

Enforcing Security and Safety Models with an Information Flow Analysis Tool

Roderick Chapman
Praxis Critical Systems Ltd.
20 Manvers Street
Bath BA1 1PX, United Kingdom
rod.chapman@praxis-cs.co.uk

Adrian Hilton
Praxis Critical Systems Ltd.
20 Manvers Street
Bath BA1 1PX, United Kingdom
adrian.hilton@praxis-cs.co.uk

ABSTRACT

Existing security models require that information of a given security level be prevented from “leaking” into lower-security information. High-security applications must be demonstrably free of such leaks, but such demonstration may require substantial manual analysis. Other authors have argued that the natural way to enforce these models automatically is with information-flow analysis, but have not shown this to be practicable for general purpose programming languages in current use.

Modern safety-critical systems can contain software components with differing safety integrity levels, potentially operating in the same address space. This case poses problems similar to systems with differing security levels; failure to show separation of data may require the entire system to be validated at the higher integrity level.

In this paper we show how the information flow model enforced by the SPARK Examiner provides support for enforcing these security and safety models. We describe an extension to the SPARK variable annotations which allows the specification of a security or safety level for each state variable, and an extension to the SPARK analysis which automatically enforces a given information flow policy on a SPARK program.

Categories and Subject Descriptors

D2.4 [Software Engineering]: Software/Program verification; F3.1 [Logics and Meanings of Programs]: Specifying and verifying and reasoning about programs

General Terms

Design, Measurement, Security, Verification

Keywords

Information flow, SPARK Ada, Dolev-Yao, Bell-LaPadula, security, safety

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGAda'04, November 14–18, 2004, Atlanta, Georgia, USA.
Copyright 2004 ACM 1-58113-906-3/04/0011 ...\$5.00.

1. INTRODUCTION

Software is often used to develop security-critical applications. Some of these applications are required to manage information of different classification levels, ensuring that each user may only access the data for which he or she has adequate authorisation. This requirement has two components; the developer must produce a design and implementation which supports this model and its constraints, and the implementation must support *verification* that the constraints are never violated. It is not enough for the implementation to be correct; for it to be secure, it must be *seen* to be correct.

In this paper we examine the problem of developing and verifying a security-critical application containing this security structure (often termed *multi-level security*). We will analyse recent work on information flow and security and show how the information flow analysis of the SPARK Examiner tool is appropriate for solving this problem. We will show how it is also important for the efficient analysis of safety-critical systems. We will then describe modifications to the current definition of the SPARK Ada language[2] and the Examiner which permit complete validation of Ada programs against defined security models such as the Bell-LaPadula model[3].

We intend that the additional flow analysis features described here will appear in a future commercial Examiner release. Although we anticipate possible minor changes to the syntax and analysis performed, we expect the released version to implement the analysis described here in all substantial aspects.

2. EXISTING WORK

In this section we identify typical standards which will inform the development and verification of a security-critical or safety-critical application. We then analyse a survey paper on information flow and security, and compare its conclusions against the information flow model of the SPARK annotated subset of Ada95.

2.1 Standards

The Common Criteria for IT Security [5] specify the development and verification activities that are suitable for applications at differing levels of security. These Evaluation Assurance Levels (EALs) range from EAL-1 (lowest) to EAL-7 (highest). As the EAL number rises, the required activities become more rigorous; at the higher levels, formal specification and analysis of the software becomes required.

For safety-related applications there are similar concepts for the safety criticality of software; RTCA DO-178B[10] for civil avionics software defines criticality levels E (lowest) through to A (most critical). As one would expect, the required development and verification activities become increasingly more onerous (and expensive) with increasing criticality level. As a result, if an avionics system were to contain a “core” of critical functionality at Level A and a larger body of utility code at Level D then either the entire system would have to be developed and verified at Level A or a rigorous argument would have to be applied that the Level D code could not in any way affect the integrity of the Level A code.

Another notation for safety criticality is the Safety Integrity Level (SIL), described in IEC 61508[8]. SIL-1 is the lowest level of safety integrity, and SIL-4 the highest; DO-178B level A approximates to SIL-3/SIL-4 when comparing the development and verification activities required.

In this paper we assume that we are attempting to validate systems at EAL-5 or greater (for security) and RTCA Level B or greater (for safety). We are therefore required to provide a rigorous and comprehensive justification for any statements which we make about the separation of data. Therefore we now look at how such statements may be expressed.

2.2 Information Flow

“Information flow” as it applies to conventional imperative computer programs has a range of definitions, and is often confused with “data flow”; we shall take the definition as expressed in Barnes[2] p.13:

- data flow analysis is concerned with the *direction* of data flow; whereas
- information flow analysis also considers the *coupling* between variables.

An example of the difference between data flow and information flow comes from the following (Ada) code:

```
while (X < 4) loop
  A := A + 2 * B;
  X := X * 2;
end loop;
```

Here, data flow analysis would state that data flows from X to X and from A and B to A. Information flow analysis would note additionally that the final value of A is affected by the initial value of X, and hence that there is information flow from X to A.

Sabelfeld and Myers[11] recently surveyed the use of information flow analysis in software. They viewed the fundamental problem of maintaining security as one of tracking the flow of information in computing systems. They examined the various ways that secure information could leak into less secure information, overtly and covertly. They identify in particular the concept of *implicit* information flows, an example of which is given in the `while`-loop example above.

Their survey characterised language-based information flow as requiring:

1. semantics-based security, so that rigorous argument could be made about a variation of a high-security value not affecting a low-security value; and

2. a security type system, so that a program or expression can be assigned security values (types) and the type changes of the program or expression can be characterised.

They concluded that “standard” security practices do not and cannot enforce the end-to-end confidentiality required by common security models. They characterised the existing work in security and information flow analysis, but notably did not address the work of Bergeretti and Carré[4].

2.3 Security models

The Bell-LaPadula (BLP) model of computer security[3] enforces two properties:

1. no process may read data from a higher security level; and
2. no process may write data to a lower security level.

Such multi-level security has a number of problems; Anderson[1] provides a list of them including:

- “blind write-up”: the inability to inform low-security data whether a write to high-security data has happened correctly;
- “downgrading”: moving information from a high security level to a lower level is sometimes desirable; and
- “TCB bloat”: a large subset of the operating system may end up in the Trusted Computing Base (TCB).

The Dolev-Yao security model[6] makes secret information indivisible; it cannot be leaked in part but only in total.

2.4 Information Flow in Ada

Bergeretti and Carré wrote a seminal paper[4] describing a practical implementation of information flow analysis in the SPADE Pascal language (although the principles were applicable to most conventional imperative programming languages). This was managed by the composition of matrices representing information flow dependencies between variable imports and exports. Notably, conditional and infinite loops were permissible and analysable within the framework of such a language; these were managed by computing the transitive closure of the information flow matrix corresponding to one execution of the loop.

This information flow model was implemented in the SPARK annotated Ada subset[2]. The subset requires each subprogram to be annotated with the required information flow (“derives” annotations) if information flow analysis is required. The SPARK subset is enforced by the SPARK Examiner tool which checks the required information flow of each subprogram against the actual information flow. The annotations (and other SPARK rules such as the ban on circular dependency) are necessary to make this information flow analysis tractable.

The result of this is that it is possible, in a fully-analysed SPARK program, to be certain that a given exported variable is independent of a given imported variable. This is a key step towards supporting multi-level security in SPARK Ada, and makes it easier to write demonstrably secure and correct code, but is not a complete solution. In Section 3 we will describe how SPARK Examiner analysis may be extended better to implement these checks.

We now examine case studies of the use of SPARK, and the utility of information flow in real applications.

