Chapter 11

Improving Memory Management Security for C and C++

Yves Younan
Katholieke Universiteit Leuven, Belgium

Wouter Joosen
Katholieke Universiteit Leuven, Belgium

Frank Piessens
Katholieke Universiteit Leuven, Belgium

Hans Van den Eynden
Katholieke Universiteit Leuven, Belgium

ABSTRACT

Memory managers are an important part of modern language and are used to dynamically allocate memory. Many managers exist; however, two major types can be identified: manual memory allocators and garbage collectors. In the case of manual memory allocators, the programmer must manually release memory back to the system when it is no longer needed. Problems can occur when a programmer forgets to release it, releases it twice, or uses freed memory. These problems are solved in garbage collectors. However, both manual memory allocators and garbage collectors store management information. This paper describes several vulnerabilities for C and C++ and how these could be remedied by modifying the management information of a representative manual memory allocator and garbage collector. Additionally, the authors present an approach that, when applied to memory managers, will protect against these attack vectors.

INTRODUCTION

Security has become an important concern for all computer users. Worms and hackers are a part of everyday internet life. A particularly dangerous attack is the code injection attack, where attackers are able to insert code into the program’s address space and can subsequently execute it. Programs written in C are particularly vulnerable to such attacks. Attackers can use a range of vulnerabilities to inject code. The most well known and most exploited is of course the standard buffer overflow: attackers write past the boundaries of a stack-based buffer and overwrite the return address of a func-
Improving Memory Management Security for C and C++

There are a number of vulnerabilities that occur frequently and as such have become a favorite for attackers to use to perform code injection. We will examine how different memory allocators might be misused by using one of three common vulnerabilities: “heap-based buffer overflows”, “off by one errors” and “dangling pointer references”. In this section we will describe what these vulnerabilities are and how they could lead to a code injection attack.

Heap-Based Buffer Overflow

Heap memory is dynamically allocated at runtime by the application. Buffer overflow, which are usually exploited on the stack, are also possible in this kind of memory. Exploitation of such heap-based buffer overflows usually relies on finding either function pointers or by performing an indirect pointer attack (Bulba & Kil3r, 2000) on data pointers in this memory area. However, these pointers are not always present in the data stored by the program in this memory. As such, most attackers overwrite the memory management information that the memory allocator stores in or around memory chunks it manages. By modifying this information, attackers can perform an indirect pointer overwrite. This allows attackers to overwrite arbitrary memory locations, which could lead to a code injection attack (anonymous, 2001; Younan, 2003). In the following sections we will describe how an attacker could use specific memory managers to perform this kind of attack.
Related Content

A Service Oriented SLA Management Framework for Grid Environments
[www.igi-global.com/article/service-oriented-sla-management-framework/47036?camid=4v1a](www.igi-global.com/article/service-oriented-sla-management-framework/47036?camid=4v1a)

New Optimal Solutions for Real-Time Reconfigurable Periodic Asynchronous OS Tasks with Minimizations of Response Times
[www.igi-global.com/chapter/new-optimal-solutions-real-time/76959?camid=4v1a](www.igi-global.com/chapter/new-optimal-solutions-real-time/76959?camid=4v1a)

Utilizing UML, cTLA, and SRN: An Application to Distributed System Performance Modeling
[www.igi-global.com/chapter/utilizing-uml-ctla-and-srn/135879?camid=4v1a](www.igi-global.com/chapter/utilizing-uml-ctla-and-srn/135879?camid=4v1a)

Evaluating the Consistency between Navigation and Data Models for Web Applications
[www.igi-global.com/article/evaluating-consistency-between-navigation-data/45926?camid=4v1a](www.igi-global.com/article/evaluating-consistency-between-navigation-data/45926?camid=4v1a)