory domain model [10], but extends the concept. According to Nielson's semantic definition [13], the concept of symbols in the program is divided into three categories: semantics, storage, and mathematics. Semantics of the various types of source code in the program expression; storage refers to the variable corresponding to the storage space in memory, that is, the expression of the left value; maehematics refers to the right-hand side of the expression. For an expression, we can use a triplet to describe it: <expression, memoryobject, value>. According to this idea, two mapping relations can be abstracted. The mapping from the expression to the storage location, which represents each expression and its location: (Expr,Location). The mapping from position to value, which represents the value of each position: (Location,Value)

According to the C language standard [14], an lvalue is an expression with an object type.This paper uses the concept of memory domain to abstract C left value expression. Therefore, in a domain-sensitive memory model, each left value expression has a corresponding memory field. Also, the lvalue expression that points to the same position should point to the same memory domain object to eliminate the effect of the alias problem on symbol execution.

2.1 The hierarchical relationship of the memory domain

A domain- sensitive memory model defines multiple types of memory domains, such as variables explicitly declared, with a memory domain type of VarMD. If a variable is an array or a structure that contains subtype data. In order to represent the memory hierarchy relationship of variables of this type, this paper proposes the concept of sub-memory domain. A memory domain can be a sub-domain of another memory domain. Each sub-domain also has a pointer to its parent domain (Father MD). For example, for array types, this paper defines an IndexDomain to represent each child element, and each sub-domain has a pointer to its array memory type field. And use a mapping <domain, length> to represent the size of each block of memory length.

Three kinds of StoreMD are used to represent three kinds of storage types of C: stack type, global type and heap type. All local variables have a StackDomain as their parent memory domain; all dynamically allocated memory has a HeapDomain as its parent memory domain; all static variables have a GlobalDomain as their parent memory domain.

2.2 Type conversion

In C Language, type conversions between pointers of any type are allowed. This syntax presents a great deal of difficulty for static analysis, as shown in the following code.

```
void* p1 = (void*)malloc(100);
char* p_char = (char*)p;
p_char[0] = ‘b’;
int *p_int = (int*)p;
p_int[0] = 0;
```

This is an artificial example, but similar code is widespread in reality. Similar code will raise type unsafe access. The problem is the C language syntax, you can apply for a block of memory without type, and then converted to any type. This paper establishes a concept of TypeDomain, which is used to record memory type information on a specific data flow node of the memory domain. In the process of type conversion, equivalent to assigning a different TypeDomain type to a specific pointer for the memory domain model. A memory domain at the same time, that is, a symbol on the implementation of the state point, only one TypeRegion effective. The specific conversion process shown in Figure 1.

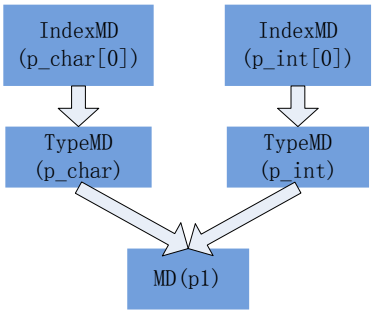

Figure 1 Symbol type conversion

Any memory object with a name, such as Type a, has two SValue objects in the memory domain model. One on behalf of its left value l_a, the other one on behalf of its right r_a. We classify SValues into two types, location types and symbolic types. The Location type represents an abstract memory location. All C language pointer type is location type, the other variables are symbolic type, Symbolic type have type pointer used to point to TypeDomain.

2.3 Memory Model and Language

Usually the memory model and the programming language is the corresponding, the more complex the language of the corresponding memory model is also more complex. The high-level language syntax structure is complex, such as the traditional C language in the interpretation of the implementation must correctly handle a [b], ab, a-> b, * a, & a and other basic expressions composed of these basic expressions Complex complex expressions such as a[b] [c].d->e[f],*G-> h and so on. Implementing such a system is very complex, and it does not make much sense to find the bug itself. Another possible approach is to translate the high-level language into an intermediate language model first and then analyze it in intermediate languages. This approach avoids the complexity of high-level languages because complex language structures have been decomposed into smaller and simpler intermediate language constructs in the process of converting high-level languages into intermediate languages. The symbolic execution data stream is an intermediate expression type.As shown in Table 1, the paper defines several symbolic rules to represent the syntax in C: LVal (expr) returns the left value of the expression expr, expressed as a Location type.RVal (expr) returns the right value of the expression expr, expressed as a Location type or symbolic type.The domain-sensitive memory model supports almost all C language expressions: pointers at any level, array data types, structure types, dynamically allocated memory types, and so on.

Table 1 Domain - sensitive memory model syntax correspondence table

	LVal	RVal
Constant n Variable x Array a	N/A VD(x) VD(x)	n SV(LVal(x)) N/A
A[e] s.d p->d	IndexMD(LVal(a),RLval(e)) FieldMD(LVal(s),d) FieldMD(RVal(p),d)	SV(LVal(a[e])) SV(LVal(s.d)) SV(LVal(p->d))
&expr *expr	N/A LVal(expr)	LVal(expr) SV(RVal(expr))
Malloc(n)	N/A	MallocDomain(n)

3. Memory leak detection

In this section, we will first design a static detection system, and then describe in detail how to write a detector using the new memory model to detect memory leak security vulnerabilities.

According to the characteristics of static detection, this paper designs the symbol execution engine system. The specific symbolic execution logic flow is shown in Figure 2.

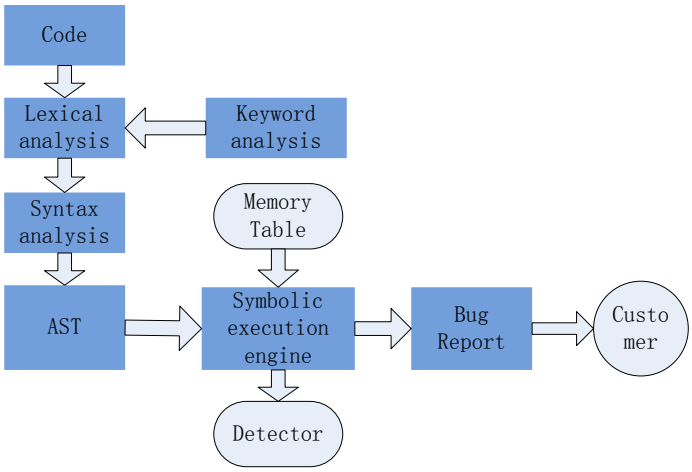

Figure 2 The flow chart of symbolic execution

This system contains several modules, such as semantic analysis, symbol execution, and report output. We can add different types of security vulnerability detectors by adding custom detectors. This paper takes memory leak as an example.

The process of detecting memory leaks can be thought of as a state machine transformation process. It contains a set of state sets, a starting state, one or more termination status, a state between the conversion table. In addition, this article also adds an error state, used to characterize the program failure. Memory leak state machine described in Figure 3.

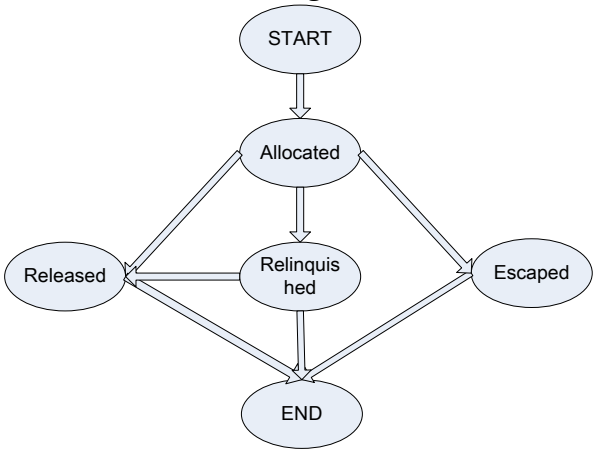

Figure 3 Memory leak state automaton

Considering all the memory calls in the program, each memory request statement in the program affects the state machine state. The system creates a variable-independent and path-related state machine instance for each allocated memory and maintains a memory field for each memory variable, and then analyzes the program according to the state machine.

4. Testing and results analysis

In this section, we will test the theory and model above. We use NoSQL database redis-3.0 and C language open source compiler ucc and open source query engine OLAP-NG as the test cases. Table 2 shows the results of using our algorithm for memory leak detection, Table 3 shows the results obtained using the TMM model test.

The table includes the number of reports in each program, the number of false positives, false positives. Experimental results show that the memory leak detection method based on the memory domain sensitive detection algorithm can control the suspected memory leaks in a small range, and the false alarm rate is small compared with the detection algorithm based on the tree model alone. Both methods give a superset of a memory leak error that contains all of the actual memory leak errors.

Table 2 Domain-Sensitive Memory Model Test Results Table

Numble	Number of Reports	Number of false positives	False positive rate
Redis-3.0	2	0	0%
UCC	5	0	0%
OLAPNG	15	3	20%

Table 3 TMM Memory Model Test Results Table

Numble	Number of Reports	Number of false positives	False positive rate
Redis-3.0	5	3	60%
UCC	8	3	35%
OLAPNG	20	8	40%

By comparison, it can be found that the domain-sensitive memory model detection method has higher detection precision and smaller false alarm rate. Because we record more program flow analysis information, such as hierarchical relationships and type conversion information, to static detectors to provide more effective analysis of data, it can also be false alarm rate control in a small range. In conclusion, the proposed domain-sensitive memory detection model compared with the traditional single detection method, for the detection of memory leak security vulnerabilities have a higher accuracy, and has a higher detection efficiency.

5. Summary

Traditional static testing memory model suffer from false positives and false negatives. We propose a domain-sensitive memory model that can represent the memory hierarchy. Results show that our method has higher accuracy and efficiency rate.

Acknowledgments

This work is supported by NSFC (Grant Nos. 61300181, 61502044), the Fundamental Research Funds for the Central Universities (Grant No. 2015RC23).

References

[1] Robert C. Seacord, Secure Coding in C and C++ (2nd Edition) (SEI Series in Software Engineering) [M] Addison-Wesley Professional; April 12, 2013

[2] Mari a Jump, Kathryn S McKinley. Cork: dynamic memory leak detection for java [R]. Technical Report T R-06-07, Department of Computer Science, The University of Texas at Austin.Austin, TX, 2006.

[3] Nicholas Nethercote, Julian Seward. Valgrind: a framework for heavy weight dynamic binary instrumentation [C]. Proceedings of PLDI 2007, San Diego, California, USA, June 2007.

[4] Gao Hai-chang, Feng Bo-qin, He Hang -jun, et al. Dynamic memory testing based on source code instrumentation on linux platform [J]. Mini-Micro Systems, 2006, 27(9): 49-53.

[5] Yu Feng, Saswat Anand, Apposcopy:semantics-based detection of Android malware through static analysis [C] FSE 2014 Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering Pages 576-587

[6] Goran Doychev, Boris Kopf, CacheAudit: A Tool for the static Analysis of Cache Side Channels [C] ACM Transactions on information and System Security(TISSEC) June 2015

[7] Andres Letelier, Jorge Perez, Web Query base Static Analysis[C]ACM Transactions on Database System(TODS) Volume 38 Issue 4, November 2013

[8] Alexandre Bartel,Jacques Klein,Static Analysis for Extracting Permission Checks of a Large Scale Framework [C], IEEE Transactions on Software Engineering Volume 40, Issue 6, June 1 2014

[9] Xie Y, Aliken A. Saturn: A scalable framework for error detection using Boolean satisfiability. ACM Trans on Programming Languages and Systems, 2011, 29(3):1-42

[10] Barker C. Static error checking of C applications ported from UNIX to WIN32 systems using LCLint [R]. Senior Thesis, Dept. Computer Science, University of Virginia, Charlottesville, 2011.

[11] Zhongxing Xu and Jian Zhang. A test data generation tool for unit testing of C programs. In Proceedings of the International Conference on Quality Software,pages 107–116, 2006.

[12] QiJun Li ,Based on the symbolic execution code research and implementation of static test method [D], 2012, Chengdu: University of Electronic Science and Technology of China

[13] Jian Zhang. Symbolic execution of program paths involving pointers and structure variables.In Proceedings of the Fourth International Conference on Quality Software, pages 87–92, 2004.

[14] WG14 ISO/IEC 9899:201x, editor. Programming Languages - C. ISO

[15] Uday Khedker, Amitabha Sanyal, Bageshri Sathe. Data Flow Analysis: Theory and Practice [M]. CRC Press; March 27, 2009