International Journal of Security and Its Applications
Vol. 9, No. 5 (2015), pp. 95-104
http://dx.doi.org/10.14257/ijsia.2015.9.5.10

Research on Trustiness of Software Behavior Based on Cross-References

Wei LIU[1], Linlin CI[2] and Liping LIU[3]

[1,3]*Computer Department of Beijing Institute of Technology*
[2]*Research on High Technology Information Institute of Beijing*
liuwei_bit@126.com, huofolw@126.com, ningxiahuofo@126.com

Abstract

Trustiness of software dynamic behavior is always the research hotpot and difficult point in the field of trusted computing. Dynamic trustiness model of software behavior relies on three basic aspects including the extraction and description of intended software behavior, real-time monitoring of software behavior and a contrastive analysis of intended software behavior and running-time behavior. It is a hard nut to extract and describe intended software behavior. In our paper, we present a method of extracting and describing of software trajectory based on cross-references of binary file. Our method is thinking from the perspective of the attacker, and constructing suitable size function chain starting at input variables. Then we optimize and simplify of the chain in order to make it practical. Our specific experiment indicates the effectiveness of our way.

Keywords: *Software behavior, trusted computing, cross-reference, reverse analysis*

1. Introduction

Trusted computing is becoming a new trend of computer security fields. Trustiness of software is a focus of trusted computing research. Current static trusted authentication of software is via checking hash value of binary codes or critical files before software running. However, it just guarantees data integrity of program before running.

The aim of dynamic trusted authentication of software is to make sure the real time behavior of software according to expecting behavior. Dynamic trusted authentication is an unavoidable obstacle in the trusted computing field. Dynamic trustiness model of software behavior depends on three key aspects, including the extraction and description of intended software behavior, real-time monitoring of software behavior and a contrastive analysis of intended software behavior and running-time behavior. How to abstract and describe the behavior of software is the paramount task of dynamic authentication. The accuracy and simplicity of the software behavior affect validity and practicability of the model. There is no generally acceptable theoretical method and the theoretical research lags behind the technical practice.

In our article, we present a means of extracting and describing of software trajectory based on cross-references of binary file. Our method is thinking from the perspective of the attacker, and constructing suitable size function chain starting at input variables. Then we optimize and simplify of the chain in order to make it practical and valid. Our specific experiment indicates the effectiveness of our way.

In recent years, many artful methods based on dynamic software behavior have been proposed for defending against these software abnormal behaviors; these include subverting the intended data flow in the program, subverting machine-code execution. For example, [2, 4, 7, 8, 9, 11, 12]. In general, the methods of software behavior were two patterns. Both of them are supported by reverse technology.

One of the patterns is that based on vitual technology, namely, in the vitual environment, researchers observe and study the behavior of the software, or, researchers

ISSN: 1738-9984 IJSIA

implant codes to source codes and set checking points. And researchers will propose various intelligent learning algorithms to the modeling of software motion. But this pattern is not having completeness of the behavior. It depends on specific input. Dynamic analysis involves allowing the malware to execute in a carefully controlled environment (sandbox) while recording every observable aspect of its behavior using any number of system instrumentation utilities [22].

The other one is that based on these analytic techniques, these techniques include analysis of control flow, analysis data flow, codes dependence. Researchers divide the whole disassembly code into basic blocks and analyze control flow between basic blocks. Data flow analysis is different from control flow. Data flow is following with interest of data changes and relationships of data objects. Data flow analysis is complex than analysis of control flow. The most complex one is analysis of code dependence. It shows both information about control flow and data flow. This pattern is having completeness of the software behavior.

2. Related Works

Study of software behavior involves four aspects works. The theoretic research is lag behind the actual technology. These theoretic models are based on statistics model [14], fuzzy mathematics model [15], subjective logic model [16, 17], software behavior model [2, 18], and finite state automaton model [19].

The research area of subverting control flow and data flow gives us an angle of attacker' view [20, 21]. M. Abadi, *et al.*, has present two important papers. The one illustrates the theory of secure control flow [20]. In this article, they gives us the basic theory with precise and rigorious formulations of hypotheses and guarantees. And another shows us the principles, implementations and applications of control -flow. In companion paper, they regard that the enforcement of control-flow integrity (CFI) is simple and practical for existing software to avert shadow calling stack and accessing control for memory regions. They explore the benefits for CFI and present an implementation [21].

Reverse technology is often to facilitate understanding of programs when source code is unavailable. Common situations in which disassembly is used include these: analysis of malware, closed-source software for vulnerabilities, closed-source software for interoperability. That is a very important supporting technology [22]. Static analysis attempts to understand the behavior of a program simply by reading through the program code, which, in the case of malware, generally consists of a disassembly listing. Disassembly listings also provide the only means to determine exactly how a compiler has chosen to order all of the variables declared globally or within functions. Understanding the spatial relationships among variables is often essential when attempting to develop exploits. Ultimately, by using a disassembler and a debugger together, an exploit may be developed [22].

The system call sequence is used to depict software behavior. It is also for this reason; the extracting and describing of software trajectory must match needs of practicability. The research on system call is indispensable. Remarkable works include Forrest, *et al.*, [1], Giffin, *et al.*, [23], Spivey [24], Bond and McKinley [25]. Forrest firstly proposed distinguishing program self using system call. Giffin, *et al.*, proposed considering system context. Spivey, *et al.*, presented calling context tree (CCT). Bond and McKinley presented probabilistic calling context (PCC).

Of course, other skill is indispensable, such as software testing technology and hardware technology [3, 5, 6, 10, 13] .

3. Analysis of Software Defects

3.1. The View of Program Generation

In a traditional software development model, compilers, assemblers, and linkers are used by themselves or in combination to create executable programs. The third-generation languages are generally platform independent, though programs written using them may be platform dependent as a result of using features unique to a specific operating system [22]. So the software defects are associate with these factors, these include, operation system, compilers, assemblers, linkers, CPU and so on.

3.2 The View of Attacker

How did the attacker make it after they have gotten the fundamental principles? We have made synthesis of responsible disclosure. Briefly, there are 4 common steps. The first step in the process is to discover a potentially exploitable condition in a program. This is often accomplished using dynamic techniques such as fuzzing [22]. Secondly, attackers analyze the control flow of software and master the control flow in order to discover vulnerabilities. In addition, recognizing input data of application program. Thirdly, tracing the input data until discovering the hole of the software. Finally, attackers inject illegal code and hijack the legal control flow. So, we assert that attackers can not directly modify the PC and reserved registers without key variable data. And now, let me review previous meaningful work. The enforcement of control flow integrity is an effective method adopted for software behavior. A control flow graph is considered as the normal behavior of software after statically specified. We suppose that the software is trusted when the program run dynamically follows the former path.

4. The Models of Extracting the Software Intended Behavior

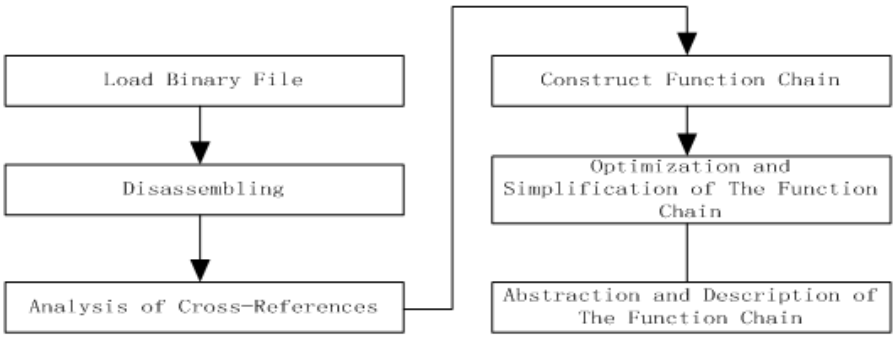

Figure 1. The Flow Chart of Abstracting Intended Behavior

Figure 1. Shows 6 Steps of our Abstracting Model

Step 1. Load the new file of binary file.

The authors of application software seldom to provide the source code of their productions. It is virtually important to know the file type before loading the binary file.

Step 2. Preliminary analysis of Binary File by IDA pro

IDA is a comprehensive and recursive disassembler. It makes every effort to present code as close to source code as possible. It also goes to great lengths to annotate generated disassemblies with not only datatype information but also derived variable and function names. We mainly use the information of data cross-reference and code cross-reference in our analysis of intended software behavior.

Step 3. Analysis of Cross-references

Some of the more common questions asked while reverse engineering a binary are along the lines of “Where is this function called from?” and “What functions access this data?” These and other similar questions seek to catalog the references to and from various resources in a program. Two examples serve to show the usefulness of such

questions [22].

Program changed the status one after another along with the control-flow. Traditionaly, analyzer lists all the possible path including conditional branches, jumps, procedure calls and procedure returns. All the instructions likely composite infinite combination paths. Our strategy is used to optimize the set of combination paths, concentrating on the most possible path which contains data closely related to input. We must analyze all cross-references of code and data firstly, and then monitor all paths containing variables related to the input, including direct and indirect. The overhead of prior approach is huge and waste more expansive time .We only tried to record paths that are closely related to the input data without all variables in all paths. We focused on optimization program to eliminate the overhead and tracing time.

Step 4.Construct function chain

Our object of study in our work is function chain. The function is a suitable size object in software behavior research. The cross-references of data, codes and functions are made from one address to another address in IDA. We exploit cross-references information that IDA makes available and the tools for accessing cross-references data. We choose to think of the addresses of functions as nodes in a directed graph and cross-references as the edges in that graph. In Figure 2, it shows a simple graph that was consisted of three node and two directed edges. Firstly, we create a graph node for each function. Secondly, on the basis of the call cross-reference between two functions, we generated function call graph by connecting function nodes.

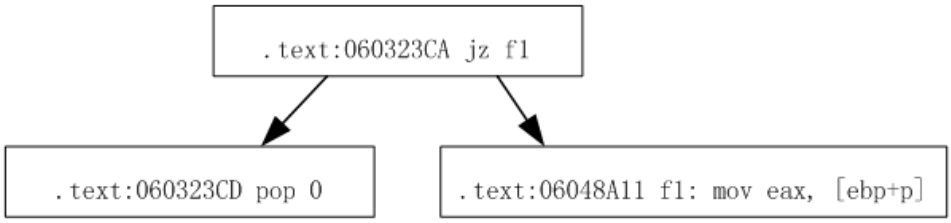

Figure 2. Basic Directed Graph

Step 5. Optimization and simplification the function chain.

In general, we will get a nasty mess graph. Because the compiler has inserted wrapper code for initialization, termination and configuration. We will optimize and simplify the whole function calling graph in virtue of graph theory and restrictive condition of input parameters. Finally, the whole calling graph will transformed into function chain with 6 functions at most.

Step 6. Abstraction and Description of the function chain.

We must abstract and description the function chain in order to contrast with the real-time behavior. We choose a linear structure to save the function nodes information. It shows in Figure 3. The precursor domain represents the function node before current function node. The subsequent domain represents the function node after current function node. The order domain represents the location order of current function in the function chain. And the name domain indicates the name of current function node.

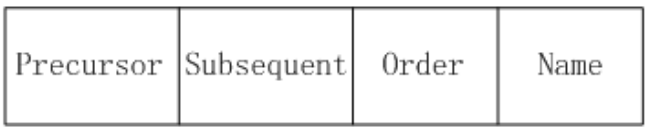

Figure 3. The Storage Structure

5. Experiment and Evaluation

In the previous section we provide 6 steps for abstracting intended behavior, based on short sequences of system calls. These sections propose the details and related experiment.

We ran the experiment on a laptop with i3-370M 2.4G processor and 4G memory. The operating system is windows 7 Ultimate. We selected the trivial example program from website in order to illustrate the problem. Figure 4 shows the source code of the program. Figure 5 shows the function calls of disassembling binary file of the source code after compiling by gcc.

```
#include <stdio.h>
int main (){
layer1_1();
layer1_2();}
void layer1_1() {
layer2_1();
layer2_2();
Printf("This is the first layer \n");}
void layer1_2() {
layer2_3();
layer2_4();
Printf("This is the first layer \n");}
void layer2_1() {
printf("This is the second layer  \n");}
void layer2_2() {
fprintf("This is the second layer  \n");}
void layer2_3() {
printf("This is the second layer  \n");}
void layer2_4() {
fprintf("This is the second layer  \n");}
```

Figure 4. Source Code

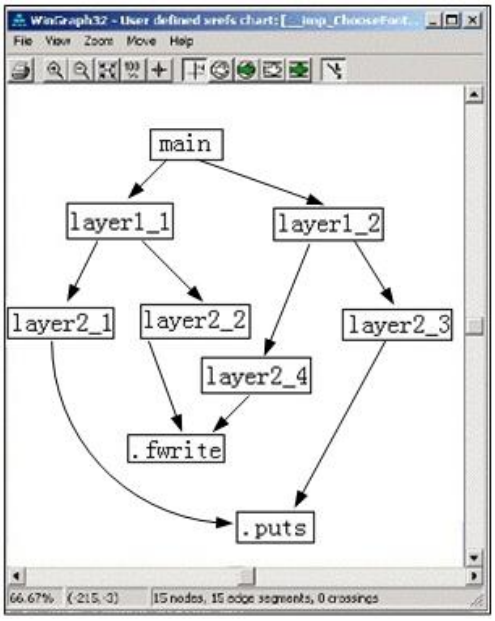

Figure 5. Function Calls

There are 4 function chains from function main (In Figure 6). These four chains are the optimal chains. Then we save all the function nodes as shown in the Figure 3. We give the node of layer2_2 as shown in the Figure 7.

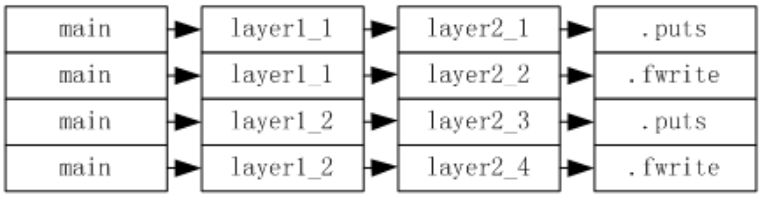

Figure 6. Function Chains

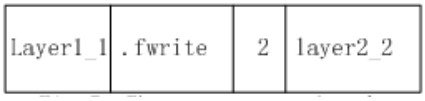

Figure 7. The Structure of Node

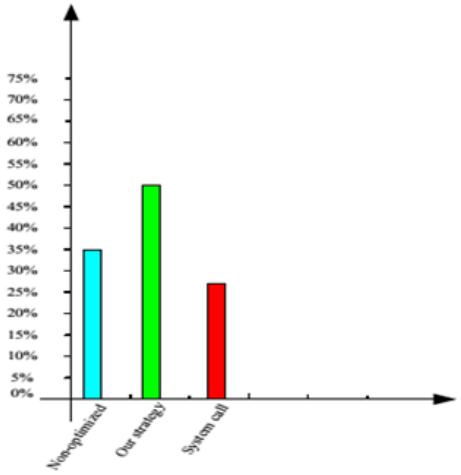

Figure 8. Radio of Successful Execution

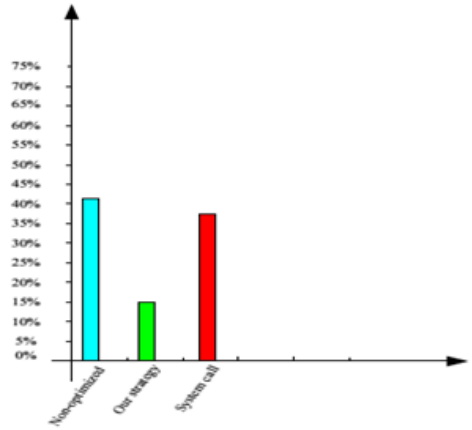

Fig 9. Radio of False Execution

Figure 9. Radio of False Execution

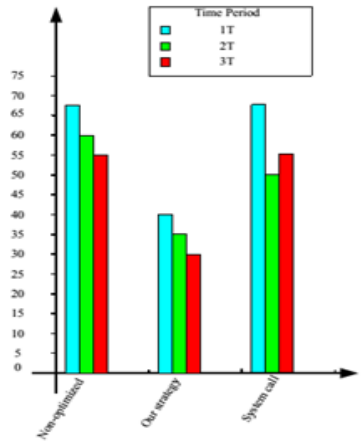

Figure 10. Instruction Number

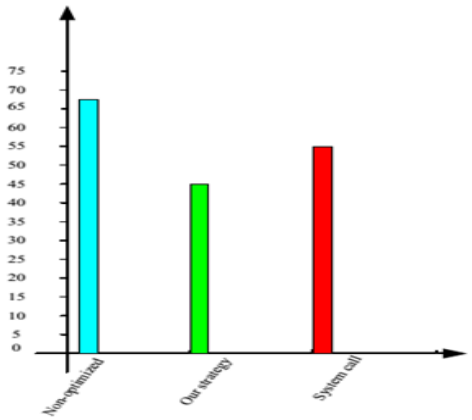

Figure 11. Mean Time

In Figure 8, our strategy shows higher radio of successful execution than nonoptimized program and program based on system calling, and our strategy shows lower radio of false execution as show in Figure 9. In Figure 10, our strategy shows lower number in 1-3T than non-optimized program and program based on system calling, and our strategy shows lower mean time execution as show in Figure 11.

6. Conclusions

We present an effective strategy to abstract and describe the intended behavior of software. We are following 6 steps to build the trace model of software based on function chains. The evaluation of the example shows that our model is an effective model. However the experiment of our model was applied in trivial program. Our next target is complex program.

Acknowledgments

We wish to thank the colleagues in our research group for their help and encouragement.

Reference

[1] S. Forrest, S. A. Hofmey, A. Somayaji, *et al.*, "A Sense of Self for Unix Processes", Proceeding of the IEEE Symposium on Security and privacy, Washington, DC IEEE Computer Society, **(1996)**, pp.

120-128.
[2] Y. Xiaohui, Z. Xuehai and T. Jjunfeng, "Hierarchiealand Dynamic Trusted Evaluation Model Based on Agent", Proceedings of 2009 Conference on Communication Faculty. Irvine, CA, USA: Scientific Research Publishing, **(2009)**, pp. 312-316.
[3] D. Jackson, "A direct path to dependable soft", Commun ACM, vol. 52, **(2009)**, pp. 78-88.
[4] P. K. Manadhata, K. M. C. Tan, R. A. Maxion, *et al.*, "An Approach to Measuring A System's Attack Surface", CMU-CS07-146, **(2007)**.
[5] M. R. Lyu, "Handbook of Software Reliable Engineering", New York IEEE Computer Society Press, McGraw-Hill Book Company, **(1996)**.
[6] P. Godefroid, M. Y. Levin and D. Molnar, "Auto Whitebox Fuzz Testing", Proceedings of 16th Annual Network & Distributed System Security Symposium. San Diego, CA, USA: Internet Society, **(2008)**.
[7] Y. Deng and G. Wang, "A Time-Related Trust Model Based on Subjective Logic Theory", In Proceedings of The 9th International Conference for Yong Computer Scientists (ICYCS 2008), Hunan China, **(2008)**, pp. 1902-1907.
[8] M. Castro, M. Costa and S. T. Harris, "Securing software by enforcing data-low integrity", **(2006)**.
[9] M. Abadi, M. Budiu, U. Erlingsson and J. Ligatti, "Control-flow integrity: Principles, implementations and applications", In ACM computer and Communication Security conf., **(2005)**.
[10] G. Necula, "Proof-carrying code", In Proceeding of the 24thACM Symposium on Principles of Programming Languages, **(1997)** January, pp. 106-119.
[11] S. McCamant and G. Morrisett, "Efficient, verifiable binary sandboxing for a CISC architecture", Technical Report MIT-LCS-TR-988, MIT Laboratory for Computer Science, **(2005)**.
[12] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula and N. Fullagar, "Native client: A sandbox for portable, untrusted x86 native code", In the 30th IEEE Symposium on Security and Privacy (Oakland), **(2009)**.
[13] D. Sehr, R. Muth, C. Biffle, V. Khimenko, E. Pasko,K. Schimpf, B. Yee and B. Chen, "Adapting software fault isolation to contemporary cpu architectures", In the 19th conference on USENIX Security Symposium, **(2010)**.
[14] J. Patel, W. T. L. Teacy, N. R. Jennings, *et al.*, "A Probabilistic Trust Model for Handling Inaccurate Reputation Sources", Trust Management, 3rd International conference on iTrust Heidelberg Springer Berlin, **(2005)**, pp. 193-209.
[15] T. Flaminioa, G. M. Pinnab and E. B. P. Tiezzia, "A Complete fuzzy logical system to deal with trust management System. Fuzzy Sets and Systems, vol. 159, no. 10, **(2007)**, pp. 1191-1207.
[16] A. Josang, R. Hayward and S. PoPe, "Trust network analysis with subjective logic", ACM International Conference Proceeding series, Proceedings of the 29th Australia Computer Science Conference (VOI.48).Hobart, Australia: ACM, vol. 171, **(2006)**, pp. 85-94.
[17] A. Josang, "Conditional Reasoning with Subjective Logic", Journal of Multiple-Valued Logic and Soft Computing, vol. 15, no. l, **(2008)**, pp. 5-38.
[18] W. Wang, X. H. Guan, X. L. Zhang, *et al.*, "Profiling Program Behavior for Anomaly Intrusion Detection Based on The Transition and Frequency Property of Computer Audit Data", Computers & Security, vol. 25, no. 5, **(2006)**, pp. 539-550.
[19] D. Wagller and D. Drew, "Intrusion Detection via Static Analysis", Proceedings of the IEEE Symposium on Security and Privacy. Washington, DC: IEEE Computer Society, **(2001)**, pp. 156-169
[20] M. Abadi, M. Budiu, U. Erlingsson and J. Ligatti, "A theory of secure control flow", In International Conf. on Formal Engineering Methods, **(2005)**.
[21] M. Abadi, M. Budiu, U. Erlingsson and J. Ligatti, "Control-flow integrity: Principles, implementations and applications", In ACM computer and Communication Security conf., **(2005)**.
[22] C. Eagle, "The IDA Pro Book", William Pollock, **(2009)**.
[23] J. T. Giffin, D. David, S. Jha, *et al.*, "Environment-Sensitive Intrusion Detection", Proceedings of the 8th International symposium on Recent Advances in Intrusion Detection, Heidelberg Springer Berlin, **(2005)**, pp. 185-206.
[24] J. M. Spivey, "Fast, Accurate Call Graph Profiling", Software-Practice & Experience, vol. 34, no. 3, **(2004)**, pp. 249-264.
[25] M. D. Bond and K. S. Mckinley, "Probabilistic Call Context", Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-oriented Programming systems and applications, New York: ACM, **(2007)**, pp. 97-112.

Authors

Wei LIU, PhD. Candidate, Beijing Institute of technology. His researches include computer security and trusted computing.

Liping LIU, PhD. Candidate, Beijing Institute of technology. His researches include computer security and trusted computing.

Linlin CI is a professor and doctoral supervisor of Beijing Institute of technology. His research interests include computer security, sensor networks, and distributed system.

