Atlantis Highlights in Engineering, volume 3
3rd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2018)

A Review of Researching on Dynamic Taint Analysis Technique

Peiwu Dai[1*],Zulie Pan[2*],Yang Li[3*]
1. College of Electronic Engineering, National University of Defense Technology
petrichor0324@163.com,18955185317@189.cn, 411391400@qq.com

Abstract—Taint analysis technique is the key technique means for analyzing the robustness of programs and vulnerability mining. By marking the data which are sensitive or untrusted, one can observe the flow of these tainted data during program execution, then determine whether the marked data affects the key nodes of the program. According to the implementation mechanism, the taint analysis can be divided into static taint analysis and dynamic taint analysis. As an auxiliary technique, it can be combined with mainstream vulnerability mining techniques such as fuzzing test and symbol execution, playing a great role in test case construction and path feasibility analysis. This article firstly introduces the basic concepts of dynamic taint analysis technique. Second, it focuses on the process of dynamic taint marking, propagation and detection. Then it summarizes the main defects in taint analysis and the application status of dynamic taint analysis technique. Finally, it is compared with the current mainstream taint analysis tools and explores the future trends of taint analysis technique.

Keywords—taint analysis; vulnerability detection; information flow tracking; taint propagation

Ⅰ. INTRODUCTION

With the rapid development of network technique, Internet products are blooming and bringing convenience to people's lives. At the same time, due to the weak skill level and security awareness of programmers, these applications have huge security risks. In recent years, the number of vulnerabilities and the proportion of Zero day Vulnerabilities has increased obviously year after year [1]. Not only that, due to the users weak security awareness, they do not update the patch in time after the manufacturer released the patch. The number of attacks on the 1day vulnerabilities increases gradually. These attacks are becoming more destructive. Hackers can create malicious and aggressive programs based on official patches in a short period of time to attack unpatched systems or servers [2]. According to the statistics of 360 Threat Intelligence Center, the vulnerabilities used by top APT organizations such as equations in recent years are mainly classified about ten categories, including two types of server vulnerabilities, such as the SMB protocol vulnerability "Eternal Blue" and the firewall device vulnerabilities in the NSA network arsenal. Eight types of client vulnerabilities, are mainly included in IOS and Android, flash vulnerabilities and Windows privilege vulnerabilities and other four types of office vulnerabilities. With the popularity of mobile devices, the target of APT attacks has extended to the mobile device field in recent years. For example, the famous Trident IOS vulnerability and the remote2local Android vulnerability are very common and destructive. The mobile network attack technique is more casual and concealed. This will also become the focus of work in the field of network attack and defense [3].

Based on the technical principle of taint analysis, this paper attempts to comprehensively introduce, analyze and summarize the technical status and application prospects of dynamic taint analysis. Sections 2 detail the principles of dynamic taint analysis, Section 3 explores some of the shortcomings of dynamic taint analysis and current improvements for these defects. And section 4 discusses about the main applications of the dynamic taint analysis technique. Finally, the paper is summarized and the future development trend of dynamic taint analysis technique is estimated.

Ⅱ. TECHNICAL THEORY

A. *Approximately classification*

Dynamic taint analysis for taint marking, tracking, and detection during actual program operation. According to the different streams, it can be divided into dynamic flow analysis technology based on data flow and control flow. Marking external taint data and tracking its explicit propagation method in memory is a common analysis method based on dynamic taint analysis of data flow. Such as TaintCheck [4], Flayer [5], Vigliante and so on. Establishing a control flow graph (CFG) of the program by analyzing the control flow of the program is the usually way of taint analysis based on control flow. Using a specific algorithm to monitor and analyzing the implicit information flow of the propagation is a way based on data flow. Such as Dytan [6], libdft [7], Bitblaze [8], Agros [9] and so on.

B. *Taint mark (source point) and detection (sink point)*

The taint analysis structure framework is shown in Figure 1: Source points are generally composed of entry points or external function calls that the program accepts input data. These input data can be roughly divided into three categories: file input, network input, and external device input. Using machine learning combined with statistics to automatically identifies the source and sink points [10]. And the second type, such as DroidSafe [11], is manually labeled based on API which called by programs. The third class considers external data to be untrustworthy. Based on the intuitive or empirical, the tagged variable is gradually optimized for heuristics. Such as the tool for PHP vulnerability analysis, Aspis [12]. There are three main rules for detection: the first one uses taint data as the jump address. The memory data pointed to by the EIP is also the contaminated data. One is over-tainting. Marking non-dependent variables as taint, causing the analysis path to explode during the propagation process, and the system resources are consumed more. The other type is under-tainting. Variables that control dependencies (implicit streams) may be mislabeled as non-tainted during taint propagation, resulting in underreporting.

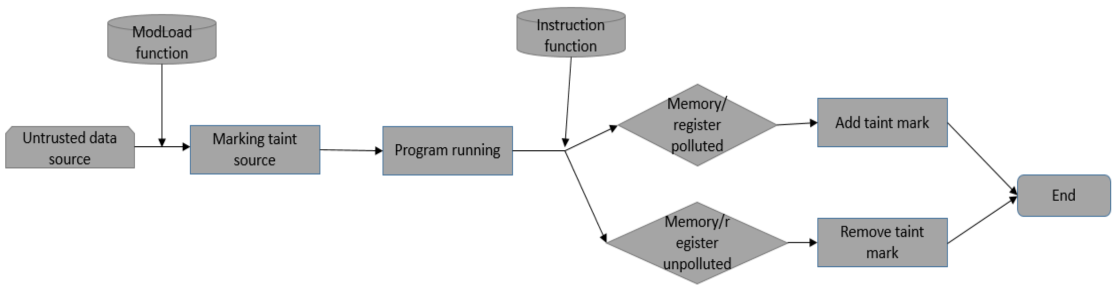

Fig.1 The structure of taint marking module

At present, there are two main methods for the taint propagation rules of binary dynamic taint analysis. The table 1 outlines the approximate instructions used by the spread of the taint. Table 2 refers to the taint propagation logic applied by the TaintDroid tool.

Table 1.Taint analysis instruction/function classification

Instruction/function type	Instruction	example	proporgation rule
Data transfer instruction	MOV、XCHG、LEA、LDS、LES、PUSH、POP、PUSHF、POPF、CBW、CWD、CWDE	mov a, b	R(a)←R(b)
Arithmetic instruction	ADD、ADC、INC、SUB、SBB、DEC、CMP、MUL、DIV、DAA、DAS、AAA、AAS	Add a, b	R(a)←R(b)
Logical instruction	AND、OR、XOR、NOT、TEST、SHL、SAL、SHR、SAR、RCL、RCR、ROL、ROR	Xor a, a	R(a)←false
Control transfer instruction	JMP、JCC、JCXZ、LOOP、LOOPZ、LOOPNZ、LOOPNE、CALL、RET、INT	Call memcpy()	R(dst)←R(src)
String operation instruction	MOVS、LODS、STOS、CMPS、SCAS	Movs %dx %eax	R(edx)←R(eax)
Indirect taint function	Strncpy(buf,source,len)	Call strncpy()	R(dst)←R(src)
Sensitive function	Scanf、sprintf、strcpy	Call strcpy()	R(dst)←R(src)

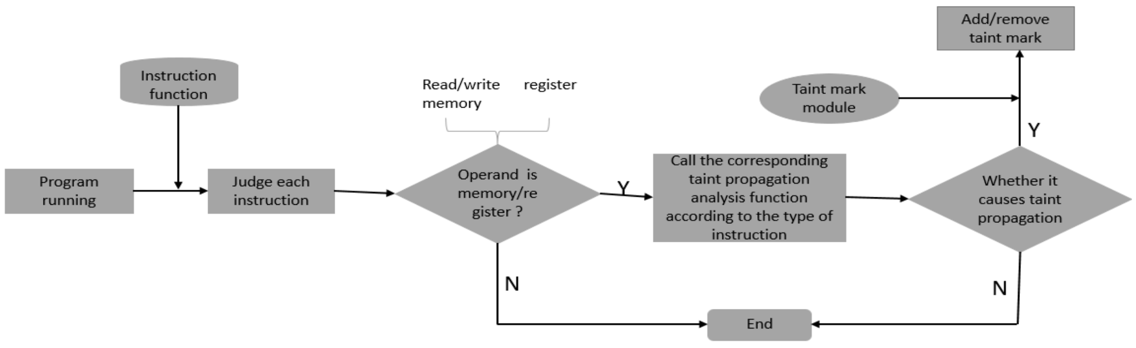
Add/remove taint mark
Instruction function
Read/write memory
register
Taint mark module
Y
Program running
Judge each instruction
Operand is memory/register ?
Y
Call the corresponding taint propagation analysis function according to the type of instruction
Whether it causes taint propagation
N
N
End

Fig.2.The structure of taint tracing module

Taint deletion aims to clear some taint marks in the data during the propagation process. Otherwise it will spread indefinitely, and the taint will be deleted when the following three types of operations are encountered:

1) Pass non-taint data to the registers and memory instructions that store the taint.

2) Some special logic operations will remove the taint marks, such as xor eax, eax, sub eax, eax, move eax Imm, etc.

3) When a tainted register is passed to a tainted register, the passed register loses its original taint mark.

III. Defects and Measures

The application of taint analysis technique to the field of vulnerability mining mainly involves the following aspects:

A．Issues on dynamic implicit flow tag range

Accurate implicit flow propagation analysis needs to analyze whether each branch control condition propagates the taint mark in the actual taint analysis process. This often leads to over-tainting. It causes a large scope of spread of the taint, eventually producing a path explosion, which not only consumes a large amount of system resources. Moreover, the amount of information obtained by users is huge. It is difficult to screen, or insufficiently labeled, resulting in inaccurate analysis.

B. Missing reports and false reports

Partially leaked means that the taint information propagated and leaked through part of the program that is not executed. The main influencing factors of taint analysis accuracy are false report which caused by over-tainting and missing report which caused by under-tainting. These affect the efficiency of vulnerability mining.

C. The defects of Current taint analysis tools which cannot balance speed and accuracy

Dynamic taint analysis technique can be divided into fine-grained dynamic taint analysis technique and coarse-grained dynamic taint analysis technique according to the analysis granularity. Fine-grained taint analysis solves the problem of backtracking of data flow with high precision. It is mainly used to detect vulnerable points of attack. The coarse-grained taint analysis is fast and takes up little space.

D. Problems such as not supporting the floating point instructions, low execution efficiency, and taint loss in JIT (just in time) translation execution.

Jinxin Ma [13] proposed a taint analysis method based on the offline index of the execution trace. The offline index can be used to skip instructions that are unrelated to taint data. It can increase efficiency in this way. A more perfect taint propagation algorithm is also proposed, which supports the propagation of floating-point instructions and improves the detection capability of the vulnerability.

Ⅳ. APPLICATION FIELD STATUS

A. Crash verifing

Fuzzing technique and dynamic symbol execution generate a large number of program crashes. There are mainly three tools which use crash analysis technique to determine whether the crashes are available. ! Exploitable [14] that released by Microsoft. BitBlaze, a binary information analysis platform developed by UC Berkely. And CRAX [15], an automated vulnerability utilization framework based on the AEG method proposed by IEEE in 2012.

B. Program vulnerability detection

Using the taint analysis to detect program vulnerability and the propagating taints to various types of vulnerability functions can find out the buffer overflow and formatting strings in the program. Such as the dynamic and static detection technique based on the JavaScript platform proposed by SHYI. The Java web prototype system for web XSS vulnerability designed by BH Liang [16] can track the flow of web applications. It is a good way to detect XSS vulnerabilities. Pin instrumentation and TaintCheck can also detect program vulnerability.

C. Information leak prevention

The information carrying capacity of mobile terminal devices is increasing day by day. So, the information leakage detection for Android is one of the most widely used fields of current taint analysis technique. TaintDroid can track multiple sensitive data sources at the same time.

D. Attacking feature generation and detection

The existing black box or white box attack feature generation method faces difficulties in sample collection, low degree of automation, and dependence on source code. Yu Liu [17] proposed an attack feature generation method based on backtrackable dynamic taint analysis. It generates the attack features of the Turing machine by monitoring the dynamic execution of the process, extracting the operational sequences and extracting the sequence of operations and constraints associated with the attacking input. ERIKA [18] modified the Java library class and the Java servlet framework to prevent command injection vulnerabilities for the Java web.

E. Vulnerability mining

The most widely used field of dynamic taint analysis technique is vulnerability mining. Combining with Fuzzing technique, it automatically tracks the areas of the input samples that affect the key positions of the program. It can verify some logic errors in large programs, such as BuzzFuzz. Dynamic taint analysis combined with symbolic execution technique can automate the verification of crash input in a program, improve the efficiency of symbol execution, and mitigate the path explosion problem of symbol execution to some extent, such as TaintScope [19].

F. Malware detection

A big challenge for malware detection is that the software and hardware resource costs required by software detector makes it impossible to monitor each application. SwordDTA [20] proposed by CAJ Jun detects malware through vulnerability modeling and taint propagation. For mobile malware detection, Y Aafer proposed DroidAPIMiner [21] to extract the relevant functions of the captured API-level malware. And using the generated feature set to evaluate different classifiers can achieve higher accuracy. Combining taint analysis technique with machine learning and training malware sample sets, online identification and detection of malware is an important technique point in the future.

Ⅴ. SUMMARY AND OUTLOOK

According to the analysis of this paper and combining the research status of this field, it is preliminarily believed that the future development trend of taint analysis technique will have the following points:

A. Program performance analysis, defect location, and error detection

How to design mining tools based on taint analysis technique to reduce the false reporting rate and discover the security vulnerabilities from common fuzzing testing tools. It will be a hot research direction. The RETracer [22] tool performs binary-level back-taint analysis. It understands how functions on the stack cause crashes without recording execution traces.

B. Optimize the efficiency of dynamic taint analysis

How to reduce the analysis cost as much as possible without affecting the analysis performance is a difficult point. How to selectively control the number of taints that you want to spread is one of the solutions. It is good to delete unnecessary taints in time. Reducing unnecessary system resource consumption and combining the scope of semantic rule deletion analysis are also good solutions.

C. Hidden channel issues

At present, the pollution detection methods for implicit flow generally have the status of over-tainting and under-tainting. For issues such as implicit flow, shutdown channel, probability channel, how to choose a reasonable propagation branch is a difficult point. In addition, for some detection tools for Android (such as TaintDroid, Droid Analytics [23], etc.) can only be improved through some heuristic strategies.

D. Automation

One idea is to provide a unified mapping language F4F [24] for analysis. It solves the problem of callback function characteristics by adding virtual access points. Due to the complexity of the analysis language and some of the drawbacks of the dynamic taint analysis technique itself (over-contamination, under-contamination, etc.), the implementation of automation is still very difficult.

References

[1] Lai Y P, Hsia P L. Using the vulnerability information of computer systems to improve the network security[J]. Computer Communications, 2007, 30(9):2032-2047.

[2] Zero Day Initiative.www.zerodayinitiative.com/[2018-8-27].

[3] Paper. https://paper.seebug.org/565/[2018-08-27].

[4] Zhu G M, Zeng F P, Yuan Y, et al. Blackbox Fuzzing Testing Based on Taint Check[J]. Journal of Chinese Computer Systems, 2012, 33(8):1736-1739.

[5] Drewry W, Ormandy T. Flayer: exposing application internals[C]// Usenix Workshop on Offensive Technologies. USENIX Association, 2007.

[6] Clause J, Li W, Orso A. Dytan: a generic dynamic taint analysis framework[C]// Proceedings of the 2007 international symposium on Software testing and analysis. ACM, 2007:196-206.

[7] Kemerlis V P, Portokalidis G, Jee K, et al. libdft: practical dynamic data flow tracking for commodity systems[C]// ACM Sigplan/sigops Conference on Virtual Execution Environments. ACM, 2012:121-132.

[8] Miller C, Caballero J, Berkeley U, et al. Crash analysis with BitBlaze[J]. Revista Mexicana De Sociologí a, 2010, 44(1):81-117.

[9] Argos. http://www.few.vu.nl/argos/[2018-8-27].

[10] Rasthofer S, Arzt S, Bodden E. A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks[C]// Network and Distributed System Security Symposium. 2014.

[11] Gordon M I, Kim D, Perkins J, et al. Information-Flow Analysis of Android Applications in DroidSafe[C]// Network and Distributed System Security Symposium. 2015.

[12] Crandall J R, Chong F T. Minos:Control Data Attack Prevention Orthogonal to Memory Model[C]// International Symposium on Microarchitecture. IEEE, 2004:221-232.

[13] Jinxin Ma, Zhoujun Li, Tao Zhang. Research on Taint Analysis Method Based on Execution Trace Offline Index[J]. Journal of Software, 2017,

28(9): 2388-2401.

[14] Microsoft. ! exploitable crash analyzer------ MSEC debugger extensions [EB /O L]. [2018-09-05]. http：/ /msecdbg. codeplex. com.

[15] Huang S K, Huang M H, Huang P Y, et al. CRAX: Software Crash Analysis for Automatic Exploit Generation by Modeling Attacks as Symbolic Continuations[C]// IEEE Sixth International Conference on Software Security and Reliability. IEEE Computer Society, 2012:78-87.

[16] Pezzè M, Harman M. Proceedings of the 2013 International Symposium on Software Testing and Analysis[J].

[17] Yu LIU, Mei-ning NIE, Yi-rui SU. Attack feature generation method based on backtrackable dynamic taint analysis[J]. Transactions of Communications, 2012, 33(5): 21-28.

[18] Chin E, Wagner D. Efficient character-level taint tracking for Java[C]// ACM Workshop on Secure Web Services. ACM, 2009:3-12.

[19] Wang T, Wei T, Gu G, et al. TaintScope: A Checksum-Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection[C]// IEEE Symposium on Security and Privacy. IEEE Computer Society, 2010:497-512.

[20] Cai J, Zou P, Ma J, et al. SwordDTA: A dynamic taint analysis tool for software vulnerability detection[J]. Journal of Wuhan University (Natural Science English Edition), 2016, 21(1):10-20.

[21] Aafer Y, Du W, Yin H. DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android[M]// Security and Privacy in Communication Networks. Springer International Publishing, 2013:86-103.

[22] Cui W, Peinado M, Sang K C, et al. RETracer:triaging crashes by reverse execution from partial memory dumps[J]. 2016:820-831.

[23] Zheng M, Sun M, Lui J C S. Droid Analytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware[C]// IEEE International Conference on Trust, Security and Privacy in Computing and Communications. IEEE Computer Society, 2013:163-171.

[24] Sridharan M, Artzi S, Pistoia M, et al. F4F: taint analysis of framework-based web applications[J]. ACM SIGPLAN Notices, 2011, 46(10):1053-1068.