Value-Range Analysis of C Programs

Axel Simon

Value-Range Analysis of C Programs

Towards Proving the Absence of Buffer Overflow Vulnerabilities

Springer

Axel Simon

ISBN: 978-1-84800-016-2 e-ISBN: 978-1-84800-017-9
DOI: 10.1007/978-1-84800-017-9

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2008930099

© Springer-Verlag London Limited 2008
Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licenses issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms should be sent to the publishers.
The use of registered names, trademarks, etc., in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant laws and regulations and therefore free for general use.
The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Printed on acid-free paper

Springer Science+Business Media
springer.com

To my parents.

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`.

Acknowledgments

First and foremost, I would like to thank Andy King, who has become much more to me than a Ph.D. supervisor during these last years. He not only chose an interesting topic but also supported me with all his expertise and encouragement in a way that went far beyond his duties. Furthermore, my many friends at the Computing Laboratory at the University of Kent – who are too numerous to list here – deserve more credit than they might realise. I wish to thank them for their support and their ability to take my mind off work. My special thanks go to Paula Vaisey for her undivided support during the last months of preparing the manuscript, especially after I moved to Paris. I would also like to thank Carrie Jadud for her diligent proofreading.

Paris, *Axel Simon*
May 2008

Contents

Preface vii

Contributions xvii

List of Figures xix

1 Introduction 1
- 1.1 Technical Background 2
- 1.2 Value-Range Analysis 4
- 1.3 Analysing C 6
- 1.4 Soundness 7
  - 1.4.1 An Abstraction of C 7
  - 1.4.2 Combining Value and Content Abstraction 8
  - 1.4.3 Combining Pointer and Value-Range Analysis 9
- 1.5 Efficiency 11
- 1.6 Completeness 15
  - 1.6.1 Analysing String Buffers 16
  - 1.6.2 Widening with Landmarks 16
  - 1.6.3 Refining Points-to Analysis 17
  - 1.6.4 Further Refinements 17
- 1.7 Related Tools 18
  - 1.7.1 The Astrée Analyser 18
  - 1.7.2 SLAM and ESPX 19
  - 1.7.3 CCured 20
  - 1.7.4 Other Approaches 20

2 A Semantics for C 23
- 2.1 Core C 23
- 2.2 Preliminaries 28
- 2.3 The Environment 28
- 2.4 Concrete Semantics 32

2.5 Collecting Semantics 37
2.6 Related Work 42

Part I Abstracting Soundly

3 Abstract State Space 47
3.1 An Introductory Example 48
3.2 Points-to Analysis 51
3.2.1 The Points-to Abstract Domain 54
3.2.2 Related Work 55
3.3 Numeric Domains 56
3.3.1 The Domain of Convex Polyhedra 56
3.3.2 Operations on Polyhedra 59
3.3.3 Multiplicity Domain 62
3.3.4 Combining the Polyhedral and Multiplicity Domains 65
3.3.5 Related Work 68

4 Taming Casting and Wrapping 71
4.1 Modelling the Wrapping of Integers 72
4.2 A Language Featuring Finite Integer Arithmetic 74
4.2.1 The Syntax of Sub C 74
4.2.2 The Semantics of Sub C 75
4.3 Polyhedral Analysis of Finite Integers 76
4.4 Implicit Wrapping of Polyhedral Variables 77
4.5 Explicit Wrapping of Polyhedral Variables 78
4.5.1 Wrapping Variables with a Finite Range 78
4.5.2 Wrapping Variables with Infinite Ranges 80
4.5.3 Wrapping Several Variables 80
4.5.4 An Algorithm for Explicit Wrapping 82
4.6 An Abstract Semantics for Sub C 83
4.7 Discussion 86
4.7.1 Related Work 87

5 Overlapping Memory Accesses and Pointers 89
5.1 Memory as a Set of Fields 89
5.1.1 Memory Layout for Core C 90
5.2 Access Trees 93
5.2.1 Related Work 99
5.3 Mixing Values and Pointers 100
5.4 Abstraction Relation 106
5.4.1 On Choosing an Abstraction Framework 108

6 Abstract Semantics 111
6.1 Expressions and Simple Assignments 116
6.2 Assigning Structures 118
6.3 Casting, &-Operations, and Dynamic Memory 121
6.4 Inferring Fields Automatically 123

Part II Ensuring Efficiency

7 Planar Polyhedra 127
7.1 Operations on Inequalities 129
7.1.1 Entailment between Single Inequalities 130
7.2 Operations on Sets of Inequalities 131
7.2.1 Entailment Check 131
7.2.2 Removing Redundancies 132
7.2.3 Convex Hull 134
7.2.4 Linear Programming and Planar Polyhedra 144
7.2.5 Widening Planar Polyhedra 145

8 The TVPI Abstract Domain 147
8.1 Principles of the TVPI Domain 148
8.1.1 Entailment Check 150
8.1.2 Convex Hull 150
8.1.3 Projection 151
8.2 Reduced Product between Bounds and Inequalities 152
8.2.1 Redundancy Removal in the Reduced Product 155
8.2.2 Incremental Closure 156
8.2.3 Approximating General Inequalities 160
8.2.4 Linear Programming in the TVPI Domain 160
8.2.5 Widening of TVPI Polyhedra 161
8.3 Related Work 163

9 The Integral TVPI Domain 165
9.1 The Merit of $\mathbb{Z}$-Polyhedra 166
9.1.1 Improving Precision 166
9.1.2 Limiting the Growth of Coefficients 167
9.2 Harvey's Integral Hull Algorithm 168
9.2.1 Calculating Cuts between Two Inequalities 169
9.2.2 Integer Hull in the Reduced Product Domain 172
9.3 Planar $\mathbb{Z}$-Polyhedra and Closure 177
9.3.1 Possible Implementations of a $\mathbb{Z}$-TVPI Domain 177
9.3.2 Tightening Bounds across Projections 179
9.3.3 Discussion and Implementation 180
9.4 Related Work 182

10 Interfacing Analysis and Numeric Domain . . . 185
10.1 Separating Interval from Relational Information . . . 185
10.2 Inferring Relevant Fields and Addresses . . . 187
10.2.1 Typed Abstract Variables . . . 189
10.2.2 Populating the Field Map . . . 190
10.3 Applying Widening in Fixpoint Calculations . . . 192

Part III Improving Precision

11 Tracking String Lengths . . . 197
11.1 Manipulating Implicitly Terminated Strings . . . 198
11.1.1 Analysing the String Loop . . . 199
11.1.2 Calculating a Fixpoint of the Loop . . . 203
11.1.3 Prerequisites for String Buffer Analysis . . . 209
11.2 Incorporating String Buffer Analysis . . . 209
11.2.1 Extending the Abstraction Relation . . . 212
11.3 Related Work . . . 213

12 Widening with Landmarks . . . 217
12.1 An Introduction to Widening/Narrowing . . . 217
12.1.1 The Limitations of Narrowing . . . 218
12.1.2 Improving Widening and Removing Narrowing . . . 220
12.2 Revisiting the Analysis of String Buffers . . . 220
12.2.1 Applying the Widening/Narrowing Approach . . . 222
12.2.2 The Rationale behind Landmarks . . . 222
12.2.3 Creating Landmarks for Widening . . . 225
12.2.4 Using Landmarks in Widening . . . 225
12.3 Acquiring Landmarks . . . 226
12.4 Using Landmarks at a Widening Point . . . 227
12.5 Extrapolation Operator for Polyhedra . . . 229
12.6 Related Work . . . 231

13 Combining Points-to and Numeric Analyses . . . 235
13.1 Boolean Flags in the Numeric Domain . . . 237
13.1.1 Boolean Flags and Unbounded Polyhedra . . . 238
13.1.2 Integrality of the Solution Space . . . 239
13.1.3 Applications of Boolean Flags . . . 240
13.2 Incorporating Boolean Flags into Points-to Sets . . . 241
13.2.1 Revising Access Trees and Access Functions . . . 241
13.2.2 The Semantics of Expressions and Assignments . . . 244
13.2.3 Conditionals and Points-to Flags . . . 246
13.2.4 Incorporating Boolean Flags into the Abstraction Relation . . . 249

13.3 Practical Implementation . . . 250
13.3.1 Inferring Points-to Flags on Demand . . . 251
13.3.2 Populating the Address Map on Demand . . . 251
13.3.3 Index-Sensitive Memory Access Functions . . . 253
13.3.4 Related Work . . . 255

14 Implementation . . . 259
14.1 Technical Overview of the Analyser . . . 260
14.2 Managing Abstract Domains . . . 262
14.3 Calculating Fixpoints . . . 264
14.3.1 Scheduling of Code without Loops . . . 265
14.3.2 Scheduling in the Presence of Loops and Function Calls . . . 267
14.3.3 Deriving an Iteration Strategy from Topology . . . 268
14.3.4 Related Work . . . 269
14.4 Limitations of the String Buffer Analysis . . . 271
14.4.1 Weaknesses of Tracking First NUL Positions . . . 271
14.4.2 Handling Symbolic NUL Positions . . . 272
14.5 Proposed Future Refinements . . . 276

15 Conclusion and Outlook . . . 277

A Core C Example . . . 281

References . . . 285

Index . . . 297

Contributions

This section summarises the novelties presented in this book. Some of these contributions have already been published in refereed forums, such as our work on the principles of tracking NUL positions by observing pointer operations [167], the ideas behind the TVPI domain [172], a convex hull algorithm for planar polyhedra [168], the idea of widening with landmarks [170], the idea of an abstraction map that implicitly handles wrapping [171], and the use of Boolean flags to refine points-to analysis [166]. Overall, this book makes the following contributions to the field of static analysis:

1. Chapter 2: Defining the Core C intermediate language, which is concise yet able to express all operations of C.
2. Chapter 3: The observation of improved precision when implementing congruence analysis as a reduced product with $\mathbb{Z}$-polyhedra.
3. Chapters 4–6: A sound abstraction of C; in particular:
   a) Sound treatment of the wrapping behaviour of integer variables.
   b) Automatic inference of fields in structures that are relevant to the analysis. In particular, fields on which no information can be inferred are not tracked by the polyhedral domain and therefore incur no cost.
   c) Combining flow-sensitive points-to analysis with a polyhedral analysis of pointer offsets.
   d) Sound and precise approximation of pointer accesses when the pointer may have a range of offsets using access trees.
   e) A concise definition of an abstraction map between concrete and abstract semantics.
4. Chapter 7 presents a complete set of domain operations for planar polyhedra; in particular, a novel convex hull algorithm [168].
5. Chapter 8 presents the two-variables-per-inequality (TVPI) domain [172].
6. Chapter 9 describes how integral tightening techniques can be applied in the context of the TVPI domain.

7. Chapter 10 discusses techniques for adding polyhedral variables on-the-fly. Specifically, this chapter introduces the notion of typed polyhedral variables.
8. Chapter 11 details string buffer manipulation through pointers. The techniques presented in this book are a substantial refinement of [167].
9. Chapter 12 presents widening with landmarks [170], a novel extrapolation technique for polyhedra.
10. Chapter 13 discusses techniques for analysing a path of the program several times using a single polyhedron [166]. It uses the techniques developed to define a very precise points-to analysis.

The most important contribution of this book is a formal definition of a static analysis of a real-world programming language that is reasonably concise and – we hope – simple enough to be easily understood by other researchers in the field. We believe that the static analysis presented in this book will be useful as a basis for similar analyses and related projects.

List of Figures

1.1 View of the Stack . . . 3
1.2 Counting Characters . . . 5
1.3 Incompatible Points-to Information . . . 10
1.4 Control-Flow Graphs . . . 13
1.5 State Spaces in the **`for`** Loop . . . 14

2.1 Syntactic Categories . . . 26
2.2 Core C Syntax . . . 26
2.3a Concrete Semantics of Core C . . . 34
2.3b Concrete Semantics of Core C . . . 35
2.4 Other Primitives of C . . . 37
2.5 Echo Program . . . 39

3.1 Points-to and Numeric Analysis . . . 48
3.2 Flow Graph of Strings Printer . . . 49
3.3 Simple Fixpoint Calculation . . . 50
3.4 Tracking `NULL` Values . . . 52
3.5 Flow-Sensitive vs. Flow-Insensitive Analysis . . . 53
3.6 $\mathbb{Z}$-Polyhedra are not Closed Under Intersection . . . 60
3.7 Right Shifting by 2 Bits . . . 61
3.8 Core C Example of Array Access . . . 62
3.9 Updating Multiplicity . . . 64
3.10 Reducing Two Domains . . . 66
3.11 Topological Closure . . . 68

4.1a The Initial Code . . . 73
4.1b Removing the Compiler Warning . . . 73
4.1c Observing that **`char`** May By Signed . . . 73
4.2 Concrete semantics of Sub C . . . 75
4.3 Signedness and Wrapping . . . 76
4.4 Wrapping in Bounded State Spaces . . . 79

4.5 Wrapping in Unbounded State Spaces . . . 80
4.6 Wrapping of Two Variables . . . 81
4.7 Abstract Semantics of Sub C . . . 84
4.8 Merging Wrapped Variables . . . 86

5.1 Overlapping Write Accesses . . . 92
5.2 Read Operations on Access Trees . . . 95
5.3 Write Operations on Access Trees . . . 98
5.4 Modifying l-Values and Their Offsets . . . 102
5.5 Abstract Memory Read . . . 103
5.6 Abstract Memory Write . . . 105

6.1 Abstract Semantics: Basic Blocks . . . 113
6.2 Abstract Semantics: Expressions and Assignments . . . 117
6.3 Functions on Memory Regions . . . 119
6.4 Abstract Semantics: Assignments of Structures . . . 120
6.5 Abstract Semantics: Miscellaneous . . . 122

7.1 Classic Convex Hull Calculation in 2D . . . 127
7.2 Classic Convex Hull Calculation in 3D . . . 128
7.3 Measuring Angles . . . 129
7.4 Planar Entailment Check Idea . . . 131
7.5 Redundant Chain of Inequalities . . . 133
7.6a Calculating a Containing Square . . . 138
7.6b Translating Vertices . . . 138
7.6c Calculating the Convex Hull . . . 139
7.6d Creating Inequalities . . . 139
7.7a Creating a Vertex for Lines . . . 141
7.7b Checking Points . . . 141
7.8 Convex Hull of One-Dimensional Output . . . 142
7.9 Creating a Ray . . . 143
7.10 Pitfalls in Graham Scan . . . 144
7.11 Linear Programming and Planar Polyhedra . . . 145
7.12 Widening of Planar Polyhedra . . . 146

8.1 Approximating General Polyhedra . . . 148
8.2 Representation of TVPI . . . 153
8.3 Removal of a Variable . . . 154
8.4 Entailment Check for Intervals . . . 155
8.5 Tightening Interval Bounds . . . 156
8.6 Incremental Closure for TVPI Systems . . . 157
8.7 Polyhedra with Several Representations . . . 162

9.1 Cutting Plane Method . . . 166
9.2 Precision of $\mathbb{Z}$-Polyhedra . . . 167
9.3 Calculating Cuts . . . 169

9.4 Transformed Space 170
9.5 Tightening Interval Bounds 172
9.6 Calculating Cuts for Tightening Bounds 174
9.7 Redundancies Due to Cuts 176
9.8 Closure for $\mathbb{Z}$-Polyhedra 178
9.9 Tightening in the TVPI Domain 180
9.10 Redundant Inequality in Reduced Product 181

10.1 Separating Ranges and TVPI Variables 186
10.2 Allocating Memory in a Loop 188
10.3 Populating the Fields Map 190
10.4 Closure and Widening 193

11.1 Abstract Semantics for String Buffers 199
11.2 Core C of String Copy 200
11.3 Control-Flow Graph of the String Loop 201
11.4 Fixpoint of the String Loop 205
11.5 Joins in the Fixpoint Computation 207
11.6 String-Aware Memory Accesses 210
11.7 String-Aware Access to Memory Regions 211

12.1 Jacobi Iterations on a **`for`**-Loop 218
12.2 Unfavourable Widening Point 219
12.3 Imprecise State Space for the String Example 222
12.4 Applying Widening to the String Example 223
12.5 Precise State Space for the String Example 224
12.6 Fixpoint Using Landmarks 224
12.7 Landmark Strategy 227
12.8 Non-linear Growth 230
12.9 Standard vs. Revised Widening 231
12.10 Widening from Polytopes 232

13.1 Precision Loss for Non-trivial Points-to Sets 236
13.2 Boolean functions in the Numeric Domain 237
13.3 Control-Flow Splitting 238
13.4 Distinguishing Unbounded Polyhedra 239
13.5 Modifying l-Values 242
13.6 Abstract Memory Accesses 243
13.7 Semantics of Expressions and Assignments 245
13.8 Semantics of Conditionals 247
13.9 Accessing a Table of Constants 253
13.10 Precision of Incorporating the Access Position 255

14.1 Structure of the Analysis 261
14.2 Adding Redundant Constraints 263
14.3 Iteration Strategy for Conditionals 265

14.4 Iteration Strategy Loops 267
14.5 Deriving SCCs from a CFG 268
14.6 CFG of Example on Symbolic NUL Positions 272
14.7 Limitations of the TVPI Domain 273