Introduction

COMSEC – Communication Security
- Manage encrypted communications

CD – a COMSEC device
- Provides cryptographic processing for a radio receiver

PEIP (Programmable, Embeddable INFOSEC Product)
- Technology used for implementing CD
Software Cost Reduction (SCR)

- A Formal Method for specifying system requirements
- Uses tabular notation
  - E.g. mode transition tables and event tables
- Designed for safety-critical control systems

SCR Method

- SCR Requirements Specification
  - System environment
  - Required system behavior
- System Representation
  - State machine
  - Current state determined by values of state variables
  - Input events change state and produce output events
- SCR Specification
  - Defines transitions of system using a set of tables
  - Contains various dictionaries
  - Specification assertion dictionary records security properties

Overview SCR*

- SCR*
  - Set of tools for detecting flaws in requirements specification
  - Often automatic
  - Complimentary
  - Supportive
**SCR* - Included tools**
- Specification editor
- Dependency graph browser
- Automated consistency checker
- Simulator
- Spin interface (model checking)
- Invariant generator
- TAME (Timed Automated Modeling Environment)
- PVS interface (theorem proving)
- Validity checker
- Test set generator

**Applying SCR* to CD**

**Prose to SCR Requirements**
- **SCR Specification Development**
  - Used CD Systems Requirement Document (SRD)
  - States required system behavior
  - Consistent with SCR model of black-box requirements
    - Purpose of CD requires some memory management outside of black-box
- **Security Properties**
  - Extracted from SCR specification
  - Based upon required behavioral goals and interpretation of function descriptions from CD SRD
The Consistency Checker
- Fully automatic
  - Provides detailed feedback for error correction
  - Uses static analysis techniques
- What can be found with checker?
  - Syntax and type errors
  - Variable name discrepancies
  - Unwanted nondetermination (disjointness errors)
  - Missing cases (coverage errors)
  - Circular definitions
    - Determines if dependency graph cycles have circular dependencies

Simulating the CD Specification
- Validates the specification defined behavior
- Represents current state of execution
- Scenarios – a sequence of input events
- Running scenarios
  - Each event updates values of dependant variables
- Scenarios Produce
  - Execution history
  - Report of violated specified properties
- Generic interface
  - Current state of execution
  - Simulator supports rapid construction of customized graphical front-ends

Automatic Invariant Generation
- Generation is “based on an algorithm for constructing state invariants from the functions defining the dependent variables.”
- Types of invariants
  - State invariants: property that holds in every reachable state
  - Transition invariants: property that holds in every reachable transition (prestate-poststate pair)
Automatic Invariant Generation

**Invariant Format:**
- $v$ = dependent variable with values $\{a_1, a_2, \ldots, a_n\}$
- $C_i$ = terms of variables on which $v$ depends
- Examine conditions that may cause the value of $v$ to change
- For each $a_i$, generate an invariant
  - $(v = a_i) \Rightarrow C_i$
- For variable $v$ with value $a_i$, depends on $C_i$

**Example Invariant:** (smOperation is a mode class)
- smOperation = sIdle
  - $\Rightarrow$ mHealthyBackground AND mBackupPower $\neq$ overvoltage

In Idle mode, the system is healthy and backup power is not overvoltage.

**Generating Invariants**
- SCR partially implements the algorithm from a mode transition table
- Full algorithm (by hand!) implements from event tables, condition tables, and stronger method from a mode transition table
- Invariants generated are not always strongest
- Applying invariant generation may be a useful first step for verifying a set of properties

**Model Checking Properties**
- Exhaustive search of automation state space
- Mostly Automatic process
  - Beneficial to validate a property before performing theorem proving
- Automatic abstraction methods
  - Reduce state space
    - Variable restriction
    - Variable abstraction
  - Partial search for property violation
- Spin will provide counterexamples of violations
Model Checking Properties

Model Checking Steps
- Step 1: Reduce state space
  - Step 1a: Variable restriction
    - Automated
  - Step 1b: Variable abstraction
    - By hand
- Step 2: Implement Spin
  - Found property violations usually were a result of incorrect formulation of the property

Strategy for checking properties:
- Use model checker to check validity before using a theorem prover to establish the property

TAME: Checking Properties

- Provide interface for PVS to prove properties of automata models
- Goal of TAME
  - Reduce human effort to use PVS
- Goals of PVS
  - Specify automata models
  - Prove state invariant properties for models

(continued)
TAME: Checking Properties

- How TAME works
  - Offers +20 strategies that implement proof steps mimicking humans in proving invariant properties
- Developed
  - SCR-to-TAME automatic translator
  - TAME strategies for automatic analysis of properties of SCR automata
TAME: Checking Properties

- State invariants
  - Proved by induction
  - SCR_INDUCT_PROOF
- Transition invariants
  - Proved by reasoning about transition relation of the SCR automation.
  - SCR_DIRECT_PROOF

Proof: Certain properties were invariants
- One property that Spin could not find a violation of was not an invariant
- If proof fails
  - Invariant is false, or
  - Additional invariants are needed in the proof
- Proof “dead-ends” have a problem transition
  - One-state properties – transition of the action case in the induction proof resulting in dead-end
  - Two-state properties – transition from the given state through some enabled automation action to the successor state
- TAME provides analysis strategy for details of problem transition

The Validity Checker

- VC
  - Checks validity of first-order or two-state properties
  - Three step process
    1. Initial term-rewriting phase
    2. Application of a decision procedure using binary decision diagrams to evaluate propositional formulae
    3. Constraint solver to reduce simple integer arithmetic formulae
  - Can perform an induction proof of a property
- Automatic translation of SCR specifications into VC input
- VC can demonstrate a property to be invalid
- VD cannot be used to prove properties interactively
Generating Test Sets

- Automated
- Constructs suite of test sets from SCR requirements
  - Uses model checker’s counterexamples to construct test sets
  - Sequence of system inputs coupled with required system outputs
  - Requirements specification used for inputs and outputs
- Traces organized into equivalence classes
  - One or more test sets for each class are built
- Test sets automatically used to evaluate implemented software

Process Analysis:
The Complexity of CD

- Significant Complexity with Moderate Size
- SCR specification has 39 variables
  - Variables have complex relationships
- In any state of the CD, just for mHostCommand, 16 possible input events involving changes
  - 17 variables similar to mHostCommand
- Mode transaction table
  - 55 events
  - 25 mode transactions defined
- Large event tables
  - Average number of events = 8
  - Largest number of events = 16

Process Analysis:
Time and Effort

Total time: 1 one-person month
(with additional time for improving the SCR* techniques)
- Prow to SCR time period: 2 one-person weeks
  - Includes refinement with consistency checker
- Consistency checker time: few seconds
  - Except for coverage & disjointness errors that took up-to minutes
- Simulator time = Graphical front-end built < 1 day
  - Improved interactions with project manager
- Property formulation improvements (model checker feedback) time = few days
- TAME & validity checker (w/new improvements) time/property = few minutes
- Analysis of proof dead-ends for cause and resolution, time = not given
  - Stated to be most labor-intensive
- Generating Testing Sets not completed
Process Analysis: Practicality of SCR

- Analyzed 8 security properties out of hundreds of security properties
- Does the operational specification satisfy this property?
  - When “No”, counterexample was given
- Is this really a practical, low-cost approach to providing high assurance?

References


Questions

- Discuss verification and/or validation in relation to the SCR tools. (Slide 28)
- What is the strategy used to check security properties? What reason(s) is (are) given for using this strategy? (Slide 16)
- The authors state that "SCR and the analysis techniques supported in SCR provide a practical, low-cost approach to providing high assurance." Argue for or against this statement.
Verification vs. Validation

- Validation
  - "to confirm that the specification captures the operational system behavior intended by the customer"
  - Simulator tool
- Verification
  - "to prove that the specification satisfies selected system properties"
  - All tools but Simulator

Example Properties

- If CD is tampered with, then key 1 in keybank 1 is zeroized
- When the zeroize switch is activated, key 1 in keybank 1 is zeroized
- No key can be stored in location 1 of keybank 1 before an algorithm has been loaded into the first location of algorithm storage segment 1
- If backup power has an undervoltage when primary power is unavailable, the CD enters either Alarm mode or Off mode
- If backup power is overvoltage then the CD is in Initialization, Standby, Alarm, or Off mode
- If primary power has an overvoltage then either the CD is in Initialization, Standby, Alarm, or Off mode, or the CD enters Initialization mode
- If primary power has an undervoltage then either the CD is in Initialization, Standby, Alarm, or Off mode, or the CD enters Initialization mode