PVS를 이용한 수정된 BLP 모델의 안전성 검증

구하성* · 박태규* · 송호근*

A Safety Verification of the Modified BLP Model using PVS

Ha-Sung Koo* · Tae-Kue Park* · Ho-Keun Song*

요 약

안전성에 대한 이상적인 평가방법은 운영체제내의 모든 가능한 연산을 대상으로 실행 결과의 안정성 여부를 검사하는 것이다. 하지만 이는 현실적으로 불가능하다. 그러나 정형기법을 사용할 경우 운영체제 동작논리상의 안전성 보장 여부를 이론적으로 증명할 수 있다. 따라서 본 논문에서는 보안커널의 안정성 검증방법에 대하여 논하고, 정형검증의 대표적인 도구들에 대하여 비교분석을 수행하였다. 그리고 보안커널에 기반한 다중레벨 접근통제모델인 수정된 BLP(Bell & LaPadula) 모델을 검증하기에 적합한 PVS(Prototype Verification System) 검증도구를 선정하였다. 마지막으로 PVS 검증도구를 활용하여 정형명세를 작성하고, 작성된 정형명세의 검증을 통하여 수정된 BLP 모델이 안전한 보안모델이라는 것을 검증하였다.

ABSTRACT

The ideal method of safety evaluation is to verify results of execution against all possible operations within operating system, but it is impossible. However, the formal method can theoretically prove the safety on actual logic of operating system. Therefore we explain the contents of the art of the safety verification of security kernel, and make a comparative study of various standardized formal verification tools. And then we assigned PVS(Prototype Verification system) of SRI(Stanford Research Institute) to verify the safety of a modified BLP(Bell & LaPadula) model, the core access control model of multi-lavel based security kernel. Finally, we describe formal specification of the revised BLP model using the PVS, and evaluate the safety of the model by inspecting the specification of the PVS.

키워드

Safety verification, PVS, BLP, Security model

Ⅰ. 서 론

현재까지 개발된 보안 운영체제 관련 보호프로파일은 통제된 접근 보호 프로파일(Controlled Access Protection Profile), 레이블 보안 보호 프로파일(Labeled Security Protection Profile), 상업용 운영체제 보안 보호 프로파일(Commercial- Off-the-Shelf OS Security Protection Profile), 국내의 국가기관용 등급기반 접근통제 보호 프로파일(Label-Based Access Control Protection Profile)[1,2] 등이 있으며, 이들은 각각 CC (Common Criteria)의 해당 보안등급에 맞추어 생성되었다. 이 기준에 의해서 개발된 보안 운영체제의 안전성을 입증할 수 있는 근거를 제시하기란

* 한서대학교 컴퓨터정보학과

접수일자 : 2006. 5. 24

그러 쉽지 않다. 안전성에 대한 직접적인 검증 방법은 운영체제 내의 모든 가능한 연산을 대상으로 실행 결과의 안전성 여부를 검사하는 것이지만, 이는 현실적으로 불가능하다. TCSEC (Trusted Computer System Evaluation Criteria)의 B2급 이상의 운영체제에서는 정형화 검증이 필수적인 요소이며 정형기법(formal method)을 사용할 경우 운영체제 동작 논리상의 안전성 보장 여부를 이론적으로 증명해 볼 수 있다[3]. 정형기법은 정형명세(formal specification)와 정형검증 (formal verification)으로 나눌 수 있다. 정형명세는 수학적 개념을 기초로 한 의미(semantics)와 제한된 구문(syntax)으로 구성된 언어를 이용하여 작성된다. 이러한 정형명세의 대표적 언어로는 Z, VDM, CSP, Petri Net, Lisp 등이 있다. 정형검증의 대표적인 도구로는 HDM, GVE, PVS, SMV, Spin 등이 있다. 본 논문은 보안 운영체제를 이루는 핵심 요소인 보안 커널에 대하여 특정 보안 정책을 설계하고 구현함에 있어서 반드시 필요한 안전성 검증을 위하여 정형명세 작성 및 검증 도구로 PVS(Prototype Verification System)를 선정하게 된 기준 및 배경을 설명하고, 현재 PVS 검증도구를 사용하여 수정된 BLP (Bell & LaPadula) 모델[4]의 안전성을 검증함으로써 구현 보안 커널의 안전성을 제시하고자 한다.

II. 검증 도구의 선정

도구들이 평가되는 이유는 도구들의 능력이 차이가 있다는 것에 기인하며, 도구의 정보는 각 도구에서 제공하는 홈페이지 정보를 기준으로 하였다[5]. 평가 대상의 도구는 현재 가장 많이 사용되는 ACL2, AutoFOCUS, Coq, Elf/Twelf, HOL, IMPS, Otter, PVS, SteP, TAME, TPS, Vienna, Z/Eves를 선정하였다. PVS는 SRI (Stanford Research Institute)에서 20년 이상의 경험을 토대로 제작되었으며 다양한 정보를 제공하고 있으며 명세 언어를 작성함에 다수의 미리 정의된 이론 (Predefined Theories)을 통해 작성된 명세의 이론검증이 가능하며 여러 가지 유용한 도구를 지원한다. 또한 명세를 작성하기 다양한 문서를 제공하며 사용자 이해를 돕기 위해 각 각의 응용 분야에서 시스템의 사용 방법을 설명하는 예제로 이루어져있다[7]. <표 1>는 본 논문에서 평가한 것으로 검증도구의 평가 요소를 중심으로 3 단계(좋음, 보통, 나쁨)로 평가 비교한 결과의 도식이다.

표 1. 검증 평가 비교
Table. 1 Comparison of Verifying Evaluation

	ACL2	Auto-FOCUS	Coq	Elf/Twelf	HOL	IMPS	Isabelle	Ketl	Otter	PVS	Step	TAME	TPS	Vienna	Z/Eves
	O	△	O	O	△	△	O	X	X	O	X	△	△	△	O
	O	△	O	△	△	△	O	O	△	O	△	△	△	△	O
	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
	△	△	O	△	O	△	△	△	△	O	O	△	O	△	O
	△	△	O	△	O	O	△	O	△	O	△	△	△	△	O
	O	△	X	O	O	O	△	O	O	O	O	△	△	△	△
	사용자 마다 다를 수 있다.														
	O	△	△	△	O	△	O	O	△	O	△	O	△	△	O
	O	△	O	△	O	△	△	O	△	X	△	△	△	O	△
	O	△	△	△	O	O	△	△	△	O	△	△	△	△	△
	O	△	O	O	O	O	O	O	△	O	△	O	O	△	O

(O:좋음, △:보통, x:나쁨)

언어적 측면에서 보면 다양한 표현이 가능한 풍부한 명세 언어와 효과적인 자동화로 추론적(deduction) 기능을 사용할 수 있으며, 데이터의 표현은 기초형에서부터 사용자로부터 작성된 형, 추상 데이터를 포함한다. 증명은 이론 증명 기능에 의해 자동적으로 수행 되며 이론증명은 다음의 순차적인 수학적 프레임워크 안에서 사용자의 명령으로 추론 과정 수집을 제공한다. 이런 원시적인 추론들은 제한적 또는 한정 기호 규칙, 유도, 재사용, 선형의 산술을 위한 알고리즘 등을 포함한다. 또한, PVS는 그 명세 언어와 증명의 완전한 인터페이스를 제공하기 위해 GNU 또는 X Emacs를 사용한다. 명령들은 풀다운 메뉴들에 의해 또는 확장의 Emacs 명령들에 의해 선택될 수 있다. 본 논문에서 검증도구로 선택된 PVS는 주로 필요조건들과 설계수준 명세들의 형식화와 복잡하고 어려운 문제들의 분석을 위해 사용한다. 주로 고장 방지능력이 있는 비행 컨트롤 시스템들의 문제 해결을 위한 단계적 수법들과 건축, 하드웨어와 리얼 타임 시스템 설계 등에 많이 적용되고 있으며, 국내에서는 원전분야 소프트웨어를 검증하는 분야에서 적용된 사례가 있다.

Ⅲ. 보안모델의 정형명세

3.1. BLP 모델

1973년에 발표된 BLP(Bell & LaPadula) 모델은 군사용 보안 모델에서 출발하여 정보흐름(information flow)에 대한 보안 기능을 강화한 정보 흐름 모델의 일종으로 안전한 시스템에서 정보 흐름의 허용 가능한 경로에 대해 기술하고 있다.

즉, 이 모델은 서로 다른 보안등급을 가지고 있는 데이터를 동시에 취급하는 시스템에서 필요한 보안 요구조건에 대해 정의하고 있다. BLP 모델의 구성은 주체의 집합(S)과 객체의 집합(O), 주체와 객체 각각에 대한 보안등급(C)으로 구성 된다. BLP 모델은 다중등급 보안을 위한 가장 기본적인 모델이다. 주체와 객체에 각각 보안등급을 부여하여 주체의 객체접근 허용 여부를 일차로 점검하고 접근이 허용된 여러 객체간의 보안등급을 고려하여 상호간의 정보 교환을 감시함으로써 허용 되지 않는 정보 흐름을 방지한다. 시스템은 주체의 집합 S와 객체의 집합 O로 구성되며, S에 속한 모든 주체 s와 O에 속한 모든 객체 o에 대해 고정된 보안등급 C(s), C(o)가 존재한다. BLP 모델은 시스템 요소와 시스템의 상태를 변경시키는 규칙들을 정형적으로 정의한 상태전이 모델이다. BLP 모델은 정보의 불법유출 방지를 위한 보안정책으로 채택하고 있으며, 안전한 정보시스템의 설계를 위한 기준이 되고 있다. 이 모델은 TCSEC에서 보안정책의 기본으로 사용되고 있다. 운영체제에서 프로세스는 유일한 주체이다. 프로세스는 객체를 생성, 삭제하고, 액세스 매트릭스 내의 액세스 모드와 현재 액세스 집합의 내용을 변경시킨다. 객체에 대한 프로세스의 모든 액세스 취득 행위는 3가지 보안 특성을 만족할 경우에만 허용된다. 프로세스의 보안등급은 사용자의 보안 등급을 계승하며 이는 주체의 최대 보안등급과 현재 보안등급으로 구성된다. 운영체제에서 객체로 인식되는 것은 사용자가 생성하는 파일, 디렉터리와 특수한 파일인 디바이스 등이 있으며, 그 이외에 파이프, 메시지 큐, 공유메모리 등도 있다. 이 객체의 특성은 모델의 액세스 모드를 해석하는데 중요한 고려 사항이 된다. 즉 이들의 특성에 따라 모델의 액세스 모드는 의미가 달라진다. BLP 모델에서의 액세스 모드는 read, append, execute, write의 네 가지 종류가 있으며, BLP 모델의 액세스 모드는 실제 운영체제에서의 액세스모드와 완전히 1 : 1로 매핑 되지 않으며, 그 이유는 객체의 특성 때문이다. 현재 액세스 집합 b는 임의 주체가 임의 객체에 대하여 DAC(Discretionary Access Control)와 MAC(Mandatory Access Control)에 의해 허가 받은 액세스 모드를 의미한다. 운영체제에서 개인 단위로 DAC의 제어를 가능케 하도록 하기 위해 ACL(Access Control List)이 도입되었다. 보안 함수는 주체의 최대 보안 등급, 현재 보안등급의 할당과 객체의 보안 등급 할당을 담당한다. 서로 다른 보안등급을 가지고 있는 데이터를 다루는 시스템에서는 불법적인 정보 유출을 막기 위하여 필요한 보안 요구 조건에 대하여 정의하며, 기본적인 성질은 다음과 같다.

① 주체 S는 객체 O를 오직 Clearance(S) >= Clearance(O) 일 경우에만 읽을 수 있다.
(Simple Security : ss-Property).

② 주체 S는 객체 O를 오직 Clearance(S) =< Clearance(O) 일 경우에만 쓸 수 있다.
(Star : *-Property).

원래의 BLP 모델은 ②에서와 같이 Clearance (S) < Clearance(O)일 경우에도 쓰기(write) 연산을 허용하였다. 이 쓰기 연산은 객체의 변경(modify), 생성(create), 삭제(delete), 붙여 쓰기(append) 등의 포괄적인 연산이다. 즉, [그림 1]과 같이 표현될 수 있다. 그러나 낮은 등급의 주체가 더 높은 등급의 객체를 붙여 쓰기, 삭제가 가능하다는 것은 현실적으로 보안상 문제가 발생되므로, 쓰기 연산은 제한되어야 한다. 따라서, 많은 보안 운영체제의 보안 커널 구현에서는 ②'와 같이 수정하여 적용하고 있다.

②' 주체 S는 객체 O를 오직 Clearance(S) = Clearance(O) 일 경우에만 쓸 수 있다
(수정된 Star : *-Property).

즉, 주체 S는 객체 O를 등급이 같은 경우에만 read/write가 가능하고, 객체보다 등급이 높은 경우에는 낮은 등급의 객체를 read만 가능하다. 주체는 자신보다 높은 객체에 접근할 수 없다. 이런 BLP 모델의 성질을 시스템에 적용하여 더 높은 등급의 정보를 읽는다거나 더 높은 등급의 주체가 더 낮은 등급의 객체에 정보를 쓰는 것을 방지하여 정보의 불법적인 흐름을 차단하게 된다. 수정된 BLP 모델은 [그림 2]와 같이 표현될 수 있다.

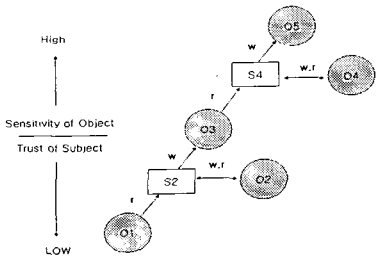

그림 1. 원래의 BLP 모델
Fig. 1. Original BLP Model

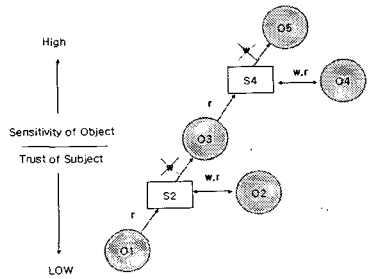

그림 2. 수정된 BLP 모델
Fig. 2. Modified BLP Model

3.2. BLP 모델의 정형명세

보안 모델의 정형명세 작성하려는 운영체제에 대한 정보나 지식 등을 우선 숙지해야한다. 첫 단계는 작성하고자 하는 논리나 명제를 이해하고. 둘째 단계는 작성된 논리나 명제를 수학적으로 표현해야 한다. 마지막 단계로, 정형 검증도구로 모델을 명세하고 검증해야한다. 어떤 사실의 원인을 설명하거나 어떤 이론 체계를 연역하기 위하여 가설을 설정하게 되는데 PVS 또한, ASSUMING이라는 문법을 통해 BLP 모델의 증명될 수 있는 근본적인 전제조건을 설정하게 된다. 그 결과 가설의 타당성이 판단되면 최초의 가설적 대상은 이론으로 인식된다. 우선 BLP 모델의 전반적인 모델의 흐름을 가설로 명세하고자 한다. 매개체로는 Access_State, SecState, Seq, Transform, Transform Instance로 지정한다. Access_State는 주체가 개체에게 접근하는 상태를 표현하며 SecState는 BLP 모델의 보안 상태를 표현한다. Seq는 주체가 개체에게 접근하는 순서적 의미를 포함하며 Transform은 주체가 개체에게 접근하는 함수를 표현하며 TransformInstance는 Transform에 접근하는 인스턴스를 표현하였다. BLP 모델의 흐름은 [그림 3]과 같이 명세하였다. 이 명세에서는 BLP 모델의 주체 및 객체를 통하여 명세한 가설에 접근함으로써 작성된 명세를 증명하고자 한다.

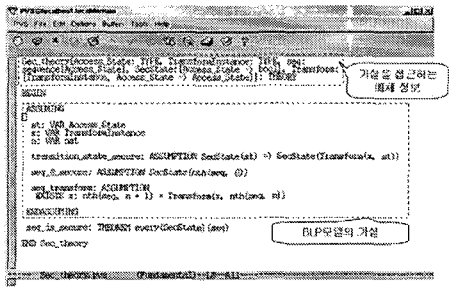

그림 3. BLP 모델 흐름
Fig. 3. Flow of the BLP Model

BLP 모델 상태변화는 주체의 명령 혹은 주체가 객체에 접근할 때마다 BLP 모델의 상태가 변화하게 된다. 시스템의 상태를 명세하기 위해서 보안 모델의 ss-Property, *-Property를 명세하고 두 가지 공리를 표현하기 위해 주체와 객체의 보안 등급을 표현할 필요성이 있다. 실제 환경에서는 운영체제의 관리자가 다양한 등급으로 나누어 표현할 수 있지만, 본 연구에서는 총 4 단계로 등급을 표현하였으며, 주체가 객체로의 접근 방식을 두 가지로 나누었다. 실행 및 삭제를 표현하는 "Action", 그리고 읽기, 삭제를 표현하는 "Mode"로 표현했다. BLP 모델의 상태변화의 매개체는 Subject, Object, slSubject, slObject, Action, addit, del, no_op, Mode, rd, wr로 지정하고 3개의 가설 보안 모델을 공리를 명세하고 BLP 모델의 흐름을 참조(Import)함으로써 보안모델의 상태는 BLP 모델의 흐름아래에서 진행할 수 있도록 한다. [그림 4]는 수정된 BLP 모델의 시스템 상태변화를 명세한 것이다.

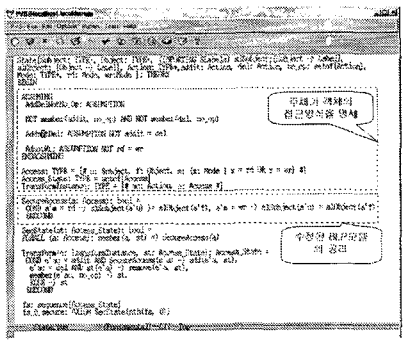

그림 4. 수정된 BLP 모델의 시스템 상태변화
Fig. 4. System status flow of the Modified BLP Model

Ⅳ. 보안모델의 정형검증

4.1. BLP 모델의 정형검증

모든 정형 명세가 완료되었다면 정형 검증을 통해 이를 검증한다. PVS의 정형 검증은 Parsing, Type Checking, Prove 등의 3 단계로 구성되어 있다. 먼저 증명하려는 정형 명세를 작성한 다음 Parsing을 통해 문장의 문법을 검사한다. 문법의 오류가 없다면 Type Checking을 통해 작성된 명세에서 증명하려는 이론이나 자료형의 일관성을 요구하게 된다. 증명의 필요한 이론들은 Type CorreCtness (TCCs)이라고 하며 TCCs는 증명 명령을 통해 TCCs 일관성을 유지할 수 있도록 한다. 일관성이 유지되었다면 다양한 명령을 통해 정형 명세를 증명하게 된다. [그림 5]는 BLP 모델의 정형 검증 절차를 나타낸 것이며, 검증 절차는 그림의 오른쪽 검증 트리를 통해 알 수 있다. 수정된 BLP 모델의 경우에도 동일하다. BLP 모델의 검증절차의 세부사항은 [그림 6]에서 설명하고 있으며, 검증 시에 발생하는 TCCs는 TCCs 증명 명령을 통해 일관성을 유지 할 것이다.

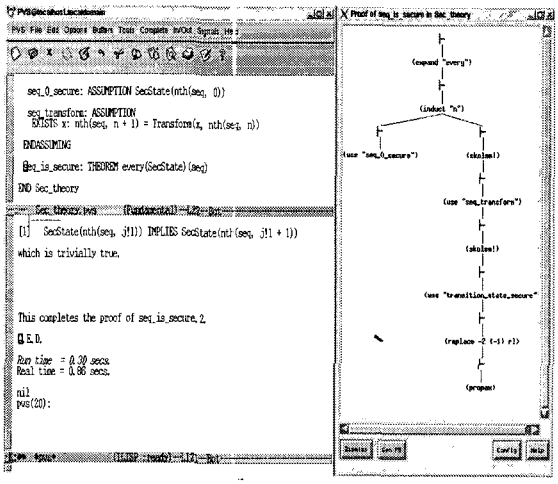

그림 5. BLP 모델의 정형 검증 절차
Fig. 5. Formal Verification Steps for BLP Model

```
seq_is_secure :
|-------
{1}   every(SecState)(seq)
seq_is_secure :
|-------
{1}   every(SecState)(seq)
Rule? (expand "every") %every 이론을 확장함
Expanding the definition of every,
this simplifies to:
seq_is_secure :
|-------
{1}   FORALL n: SecState(nth(seq, n))%SecState로 구성됨

Rule? (induct "n") %n으로부터 유도함
|-------
Inducting on n on formula 1,
this yields  2 subgoals:
seq_is_secure.1 :
{1}   SecState(nth(seq, 0)) % 0일 경우
Rule? (use "seq_0_secure")
Using lemma seq_0_secure,
This completes the proof of seq_is_secure.1.
seq_is_secure.2 :
|-------
{1} FORALL j: SecState(nth(seq, j)) IMPLIES SecState(nth(seq, j
+ 1))
Rule? (skolem!) %한정기호를 통해 j를 한정한다.
Skolemizing,
this simplifies to:
seq_is_secure.2 :
  |-------
{1}SecState(nth(seq, j!1)) IMPLIES SecState(nth(seq, j!1 + 1))
Rule? (use "seq_transform") %Seq_transform을 통해 유도함
Using lemma seq_transform,
this simplifies to:
seq_is_secure.2 :
{-1} EXISTS x: nth(seq, j!1 + 1) = Transform(x, nth(seq, j!1))
  |-------
[1]SecState(nth(seq, j!1)) IMPLIES SecState(nth(seq, j!1 + 1))
Rule? (skolem!) %한정기호를 통해 nth를 한정함
Skolemizing,
this simplifies to:
seq_is_secure.2 :
{-1}  nth(seq, j!1 + 1) = Transform(x!1, nth(seq, j!1))
  |-------
[1]SecState(nth(seq, j!1)) IMPLIES SecState(nth(seq, j!1 + 1))
Rule? (use "transition_state_secure")
%위의 결과는 transition_state_secure을 통해 유도함
Using lemma transition_state_secure,
this simplifies to:
seq_is_secure.2 :
{-1}    SecState(nth(seq,  j!1))  =>  SecState(Transform(x!1,
nth(seq, j!1)))
[-2]  nth(seq, j!1 + 1) = Transform(x!1, nth(seq, j!1))
  |-------
[1]SecState(nth(seq, j!1)) IMPLIES SecState(nth(seq, j!1 + 1))
Rule? (replace -2(-1) rl) % -1,-2 의 결과는 삭제
Replacing using formula -2,
this simplifies to:
seq_is_secure.2 :
{-1}  SecState(nth(seq, j!1)) => SecState(nth(seq, j!1 + 1))
[-2]  nth(seq, j!1 + 1) = Transform(x!1, nth(seq, j!1))
  |-------
[1]SecState(nth(seq, j!1)) IMPLIES SecState(nth(seq, j!1 + 1))
which is trivially true.
This completes the proof of seq_is_secure.2.
Q.E.D. %증명완료
```

그림 6. BLP 모델의 흐름 검증과정
Fig. 6. Verification Steps for BLP Model

검증이 완료되면 "Q. E. D"로 표시된다. "Q. E. D".란 Latin어로 Quod erat demonstrandum(증명완료), 즉 "Proved"로서 검증이 완료되었다는 의미이다. 이로써 수정된 BLP 모델은 작성된 정형명세에 의해서 검증이 완료되었으며, 안전성의 문제가 없이 증명되었음을 의미한다. 이렇게 보안 모델의 흐름이 안전하다는 것을 검증하였다면 보안 모델의 흐름과 매핑되어 있는 보안모델의 상태변화에 대해 정형명세가 일관성을 유지해야 한다. 보안 모델 상태에서 발생하는 TCCs는 [그림 7]과 같다.

4.2. BLP 모델의 TCCs 검증

이론 증명에서는 발생된 TCCs에 대해 일관성을 유지할수 없기 때문에 발생된 TCCs는 일관성 유지 명령으로 일관성을 유지할 수 있다. 일관성이 유지되지 않는다면 BLP 모델은 검증이 불가능하므로 "BLP 모델의 상태변화" 명세에서 발생된 TCCs 증명을 통하여 일관성을 유지한다. [그림 8]은 "BLP 모델의 상태흐름"에서 발생된 TCCs의 세부사항이이며 수정된 BLP 모델과 원래의 BLP 모델의 상태변화에 대한 TCCs를 표기하였다.

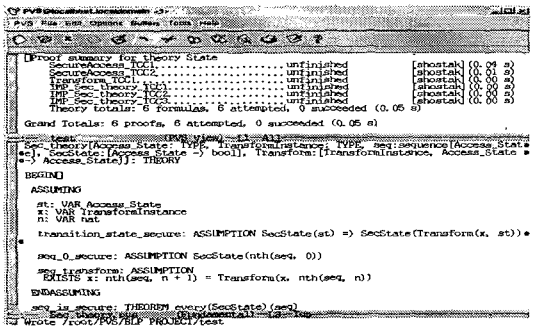

그림 7. BLP 모델의 상태변화 명세에서 발생한 TCCs
Fig. 7 TCCs of Status Specification for BLP Model

```
%수정된 BLP 모델
% Disjointness TCC generated (at line 20, column 2) for
% COND a`m = rd -> slSubject(a`u) >= slObject(a`f),
%      a`m = wr -> slSubject(a`u) = slObject(a`f)
% ENDCOND
% unfinished
SecureAccess_TCC1: OBLIGATION FORALL (a: Access): NOT (a`m =
rd AND a`m = wr);
%원래의 BLP 모델
% Disjointness TCC generated (at line 20, column 2) for
% COND a`m = rd -> slSubject(a`u) >= slObject(a`f),
%      a`m = wr -> slSubject(a`u) =< slObject(a`f)
% ENDCOND
% unfinished
SecureAccess_TCC1: OBLIGATION FORALL (a: Access): NOT (a`m =
rd AND a`m = wr);
% Coverage TCC generated (at line 20, column 2) for
% COND a`m = rd -> slSubject(a`u) >= slObject(a`f),
%      a`m = wr -> slSubject(a`u) = slObject(a`f)
% ENDCOND
% unfinished
SecureAccess_TCC2: OBLIGATION FORALL (a: Access): a`m = rd OR
a`m = wr;
% The disjointness TCC (at line 20, column 2) in decl
SecureAccess for
% COND a`m = rd -> slSubject(a`u) >= slObject(a`f),
%      a`m = wr -> slSubject(a`u) = slObject(a`f)
% ENDCOND
% was not generated because it simplifies to TRUE.
% The coverage TCC (at line 20, column 2) in decl SecureAccess
for
% COND a`m = rd -> slSubject(a`u) >= slObject(a`f),
%      a`m = wr -> slSubject(a`u) = slObject(a`f)
% ENDCOND
% was not generated because it simplifies to TRUE.
% Disjointness TCC generated (at line 27, column 2) for
% COND e`ac = addit AND SecureAccess(e`a) -> add(e`a, st),
%      e`ac = del AND st(e`a) -> remove(e`a, st),
%      member(e`ac, no_op) -> st,
%      ELSE -> st
% ENDCOND
% unfinished
Transform_TCC1: OBLIGATION
  FORALL (e: TransformInstance, st: Access_State):
    NOT ((e`ac = addit AND SecureAccess(e`a)) AND e`ac = del
 AND st(e`a)) AND NOT ((e`ac = addit AND SecureAccess(e`a))
 AND member(Action) (e`ac, no_op))
     AND NOT ((e`ac = del AND st(e`a)) AND member(Action)
(e`ac, no_op));
% The disjointness TCC (at line 27, column 2) in decl Transform
for
% COND e`ac = addit AND SecureAccess(e`a) -> add(e`a, st),
%      e`ac = del AND st(e`a) -> remove(e`a, st),
%      member(e`ac, no_op) -> st,
%      ELSE -> st
% ENDCOND
% was not generated because it simplifies to TRUE.
% Assuming TCC generated (at line 44, column 12) for
% Sec_theory[Access_State, TransformInstance, fs, SecState,
Transform]
%    generated from assumption Sec_theory.transition_
state_secure
% unfinished
IMP_Sec_theory_TCC1: OBLIGATION
  FORALL (st: Access_State, x: TransformInstance):
    SecState(st) => SecState(Transform(x, st));
% Assuming TCC generated (at line 44, column 12) for
% Sec_theory[Access_State, TransformInstance, fs, SecState,
Transform]
% generated from assumption Sec_theory.seq_0_secure
% unfinished
IMP_Sec_theory_TCC2:  OBLIGATION  SecState(nth[Access_State]
(fs, 0));
% Assuming TCC generated (at line 44, column 12) for
% Sec_theory[Access_State, TransformInstance, fs, SecState,
Transform]
% generated from assumption Sec_theory.seq_transform
% unfinished
IMP_Sec_theory_TCC3: OBLIGATION
  FORALL (n: nat):
    EXISTS (x: TransformInstance):
      nth[Access_State](fs,  n  +  1)  =  Transform(x,  nth
[Access_State](fs, n));
```

그림 8. BLP 상태변화 명세에서 발생한 TCCs 세부사항
Fig. 8 TCCs Specification from the BLP Status Variation Specification

[그림 8]에서 발생한 TCCs는 검증명령을 통해 명제의 일관성을 유지할 수 있다. [그림 9]는 [그림 8]에서 발생한 TCCs를 검증한다. 원래의 BLP 모델과 수정된 BLP 모델의 TCCs는 모두 동일하게 검증되었다. 그 이유는 수정된 BLP 모델(즉, slSubject(a`u) = slObject(a`f))과 원래의 BLP 모델 (즉, slSubject(a`u) =< slObject(a`f))의 차이점인 *-Property의 주체와 객체 사이에서 "같음"과 "같거나 작음"의 차이가 TCCs 검증에 위배되지 않기 때문이다.

추가적으로 BLP 모델의 상태변화에서는 원래의 BLP 모델과 수정된 BLP 모델 그리고 임의로 수정된 BLP 모델을 TCCs를 통하여 검증해 보았다. 원래의 BLP 모델과 수정된 BLP 모델은 모두 일관성이 유지 되었으나, 임의로 수정하여 작성된 BLP 모델은 TCCs의 검증이 불가능하였으며, 일관성이 유지되지 않았다. 위의 결과로 수정된 BLP 모델은 원래의 BLP 모델과 마찬가지로 수학적으로 검증이 완료되었으며 안전한 운영체제에 커널에 사용될 모델에 적합하다는 것을 검증하였다.

```
SecureAccess_TCC1 :
  |-------
{1}   FORALL (a: Access): NOT (a`m = rd AND a`m = wr)
Rule? (skolem!) %한정기호를 사용하여 식을 간단히 바꿈
Skolemizing,
this simplifies to:
SecureAccess_TCC1 :
{-1}  a!1`m = rd AND a!1`m = wr
  |-------
Rule? (use "RdnotWr") %Read와 Write는 같지 않음을 지정
Using lemma RdnotWr,
this simplifies to:
SecureAccess_TCC1 :
[-1]  a!1`m = rd AND a!1`m = wr
  |-------
{1}   rd = wr

Rule? (flatten) %위의 식을 분할함
Applying disjunctive simplification to flatten sequent,
this simplifies to:
SecureAccess_TCC1 :

{-1}  a!1`m = rd
{-2}  a!1`m = wr
  |-------
[1]   rd = wr
Rule? (simplify) %식을 간단히 함

Simplifying with decision procedures,
this simplifies to:
SecureAccess_TCC1 :

[-1]  a!1`m = rd
[-2]  a!1`m = wr
  |-------
[1]   rd = wr
Rule? (grind)
Trying repeated skolemization, instantiation, and if-lifting,
Q.E.D.
Run time  = 0.42 secs.
Real time = 69.38 secs.
```

그림 9. BLP 모델의 상태변화 명세의 TCCs 검증 화면
Fig. 9 TCCs Verification for the BLP Status Variation Specification

V. 결 론

정보 산업이 발전함에 따라 컴퓨터 하드웨어 및 소프트웨어 시스템의 개발 주기가 매우 짧아지고 있다. 또한, 시스템의 동작 역시 예전의 중앙집중형 구조에서 분산형으로, 그리고 객체지향형 프로그래밍 언어에 의한 개발로 발전하고 있다. 그리고 내장형 시스템과 같이 크기는 매우 작지만 하드웨어 및 소프트웨어가 유기적으로 연결되어 기능을 발휘하는 시스템이 활발히 연구되고 있다. 이러한 발전에 따라 시스템의 동작상의 안정성을 보장하기 위한 노력이 꾸준히 이뤄지고 있다. 전통적으로 안전한 시스템을 설계하기 위해 테스트와 같은 방법이 주로 사용되어 왔다. 그러나 이 방법은 일단 시스템을 설계한 후 수행되고, 테스트에 사용할 시나리오에 따라 그 신뢰도가 결정된다. 따라서 최근에는 시스템을 설계하는 단계에서 안정성을 증명하는 정형 기법에 대한 연구 및 정형 검증 도구가 많이 개발되고 있다. 그러나 정형 검증에 필요한 지식이 매우 이론적이고, 일반 설계자들에게는 어렵게 느껴지는 것이 사실이다.

본 논문에서는 보안 운영체제를 개발함에 있어 필수적으로 제기되는 보안 커널의 안전성을 검증하기 위해서 커널 구현시 적용되는 수정된 BLP 모델에 대하여 PVS라는 정형 검증 도구를 통해서 정형 명세를 작성하고 그 안전성을 검증하였다. 이러한 정형 명세를 통하여 보안 모델뿐만 아니라 보안 운영체제 전체에 적용시킨다면 좀 더 안정성있는 안전한 컴퓨터 시스템을 개발할 수 있을 것이다. 따라서 향후 연구에서는 수정된 BLP 모델의 검증과 아울러 실제 보안 운영체제의 실제 환경에 맞추어 세부 구현 내용까지 포함할 수 있는 명세를 작성하여 검증할 예정이다.

참고문헌

[1] 박 태규, "보안 리눅스(Secure Linux) 연구개발 동향," 情報保護學會誌 13권,4호, pp37-48, 2003.

[2] 한국정보보호진흥원, 국가기관용 등급기반 접근통제 보호 프로파일, 2004. 2. 26.

[3] Bell. D. and LaPadula., "Secure Computer System : Mathematical Foundations and Model," MITRE Report MTR 2547, v2, Nov. 1973.

[4] 티에스온넷(주), RedOwl SecuOS Administration Guide, 2004.

[5] http://pvs.csl.sri.com/introduction.shtml

[6] White, G., Fisch, E., Pooch, U., "Government- based security standards," Auerbach Publications Inf. Syst. Secur. (USA), 6권, 3호, pp9-19, 2003.

[7] 김 의탁 "접근통제 기술 동향," 通信情報保護學會 8권, 4호, pp77-96, 1998.

[8] Andrews, Peter. System Description: TPS: A Theorem Proving System for Type Theory, 2000.

저자소개

구 하 성(Ha-Sung Koo)

1991년 광운대,전자통신공학,석사
1995년 광운대,전자통신공학,박사
1995-97년 기아정보시스템
1997-현재 한서대학교 컴퓨터 정보학과 부교수
※관심분야 : 영상처리, 생체인식, 생체보안등.

박 태 규(Tae-Kue Park)

1989년 충남대,전산학과,석사
1996년 성균관대,정보공학과,박사
1981~82년한국국방연구원연구원
1982~92년 ETRI 선임연구원
1997~98년 Univ. of W. Sydney(Post-doc)
1992년~현재 한서대학교 컴퓨터 정보학과 교수
※관심분야 : 보안운영체제, 네트워크 보안등.

송 호 근(Ho-Keun Song)

1993년 중앙대,전자공학과,석사
1997년 중앙대,전자공학과,박사
1996-현재 한서대학교 컴퓨터 정보학과 부교수
※관심분야 : 3차원영상처리/인식, 영상보안 등.