A Backwards-Compatible Technique for Detection of All Memory Errors in C Programs

Wei Xu, R. Sekar, and Daniel C. DuVarney
Department of Computer Sciences, Stony Brook University
Stony Brook, NY 11794-4400
{weixu,sekar,dand}@cs.sunysb.edu

The C programming language is commonly used for systems programming because of its speed and the precise control it provides over memory allocation. Unfortunately, this control is more than most programmers can fully manage, and as a result, memory-related errors remain one of the principal causes of failure in C programs and security vulnerabilities in software.

Memory errors can be broadly classified into *spatial errors*, in which a pointer dereference occurs while the reference points to an address outside the bounds of the referent (e.g., *array bounds errors*), and *temporal errors*, in which the referent no longer exists (i.e., the storage for the referent has been recycled) (e.g., *dangling pointers dereferences*).

A number of recent research efforts have focused on the problem of detection/elimination of memory errors in C programs. All these approaches suffer from one or more of the following problems: the inability to detect all memory errors (e.g., Purify), requiring modifications to existing C programs (e.g., Cyclone), changing the memory management model of C to use garbage collection (e.g., CCured), and excessive performance overheads (e.g., Bcc).

In this paper, we present a new approach that does not suffer from these problems. Our approach operates via source code transformation and combines efficient data-structures with simple, localized optimizations to obtain low performance overheads.

Although our technique is capable of dealing with all C programs, our current implementation restricts pointer casts in certain ways, e.g., casts between pointers to different, unrelated structure types are not supported. (But casts among pointers to structures that can be viewed as subtypes/supertypes are supported). Most C programs seem to follow this restriction, as evidenced by the fact that our prototype can successfully compile many programs ranging from hundreds to tens of thousands of lines of code, and do so with requiring almost no modifications to these programs.

For programs which obey the casting restrictions, our approach can promptly detect all spatial and temporal errors, without changing the memory allocation model. This has some important benefits over approaches which rely on garbage collection to ensure temporal memory safety. Garbage collectors for C are usually non-incremental and conservative, which may make the runtime overhead of a program unpredictable and result in memory leaks. Also, prompt detection of temporal errors allows for the program to be corrected.

We present experimental results on Olden benchmarks and SPECINT benchmarks showing that the performance overhead of our approach improves over previous techniques for handling temporal errors by at least a factor of two.