Monitoring Buffer Overflow Attacks:
A Perennial Task

Hossain Shahriar, Queen's University, Canada

Mohammad Zulkernine, Queen's University, Canada

ABSTRACT

Buffer overflow (BOF) is a well-known, and one of the worst and oldest, vulnerabilities in programs. BOF attacks overwrite data buffers and introduce wide ranges of attacks like execution of arbitrary injected code. Many approaches are applied to mitigate buffer overflow vulnerabilities; however, mitigating BOF vulnerabilities is a perennial task as these vulnerabilities elude the mitigation efforts and appear in the operational programs at run-time. Monitoring is a popular approach for detecting BOF attacks during program execution, and it can prevent or send warnings to take actions for avoiding the consequences of the exploitations. Currently, there is no detailed classification of the proposed monitoring approaches to understand their common characteristics, objectives, and limitations. In this paper, the authors classify runtime BOF attack monitoring and prevention approaches based on seven major characteristics. Finally, these approaches are compared for attack detection coverage based on a set of BOF attack types. The classification will enable researchers and practitioners to select an appropriate BOF monitoring approach or provide guidelines to build a new one.

Keywords: *Buffer Overflow, Monitor Security, Monitoring Objective, Overhead, Program State Utilization*

INTRODUCTION

A vulnerable program can be exploited at runtime by providing specially crafted inputs. Buffer overflow (BOF) is a well known and one of the worst and oldest vulnerabilities in programs (Aleph One, 1996). It allows attackers to overflow data buffers that might be exploited to execute arbitrary code. Several mitigation techniques are widely used to mitigate BOF vulnerabilities. These include static analysis (e.g., Hackett et al., 2006), testing (e.g., Xu et al., 2008), and fixing of vulnerable code (e.g., Dahn et al., 2003). However, BOF vulnerabilities are widely discovered in programs (e.g., CVE, 2010). Moreover, some BOF vulnerability exploitations (or attacks) might not appear until a program is operational. Thus, BOF attack detection is a perennial task.

Monitoring is a widely used technique that can detect BOF attacks at an early stage and mitigate some of the consequences at runtime. In a monitoring approach, vulnerability exploitation symptoms are checked by comparing the current state of a program with a known state under attack. When there is a

DOI: 10.4018/jsse.2010070102

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

match (or mismatch) between the two states, a successful exploitation of a particular vulnerability occurs. A program might be stopped for further execution. A monitor remains silent as long as a program is not under an attack at the cost of additional memories and execution time (e.g., Jones et al., 1997). Nevertheless, a program monitor is accurate in detecting attacks compared to other complementary mitigation techniques such as static analysis. This unique feature makes it a useful prevention mechanism in a deployed program.

Although many monitoring approaches have been introduced in the literature to detect the exploitations of BOF vulnerabilities (or attacks) (e.g., Berger et al., 2006; Chiueh et al., 2001), there is no classification to understand the common characteristics, objectives, and limitations of these approaches. Moreover, the lack of a comprehensive comparative study provides little or no direction on choosing the appropriate monitoring techniques for particular needs.

In this paper, we perform an extensive survey on the state of the art runtime monitoring approaches that detect BOF attacks[1]. We classify the monitoring approaches based on seven most common characteristics: *monitoring objective*, *program state utilization*, *implementation mechanism*, *environmental change*, *attack response, monitor security*, and *overhead*. Moreover, for each of the characteristics, we further classify the current work to identify fine grained features that might be present in BOF monitoring techniques. We then perform a comparative analysis of existing approaches for BOF attack detection coverage. We identify BOF attack types based on both vulnerable program code (operation, data type, overflow among object members, and pointer arithmetic) and runtime state (BOF location, BOF magnitude). The survey will help secure software developers, researchers, and practitioners to select a tool from the existing monitoring approaches by highlighting the BOF attack type detection capabilities. Moreover, it will provide a guideline to build a new monitoring technique based on their particular application needs.

This paper is organized as follows: the next section provides an overview of program monitor and BOF attack. Then we discuss the classification of the monitoring works followed by comparison of the works based on BOF attack types. We then review other similar efforts on comparing BOF attack monitoring approaches. Finally, we draw conclusions.

OVERVIEW

Program Monitor

In general, a program takes inputs, processes them with or without the help of runtime environment (e.g., API calls), and generates outputs as shown in Figure 1(a). A program monitor is deployed in a post-release stage. It provides an additional layer between a program and its execution environment (Figure 1(b)). A monitor passively checks runtime program states for the occurrence of attacks. The underlying assumption is that attack symptoms can be captured by program states. Program states are entities involved in program execution such as program memories containing modified (or unmodified) inputs and outputs, registers, opcode, and attribute of inputs (e.g., sizes). While executing, program states are captured at specific execution points (e.g., beginning of a function call, return from a function) and matched with known states under attacks. Any match (or mismatch) might indicate the occurrence of a successful attack. A monitor might have access to various elements of programs (e.g., source code) and environments (e.g., stack, code, data, inputs, APIs, processors).

Buffer Overflow

A buffer overflow (BOF) vulnerability allows overflowing a data buffer in a program. The overwriting might corrupt sensitive neighboring variables of the buffer such as the return address of a function or the stack frame pointer. A BOF can occur due to vulnerable ANSI C library function calls, lack of null characters at

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

21 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage:
www.igi-global.com/article/monitoring-buffer-overflow-attacks/46150

Related Content

The Practice of Participatory Enterprise Modelling: A Competency Perspective
Anne Persson (2008). *Information Systems Engineering: From Data Analysis to Process Networks (pp. 129-158).*
www.irma-international.org/chapter/practice-participatory-enterprise-modelling/23414/

Software and Systems Engineering: Conflict and Consensus
Rick Gibson (2003). *Practicing Software Engineering in the 21st Century (pp. 26-41).*
www.irma-international.org/chapter/software-systems-engineering/28108/

Evaluating the Consistency between Navigation and Data Models for Web Applications
Jordi Cabot and Cristina Gómez (2010). *International Journal of Information System Modeling and Design (pp. 68-85).*
www.irma-international.org/article/evaluating-consistency-between-navigation-data/45926/

Enforcing ASTD Access-Control Policies with WS-BPEL Processes in SOA Environments
Michel Embe Jiague, Marc Frappier, Frédéric Gervais, Régine Laleau and Richard St -Denis (2011). *International Journal of Systems and Service-Oriented Engineering (pp. 37-59).*
www.irma-international.org/article/enforcing-astd-access-control-policies/55122/

Access Control Specification in UML
Manuel Koch, Francesco Parisi-Presicce and Karl Pauls (2009). *Software Applications: Concepts, Methodologies, Tools, and Applications (pp. 2775-2794).*
www.irma-international.org/chapter/access-control-specification-uml/29534/