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-

DOI: 10.4018/978-1-4666-1580-9.ch011

tion and point it to their injected code. When the function subsequently returns, the code injected by the attackers is executed (Aleph One, 1996).

These are not the only kind of code injection attacks though: a buffer overflow can also exist on the heap, allowing an attacker to overwrite heap-stored data. As pointers are not always available in normal heap-allocated memory, attackers often overwrite the management information that the memory manager relies upon to function correctly. A double free vulnerability, where a particular part of heap-allocated memory is de-allocated twice could also be used by an attacker to inject code.

Many countermeasures have been devised that try to prevent code injection attacks (Younan, Joosen, & Piessens, 2004). However most have focused on preventing stack-based buffer overflows and only few have concentrated on protecting the heap or memory allocators from attack.

In this paper we evaluate a commonly used memory allocator and a garbage collector for C and C++ with respect to their resilience against code injection attacks and present a significant improvement for memory managers in order to increase robustness against code injection attacks. Our prototype implementation (which we call *dnmalloc*) comes at a very modest cost in both performance and memory usage overhead.

This paper is an extended version of work described in (Younan, Joosen, & Piessens, 2006) which was presented in December 2006 at the Eighth International Conference on Information and Communication Security. The paper is structured as follows: section explains which vulnerabilities can exist for heap-allocated memory. Section describes how both a popular memory allocator and a garbage collector can be exploited by an attacker using one of the vulnerabilities of section to perform code injection attacks. Section describes our new more robust approach to handling the management information associated with chunks of memory. Section contains the results of tests in which we compare our memory allocator to the original allocator in terms of performance overhead and memory usage. In section related work in improving security for memory allocators is discussed. Finally, section discusses possible future enhancements and presents our conclusion.

HEAP-BASED VULNERABILITIES FOR CODE INJECTION ATTACKS

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.