Computers and Electrical Engineering 38 (2012) 1768–1776

Contents lists available at SciVerse ScienceDirect

Computers and Electrical Engineering

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

Memory monitor module for embedded systems ☆

Zhenglin Liu *, Qidi Zhao, Dongfang Li, Xuecheng Zou

Department of Electronic Science and Technology, Huazhong University of Science and Technology, China

ARTICLE INFO

Article history:
Received 5 May 2011
Received in revised form 10 August 2012
Accepted 13 August 2012
Available online 8 September 2012

ABSTRACT

Embedded systems are vulnerable to buffer overflow attacks. In this paper, we propose a hardware memory monitor module that aims to detect buffer overflow attacks by analyzing the security of an embedded processor at the instruction level. The functionality of the memory monitor module does not rely on the source code and can perform security check through dynamic methods. Compared with several existing countermeasures that protect only part of the program's data space, our proposed memory monitor module can protect the program's entire data space. The proposed memory monitor module has negligible performance overhead because it runs in parallel with the embedded processor. As demonstrated in an FPGA (Field Programmable Gate Array) based prototype, the experimental results show that our memory monitor module can effectively resist several types of buffer overflow attacks with approximately a 15% hardware cost overhead and only a 0.1% performance penalty.

1. Introduction

The buffer overflow attack [1–3] has been considered one of the top software vulnerabilities in recent years. According to National Vulnerability Database [4], 414 and 393 buffer overflows vulnerabilities were published in 2010 and 2011, respectively. Buffer overflow attacks can cause serious damage to general purpose systems and embedded systems.

A buffer overflow occurs when a program writes data to a buffer, overruns the buffer's boundary and overwrites adjacent memory. According to the overflowed buffer's memory region, there are two main types of buffer overflows: stack and heap overflows. The stack overflow is the most exploited vulnerability because the return addresses of function calls are stored together with buffers on the stack and can be overwritten by overflowing a local buffer. When the function returns, it is redirected to the malicious code. The heap overflow is similar to the stack overflow except that there are no return addresses on the heap. For some widely used memory allocators, such as Doug Lea's *malloc* for the *glibc* library, altering memory management information to achieve arbitrary memory overwrites is a general way to exploit a heap overflow.

An attacker may use this effect to modify the return address to change the control flow of the hostile program. If the embedded processor itself has the ability to identify security threats and prevent attacks from buffer overflows, the security of the embedded system will be greatly enhanced. Control flow monitoring, information flow tracking and memory monitoring are the three main solutions to enhance the security at the hardware architecture level [5]. To target the buffer overflow attack, which is a special case violation of memory safety, this paper focuses on novel memory monitor designs. After analyzing the security mechanism at the instruction level, we propose a novel hardware monitor module to enhance the security of embedded processors.

☆ Reviews processed and approved for publication by Editor-in-Chief Dr. Manu Malek.

* Corresponding author. Address: No. 1037, Luoyu Road, Hongshan District, Wuhan, Hubei 430074, China. Tel.: +86 13396040059; fax: +86 02787611255.
E-mail addresses: liuzhenglin@mail.hust.edu.cn (Z. Liu), zqd_373330321@163.com (Q. Zhao), lidongfang0375@163.com (D. Li), estxczou@gmail.com (X. Zou).

0045-7906/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compeleceng.2012.08.005

The remainder of the paper is organized as follows. In Section 2, we briefly review related works and show our improvements. We present the design overview in Section 3. Section 4 describes the architecture design and hardware implementation. A case study of buffer overflow attacks is analyzed in Section 5. We discuss the evaluation results in Section 6. Section 7 concludes the work with some future works.

2. Related work

The Stackguard proposed by Cowan et al. [6] protects the return address but leaves the frame pointer unprotected, which can also be easily attacked [7]. Moreover, the effectiveness of the protection mechanism may also depend on whether the return address is covered by constructing a pointer [7]. In 2006, Shao and Sha proposed HSDdefender [8], which detects the stack buffer overflow by checking whether the target address is over the frame pointer. This method can protect the return address and frame pointer effectively, but this method may produce many false warnings caused by ignoring the legal operations of data exchanges between functions. Shetty et al. proposed HeapMon [9] to protect the data in the heap segment. It signs the memory location with a few state bits according to different operations and in the meantime a legal state change is defined. A buffer overflow attack can be found by following the state change of the memory location. However, this method has been found to be inefficient because the buffer bound information can be easily obtained through dynamic checking of the operational characteristics in the heap segment. Therefore, a buffer overflow attack can be detected through the bound checking method directly without defining the state of the memory location. In 2008, Devietti et al. proposed HardBound [10], which can completely resist buffer overflow attacks. A big limitation of HardBound is that the extraction of most buffer bound information relies on scanning the source code, which is difficult to obtain in most cases.

To summarize, Stackguard, HeapMon and HSDdefender protect either the stack segment or the heap segment, but none of them protect both of the stack and heap segments at the same time. Moreover, none of them protect the global data segment. In our memory monitor module design, the complexity and granularity of the defense strategy depend on how much bound information we can obtain. Compared with all existing works, our memory monitor module features three new improvements:

i. Based on the operation characteristics of different data spaces and the bound information obtained, we construct a corresponding monitoring strategy to protect the entire data space.
ii. The extraction of the bound information does not rely on the source code and can be obtained through dynamic methods in most cases.
iii. The proposed memory monitor module operates in parallel with the embedded processor, so it has negligible impact on system performance.

3. Defense policy

3.1. Program data space

The amount of buffer information obtained through dynamic monitoring determines the defense policy of the hardware security module. All buffers defined in the program are contained within the data space of the program. We should first understand the composition of the program data spaces and then construct the protection mechanism for each part of the data space according to the operation characteristics.

A typical program data space as shown in Fig. 1 includes a stack segment, a heap segment, a global data segment and a text segment. The local pointers, variables and local arrays defined in the program are stored in the stack segment. A buffer defined by a memory allocated function, such as *malloc*, is located in the heap segment. The global pointers, variables and global arrays defined in the program are saved in the global data segment. Buffer overflow attacks usually occur in these three segments, but the stack segment is the most popular attack target of the three. The program's binary executable code

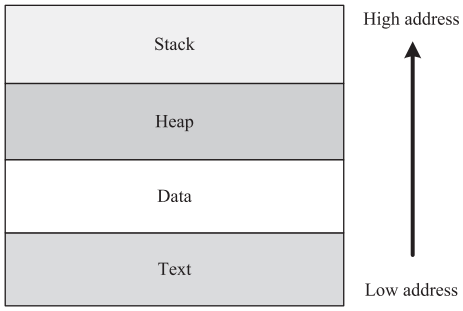

Fig. 1. Diagram of a typical program data space.