Available online at www.sciencedirect.com

ScienceDirect

Information and Software Technology 50 (2008) 449–461

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

An approach for the maintenance of input validation

Hui Liu *, Hee Beng Kuan Tan

School of Electrical and Electronic Engineering, Block S2, Nanyang Technological University, Nanyang Avenue, Singapore 639798, Singapore

Received 29 November 2006; received in revised form 14 May 2007; accepted 15 May 2007
Available online 31 May 2007

Abstract

Input validation is the enforcement of constraints that an input must satisfy before it is accepted in a program. It is an essential and important feature in a large class of systems and usually forms a major part of a data-intensive system. Currently, the design and implementation of input validation are carried out by application developers. The recovery and maintenance of input validation implemented in a system is a challenging issue. In this paper, we introduce a variant of control flow graph, called validation flow graph as a model to analyze input validation implemented in a program. We have also discovered some empirical properties that characterizing the implementation of input validation. Based on the model and the properties discovered, we then propose a method that recovers the input validation model from source and use program slicing techniques to aid the understanding and maintenance of input validation. We have also evaluated the proposed method through case studies. The results show that the method can be very useful and effective for both experienced and inexperienced developers.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Software maintenance; Input validation; Empirical properties; Program slicing; Feature recovery

1. Introduction

A large class of systems process inputs submitted from users. Information systems and database applications belong to this class. In such systems, an important component is to properly handle both expected and unexpected inputs, so that only valid inputs are accepted to raise external effects, while invalid inputs are rejected and no external effects are raised. The enforcement in the systems is called **input validation**, and we refer it in this paper as the *input validation feature*. For example, in an order processing system, the outstanding balance of a customer must not exceed the credit limit. A database transaction is taken place only if the new order placed by the customer does not lead to the exceeding of the limit; hence, input validation is incorporated into the system to enforce that only a valid new order is accepted for the transaction to occur.

Input validation plays a key role in the control and accuracy of user inputs submitted to a system. It is vital for the robustness of the system. It is also one of the strongest measures of defense against today's application attacks [23]. Design and implementation of input validation is a challenging issue, and currently, the primary burden falls on application developers. Furthermore, many systems that depend on user inputs have been developed and need to be maintained for years. During the course of the time, as the requirements change and functionalities evolve, both the documents and the program source code become more complex and increasingly difficult to understand and maintain. Many of the documents become out of date and many systems become largely undocumented [9]. This creates problems in the maintenance of the input validation feature implemented in those systems. When an update of an input validation feature implemented in a system occurs, the developers need to understand how the feature was implemented before they can initiate the update. However, in many cases, the system documents are not trustable because the developers are not sure that they have made

* Corresponding author.
E-mail addresses: liuh0007@ntu.edu.sg (H. Liu). ibktan@ntu.edu.sg (H.B.K. Tan).

all the updates. On the other hand, the developers are not confident to manually recover the input validation feature from the code either. Therefore, there is a need for an assisted approach to recover the input validation feature from program source code and also aid the maintenance of the feature.

This paper is an enhancement and extension of our previous work [19]. In the previous work, we introduce a variant of control flow graph, called validation flow graph (VFG) for analyzing input validation implemented in a program. Based on that, we then propose a method for the recovery of input validation from source code and also briefly introduce the techniques of VFG-guided slicing and effect-oriented decomposition slicing that can be used to aid the maintenance of input validation. In this paper, the model for analyzing input validation is further improved by refining the definitions and incorporating empirical properties that characterize the implementation of input validation. All the properties have been statistically validated. In addition, an algorithm is presented for the recovery of input validation from source code. We also propose several guidelines to elaborate the use of VFG-guided slicing in the maintenance of input validation. The case study is further enhanced with more details. The major contributions of the paper include the followings:

(1) The introduction of the validation flow graph for modeling input validation.
(2) The discovery of empirical properties that characterize the implementation of input validation through empirical study and hypothesis testing.
(3) An automated method for the recovery of input validation from source code.
(4) The techniques of VFG-guide slicing and effect-oriented decomposition slicing for aiding the understanding and maintenance of input validation.

The paper is organized as follows. Section 2 introduces a model and some empirical properties for characterizing input validation. Section 3 discusses the proposed approach. Section 5 compares our work with related work. Finally, Section 6 concludes the paper.

2. Characterizing input validation

2.1. Analysis model

The analysis of the input validation feature implemented in a program is based on the control flow graph of the program. We shall adopt the formalism of control flow graph [27]. A **control flow graph** (**CFG**) of a program P is a directed graph $G = (N, E)$, in which N contains a set of nodes and $E = \{(n,m) \mid n, m \in N\}$ contains edges that connecting the nodes. Each node in G represents a statement in P, and each edge in the G represents possible flow of control between two statements in P. There is an entry node and an exit node in a CFG representing entry to and exit from P, respectively.

In a control flow graph, the data dependence represents the data flow between program statements. A node y is **data dependent** on a node x if and only if there exists a variable v such that x defines v, y uses v and there is a path from x to y along with which v is not redefined. The control dependence is defined between conditional statement and the statements whose executions are controlled by the conditional statement. A node y **dominates** a node x if and only if every path from the entry node to x contains y. A node y **postdominates** a node x if and only if every path from x to the exit node contains y. A node y is **control dependent** on a node x if and only if x has successors x' and x'' such that y postdominates x' but y does not postdominate x''.

Let G be the CFG of a program. A node in G is called an **input node** if it reads user inputs. A node in G is called an **effect node** if it raises external effects. An external effect represents an output state or action that a program interacts with its external environment. For example, in a database application, once the input data is submitted by a user, a program is executed to process the data submitted and update system database. In the control flow graph of such program, a node that read the data submitted is an input node, and a node that updates the database maintained is an effect node.

Next, we shall introduce a variant of control flow graph, called validation flow graph (VFG), which provides essential information on the input validation feature implemented in a program. The VFG of a program is constructed from a set of nodes selected from the original CFG of the program. An Edge is added to the VFG if two nodes are connected in the original CFG via nodes that are not in the VFG. For a program P that reads user inputs and raises external effects, the VFG of P is constructed by identifying the input nodes, effect nodes, and predicate nodes that control the execution of the input nodes and effect nodes. The formal definition of validation flow graph is given below:

Let $G = (N, E)$ be the CFG of a program P where N and E are its set of nodes and edges, respectively. Let N' be the subset of nodes in G that includes the following types of nodes:

(1) Entry node and Exit node.
(2) All the input nodes.
(3) All the effect nodes.
(4) Node n in G such that a node in N' is control dependent on n.

Let E' be the set of edges that connects the nodes in N'. For each unordered pair of nodes (n,m) in N', if there is a path in G from n to m without passing through another node in N', an edge (n,m) in is included in E'. Formally, we define $E' = \{(n,m) \mid n, m \in N', n \neq m$ and there exists a path $(n,n_1,\ldots,n_k,m)$ in G such that $k \geqslant 0$ for all j, $1 \leqslant j \leqslant k$, $n_j \notin N'\}$.