Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Towards designing an extendable vulnerability detection method for executable codes[☆]

Maryam Mouzarani, Babak Sadeghiyan[*]

Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran

ARTICLE INFO

Article history:
Received 16 September 2015
Revised 31 July 2016
Accepted 12 September 2016
Available online 13 September 2016

Keywords:
Software vulnerability
Executable codes
General specification
Extendable method

ABSTRACT

Context: Software vulnerabilities allow the attackers to harm the computer systems. Timely detection and removal of vulnerabilities help to improve the security of computer systems and avoid the losses from exploiting the vulnerabilities.

Objective: Various methods have been proposed to detect the vulnerabilities in the past decades. However, most of these methods are suggested for detecting one or a limited number of vulnerability classes and require fundamental changes to be able to detect other vulnerabilities.

In this paper, we present a first step towards designing an extendable vulnerability detection method that is independent from the characteristics of specific vulnerabilities.

Method: To do so, we first propose a general model for specifying software vulnerabilities. Based on this model, a general specification method and an extendable algorithm is then presented for detecting the specified vulnerabilities in executable codes.

As the first step, single-instruction vulnerabilities–the vulnerabilities that appear in one instruction–are specified and detected. We present a formal definition for single-instruction vulnerabilities. In our method, detection of the specified vulnerabilities is considered as solving a satisfaction problem. The suggested method is implemented as a plug-in for Valgrind binary instrumentation framework and the vulnerabilities are specified by the use of Valgrind intermediate language, called Vex.

Results: Three classes of single-instruction vulnerabilities are specified in this paper, i. e. division by zero, integer bugs and NULL pointer dereference. The experiments demonstrate that the presented specification for these vulnerabilities are accurate and the implemented method can detect all the specified vulnerabilities.

Conclusion: As we employ a general model for specifying the vulnerabilities and the structure of our vulnerability detection method does not depend on a specific vulnerability, our method can be extended to detect other specified vulnerabilities.

1. Introduction

Detecting software vulnerabilities has been studied widely in the past decades. As a result various methods are presented to detect vulnerabilities more accurately, with less false positives and false negatives. However, most of these methods are suggested for detecting one or a limited number of vulnerabilities. Thus, their algorithms should be changed to detect new vulnerabilities [1]. Therefore, the enhanced techniques applied in one algorithm are not usable for detecting other vulnerabilities.

As an example, many advances have occurred in detecting the buffer overflow vulnerability during the past years. Different techniques have been suggested to detect this vulnerability, such as pattern matching [2], program annotation [3,4], constraint analysis [5] and taint analysis [6,7]. If we call the algorithm that analyzes the program and searches for a vulnerability in it as vulnerability seeking algorithm, most of vulnerability seeking algorithms in these works are designed based on the mechanism of the buffer overflow vulnerability. For example, the vulnerability seeking algorithm presented in [5] considers the strings in a C program as an abstract data type with pre-defined functions, such as *str-*

☆ This work is supported in part by APA Research Center of Amirkabir University of Technology (Tehran Polytechnic).

* Corresponding author.
E-mail address: basadegh@aut.ac.ir (B. Sadeghiyan).

http://dx.doi.org/10.1016/j.infsof.2016.09.004
0950-5849/© 2016 Elsevier B.V. All rights reserved.

cpy(), strcat(), etc. The state of each string is also summarized with two integer values, i. e. its allocated size and its current length. Thus, the content of the strings is not important for this algorithm. For each string buffer, the algorithm examines string manipulation statements to check whether the maximum length of a string exceeds its allocated size. If such condition is detected, a buffer overflow vulnerability would be reported. This algorithm requires fundamental changes to be able to detect another vulnerability, such as format string, command injection or dangling pointers.

It is worth mentioning how we differentiate a vulnerability detection technique from a vulnerability seeking algorithm. A vulnerability detection technique is the general instruction for finding vulnerabilities in the programs and is not usually specific to a particular vulnerability. For example, taint analysis is a detection technique that has been used for detecting various vulnerabilities, e. g. SQL injection [8,9], XSS [8,10], buffer overflow [6,7] and format string [11]. A vulnerability seeking algorithm is an accurately defined instruction for analyzing particular programs and seeking specific vulnerabilities based on one or a combination of vulnerability detection techniques. For example, the vulnerability seeking algorithm in [5] is designed based on the constraint analysis technique to detect buffer overflow in C programs. The design of most of the vulnerability seeking algorithms depends on the mechanism of the intended vulnerabilities. For example, a successful buffer overflow seeking algorithm may not be easily used to detect other types of vulnerabilities.

We believe that an extendable vulnerability seeking algorithm could be a solution for this limitation. By an "extendable vulnerability seeking algorithm", we mean an algorithm that is able to detect the specified vulnerability classes in the target program, even the vulnerabilities that are specified in the future. Vulnerabilities have some common characteristics and can be defined in a general structure. Also, there are vulnerability detection techniques that have been used separately for detecting various vulnerabilities, such as taint analysis or symbolic execution [12–14]. Thus, an extendable vulnerability seeking algorithm that is designed based on such techniques may be able to detect different vulnerabilities at the same time.

To be extendable, the vulnerability seeking algorithm should be independent from the specification of the vulnerabilities as much as possible. In this way, the vulnerability seeking algorithm can be improved separately and get benefit from the enhancements in other detection techniques. In this paper, we present a first step towards designing a general extendable vulnerability detection method. This method consists of a general specification model for specifying vulnerabilities in a way that is understandable by the vulnerability seeking algorithm. It also contains an extendable vulnerability seeking algorithm that searches for any specified vulnerabilities in the program automatically.

There are a limited number of extendable vulnerability seeking algorithms and vulnerability specification methods presented by now, that will be reviewed in Section 2. However, there is no extendable vulnerability detection method for executable codes. The advantages of analyzing executable codes, instead of source codes, in detecting software vulnerability have propelled us to take steps in designing an extendable vulnerability detection method for executable codes. Reflection of the exact behavior of the program, optimizations and bugs in the compilers, unavailability of the source codes and platform-specific details are some reasons that make analyzing executable codes more preferable [15]. Moreover, analyzing the executable codes for detecting the vulnerabilities makes the method independent from the development language and thus the method would cover more programs.

In this paper, we consider the vulnerabilities that appear in a single instruction. Therefore, the paper is regarded as a first step towards designing an extendable detection method. Single-instruction vulnerabilities can be specified based on the arguments of one instruction, such as division by zero [16] and some integer bugs [17–19]. Other vulnerabilities that appear in a scenario, in more than one instruction, are not considered in this paper and will be studied in our future works.

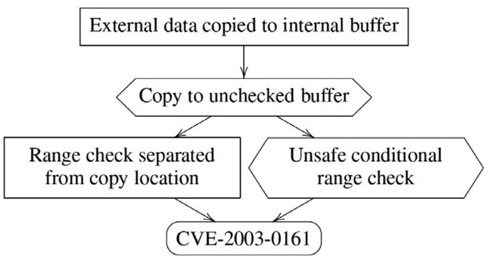
External data copied to internal buffer
Copy to unchecked buffer
Range check separated from copy location
Unsafe conditional range check
CVE-2003-0161

Fig. 1. A sample VCG for CVE-2003-0161 [22].

We present a general model for specifying the vulnerabilities. Based on this model, vulnerabilities are specified by the use of Vex language. Vex is an intermediate representation for the executable codes that is used in Valgrind binary instrumentation framework [20]. The vulnerability seeking algorithm is implemented as a plug-in for Valgrind. It automatically searches in executable codes for any specified single-instruction vulnerability. The ease of extending the method to other single-instruction vulnerabilities will be demonstrated.

Hence, the followings can be considered as the contributions of this paper:

- Presenting a general model for specifying software vulnerabilities.
- Presenting a method for specifying the single-instruction vulnerabilities to be detected in the executable codes.
- Presenting an extendable vulnerability detection method for executable codes based on the proposed specification model.
- Specification and detection of vulnerability classes division by zero [16], NULL pointer dereference [21], integer overflow [17], integer underflow [18] and incorrect width conversion in numeric type (for integer type) [19] using the proposed extendable detection method.

This paper is organized as following: Section 2 reviews the related works. The general model for specifying software vulnerabilities is presented in Section 3. Based on this model, a general extendable vulnerability seeking algorithm is presented in Section 4. Section 5 presents the details of designing and implementing an extendable vulnerability detection method for executable codes. The implemented method is evaluated in Section 6. We conclude the paper and suggest some future works in Section 7.

2. Related works

One of the well-known vulnerability specification methods is called Vulnerability Cause Graph (VCG). A VCG is a directed non-cyclic graph that illustrates how and why a vulnerability appears in a program [22]. It has one leaf node that defines a specific vulnerability. The other nodes are *causes* that explain the conditions and events during the development process that make software vulnerable. Fig. 1 illustrates an example VCG. This graph explains how the vulnerability CVE-2003-0161 is created in Sendmail mail server. Although these graphs help the developers learn about different vulnerabilities, the narrative specification of causes prevents them to be automatically understandable. Thus, this specification method is not usable in an extendable vulnerability detection method.

Mallouli et al. specify vulnerabilities formally in [23] based on Vulnerability Detection Conditions (VDCs). A VDC characterizes