VERIFYING A SEPARATION KERNEL FOR A CRYPTOGRAPHIC DEVICE: A STATUS REPORT

CONNIE HEITMEYER
CENTER FOR HIGH ASSURANCE COMPUTER SYSTEMS
NAVAL RESEARCH LABORATORY
WASHINGTON, DC 20375

(202) 767-3596
heitmeyer@itd.nrl.navy.mil

Joint work with Myla Archer, Jim Kirby, and Beth Leonard
Sponsored by ONR
OUTLINE

• Background
  – SCR Tools
  – Applying SCR Tools in the PEIP I case study

• Verifying that PEIP II enforces separation
  – Proposal 1: Start with requirements (Top-down)
  – Proposal 2: Start with Separation Kernel (Bottom-up)

• Current status
  – State-Based spec of separation kernel
  – Proving separation

• Summary
SCR METHOD AND TOOLS
HISTORY OF
SCR (SOFTWARE COST REDUCTION) METHOD

● **1978:** Parnas et al. introduced SCR requirements method
  – Tabular notation
  – Events and conditions
  – Mode classes and terms

● **1980s-early 1990s:** SCR applied to a wide range of systems
  – Telephone networks (AT&T Bell Labs)
  – Submarine communications (NRL)
  – Control software for nuclear plants (Ontario Hydro)
  – Avionics software (Grumman)

● **Early 1990s:** Further development of model and method
  – Parnas, van Schouwen introduce and apply Four Variable Model
  – SPC develops CoRE
  – Lockheed applies CoRE and SCR tables to C-130J flight program

● **1992-present:** NRL develops formal SCR model and tools
SCR: A PRACTICAL METHOD FOR DOCUMENTING REQUIREMENTS*

**SPECIFY THE SYSTEM PRECISELY**

Use a **TABULAR notation with an explicit formal semantics** to specify the required behavior

**APPLY “CONSISTENCY CHECKING”**

Automatically check spec for type errors, missing cases, ambiguity, circularities, etc.

As we move down the chain, we increase assurance in the spec

**SIMULATE THE SYSTEM BEHAVIOR**

Symbolically execute the system based on req. specs

**VERIFY SPECS USING MODEL CHECKING**

Check critical properties

**VERIFY SPECS USING THEOREM PROVING**

• Usable, scalable **tabular notation**
• Integrated set of robust, powerful **software tools**
  – light-weight tools whose use does not require math. sophistication/thm proving
  – heavy-duty tools

**INCREASING COST, INCREASED EXPERTISE**

**SCR TOOLS FOR DEVELOPING A REQUIREMENTS SPEC (1)**

**SCR TOOLSET**

- most mature tools
- installed at 100+ org’ns in industry, govt., and academia

**Diagram:**

- SPECIFICATION EDITOR
- CONSISTENCY CHECKER
- MODEL CHECKER
- DEPENDENCY GRAPH BROWSER
- SIMULATOR

- system spec
  - terms
  - modes
  - cont vars
  - events
  - mon vars
  - conditions
SCR TOOLS FOR DEVELOPING A REQUIREMENTS SPEC (2)

**SCR TOOLSET**

- most mature tools
- installed at 100+ org’ns in industry, govt., and academia

**New ANALYSIS TOOLS**

- TAME is an interface to PVS designed to prove properties of state machine models

---

**SPECIFICATION EDITOR**

**CONSISTENCY CHECKER**

**MODEL CHECKER**

**DEPENDENCY GRAPH BROWSER**

**SIMULATOR**

**THEOREM PROVER (TAME)**

**PROPERTY CHECKER (Salsa)**

**INARIANT GENERATOR**
METHODS THAT IMPROVE THE REQUIREMENTS SPECIFICATION

**SCR TOOLSET**
- most mature tools
- installed at 100+ org’s in industry, govt., and academia

New ANALYSIS TOOLS
- TAME is an interface to PVS designed to prove properties of state machine models

- Consistency and completeness
  - Is the spec well-formed?

- Validation
  - Is this the right spec?
  - I.e., does the spec capture the intended behavior?

- Verification
  - Is the spec right?
  - I.e., does the spec satisfy critical properties (e.g., safety, security)?
PROTOTYPE TOOLS FOR SOFTWARE TESTING & CODE SYNTHESIS HAVE ALSO BEEN DEVELOPED

**SCR TOOLSET**
- Most mature tools
- Installed at 100+ org’ns in industry, govt., and academia

**New ANALYSIS TOOLS**
- **TAME** is an interface to PVS designed to prove properties of state machine models
**Objective**

- Reduce human effort needed to verify properties with a theorem prover

**Design Goals**

- Easy to create specs
- Natural formulation of properties
- ‘Natural’ proof steps that match in size and kind the steps used in hand proofs
- Proofs recognizably similar to hand proofs

**Why build upon PVS?**

- Avoid reinventing existing, well-known techniques
- Use PVS logic as a flexible means of further proof support for automata models
- State properties in the expressive but natural logic of PVS

---

## STEPS IN HAND PROOFS VS. STEPS IN PVS PROOFS

### HUMAN-STYLE

<table>
<thead>
<tr>
<th>Proving ( A \Rightarrow B ): “suppose ( A )”</th>
<th>(FLATTEN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proving ( \forall a. P(a) ): “fix ( a = a_0 )”</td>
<td>(SKOLEM &lt;fnum&gt; “a0”)</td>
</tr>
<tr>
<td>“By the definition of &lt;function&gt;”</td>
<td>(EXPAND “&lt;function&gt;”)</td>
</tr>
<tr>
<td>To show “( \exists a. P(a) ) because ( P(a_0) )”</td>
<td>(INST &lt;fnum&gt; “a0”)</td>
</tr>
<tr>
<td>??? (A miracle happens here -- maybe)</td>
<td>(GRIND)</td>
</tr>
<tr>
<td>Knowing “event ( \pi ) precedes state ( s ) and ( P(\pi, s) ) holds” adduce “the last event ( \pi_0 ) before ( s ) such that ( P(\pi_0, s) )”</td>
<td></td>
</tr>
</tbody>
</table>

---

### PVS

<table>
<thead>
<tr>
<th>Proving ( A \Rightarrow B ): “suppose ( A )”</th>
<th>(FLATTEN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proving ( \forall a. P(a) ): “fix ( a = a_0 )”</td>
<td>(SKOLEM &lt;fnum&gt; “a0”)</td>
</tr>
<tr>
<td>“By the definition of &lt;function&gt;”</td>
<td>(EXPAND “&lt;function&gt;”)</td>
</tr>
<tr>
<td>To show “( \exists a. P(a) ) because ( P(a_0) )”</td>
<td>(INST &lt;fnum&gt; “a0”)</td>
</tr>
<tr>
<td>??? (A miracle happens here -- maybe)</td>
<td>(GRIND)</td>
</tr>
<tr>
<td>Knowing “event ( \pi ) precedes state ( s ) and ( P(\pi, s) ) holds” adduce “the last event ( \pi_0 ) before ( s ) such that ( P(\pi_0, s) )”</td>
<td></td>
</tr>
</tbody>
</table>

---

### In starting the proof of a state invariant: “Use induction.”

### Introduce the constraints applying to a nondeterministic \( \varepsilon \) value in the poststate

---

**TAME Goal:** Provide natural proof steps

4/24/02
<table>
<thead>
<tr>
<th>TAME PROOF STEP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO_INDUCT</td>
<td>Use induction based on initial state and possible transitions</td>
</tr>
<tr>
<td>APPLY_SPECIFIC_PRECON</td>
<td>Apply the specified precondition of the current action</td>
</tr>
<tr>
<td>APPLY_IND_HYP</td>
<td>Apply (quantified) inductive hypothesis to non-default values</td>
</tr>
<tr>
<td>DIRECT_PROOF</td>
<td>Set up a direct (non-induction) proof of an invariant</td>
</tr>
<tr>
<td>APPLY_INV_LEMMA</td>
<td>Apply the specified invariant to the pre- or post-state and other arguments</td>
</tr>
<tr>
<td>SUPPOSE</td>
<td>Suppose an assertion true, then suppose it false</td>
</tr>
<tr>
<td>TRY_SIMP</td>
<td>It is now obvious</td>
</tr>
<tr>
<td>SCR_INV_PROOF</td>
<td>Special for SCR: attempt automatic proof of a property using generated invariants</td>
</tr>
</tbody>
</table>
TRANSFER OF SCR TOOLS

**Universities**
- Univ. of Minn.
- CMU
- Stanford Univ.
- Univ. of Penn.
- Univ. of Utah
- Cornell Univ.
- Florida Atlantic Univ.
- Imperial College, U.K.
- Royal Military Coll., CAN
- Queens Univ., CAN
- Iowa State Univ.
- Naval Postgrad. School
- Univ. of Quebec, CAN
- George Mason Univ.
- Univ. of York, U.K.
- West Va. Univ.
- Univ. of Mass.
- George Wash. Univ.
- Univ. of Oregon
- NC State Univ.
- Rice Univ.
- Howard Univ.
- Georgia State Univ.
- Univ. of Toronto
- Colorado State Univ.
- Univ. of Cal.--Irvine
- Univ. of Arizona
- Univ. of Colorado
- Univ. of Waterloo
- Univ. of Virginia
- Upsala Univ., Sweden
- Harvey Mudd College
- Univ. of Limerick, Ireland
- Portland State Univ.
- Univ. of Tennessee
- Wayne State Univ.

**Industry**
- Ontario Hydro
- Lucent Tech.
- Praxis
- Nortel
- Honeywell
- TRW
- MCC
- SPC
- Fraunhofer Inst. (Germ.)
- Boeing
- Lockheed-Martin (Seattle)
- Lockheed-Martin (Ft. Worth)
- Rockwell-Collins Aviation
- Rockwell Science Ctr
- Intermetrics
- Fannie Mae
- Bell Atlantic
- Computing Devices Intern.
- Cleansoft
- PRC
- 21st Century
- United Technologies
- Draper Labs
- Grammatech
- NavCanada
- Siemens
- Q-Labs
- Intel
- Bell Labs (Systems and Softw. Research Ctrr)

**Government**
- NSA
- JPL
- USAF Rome Labs
- SPAWAR SSC
- NIST
- NUWC
- NSWC
- NASA IV&V Facility
- PEIP I
- Argonne National Labs
- US TACOM

**Commercial versions of SCR developed**
- **Prefer**: Product of Rockwell Science Ctr.
- **SPC** Test Automation Framework integrates SCR with their test tool T-VEC

**Lockheed-Martin using SCR tools**
- U.S. rocket programs (Atlas 5, J2, IUS rocket for satellite launch)
- software error detected in MARS Lander
- integrated into JSF software process

**Five pilot projects completed**
- Intern. Space Station (NASA IV&V)
- Flight Guidance System (Rockwell)
- NASA Deep Space-1 Spacecraft (NASA JPL)
- **WCP component of Virginia class sub** (NRL)
- **PEIP I cryptographic device** (NRL)

**New projects in progress**
- Verif. of separation kernel for **PEIP II** (NRL)
- Testing of Fault Protection Engine (NRL/JPL)
- Formal analysis of NSA's Security-Enhanced Linux operating system (NRL)
APPLYING SCR AND TAME TO PEIP I
Each member of the **PEIP** family
- Provides cryptographic processing for the US Navy
- Supports many crypto algorithms
- Is implemented in **software** and hardware
  - Software poses a challenge to the assurance community
- NSA must certify each member of the **PEIP** family

**PEIP:** **P**rogrammable **E**mbeddable **I**nfosec **P**roduct
**PEIP OVERVIEW**

To: ……
From: ……
Subj: ISR Assets

**PEIP SERVICES**

- Load (and zeroize) crypto algorithms and keys
- Configure channel (i.e., write alg and key into channel space)
- Encrypt and decrypt data using a crypto algorithm and a key
- Take emergency action when bad things happen (e.g., device is tampered with)
- **Provide the above services for** $m$ **channels**
CASE STUDY: SCR/TAME USED TO VERIFY SECURITY PROPERTIES OF PEIP I SPEC*

PEIP I provides cryptographic processing for a U.S. Navy radio receiver.

From a prose requirements spec, we developed an SCR spec of **PEIP I**
- To debug and validate the SCR spec, we applied consistency checking, the simulator, and the Spin model checker.

TAME used to analyze a set of security properties
- SCR spec automatically translated into PVS
- TAME verified each true property in a few seconds to a few minutes
- The verification of some properties required invariants
- Analysis with TAME showed that the spec violates one property

CASE STUDY: SCR/TAME USED TO VERIFY THE SECURITY OF THE PEIP I SPEC*

SECURITY PROPERTIES

1. When the zeroize switch is activated, the keys are zeroized
2. No key can be stored before an algorithm in the assoc. location is activated
3. If undervoltage occurs in backup power while primary power is unavailable, PEIP enters alarm or off mode
4. If backup power is overvoltage, then PEIP is in initialization, standby, alarm, or off mode
5. When an overvoltage occurs in primary power, then PEIP is in standby, alarm or off mode, or goes into initialization
6. When an undervoltage occurs in primary power, then PEIP is in standby, alarm, or off mode, or goes into initialization mode
7. If PEIP is tampered with, the keys are zeroized
SECURITY PROPERTIES

1. When the zeroize switch is activated, the keys are zeroized.

2. No key can be stored before an algorithm in the assoc. location is activated.

3. If undervoltage occurs in backup power while primary power is un-available, PEIP enters alarm or off mode.

4. If backup power is overvoltage, then PEIP is in initialization, standby, alarm, or off mode.

5. When an overvoltage occurs in primary power, then PEIP is in standby, alarm or off mode, or goes into initialization.

6. When an undervoltage occurs in primary power, then PEIP is in standby, alarm, or off mode, or goes into initialization mode.

7. If PEIP is tampered with, the keys are zeroized.

proved directly by induction using TAME.
SECURITY PROPERTIES

1. When the zeroize switch is activated, the keys are zeroized.

2. No key can be stored before an algorithm in the assoc. location is activated.

3. If undervoltage occurs in backup power while primary power is un-available, PEIP enters alarm or off mode.

4. If backup power is overvoltage, then PEIP is in initialization, standby, alarm, or off mode.

5. When an overvoltage occurs in primary power, then PEIP is in standby, alarm or off mode, or goes into initialization.

6. When an undervoltage occurs in primary power, then PEIP is in standby, alarm, or off mode, or goes into initialization mode.

7. If PEIP is tampered with, the keys are zeroized.

GEN’D INVARIANTS (KEEP*)

- In Configuration mode, the system is healthy, backup power is not overvoltage.
- In idle mode, the system is healthy, backup power is not overvoltage.
- In traffic processing mode, the system is healthy, backup power is not overvoltage.
- In off mode, KeyBank1Key1=0.
- ...

SECURITY PROPERTIES

1. When the zeroize switch is activated, the keys are zeroized.

2. No key can be stored before an algorithm in the assoc. location is activated.

3. If undervoltage occurs in backup power while primary power is unavailable, PEIP enters alarm or off mode.

4. If backup power is overvoltage, then PEIP is in initialization, standby, alarm, or off mode.

5. When an overvoltage occurs in primary power, then PEIP is in standby, alarm or off mode, or goes into initialization.

6. When an undervoltage occurs in primary power, then PEIP is in standby, alarm, or off mode, or goes into initialization mode.

7. If PEIP is tampered with, the keys are zeroized.

INVARIABLES GEN’D BY KEEP

- In Configuration mode, the system is healthy, backup power is not overvoltage.
- In idle mode, the system is healthy, backup power is not overvoltage.
- In traffic processing mode, the system is healthy, backup power is not overvoltage.
- In off mode, KeyBank1.Key1 = 0.
- ...

4/24/02
VERIFYING THE PEIP I SPEC (4)

SECURITY PROPERTIES

1. When the zeroize switch is activated, the keys are zeroized.

2. No key can be stored before an algorithm in the assoc. location is activated.

3. If undervoltage occurs in backup power while primary power is unavailable, PEIP enters alarm or off mode.

4. If backup power is overvoltage, then PEIP is in initialization, standby, alarm, or off mode.

5. When an overvoltage occurs in primary power, then PEIP is in standby, alarm or off mode, or goes into initialization.

6. When an undervoltage occurs in primary power, then PEIP is in standby, alarm, or off mode, or goes into initialization mode.

7. If PEIP is tampered with, the keys are zeroized.

STRENGTHENED INVARIANTS (GENERATED WITH GROUP*)

- In Initialization mode, primary power is not unavailable.
- In Configuration mode, the system is healthy, backup power is not overvoltage, and primary power is not unavailable.
- In Idle mode, the system is healthy, backup power is not overvoltage, and power power is not unavailable.
- In Traffic Processing mode, the system is healthy, backup power is not overvoltage, and primary power is not unavailable.
- In Off mode, KeyBank 1.Key1=0
- ...

SECURITY PROPERTIES

1. When the zeroize switch is activated, the keys are zeroized.

2. No key can be stored before an algorithm in the assoc. location is activated.

3. If undervoltage occurs in backup power while primary power is unavailable, PEIP enters alarm or off mode.

4. If backup power is overvoltage, then PEIP is in initialization, standby, alarm, or off mode.

5. When an overvoltage occurs in primary power, then PEIP is in standby, alarm or off mode, or goes into initialization.

6. When an undervoltage occurs in primary power, then PEIP is in standby, alarm, or off mode, or goes into initialization mode.

7. If PEIP is tampered with, the keys are zeroized.

STRENGTHENED INVARIANTS

(GENERATED WITH GROUP)

- In Initialization mode, primary power is not unavailable.
- In Configuration mode, the system is healthy, backup power is not overvoltage, and primary power is not unavailable.
- In Idle mode, the system is healthy, backup power is not overvoltage, and primary power is not unavailable.
- In Traffic Processing mode, the system is healthy, backup power is not overvoltage, and primary power is not unavailable.
- In Off mode, KeyBank1.Key1=0.
VERIFYING THE PEIP II SEPARATION KERNEL
NEW EFFORT: VERIFYING THE SEPARATION KERNEL OF **PEIP II**

- **PEIP I** has been certified by NSA and is currently installed aboard US Navy submarines
- To certify **PEIP II**, NSA requires an “EAL7-like” International Common Criteria evaluation of the **PEIP II** separation kernel
- Our plan: Use the SCR/TAME tools to support the evaluation

**Unlike PEIP I, PEIP II**

- both receives *and* transmits data
- processes data at *many* different security levels

QuickTime™ and a GIF decompressor are needed to see this picture.
### PEIP II OVERVIEW

#### PEIP II

<table>
<thead>
<tr>
<th>CHANNEL 1</th>
<th>SHARED</th>
<th>CHANNEL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>process for data encrypt/decrypt on channel 1</td>
<td>process for storing shared algorithms and keys</td>
<td>process for data encrypt/decrypt on channel 2</td>
</tr>
<tr>
<td>ded'd memory for channel 1</td>
<td>shared memory for algs/keys</td>
<td>ded'd memory for channel 2</td>
</tr>
</tbody>
</table>

#### SHARED COMMANDS

1. Load algorithm \( j \)
2. Load key \( k_1 \)
3. Configure channel 1 with alg \( j \) and key \( k_1 \)
4. Encrypt data on channel 1 (using alg \( j \), key \( k_1 \))
5. Load key \( k_2 \)
6. Configure channel 2 with alg \( j \) and key \( k_2 \)
7. Decrypt data on channel 2 (using alg \( j \), key \( k_2 \))
8. Zeroize channel 1

#### CHANNEL COMMANDS

- Algorithm \( j \) and keys \( k_1 \) and \( k_2 \) are loaded into shared memory
- When a channel is configured, an alg. & key are copied into the approp. channel mem.

---

4/24/02
WHAT SECURITY POLICY MUST **PEIP** SATISFY?

**PEIP II**

<table>
<thead>
<tr>
<th>CHANNEL i</th>
<th>SHARED</th>
<th>CHANNEL j</th>
</tr>
</thead>
<tbody>
<tr>
<td>process for data encrypt/decrypt on channel i</td>
<td>ded’d memory for channel i</td>
<td>ded’d memory for channel j</td>
</tr>
<tr>
<td>ded’d memory for channel i</td>
<td>process for storing shared algorithms and keys</td>
<td>process for data encrypt/decrypt on channel j</td>
</tr>
<tr>
<td>shared memory for algs/keys</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**COMMANDS**

Security Policy: Enforce Separation

Data never flows from channel i to channel j and vice versa
WHAT SECURITY POLICY MUST PEIP SATISFY?

Security Policy: Enforce Separation

Data never flows from channel i to channel j and vice versa

Red arrow show data flow that violates separation
WHAT SECURITY POLICY MUST **PEIP** SATISFY?

**Security Policy: Enforce Separation**

Data never flows from channel i to channel j and vice versa
WHAT SECURITY POLICY MUST PEIP SATISFY?

Security Policy: Enforce Separation

Data never flows from channel i to channel j and vice versa
HOW TO OBTAIN ASSURANCE THAT PEIP II ENFORCES SEPARATION?

**SOLUTION:** A separation kernel* ensures that data on channel $i$ is completely separate from data on channel $j$.

---

HOW TO OBTAIN ASSURANCE THAT PEIP II ENFORCES SEPARATION?

HOW TO OBTAIN ASSURANCE THAT PEIP II ENFORCES SEPARATION?

**SOLUTION:** Prove that the virtual machine VM satisfies separation. Then, prove that CM is a refinement of VM.

VM is the composition of VM$_i$ and VM$_j$.

**VIRTUAL MACHINE VM$_i$**
- DEDICATED channel $i$
- SHARED

**VIRTUAL MACHINE VM$_j$**
- SHARED
- DEDICATED channel $j$

**CHANNEL $i$ COMMANDS**
**CHANNEL $j$ COMMANDS**
**SHARED COMMANDS**

**CONCRETE MACHINE CM**
- DEDICATED channel $i$
- SHARED
- DEDICATED channel $j$

**SEPARATION KERNEL**

**TECHNICAL ISSUES**

- What process to use in verifying that PEIP enforces separation?
- What notation to use to specify the behavior of **PEIP II**?
- The choice of notation is critical -- it will be used for two different purposes:
  - For communication among stakeholders
  - As the basis for formal proofs
- We need notation for representing
  - the **PEIP** black-box requirements
  - the required behavior of the **PEIP** separation kernel

**PEIP II STAKEHOLDERS**

- development team (h/w + s/w designers)
- customers (eventual users)
- certifying organization (NSA)
- verification team (us)
1 Specify the required behavior of virtual machine $V_{M_i}$

2 Define VM, the composition of $V_{M_i}$ and $V_{M_j}$, and prove that VM separates input to and data on channels $i$ and $j$

3 Specify the required behavior of concrete machine CM, which uses a separation kernel to enforce separation -- i.e., to mediate access to input data, memory reads, and memory writes

4 Verify that CM refines VM and therefore enforces separation

5 Associate the inputs and outputs of the separation kernel interface in PEIP with those in CM. Generate test cases from the CM sep. kernel spec (using branch coverage) to test the implem. of the PEIP separation kernel
### PROCESS FOR VERIFYING PEIP SEPARATION KERNEL: INITIAL PROPOSAL

1. Specify the required behavior of virtual machine VM\(_i\)  
2. Define VM, the composition of VM\(_i\) and VM\(_j\), and prove that VM separates input to and data on channels i and j  
3. Specify the required behavior of concrete machine CM, which uses a *separation kernel* to enforce separation -- i.e., to mediate access to input data, memory reads, and memory writes  
4. Prove that CM refines VM and therefore enforces separation  
5. Associate the inputs and outputs of the separation kernel interface in PEIP with those in CM. Generate test cases from the CM sep. kernel spec (using branch coverage) to test the implem. of the PEIP separation kernel

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Specify the required behavior of virtual machine VM(_i)</td>
</tr>
<tr>
<td>2</td>
<td>Define VM, the composition of VM(_i) and VM(_j), and prove that VM separates input to and data on channels i and j</td>
</tr>
<tr>
<td>3</td>
<td>Specify the required behavior of concrete machine CM, which uses a <em>separation kernel</em> to enforce separation -- i.e., to mediate access to input data, memory reads, and memory writes</td>
</tr>
<tr>
<td>4</td>
<td>Prove that CM refines VM and therefore enforces separation</td>
</tr>
<tr>
<td>5</td>
<td>Associate the inputs and outputs of the separation kernel interface in PEIP with those in CM. Generate test cases from the CM sep. kernel spec (using branch coverage) to test the implem. of the PEIP separation kernel</td>
</tr>
</tbody>
</table>

4/24/02
1. Specify the required behavior of the PEIP Separation Kernel (SK)

2. Prove that the PEIP SK specification enforces separation

3. Do a code walk-through to validate that the PEIP implementation of the SK satisfies the PEIP SK specification
1. Specify the required behavior of the PEIP Separation Kernel (SK) in process

2. Prove that the PEIP SK specification enforces separation

3. Do a code walk-through to validate that the PEIP implementation of the SK satisfies the PEIP SK specification
EXECUTING PROPOSAL 1
The variables can be partitioned into two classes: shared and dedicated
### cBlackData (i)

<table>
<thead>
<tr>
<th>Mode M</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>ENCRYPT_DATA(x, i) WHEN channel_config (i)</td>
</tr>
<tr>
<td></td>
<td>ZEROIZE_CH (i) OR ZEROIZE_ALG(j) WHEN AlgID (i) = j OR ZEROIZE_KEY(j, k) WHEN AlgID (i) = j AND KeyID (i) = k</td>
</tr>
</tbody>
</table>

\[
cBlackData (i) = \text{encrypt} \ (\text{SelAlg (i)}, \text{SelKey (i)}, x) = 0
\]

- Command language input
- Precondition preceded by WHEN clause
- Uninterpreted functions (e.g., `encrypt`)
- Symbolic values (e.g., \( i \) to represent the index of a channel)
# EXAMPLE: SPECIFYING THE VALUE OF SHARED VARIABLE \textbf{KEY} (j, k)

<table>
<thead>
<tr>
<th>Mode M</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>\textbf{LOAD_KEY}(x, j, k)</td>
</tr>
<tr>
<td></td>
<td>WHEN RedAlg (j) (!=0)</td>
</tr>
<tr>
<td></td>
<td>AND \textbf{keyencrypt}? (x)</td>
</tr>
<tr>
<td></td>
<td>AND decryptkey (x, TrKEK) (!=0)</td>
</tr>
<tr>
<td></td>
<td>- NEVER</td>
</tr>
</tbody>
</table>

\textbf{KEY (j, k)} = decryptkey (x, TrKEK) x 0
Security policy model described formally but in prose using style of [Landwehr, Heitmeyer, McLean, *ACM Trans. on Computer Systems*, ‘84]

Components of formal model
- Primitives are sets of channels, commands, and state variables
- Primitives are used to define the notions of state and a virtual machine VM_i for each channel
- Virtual machines VM_i and VM_j are composed to form the virtual machine VM

Defined separation property in terms of the formal model

Next steps (never got to this)
- define the actual commands and state variables
- translate the model into a formal spec language (e.g., SCR or PVS)
- prove separation (probably using PVS)

We are planning to describe the formal security policy model of the separation kernel in the same prose style. Translating the model into PVS (or SCR) should be easy.
EXECUTING PROPOSAL 2
SEPARATION KERNEL FUNCTIONS

• Mediate access to storage areas
  – i.e., grant and revoke R, W, and X privs

• Handle interrupts
  – Initiate processes in response to interrupts

• Copy sensitive data from one storage area to another
SEPARATION KERNEL (SK) FUNCTIONS

• Mediate access to storage areas
  – i.e., grant and revoke R, W, and X privs
  – includes mediating access to temporary and working storage

• Handle interrupts
  – Initiate processes in response to interrupts

• Copy sensitive data from one storage area to another

• When switching from one subtask to another, sanitize storage areas
Prior to developing a formal spec of the SK, we are studying how the SK interacts with each PEIP command:

- what variables are passed from one process to the next
- what storage areas the SK must sanitize
- ...
CHECK ACCESS TABLE TO ENSURE THAT

**PEIP ENFORCES SEPARATION**

<table>
<thead>
<tr>
<th>CHANNEL i</th>
<th>BadProc2</th>
<th>SHARED</th>
<th>BadProc3</th>
<th>CHANNEL j</th>
</tr>
</thead>
<tbody>
<tr>
<td>process for data encrypt/decrypt on channel i</td>
<td>ded'd memory for channel i</td>
<td>process for storing shared algorithms and keys</td>
<td>shared memory for algs/keys</td>
<td>process for data encrypt/decrypt on channel j</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ded'd memory for channel j</td>
</tr>
</tbody>
</table>

**PEIP**

**BadProc1**

**BadProc2**

**BadProc3**

---

**STORAGE AREA**

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>Temp</th>
<th>KEY</th>
<th>...</th>
<th>...</th>
<th>Red</th>
<th>Black</th>
<th>Temp</th>
<th>Chan</th>
<th>...</th>
<th>Chan</th>
</tr>
</thead>
<tbody>
<tr>
<td>FillBfr</td>
<td>Store</td>
<td>Store</td>
<td>Store</td>
<td>Store</td>
<td>Alg</td>
<td>Store</td>
<td>KeyStore</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BadProc1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>R</td>
<td>...</td>
<td>W</td>
</tr>
<tr>
<td>BadProc2</td>
<td>W</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>R</td>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>BadProc3</td>
<td>R</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>...</td>
<td>W</td>
</tr>
</tbody>
</table>
SUMMARY

- A combined top-down/bottom-up approach is working quite well
  - The precise meaning of separation must be determined in context
    - Our early efforts to understand and represent (in a formal tabular-based notation) the required behavior of PEIP were critical for understanding the behavior of the separation kernel (SK)
    - While studying PEIP's behavior at the abstract (black-box) level was useful, it ignored certain critical issues (e.g., temporary storage and the need for the SK to sanitize memory when it is switching processes)
  - If successful, we will have addressed key aspects of J Moore’s challenge by producing
    - artifacts (a SK implementation and a formal SK specification)
    - an interesting artifact (the separation kernel)
    - a warranty in the form of a theorem (about separation)
    - a mechanically verified warranty (probably in PVS)
    - machinery used available for others to test
- In addition, we will produce (we hope)
  - an interesting and readable formal specification
REFERENCES


