The Need for an Integrated Design, Implementation, Verification and Testing Methodology

R. Alan Whitehurst

Department of Defense Computer Security Center

C411

Introduction

It is the chartered goal of the Department of Defense Computer Security Center (DoDCSC) to promote the widespread availability of trusted computer systems. Experience has shown that the necessary features and safe-guards cannot be patched into a system to successfully provide security, but that such security principles must be a driving design goal through all phases of system development. If we are to be successful in our attempt to make secure computer systems widely available, we must encourage the development of new systems which meet the security requirements as outlined in the Trusted Computer System Evaluation Criteria (henceforth referred to as "the criteria").

One such requirement for a system to be evaluated as an A level system is that the designers define a mathematical model of what security means and that the design (and possibly the implementation) be formally verified correct with respect to that security model. The process of formally specifying and verifying that the design and implementation are consistent with the relevant security principles as expressed by the security model, provide added assurance that a logical security policy is uniformly enforced throughout all the functions of the trusted computer base (TCB).

Problem

Unfortunately, the cost of producing such verifiable designs is very high. This stems from a number of factors, not the least of which are the current state of the verification tools and the relative obscurity of the verification techniques. The current state of the tools is such that they demand a high degree of scientific background and expertise from their users. Furthermore, they are very specific and limited in the scope of what they are intended to do. The relative obscurity of verification techniques compounds the problem in that the number of individuals qualified to conduct a successful verification effort is relatively small.

Another problem which makes the prospect of verifying a system design unattractive is the limited pay-back from such an effort. Most current verification systems are designed to do just that -- to verify consistency between a formal model and some given design -- and nothing more. Therefore, large amounts of time and energy go into producing a design specification which, by its very nature and purpose, is void of implementation detail, in order to prove that it is consistent with some user defined model. Normal design practices dealing with structured code, input/output analysis, composite design, testing, and so on, must still be conducted independent of the verification effort.

This brings to light another failing of the current verification systems: a verified design does not insure that the system is error free or that it is functionally correct; it simply states that the system is correct with respect to some user-defined model. Manual creation and analysis of the model must still be done to establish that it is a complete and accurate model of some portion of reality and that it embodies the principles one is interested in maintaining. Dependence upon this manual analysis of the model decreases the assurance gained by the formal (automated) verification process.

The majority of verification systems in use today do not include the capability to test the HOL implementation for consistency with the system specification. This must be done by hand, through a process of mapping the specification constructs onto the HOL constructs and attempting to show the formal correspondence. Those systems which do include code verification are viable only for limited applications of relatively small systems. This introduces another step in the verification of a system which must be done by hand without the use of automated tools and thereby decrease confidence that the system is consistent and/or the verification process meaningful.

In summary, the present verification systems are inadequate to truly meet the needs of system developers. While they do give added assurance that certain principles (such a security) are included and honored in a design, the cost of such a verification effort is high and does not yield other necessary information concerning the soundness of the design or the detailed strategy for its implementation. Several week links in the present methodologies introduce the opportunity for human error to enter the design and thereby reduce the confidence in the correctness of the system. Significant work must be done to increase the benefits of using verification systems and to reduce the cost associated with such efforts if the DoDCSC is to reach the goal of widespread availability of trusted computer systems.

Solution

Although much can be learned from the existing methodologies, there are fundamental weaknesses in each. Perhaps the most pervasive of these weaknesses is the goal of each system; they were designed primarily as verification tools rather than true design methodologies. Hence, to enable system developers to make use of these tools in the process of system design, the developers of verification tools joined a number of loosely-coupled programs into a "methodology". For this reason, it becomes questionable that any amount of work to the existing verification tools will ever truly be able to produce an integrated environment for the design, implementation, and testing of verified systems.

Rather than spending energy attempting to force the existing verification systems into the role of an integrated design and verification methodology, it would be much more productive (and have a higher probability of success) if attention and energy were directed towards the construction of a new "next-generation" design and verification tool.

An Integrated Design, Verification, Implementation and Testing Methodology

Using existing technology, but working towards the goal of producing an integrated design/verification environment rather than a verification tool, significant improvements could be made to both lower the cost of the verification process and maximize the benefits of such verification with respect to the overall process of design and implementation. A number of goals should be included in the design of such an integrated environment:

1. The system should support an approach to design and implementation of software systems which is intuitive.

2. The user interface should be constructed to maximize information while minimizing output.

3. The languages for expressing the design, specifications, and implementation should be highly integrated (if not the same).

4. The the proving process should be automated to a high degree to minimize the mathematical expertise required of the user.

5. The system should support multiple methods for gaining assurance of a correct design and implementation.

This is by no means an exhaustive list of design requirements or goals for the type of design and verification system which is being described; rather these are broad categories which, by and large, have been neglected in the design of the other verification systems. We will now look at each of the proposed goals in greater detail.

An Intuitive Approach

Thought must be expended to find or formulate a strategy for design, specification, and implementation that is both intuitive as well as compatible with the process of verification. Current verification systems have forced a rather strange view of the world upon system developers and have demanded a high degree of expertise of their users. It is essential that whatever strategy is chosen be a natural approach to system development in order to *maximize productivity while minimizing cost.*

The User Interface

One of the prime factors inhibiting the wide-spread use of verification systems today is the primitive state of the user interface. A next-generation system would strive to make the exchange of information between the system and the user as painless and as simple as possible (while avoiding simple-mindedness and tediousness). This would certainly involve the use of graphics as a prime medium for information exchange in both directions, as well as windowing and mouse capabilities. One might invision a system which allowed the designer to specify the hierarchical structure of the design as a graph, with nodes representing procedures and edges representing interfaces; then to allow the designer to pick a module at any point in the tree using a mouse, define it in terms of its interfaces, assertions, specifications and/or code, test the design through system-provided stubs or other mechanisms (rapid prototype), graphically display proof structure for the overall system and for each module, produce test data, and graphically display execution paths that have been tested. Additional information should also be generated involving information flow, cross-reference listings, and configuration management (to name a few areas) and an investigation of representing this information graphically or semi-graphically should also be investigated.

An Integrated Language

To maximize the benefit the designer/developer receives from utilizing the methodology (and to avoid duplication of effort, a problem that plagues current verification methodologies), a single, coherent and integrated language should be used to communicate to the system the design requirements and goals, the program structure, the assertations and formalism, and finally, the HOL code. While there is inherent danger in requiring a single language to have complete functionality *in each* of the areas listed above, the obvious benefit is that the designer and developer must learn only a single language. Further, any effort expended in expressing concepts or procedures in this language contribute to the final goal of verified, executable code. In other words, *the specification of design* and implementation are

not "code-to" specs, but are actually part of the code. There is no conceptual abyss between specifications and code that must be bridged. The feasibility of such an approach is substantiated by recent work in the area of non-procedural, predicate-calculus based languages (such as PROLOG). Although these languages resort to the inclusion of flow control information as part of their basic structure, and such information would generally want to be suppressed from any top level system specification, the general principles of logic-based specification/programming is both attractive and promising.

Automated Proofs

A theorem prover and proof system must be included in the methodology which is both powerful and fast. It should relieve the user from most (if not all) of the busy work. In short, the prover should not be a simple proof checker, but rather it should be capable of high-level proofs without the intervention or aid of the user. This would probably involve a sophisticated front-end to the prover that would screen the numerous trivial theorems that prove without manipulation.

Additional Assurances

There are a number of other aids/techniques which can and should be automated and included in the next generation system. These would probably include (but would not necessarily be limited to) the following:

Assertion Processing

Assertion processing is the process whereby the program's assertions are checked during program execution. As such, the techniques serve as a bridge between the more formal program correctness proof approaches and the more common "black box" testing approaches. This could also be viewed as a dynamic check on the validity of the program specification assertions.

Cause-Effect Graphing

Cause-effect graphing is a test case design methodology. It is used to select in a systematic manner a set of test cases which have a high probability of detecting errors that exist in a programs. This technique explores the inputs and combinations of input conditions of a program in developing test cases. It is totally unconcerned with the internal behaviour or structure of a program. In addition, for each test case derived, the technique identifies the expected outputs. The inputs and outputs of the program are determined through analysis of the requirement specifications. These specifications are then translated into a Boolean logic network or graph. The network is used to derive test cases for the software under analysis.

Configuration Management

Configuration management would include tools to highlight system interdependence and to provide relevant information to designers and developers relating to the impact upon changing a portion of the system on the remaining modules.

Cross-Reference Generation

Cross-reference generation produces lists of data names and labels showing all of the places they are used in a program.

Data Flow Analysis

Data flow analyzers are tools which can determine the presence of absence of data flow errors; that is, errors that are represented as particular sequences of events in a program's execution. This analysis is generally limited to sequential rather than synchronous or concurrent execution.

Interface Checking

Interface checkers analyze the consistency and completeness of the information and control flow between components, modules or procedures of a system.

Requirements Analysis

The requirements for a system will normally be specified using some formal language which may be graphical and/or textual in nature. A requirements analyzer can check for syntactical errors in the requirements specifications and then produce a useful analysis of the relationships between system inputs, outputs, processes, and data. Logical inconsistencies or ambiguities in the specifications can also be identified by the requirements analyzer.

Requirements Tracing

Requirements tracing provides a means of verifying that the software of a system addresses each requirement of that system and that the testing of the software produces adequate and appropriate responses to those requirements.

Specification- Based Functional Testing

Functional testing can be used to generate system test data from the information in requirements and design specifications. It is used to test both the overall functional capabilities of a system and functions which originate during system design.

Symbolic Execution

Symbolic execution is applied to paths through programs. It can be used to generate expressions which describe the cumulative effect of the computations which occur in a program path. It can also be used to generate a system of predicates describing the subset of the input domain which causes a specified path to be traversed. This information is generally used as a basis for data flow analysis and proof of correctness.

Test Coverage Analysis

Test coverage analyzers monitor the execution of a program during testing in order to measure the completeness of a set of program tests. Completeness is measured in terms of the branches, statements or other elementary program constructs which are used during the execution of the program over the tests.

Miscellaneous Considerations

The discussion to this point has been intentionally rather generic. The system described could be used to design, verify and test any application in any realm. However, since we at the DoDCSC are primarily interested in the verification of security principles, there are other matters relative to this application area which must be addressed: formal security models, multi-level security flow-analysis, and implementation languages.

Security modeling is a topic which is currently enjoying a high degree of attention in the verification community. It is the general consensus of most individuals involved that the existing models are inadequate and that more research and development need to be committed to creating and validating models of security. This being the case, it appears quite advantageous to exclude from the verification system any pre-conceived notion of what a security model is or should be. By the same token, however, in order to produce a MLS Flow-Analysis tool for covert channel detection, such a model must be assumed. The pros and cons of these factors must be weighed to determine what the optimum solution is going to be, but it obviously involves some sort of compromise between flexibility and MLS Flow-Analysis.

By virtue of the DoD policy on the use of Ada, to meet the need of system developers under contract to the DoD the implementation language (and therefore, the design and specification language) should probably be oriented towards Ada. While there is some fundamental concerns about the inherent security properties of this language, a thorough analysis of its strengths and weaknesses relating to security and verification has yet to be conducted. Operating under the assumption that the DoD is truly committed to using Ada for all mission critical applications (and therefore, that there would be some opportunity to influence the structure of the language were it proved that inherent security flaws existed), then we are obligated to support development using the Ada language. Since Ada is essentially a super-set of Pascal, and much work has been done in formalizing and verifying the structure of that language, it is reasonable to assume that there exists some subset of the language which is verifiable. More research will be needed to confirm that this is actually the case and to settle issues involving the formal semantics of the language and the impact of the run-time Ada environment on the security of the overall system. Further research will have to be conducted to determine the feasibility of extending the Ada language to include specification capabilities. One possible solution to this apparent paradox is to allow the design and specification of a system to be conducted in pure logical notation (with necessary extensions), and to automatically generate the Ada code. This may well be beyond the scope of what might be considered the "next generation" of tools, however.

Summary

Present verification systems are inadequate to meet the needs of the DoDCSC. To obtain the wide-spread availability of trusted computer systems qualified to process data at multi-level classification, it will be necessary to develop integrated design/verification methodologies which will reduce the cost and maximize the benefits of performing system verification by including a number of necessary and/or attractive design/verification/testing tools into a single, integrated environment. In the meantime, however, there is still much to be gained by the use of present verification systems and work should proceed to stabilize and enhance those tools to provide system developers the opportunity to develop expertise in the use of such methodologies.