IJIREEICE

ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

ISO 3297:2007 Certified

Vol. 5, Issue 3, March 2017

Review of Runtime Enforcement of Memory Safety in C Programming

Abhishek Bhosale[1], **Sankalp Biyani**[2], **Sushant Manjare**[3], **Er. Chandan Prasad**[4]

BCA Scholar, CTIS, School of Information & Technology, Pune, India[1, 2,3]

Assistant Professor, IT, Ajeenkya DY Patil University, Pune, India [4]

Abstract: The Unreliability of the C language has been in the forefront due to multiple memory access violations. Various methods have been devised to detect memory errors at runtime but are unable to detect all of them. These hurdles give rise to a various problems such as manual code modification, usage of metadata which is unsuited and high runtime outlays. This type of overwhelming conditions give rise to various compiler analysis tools which ensures the memory safety of C programs at runtime while avoiding any possible shortcomings and also help to lower the runtime cost of attaining memory safety.

Keywords: Memory Management, Memory Safety, Memory Violations, Memory Optimization.

I. INTRODUCTION

The C programming language is used fairly often despite its known memory malfunctions and the errors it generates. The main reason behind its persistent implementation is the features it provides such as low-level access to system memory which is the cause of multiple memory access violations, these indications go unnoticed which results in data corruption, buffer overflows and the much notorious dangling pointers.

Other Ideal languages such as java guarantee memory safety with multiple syntax limitations and real time inspections when security is concerned, another language known as rust provides maximum memory safety by providing functional and imperative-procedural paradigms. This dissertation provides the overview of the compilation process of the C programming language and also the low level features it gradually provides along the way, but this may be the prime cause of memory violations and also a major concern regarding the security of the C programming language.

II. LITERATURE SURVEY

C being one of the most popular languages of all time with multiple compilers supporting various architectures, it comes with its share of defects concerning with memory access violations.

Initially developed as a general purpose programming language, its main goal was to implement system software which was also a purpose to be the most widely used programming language in the distant future. It also enabled UNIX to be implemented in a high level language as assembly was system architecture specific. Due to C having low level access to the systems memory it is able to manipulate data specified in that memory.

A. Compilation

A Compiler is a program that transforms a program into a machine executable format. It is responsible for various tasks such as semantic analysis of source code and the generation and optimization of the machine code. At first the source code is pre-processed by the pre-processor then the compiler generate an assembly equivalent of that code then an object file is created and with the help of linker it is linked to its executable format and the program is executed in the form of process in the process address space.

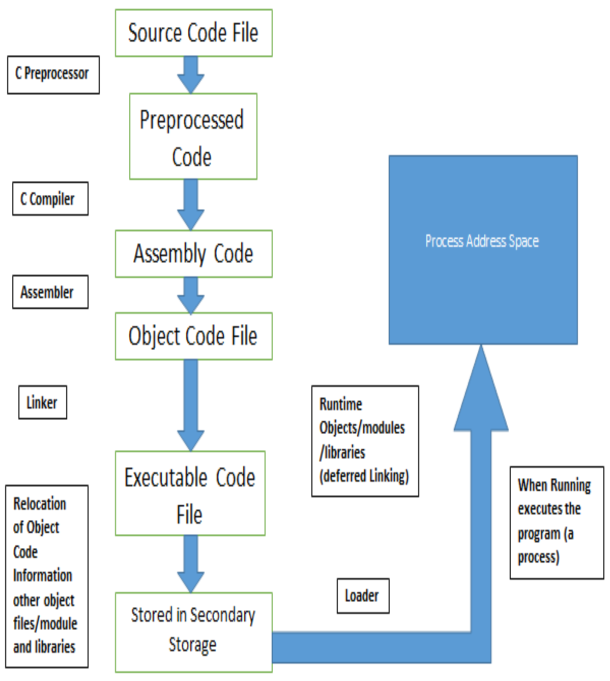
Source Code File
C Preprocessor
Preprocessed Code
C Compiler
Assembly Code
Assembler
Object Code File
Linker
Executable Code File
Relocation of Object Code Information other object files/module and libraries
Stored in Secondary Storage
Runtime Objects/modules /libraries (deferred Linking)
Loader
Process Address Space
When Running executes the program (a process)

Fig 1. Flow of Compilation

IJIREEICE

ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

ISO 3297:2007 Certified

Vol. 5, Issue 3, March 2017

In a compiler the tasks are arranged into three groups:

- Front End
- Middle End
- Back End

The main task [1] of the front end of the compiler is to develop an internal target-independent representation of the program, called the intermediate representation (IR), to be used by the middle-end. The lexical analysis of the source code is done by the front end in order to recognize the keywords, identifiers and symbols. The middle end is responsible for the semantic analysis of the source code and the backend transforms the IR into a machine executable representation.

B. Memory Security Desecrations.

Pointers enable to indirectly access [1] the data stored in system memory; they can also implement the call-back mechanism which is a prerequisite for event handlers. A function’s local storage may be reallocated for the execution of another function, reading or writing the address of a previously allocated variable results in a temporal violation. For example a scenario in which the pointer is trying to deallocate an object which has already been deallocated. The dereferencing of dangling heap pointers causes the memory [2] to be deallocated by the free function instead of deallocating it automatically during the function termination. Various attempts by the program are made to deallocate the same object twice that was originally not allocated by malloc () itself, causes a temporal safety violation. Spatial memory errors involve the use of an out of bounds pointer or array index.

These memory violations can be divided into two types (1) Spatial Safety Violation and (2) Temporal Safety Violation, the first condition occurs when you use a pointer [1] to access a particular data at a memory location which is outside the limits of an allocated object. The second condition occurs when you use a pointer [1] to deallocate an object that has already been deallocated, which means the pointer is being used at an invalid situation in the program. Various other situations include stack exhaustion in which the program runs out stack space mainly because of its deep recursive content. All of these conditions are overlooked due to the fact of fast execution as [3] execution of memory safe languages requires time as they monitor each segment of code and the syntax restrictions posed to them. Due to this Dilemma security is the one which is always compromised as people think having more speed is convenient.

III.CONCLUSION

In this paper we described that C can be made a memory safe language by following a restricted usage of certain functions and with simple optimizations of some compiler analysis tools. The goal of this work was to ease the memory safety problems of C with solutions that can be used in the real world today. Several additional compile-time optimizations are possible, aiming at immediate usability and impact, it may be possible to apply these techniques to other languages or even hardware-based solutions.

REFERENCES

[1] Runtime Enforcement of Memory Safety for the C Programming Language by Matthew Stephen Simpson and Professor Rajeev Barua.

[2] Lecture Notes on Language-Based Security Erik Poll Radboud University Nijmegen January 21, 2016.

[3] Practical memory safety for C University of Cambridge Computer Laboratory Periklis Akritidis June 2011.