
Preface

A buffer overflow occurs when input is written into a memory buffer that is not large enough to hold the input. Buffer overflows may allow a malicious person to gain control over a computer system in that a crafted input can trick the defective program into executing code that is encoded in the input itself. They are recognised as one of the most widespread forms of security vulnerability, and many workarounds, including new processor features, have been proposed to contain the threat. This book describes a static analysis that aims to prove the absence of buffer overflows in C programs. The analysis is conservative in the sense that it locates every possible overflow. Furthermore, it is fully automatic in that it requires no user annotations in the input program.

The key idea of the analysis is to infer a symbolic state for each program point that describes the possible variable valuations that can arise at that point. The program is correct if the inferred values for array indices and pointer offsets lie within the bounds of the accessed buffer. The symbolic state consists of a finite set of linear inequalities whose feasible points induce a convex polyhedron that represents an approximation to possible variable valuations. The book formally describes how program operations are mapped to operations on polyhedra and details how to limit the analysis to those portions of structures and arrays that are relevant for verification. With respect to operations on string buffers, we demonstrate how to analyse C strings whose length is determined by a NUL character within the string.

We complement the analysis with a novel sub-class of general polyhedra that admits at most two variables in each inequality while allowing arbitrary coefficients. By providing polynomial algorithms for all operations necessary for program analysis, this sub-class of general polyhedra provides an efficient basis for the proposed static analysis. The polyhedral sub-domain presented is then refined to contain only integral states, which provides the basis for the combination of numeric analysis and points-to analysis. We also present a novel extrapolation technique that automatically inspects likely bounds on variables, thereby providing a way to infer precise loop invariants.

Target Audience

The material in this book is based on the author's doctoral thesis. As such it focusses on a single topic, namely the definition of a sound value-range analysis for C programs that is precise enough to verify non-trivial string buffer operations. Furthermore, it only applies one approach to pursue this goal, namely a fixpoint computation using convex polyhedra that approximate the state space of the program. Hence, it does not provide an overview of various static analysis methods but an in-depth treatment of a real-world analysis task. It should therefore be an interesting and motivating read, augmenting, say, a course on program analysis or formal methods.

The merit of this book lies in the formal definition of the analysis as well as the insight gained on particular aspects of analysing a real-world programming language. Most research papers that describe analyses of C programs lack a formal definition. Most work that is formal defines an analysis for toy languages, so it remains unclear if and how the concepts carry over to real languages. This book closes this gap by giving a formal definition of an analysis that handles full C. However, this book is more than an exercise in formalising a large static analysis. It addresses many facets of C that interact and that cannot be treated separately, ranging from the endianness of the machine, alignment of variables, overlapping accesses to memory, casts, and wrapping, to pointer arithmetic and mixing pointers with values.

As a result, the work presented is of interest not only to researchers and implementers of sound static analyses of C but to anyone who works in program analysis, transformation, semantics, or even run-time verification. Thus, even if the task at hand is not a polyhedral analysis, the first chapters, on the semantics of C, can save the reinvention of the wheel, whereas the latter chapters can serve in finding analogous solutions using the analysis techniques of choice. For researchers in static analysis, the book can serve as a basis to implement new abstraction ideas such as shape analyses that are combined with numeric analysis. In this context, it is also worth noting that the abstraction framework in this book shows which issues are solvable and which issues pose difficult research questions. This information is particularly valuable to researchers who are new to the field (e.g., Ph.D. students) and who therefore lack the intuition as to what constitutes a good research question.

Some techniques in this book are also applicable to languages that lack the full expressiveness of C. For instance, the Java language lacks pointer arithmetic, but the techniques to handle casting and wrapping are still applicable. At the other extreme, the analysis presented could be adapted to analyse raw machine code, which has many practical advantages.

The book presents a sound analysis; that is, an analysis that never misses a mistake. Since this ambition is likely to be jeopardised by human nature, we urge you to report any errors, omissions, and any other comments to us. To this end, we have set up a Website at <http://www.bufferoverflows.org>.

<http://www.springer.com/978-1-84800-016-2>

Value-Range Analysis of C Programs
Towards Proving the Absence of Buffer Overflow
Vulnerabilities

Simon, A.

2008, XXII, 302 p. 119 illus., Hardcover

ISBN: 978-1-84800-016-2