Retracted: Explicit Untainting to Reduce Shadow Memory Usage in Dynamic Taint Analysis

Young Hyun Choi¹, Min-Woo Park¹, Jung-Ho Eom², and Tai-Myoung Chung¹,*

¹ Dept. of Electrical and Computer Engineering, Sungkyunkwan University
300 Cheoncheon-dong, Jangan-gu, Suwon-si, Gyeonggi-do, 440-746, Republic of Korea
{yhchoi,mwpark,tmchung}@imtl.skku.ac.kr
² Military Studies, Daejeon University
62 Daehakro, Dong-gu, Daejeon, Republic of Korea
eomhun@gmail.com

Abstract. As the growth of computing technologies and smart service, the dimension for importance of security of a system has been increased dramatically. Many researches for solving threats of software vulnerabilities have been proposed in worldwide. Ordinary program testing method for finding defects in software can be categorized into black-box testing and white-box testing. In Black-box testing, the tester does not need to tasks recognition of the internal structure of program, whereas in white-box testing, the tester checks to tasks recognition of internal structure of program. Taint analysis is an efficient black-box testing method for finding exploited crashes by tracking external input to the program. However, taint analysis method is too heavy and slow to provide for commercial analysis program, because of large amount of computation and shadow memory usage. Recent, many experimental approaches to weight down and to speed up the analysis process, but it were lacking in commercial use. In this paper, we propose a method to reduce shadow memory usage by selectively not trace the definite untainting memories. Our evaluation result shows that we can reduce number of taint operation by significant amount.

Keywords: Security, Dynamic Taint Analysis, Data Flow Analysis.

1 Introduction

Malicious software propagates throughout computer systems and networks leaking information and harming. Adversaries use software vulnerabilities of the system to do malicious actions. According to the annual report from Panda Security, total number of newly introduced malware in 2012 reached 27 million[1]. Moreover, Adversaries traded critical vulnerability of software directly with intruders. This means to find vulnerabilities of software is more important to defend the computer systems and networks[2].

* Corresponding author.

© Springer International Publishing Switzerland 2014.
General program testing techniques are classified into two main categories; Black-box testing and white-box testing. In Black-box testing, the tester does not need to tasks recognition of the internal structure of program, whereas in white-box testing, the tester checks to tasks recognition of internal structure of program. Most commercial programs do not release their source code, so black-box testing is usually valid on the binary analysis programs. Taint analysis traces the external input data to the program which helps the analysis to focus on crashes. It is find the higher probability of exploit. Recently, some framework of taint analysis tools are proposed in the academic, but these frameworks are not appropriate for works with commercial analysis program[7,8].

In this paper, we proposed a method to reduce shadow memory usage in dynamic taint analysis. We only trace and watch information of memories and registers that have higher probability of exploit. Therefore we do not assign space in shadow memory for memories and registers that are not being tracked. These help to more faster analysis[2].

The rest of this paper is organized as follows. In section 2 we explain the concept of taint analysis by using an example intermediate language presented in the paper. Section 3 discuss our proposal of reducing overhead in dynamic taint analysis and the we evaluate our schemes with test case in section 4. Finally we conclude this paper and present future works in section 5.

2 Taint Analysis

Taint analysis is a kind of information flow analysis. Tainted data has a considerable effect on the other data in the system. Taint analysis is a information flow tracing mechanism which tracks incoming data from the user input throughout whole process. Because of data marking ’tainted(tagged)’ proceed from the assumption which user input is untrusted[11]. The purpose of taint analysis is to track influence of tainted data with the taint policy[4]. A taint analysis determines the way of the influence of the tainted information along the execution of program [3,5,12,13]. There are a lot of researches being done on developing analysis frameworks to efficiently analyze programs [6,7,8]. Taint analysis consists of three main factors, taint introduction, taint propagation and taint checking [3]. A detail about each factor should be as follows subsections.

Taint analysis can be divided into three main factors, which are taint introduction, taint propagation and taint checking [3]. To explain taint analysis using formal methods, we present a assembly-like intermediate language in Fig. 1 and its operational semantics in Fig. 2. Operational semantic is written in \( \text{premise} \Rightarrow \text{condition} \) format. When premise is satisfied, program at \( \text{state}_1 \) will be changed to \( \text{state}_2 \). Functions with double brackets are semantic functions which returns semantic value of the given syntax and examples of them are shown in Fig. 3. \( s[ x \mapsto v ] \) means x is substituted by v in state s and \( \mu \) is replaced by \( x_2 \). For example, operational semantics of unary operation means that when expression \( e \) is evaluated into v in initial state s then the result of \( \text{unope} \) is \( U[[v]] \). [2].