정형 검증 도구를 이용한 보안 소프트웨어 개발 방안

(A Security Software Development Methodology Using Formal Verification Tools)

장 승 주 †

(Seung-Ju Jang)

요 약 본 논문에서는 정형 검증 도구를 이용한 보안 소프트웨어의 보안성을 검증하고 검증된 결과를 이용하여 안전한 보안 소프트웨어 개발 방안을 살펴본다. 정형 검증 기법에 따라 사용되어질 수 있는 각 정형 검증 도구들을 살펴보고 그 특성을 파악하여 보안 소프트웨어의 안정성과 신뢰성을 보장하는 적합한 도구들을 알아보고, 보안 소프트웨어 개발 방안을 제시한다. Z 언어를 이용하는 도구인 Z/EVES는 Z명세에 대한 분석 방법을 제시함으로써 보안 소프트웨어에서 가장 많이 사용되는 대표적인 정형 검증 도구이다. 본 논문에서는 보안 소프트웨어 검증을 위하여 C 언어로 작성된 보안 소프트웨어를 Z명세 언어로 변환하고 이것을 Z/EVES도구를 이용한다.

키워드 : 정형 방법, 정형 검증, RoZ, Z/EVES, 보안 S/W의 검증

Abstract This paper suggests method of safe security S/W by verifying and its result of formal verification tool. We will survey many formal verification tools and compare features of these tools. And we will suggest what tool is appropriate and methodogoly of developing safe security S/W. The Z/EVES is the most appropriate tool. This paper proposes formal verification of ACS by using RoZ tool which is formal verification tool to create UML model. The specification and verification are executed using Z/EVES tool. These procedures can find weak or wrong point of developed S/W.

Key words : formal method, formal verification, RoZ, Z/EVES, verification of security S/W

1. 서 론

정형 명세 기법을 적용한 보안 소프트웨어 개발 방안에서는 시스템의 안전성을 보장하는 정형화된 표준 기술이 필요하다. 소프트웨어의 정형화된 명세 규격을 정의할 때에는 SPEAR II 또는 IFAD VDM-SL과 같은 도구 등을 이용하여 구현하고자 하는 보안 소프트웨어의 개발이 가능하다. SPEAR(Security Protocol Engineering and Analysis Resource) II는 안전하고 효율적인 보안 프로토콜을 설계할 수 있고[1], IFAD VDM-SL Toolbox는 ISO VDM-SL 표준을 사용한 정형 명세의 개발을 지원하는 도구이다[2]. 정형 명세 과정을 거친 소프트웨어가 안전성을 갖춘 보안 소프트웨어라고 볼 수는 없다. 정형 검증 기법을 지원하는 도구들을 통하여 더욱 더 보안성을 잘 갖춘 소프트웨어 개발 과정을 실행할 수 있다. 검증 과정에서 정형화된 명세 코드는 보안 검증 도구를 이용하여 검증할 수 있다. 검증 단계에서 지원하는 도구들도 여러 가지가 있다. SPIN[3], SLAM[4], SMV[1], Z/EVES[6] 등과 같은 검증 도구를 이용할 수 있다.

본 논문에서는 보안 소프트웨어 개발에 사용되고 있는 SPIN, SLAM, SMV, Z/EVES 와 검증 도구 특징들을 살펴보고, 이들 도구를 이용한 안전한 보안 소프트웨어 개발 방안을 제시한다. 기존의 보안 소프트웨어에 대한 정형 검증 기법은 SPEAR-II나 VDM-SL을 이용하여 이루어지고 있다. SPEAR-II나 VDM-SL은 제한된 환경에서 정형 검증이 되는 문제점을 가지고 있다. 그나마 보안 소프트웨어 검증 방안에 대한 연구의 진행이 부진한 상태이다. 또한 기존의 보안 소프트웨어 검증 방안에 대한 연구는 모델링의 어려움으로 인해 검증 자체가 쉽지 않다. 본 논문은기존 연구의 이러한 문제점을 해결하고 보다 보안 소프트웨어 검증을 보다 용이하게 할 수 있는 방안을 제안한다. 본 논문에서는 보안 소프

† 정 회 원 : 동의대학교 컴퓨터공학과 교수
sjjang@deu.ac.kr
논문접수 : 2005년 8월 11일
심사완료 : 2005년 12월 13일

트웨어에 대한 방안을 제시하기 위하여 보안 소프트웨어 개발에 적합한 도구들의 기능을 상호 비교한다. 그리고 보안 소프트웨어에서 정형 검증 도구를 이용한 개발 방안을 제시한다. 그리고 마지막으로 제시된 보안 정형 검증 도구인 Z/EVES를 이용한 검증 방안에 대한 구체적인 방법을 제시한다.

본 논문에서 제안하는 보안 소프트웨어 검증 방안으로는 정형 검증 도구 중에서 가장 많이 사용되고 있는 Z/EVES를 이용한다. Z/EVES는 보안 소프트웨어 개발에 대한 가장 대표적인 정형 검증 도구이다. 본 논문에서는 보안 소프트웨어 검증을 위하여 C 언어로 작성된 보안 소프트웨어를 Z명세 언어로 변환하고 이것을 Z/EVES도구를 이용한다.

본 논문의 구성은 2장에서는 정형 기법의 관련 연구를 설명하고, 3장에서는 보안 소프트웨어 검증을 지원하는 도구들에 대해 설명을 하며, 4장에서는 보안 소프트웨어 설계 언어의 정형 검증 도구를 통한 보안 소프트웨어 개발 방안에 대해 설명하며, 5장에서 결론을 맺는다.

2. 관련연구 및 보안 소프트웨어 검증을 지원하는 도구

정형 기법은 수학과 논리학에 기반을 둔 방법으로 하드웨어 시스템이나 정형 소프트웨어 시스템을 명세하거나 검증하는 방법론이다. 수학적 기호를 사용하여 시스템을 명세하고 검증할 특성 또한 논리식을 통해 기술하여 시스템의 사용자가 요구하는 특정 조건에 대한 만족 여부를 수학적 성질을 이용하여 검증함으로써 자연어가 내포하는 애매모호함이나 불확실성을 최대한 줄일 수 있다. 즉, 복잡한 시스템이 특정 조건을 만족하는지를 검증하여 검증된 시스템에 대한 신뢰성을 가지게 할 수 있다[4,19].

정형 기법은 정형 명세(formal specification)와 정형 검증(formal verification)의 두 가지로 나눌 수 있는데, 정형 명세는 정형 논리(formal logic) 또는 수리 논리(mathematical logic)등을 이용하여 시스템이 동작할 환경, 시스템이 만족해야 할 요구 사항, 요구 사항을 수행할 시스템 설계 등을 기술하는 것이다. 정형 명세는 다시 요구 명세와 설계 명세로 나눌 수 있다. 요구 명세는 시스템이 무엇을 만족해야 하는가를 정의해 놓은 명세이고 설계 명세는 시스템이 어떻게 이루어져 있는가를 나타낸다. 정형 검증은 정형 논리 또는 수리 논리 등에서 제공하는 증명 방법을 이용하고, 정형 명세를 분석하여 시스템의 무모순성 및 완전성을 검증하거나, 설계가 주어진 가정에서 요구사항이 만족하는지를 검증하는 기법이다[4].

2.1 Bell Lab의 SPIN과 Feaver

Bell Lab에서는 Lucent에서 개발하는 교환기 및 각종 시스템의 안정성을 보장하기 위해 많은 연구를 수행해 오고 있다. 그 중 가장 눈에 띄는 연구 결과가 SPIN이다. SPIN은 AT&T Bell 연구소에서 1995년에 발표한 LTL model checker이다. SPIN은 비동기 프로세스 시스템을 설계하고 검증하는데 사용되는 모델체커이며, 하드웨어에서 요구되는 동기 시스템보다는 소프트웨어에서 사용되어지는 비동기적 프로세스 시스템(asynchronous process system)의 설계와 검증을 지원하는 가장 일반적인 tool이며, 프로세스 상호작용(Process interaction)의 정확성에 초점을 둔다[3].

SPIN은 기존의 모델 체킹 방법을 개선하기 위해 그림 2와 같이 On-the-Fly 방식을 사용하여 메모리의 효율적인 사용을 가능하도록 하였다. SPIN은 명세하고자 하는 시스템의 동작을 오토마타와 같은 형식으로 모델링한다. 이 때 사용하는 언어가 Promela(PROtocol MEta LAnaguage)이다[8,9].

Promela는 시스템을 프로세스의 형태로 추상화 하고, 프로세스들이 병렬적으로 수행하면서 채널을 통해 서로 간에 동기화 및 통신을 수행하게 된다. 그 결과 시스템의 각 프로세스가 동작하는 모습을 화면으로 매우 쉽게 확인할 수 있다. 범용 프로그래밍 언어로 구현된 시스템을 Promela로 모델링 할 때 모든 것이 완벽히 추상화 되는 것이 아니다. 따라서, 설계자들에 의해 구현된 소스코드를 검증하기 위해서는 매우 많은 수작업이 필요하다.

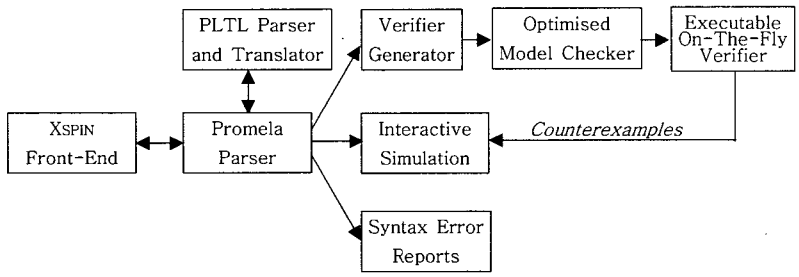

그림 1 SPIN Simulation 구조와 Verification

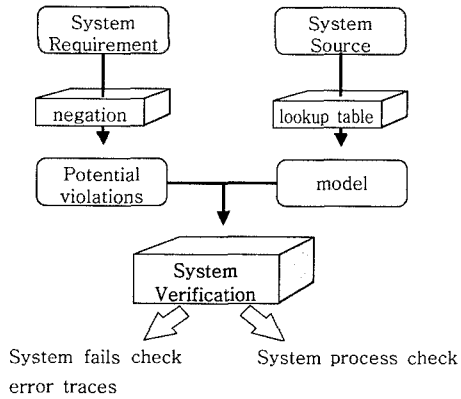

그림 2 Feaver의 동작 과정

2.2 Microsoft의 SLAM 프로젝트

Microsoft에서는 Microsoft에서 제작하는 많은 소프트웨어의 정확성을 검증하기 위해SLAM Project를 수행하고 있다. SLAM은 C 언어로 설계하는 소프트웨어를 대상으로 하고 있다. SLAM 프로젝트에서는 소프트웨어의 안정성 증명을 위한 사용자의 개입이나 추상화 작업을 필요로 하지 않는다. 대신 C 언어로 구현된 코드를 추상화한 Boolean 프로그램을 자동으로 추출하고, 그 추상 모델이 특성을 만족하는지를 증명하는 방법을 개발하고 있다[10,17].

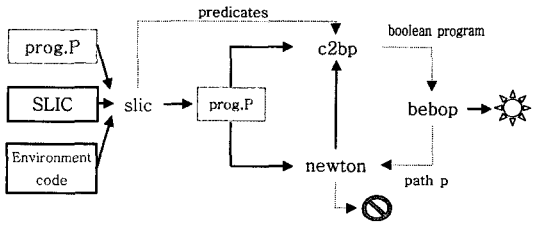

그림 3 SLAM Toolkit의 동작

SLAM toolkit은 논리적인 증명 방법을 사용한 검증을 수행하지는 않는다. 다만 Boolean 프로그램을 분석하고 순회하여 오류 상태에 도달할 수 있는 도달 가능성 분석과 같은 방법을 이용하여 그 안정성을 테스팅해 보는 방법을 사용하는 것이다. 그러나, 그 과정이 완전히 자동화 되어 있기 때문에 인간의 개입에 따른 오류를 막을 수 있고, 수행에 걸리는 시간적인 낭비를 줄일 수 있기 때문에 매우 좋은 연구로 평가받고 있다.

2.3 Bandera 프로젝트

Bandera project는 미국의 Kansas 주립대학의 SAnTOs 연구실을 중심으로 수행되는 것으로 객체지향형 언어인 Java로 구현된 소프트웨어의 정확성을 검증하는

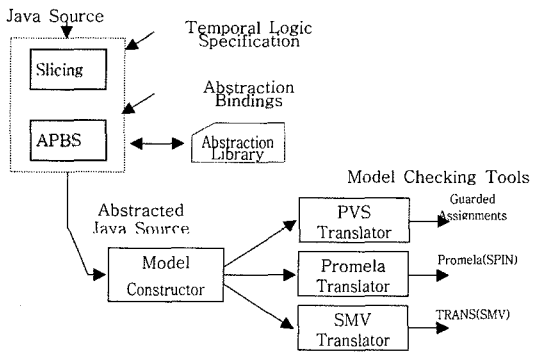

그림 4 Bandera toolset의 구조

것을 그 목표로 하고 있다. 이 도구는 자바 소스코드를 분석하고 검증언어로 변환하며 모델검증을 하기 위한 이 도구는 소스 코드의 변환과 검증이 자동으로 이뤄지기 때문에 사용자의 추가적인 개입이 없다. 결국 코드의 변환과정에서 생길 수 있는 오류를 배제할 수 있다. 또한, 오류의 발생시 역추적을 명세단계가 아닌 자바 소스코드의 레벨에서 보여주기 때문에 실제 코드의 수정을 통한 오류 수정이 가능하다. JPF II는 Bandera에서 변환한 검증할 자바코드와 역시 자바 코드로 작성된 검증할 속성을 받아들여 프로그램이 속성을 만족하는지 여부를 알려주고, 만족하지 않을 경우에는 반례를 보여준다.

2.4 SMV(Symbolic Model Verifier)

SMV 시스템은 유한 상태 기계가 *temporal logic CTL*로 쓰여진 명세가 만족하는지를 검사하는 도구이다. SMV의 입력언어는 유한 상태 기계를 묘사할 수 있도록 설계되었다. SMV는 검증하고자하는 시스템을 동기 *Mealy machine*이나, 또는 비동기 네트워크로 손쉽게 명세할 수 있다[5].

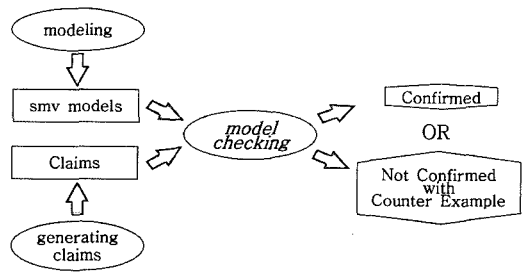

그림 5 SMV Model Checking Process

SMV 언어가 유한 상태 기계를 위해 설계된 것이기 때문에 언어가 제공하는 자료형은 유한한 것(Boolean, scalar, fixed array등)만을 제공한다. 정적이고 구조화된 자료형 또한 구현될 수 있다.

2.5 SCR(Software Cost Reduction)

NRL에서 A-7 항공기 비행 프로그램의 요구명세를 위해 개발되었다. 임베디드, 실시간 시스템, 소프트웨어를 위해 소프트웨어 요구를 명세하거나 그러한 명세를 위한 도구와 기법을 평가한다. 시스템이나 소프트웨어의 상태와 그에 관련된 변수들을 테이블로 표현하고, FSM을 이용하여 명세한다. 명세한 요구의 일관성과 무결성을 검사한다.

시스템을 5개의 테이블(*variable table, type table, mode class table, mode transition table, controlled table*)로 표현을 한다. 그림 6에서 시스템의 환경변수(*environment variable*)은 *monitored variable*과 *controlled variable*로 나뉘어 명세를 함으로써 좀더 명확한 시스템을 기술할 수 있다. 시스템의 행동이 어떻게 수행될 수 있는가가 아닌 시스템의 외형적 행동이 무엇인가를 서술하기에 유용하다는 데에 기반을 두고 있다.

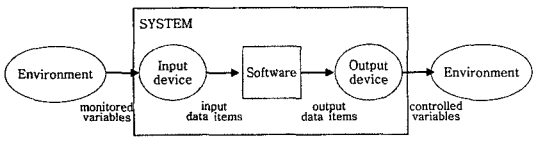

그림 6 SCR and the Four Variable Method

2.6 보안 정형 명세 도구 분류

그림 7은 검증 도구들을 분류한 결과이다. 수직 축은 도구의 사용자가 수학적 지식과 공학적 지식 중 어느 쪽에 더 친숙한지를 나타내고, 수평축은 도구가 범용인지 전용인지를 나타낸다[24].

그림 7에 나타난 NRL 프로토콜 분석기[14]는 특수 목적 검증 시스템으로 수학적 지식이 요구되는 도구로 암호, 인증, 키 분배 프로토콜 등을 위해 널리 사용되고 있다. SPIN, Murphi, SMV의 경우에는 모델 체킹을 통한 공학적 측면에서의 시스템 명세를 검증한다. Z/EVES, PVS[13], HOL[7] 등은 범용으로 사용되며, 고차원 논리를 기반으로 수학적 지식이 요구되는 자동적 정리 증명 도구 항목들이다.

3. 정형 검증 도구를 통한 보안 소프트웨어 개발 방안

3.1 보안 소프트웨어 개발을 위한 정형 검증 도구

정형 검증은 정형 논리 또는 수리 논리 등에서 제공하는 증명 방법을 이용, 정형 명세를 분석하여 시스템의 무모순성 및 완전성을 검증하거나, 설계가 주어진 가정에서 요구사항을 만족하는지를 검증하는 기법이다.

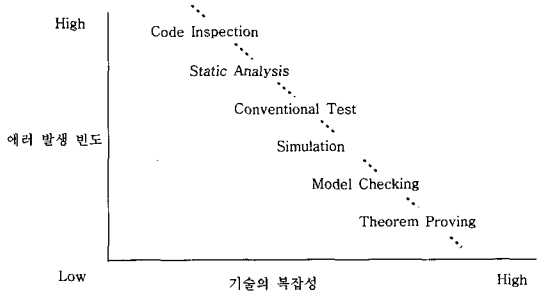

그림 8 시스템 검증 연속도

그림 8에서 시스템 인증 기술들은 수동적인 코드 조사와 계속적인 정적 분석과 같은 비교적 간단한 활동으로부터 정리 증명을 추출하기 위해 시스템 개발 라이프 사이클을 발전시킨다. 이들의 두 극단 사이가 기술적인 복잡성을 증가시키는 인증 기술들이다. 보통 실질적으로, 각 검증 기법은 발견되지 않을 수 있는 높은 가능성의 오류들이 제거될 때까지 적용된다. 더욱 강력하고 복잡한 검증 기법들이 적용됨으로써, 잠재적인 오류들이 더욱 더 적을 뿐만 아니라 감지하는 것이 어렵고 많은 손실을 가져온다. 결국, 정리 증명은 예를 들어 안전성의 특정 선언과 관련된 설계 오류들이 존재하지 않는다는 것을 보증하기 위해 중요한 시스템 컴포넌트들에 적

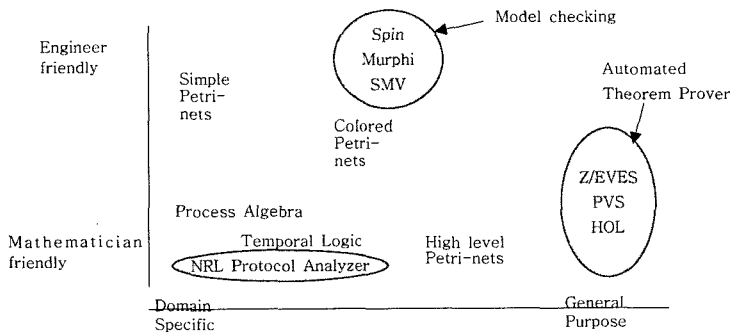

그림 7 검증 도구의 분류

용된다.

정형 검증은 프로그램이 입력 값의 모든 가능한 조합을 통해 올바르게 동작하는 것을 보여주기 위한 기술들의 모음으로 구성한다. 테스트 데이타의 모든 가능한 조합을 통해 프로그램들을 가장 잘 시험할 수 있는 것이 물리적으로 불가능하다는 사실에서 비롯되었다. 심지어 테스트들의 가장 큰 모음을 통해 정확하게 동작하는 것을 보여줬던 프로그램들조차도 데이타 값들의 사용되어질 수 없는 조합들을 통해 정확히 동작하지 않을 지도 모른다. 정형 검증은 또한 소프트웨어 개발의 비용을 증가시키는 계기가 되었다 . 정형 기술은 라이프 사이클에서 적용되어질 수 있고 테스트 데이타를 발생시키는 것을 요구하지 않기 때문에, 에러들은 비공식적 기술들보다 올바르게 고쳐지고 프로그램들이 검증되어질 수 있다.

정형 검증은 전체적인 정확성을 설명하려 보이고 있으며, 현재 기술들은 더욱 더 적절한 목표를 가지고 있다. 오늘날 프로그램의 어떠한 경향이 데이타의 모든 조합을 통해 일관성 있게 동작하는지 보여주기 위해 정형 검증을 사용하는 것은 가능하다. 이 결과 또한 부분적인 정확성이라 불려진다. 궁극적인 목적과는 거리가 멀지만, 현재 기술들은 선택된 테스트 경우들과 함께 프로그램을 실행할 필요없이 삭제되어지는 어떠한 기술들을 허용한다. 정형 검증은 자동적인 정리 증명과 그래프 이론에 근간하고 있다. 자동적인 정리 증명기와 정리 증명 기술은 우선순위 술어 계산법에 기초를 두고 기술된 문제들에 먼저 적용되어졌다. 일반적으로 자동적으로 증명되어졌던 정리들은 수학적인 논리에서 나타나는 문제들이었다.

컴퓨터 소프트웨어에 이러한 기술들의 먼저 알려진 어플리케이션은 Floyd에 의해 제안되었고, King에 의해 구현되었다. King은 ALGOL에서 쓰여지는 프로그램들의 작은 집합에서 정확한 기능을 보여주기 위해 설계된 시스템을 보여주었다. 그가 사용한 방법은 귀납적인 단언 기술로 알려져 있다. 정형 검증을 위한 King의 시스템 이후, 몇몇 주목할 만한 다른 시스템들이 구현되어졌는데, Deutch에 의한 interactive systems, Luckham에 의한 Pascal, Elspas에 의한 JOVIAL 시스템, Boyer과 Moore에 의한 LISP와 FORTRAN 시스템들이 있었다. Boyer와 Moore는 정형 검증에 대한 책을 집필하였다.

그래프 이론은 정형 검증을 위해 또한 사용되어졌었다. 첫 눈에, 흐름도에서 볼 수 있는 것들은 규제된 그래프로써 추상화될 수 있다. 그래프를 통한 프로그램의 분석은 프로그램을 모든 부분들이 함께 연결되고 그것이 일관적으로 변수들을 사용하는 것을 정형적으로 증명하기 위해 사용되어질 수 있다. 정형검증에서 그래프 이론의 어플리케이션들은 종종 그래프들의 효과적인 분석을 위해 Tarjan 알고리즘에 기초를 두고 있다.

기본적으로 보안 소프트웨어에 요구하되 사항들을 살펴보면, 인증된 사용자들의 접근을 허용할 수 있도록 확실히 하고, 접근 권한을 얻음으로 인증되지 않은 사용자들로부터의 방해에 보호를 받아야 한다. 하드웨어나 소프트웨어에 갑작스럽거나 의도적인 손상으로부터 보호되어야 하며, 손상된 시스템에서 빠른 시간 안에 복구할 수 있는 환경을 갖추어야 한다.

다음 표 1은 보안 소프트웨어 개발에 있어서 분류된 각 측면에 따른 적합한 정형 검증 도구를 나타내었다.

표 1 보안 소프트웨어의 개발에 적합한 정형검증 도구

분류	정형 검증 도구
사용자 환경 측면	Bandera 프로젝트, Z/EVES
기능적 측면	SPIN, Statechart
보안적 측면	NRL Protocol Analyzer, SPIN, SMV, Statechart
동작적 측면	SPIN, Statechart, Z/EVES
사용 범위	Z/EVES. PVS, HOL

표 2는 정형 검증 도구에서 나타나는 중점적인 특징들을 기능상 분류하여 나타내었다. 그림 10과 표 2의 내용을 통해 Z/EVES와 같은 정리 증명을 이용한 정형 검증은 보안 검증에 대한 더욱 더 막강한 기능을 지원하다.

3.2 Z/EVES

현재 Z/EVES 2.3이 이용 가능하다. 이 버전은 Z 명세를 따르는 문자 형식의 그래픽 사용자 인터페이스를 제공한다. 명세의 점진적인 분석, 명세의 수정과 함께 분석의 동시성을 관리한다. Z/EVES는 유럽과 북아메리카에서 최고 기술 수준의 정형 기법을 사용하며, 자동화된 연역적 능력과 함께 명세 표기법을 통합한다. 결과적으로 시스템은 다음 표 2와 같은 방법으로 Z 명세의 분석을 제공한다[18].

표 2 Z/EVES에서 제공하는 Z 명세의 분석 기능

기능	내용
1	신텍스와 타입 체킹
2	스키마 확장
3	필수조건 계산
4	도메인 체킹과 일반적인 원리 증명

명제 증명기를 위해서는 신텍스와 타입 체킹, 스키마 확장과 필수조건 계산에 대한 지식이 필요하며 도메인 체킹, 많은 증명 의무 또한 쉽게 풀어진다. 더욱 더 어려운 경우에는 증명 의무를 발생하는 것은 종종 명세가 의미 있는지 없는지를 결정하는 것에 대한 실질적인 목

적이다. Z/EVES는 전체적인 Z 표기법을 거의 다 제공한다. 단지 스키마 표현에 대한 특정 존재 기호를 제공하지 않고 있다. Z/EVES 증명기는 막강한 자동화 기능을 제공한다.

Z/EVES 시스템과의 상호 작용은 GUI에 증명되어지거나 *fuzz* 신텍스의 확장을 사용하는 LaTeX 기반이다. 확장은 증명기 명령들을 추가하는데 명제를 표현하는 구문과 자명하고 일반적인 박스에 속성을 나타내기 위한 라벨들을 붙이는 것에 대한 의미들을 추가하는 것이다. Z/EVES는 *fuzz* 타입체커를 위해 준비된 파일들을 읽을 수 있다. Z/EVES는 리눅스와 윈도우즈 98/NT/XP/2000에서 실행한다. 버전 2.1에서는 솔라리스에서 또한 실행한다.

3.2.1 Z/EVES 사용자 인터페이스

Z/EVES는 상호 작용적인 명령어-라인 시스템이다. Z/EVES가 유닉스 시스템에서 실행할 때, Z/EVES를 실행하기 위해서 Emacs 편집기를 사용하는 것이 가능하다. 이것은 입력 편집과 지난 입력 만회, 파일에 출력값 저장 등과 같은 편리한 작업을 제공한다. Z/EVES가 윈도우즈 기반에서 실행할 때에도 비슷한 기능을 제공한다. Z/EVES의 입력은 (=>) 프롬프트에서 작업한다. Z/EVES는 Z 구문 또는 Z/EVES 명령어들을 수용한다.

Z/EVES는 명세를 대표하는 히스토리를 유지한다. 명세는 상호작용적으로 들어가질 수 있고, 구문 단위로 또는 파일로부터 읽혀질 수 있다. 부분적이거나 전체적으로 명세를 삭제하는 명령어들이 있다. Z/EVES는 read 명령어를 사용하여 LATEX 마크업 파일을 읽을 수 있다. 파일에 있는 구문들은 히스토리에서 검사되고 추가되는 타입이다. 히스토리에 구문을 추가하지 않고 명세의 타입을 검사하는 것 또한 가능하다. Z/EVES 명령어들은 단어순이며, 세미콜론에 의해 종료된다.

3.2.2 증명

Z/EVES는 명세 내 정리의 증명과 문장을 따른다. 다양한 증명 단계의 명령어들이 있다. 증명들은 목적을 부르는 단정적인 작업을 한다. 각 단계는 목표를 동등한 새로운 목표로 전송한다. 목표를 올바르게 전송하는 것은 증명을 완벽하게 한다. Z/EVES 증명들은 우선 서술하는 계산으로 번역하는 것에 기반하고 있다.

Z 표기법은 몇 가지 술어들과 직접적인 술어 계산으로 변역되지 않을 수 있는 표현들을 허용한다. 그러한 술어들 또는 표현들이 만날 때, Z/EVES는 함수 또는 술어 이름을 정의하고 함수 응용에 의한 술어 또는 표현을 대체한다. 예를 들어, 표현 $\{ x : N \mid x < 9 \}$ 는 f0(N,9)와 같은 표현으로 번역되어질 수 있고, 함수 f0는 공리 $x \in f\ 0(X, y) \Leftrightarrow x \in X \wedge x < y$를 정의하고 선언되어진다. 보통 이러한 함축적인 선언들은 사용자에게 보여지지 않는다. 그러나 이러한 선언들을 만들기 위한 필요성은 제약을 부과한다. Z/EVES는 증명 도중에 만들어지는 선언을 허용하지 않는다. 그러므로 증명 단계에서, 함축적인 선언을 요구하는 술어나 표현을 사용하는 것이 가능하지는 않다. 그리고 사용자가 그렇게 시도하게 되면 에러가 발생하게 된다. 함축적으로 선언되어진 함수들은 가능한 다시 재사용된다, 그래서 위의 f0함수가 정의되어진 후, 표현 $\{ y : N_1 \mid y < 100 \}$ 이 만나게 되면, 그것은 $f0(N_1, 100)$으로 대체되어질 수 있고, 새로운 함축적인 선언은 요구되지 않는다.

3.3 정형 검증 도구를 통한 보안 소프트웨어 개발 방안

그림 9는 보안 소프트웨어의 명세와 검증 절차를 나타낸다.

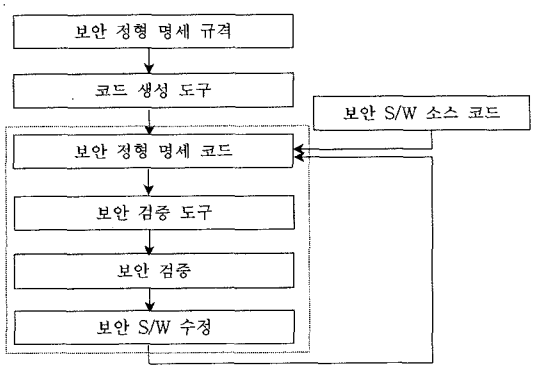

그림 9 보안 소프트웨어 명세 및 검증

그림 9에서 보여주는 바와 같이 보안 정형 명세 규격에 따라 코드 생성 도구를 이용하여 검증 절차를 수행하거나 보안 소프트웨어 소스 코드로부터 보안 검증 단계를 수행하는 과정을 통하여 보안 소프트웨어의 수정 작업을 거침으로서 정형화된 명세 코드를 다시 생성하고 또 다시 검증 과정을 거치게 된다. 명세를 통해 생성된 보안 소스 코드의 최종 검증이 완료됨으로서 개발자는 안정된 보안 소프트웨어 개발을 유지한다.

그림 8에서와 같이 보안 정형 명세 규격 및 코드 생성 도구는 SPEARII나 VDM-SL을 사용하면 된다. 그러나 보안 소프트웨어 명세 및 검증에서 보안 정형 명세 규격 및 코드 생성 도구는 그림 8에 보여 주는 바와 같이 생략 가능하다.

다음으로 보안 정형 명세 규격 및 코드 생성 도구를 사용하는 경우나 사용하지 않는 경우나 보안 정형 명세 코드가 생성이 된다. 이때 보안 정형 명세 코드 생성을 위한 환경으로는 여러 가지가 가능하다. 그중에서 일반적으로 많이 사용하는 환경이 Z 언어 이다. Z 언어의 사용 환경에서 Rational Rose는 Z 표기법을 통하여 완

성된 UML 모델을 빌드하기 위해서 사용된다. 보안 소프트웨어 검증을 위하여 C 언어로 작성된 보안 소프트웨어를 Z명세 언어로 변환하고 이것을 Z/EVES도구를 이용한다.

4. Z/EVES를 이용한 검증

기존의 보안 소프트웨어에 대한 검증은 SPEAR II나 VDM-SL을 중심으로 이루어지고 있다. 본 논문은 기존의 SPEAR II나 VDM-SL 보다 나은 환경인 Z/EVES를 통한 보안 소프트웨어 검증을 수행한다.

본 논문은 ACS(접근제어시스템)에 대한 정형 Z/EVES 검증 도구를 이용하여 보안 S/W 에 대한 검증을 수행한다. 정형화 기법을 이용한 보안 소스 코드 명세 및 검증 과정은 다음 세 가지 과정으로 이루어진다.

4.1 소스 코드로부터 UML모델 생성 과정

Rational Rose C++(실험에 사용된 버전 : 4.0)을 이용하여 C++ 코드에 대한 UML 모델 생성이 가능하다. Rational Rose C++은 C++ Analyzer를 포함하고 있는데, Rose와는 독립적으로 분리되어 실행되는 패키지이다. C++ Analyzer는 역공학을 제공한다.

4.2 UML 모델로부터 Z명세 변환 과정

본 절에서는 ACS의 UML 모델을 이용하여 Z 명세를 하는 과정에 대해 설명 한다. Rose 4.0 버전에 RoZ 스크립트를 Add-In 함으로써 UML 모델로부터 Z 명세 변환이 가능하다. 참고로 C++ 소스코드로 생성된 UML 모델이 아니며, ACS의 동작 원리를 구현하는 UML 모델을 새로 생성하면서 Z 명세 과정을 살펴본다.

4.3 Z 명세에 대한 정형 검증 과정

Z 명세 파일을 이용한 검증 절차는 Z/EVES 정형 검증 도구를 통하여 수행할 수 있다. Z/EVES를 통해 검증절차를 수행하기 위해선 먼저 Z/EVES 프로그램을 수행한 다음 메뉴에서 "File" -> "Read"를 선택하여 RoZ를 통해 생성된 Z 명세파일을 읽는다. ACS 시스템을 예로 들어, 오퍼레이션 "PERSONChangeTel_Pre" 증명을 검증하기 위해서는 우선 Z 명세 파일을 읽어들인다(read "ACS.zed"). ACS.zed을 읽고 난 후의 결과는 완전히 형성된 Z 명세 파일의 스키마를 오류없이 읽어들인다. 다음으로 증명 파일을 읽어들인다(read "ACSthms.zed"). 그림 10은 "read ACSthms.zed"에 대한 결과 화면이다.

ACSthms.zed 를 읽은 후 증명 가능한 법칙들을 보여주는 결과가 나타난다.

증명하고자하는 어떤 법칙을 보고자 한다면, Z/EVES commend 상에서 다음과 같은 명령을 실행한다.

Z/EVES에서 디스플레이 정보를 보여주는 출력 관련 명령어 중 printf status는 모든 확립된 증명 목적의 이

```
=>
Reading file D:\fm_test\RoZex_3\ACSthms.zed
theorem PERSONChangeLastname\_Pre
... theorem PERSONChangeLastname\_Pre
theorem PERSONChangeFirstname\_Pre
... theorem PERSONChangeFirstname\_Pre
theorem PERSONChangeTel\_Pre
... theorem PERSONChangeTel\_Pre
theorem PERSONChangeCardnb\_Pre
... theorem PERSONChangeCardnb\_Pre
theorem AddPerson\_Pre
... theorem AddPerson\_Pre
theorem RemovePerson\_Pre
... theorem RemovePerson\_Pre
theorem GROUPChangeGroupcode\_Pre
... theorem GROUPChangeGroupcode\_Pre
theorem GROUPChangeGroupname\_Pre
... theorem GROUPChangeGroupname\_Pre
theorem AddGroup\_Pre
... theorem AddGroup\_Pre
theorem RemoveGroup\_Pre
... theorem RemoveGroup\_Pre
Done.
=>
```

그림 10 ACSthms.zed를 읽고 난 후의 결과화면

```
=> print status;
Current status:
No current goal

Untried goals: PERSON\sdom
PersonGroupRelRel\sdomainC
PERSONChangeFirstname\_Pre
AddPerson\_Pre, RemovePer
GROUPChangeGroupname\_Pre.
=>
```

그림 11 print status 결과 화면

름들을 보여준다. 이러한 목적은 세 가지 리스트로 나타난다(proved goals(증명됨), untried goals(시도되지 않은 증명), unfinished goals(시도는 되었지만, 아직 증명되지는 않았음)). 그림 11에서 보여주는 내용들은 현재 증명 목적을 보여주는 리스트가 "No current goal"이라고 하여 현재 증명 중인 리스트가 없음을 나타내며, 증명되지 않은 목록들을 나타내고 있다. 그리고, ACS 시스템에서 "ChangeTel"을 위한 pre-condition 법칙은 다음과 같은 명령어를 사용함으로써 자동적으로 증명할 수 있다. 이는 print status 에서 나타나는 증명되지 않은 목록 중 PERSONChangeTel_Pre를 증명하기 위한 방법을 나타낸다.

본 논문은 보안 소프트웨어에서 정형 검증 도구인 Z/EVES 를 이용한 개발 방안을 제시한다. 본 논문에서 제시하는 방법은 기존의 방식이 모델링을 이용한 수학적인 방식을 사용하는데 비하여 도구를 이용함으로써 보다 체계적인 검증이 가능하다. 또한 검증이 이루어지면 개발 전 단계에서 발생할 수 있는 보안 허점을 찾을 수 있다. 본 논문에서 제안하는 보안 소프트웨어 정형 검증 방법으로 사용하는 Z/EVES 도구를 이용하였다. 실제 적용 사례로 보안 환경에 많이 적용하는 ACS 시스템을 적용하여 실제 경우에 접목되는 것을 보여주었다. ACS 시스템에 적용을 통하여 실제 적용 가능성을 보였다.

5. 결 론

본 논문에서는 보안 소프트웨어 개발에 사용되고 있는 SPIN, SLAM, SMV, Z/EVES 와 검증 도구 특징

들을 살펴보고, 이들 도구를 이용한 안전한 보안 소프트웨어 개발 방안을 제시한다.

기존의 보안 소프트웨어에 대한 정형 검증 기법은 SPEAR-II나 VDM-SL을 이용하여 이루어지고 있다. SPEAR-II나 VDM-SL은 제한된 환경에서 정형 검증이 되는 문제점을 가지고 있다. 그나마 보안 소프트웨어 검증 방안에 대한 연구의 진행이 부진한 상태이다. 또한 기존의 보안 소프트웨어 검증 방안에 대한 연구는 모델링의 어려움으로 인해 검증 자체가 쉽지 않다. 본 논문은 기존 연구의 이러한 문제점을 해결하고 보다 보안 소프트웨어 검증을 보다 용이하게 할 수 있는 방안을 제안한다. 본 논문에서는 보안 소프트웨어에 대한 방안을 제시하기 위하여 보안 소프트웨어 개발에 적합한 도구들의 기능을 상호 비교한다. 그리고 보안 소프트웨어에서 정형 검증 도구를 이용한 개발 방안을 제시한다. 그리고 마지막으로 제시된 보안 정형 검증 도구인 Z/EVES를 이용한 검증 방안에 대한 구체적인 방법을 제시한다. 본 논문에서 제안하는 보안 소프트웨어 검증 방안으로는 정형 검증 도구 중에서 가장 많이 사용되고 있는 Z/EVES를 이용한다. Z/EVES는 보안 소프트웨어 개발에 대한 가장 대표적인 정형 검증 도구이다. 본 논문에서는 보안 소프트웨어 검증을 위하여 C 언어로 작성된 보안 소프트웨어를 Z명세 언어로 변환하고 이것을 Z/EVES도구를 이용한다.

참 고 문 헌

[1] Simon Lukell, Chris Veldman, "Automated Attack Analysis and Code Generation in a Unified, Multi-Dimensional Security Protocol Engineering Framework," *COMPUTER SCIENCE HONOURS, UNIVERSITY OF CAPE TOWN.* CS02-15-00. OCTOBER 2002.

[2] The VDM Tool Group, "*User Manual for the IFAD VDM-SL Toolbox*," IFAD, Odense, Danmark, 1999.

[3] G.J. Holzmann. The Spin Model Checker - Primer and Reference Manual. Addison-Wesley, http://www.spinroot.com/. 2003.

[4] M.Bishop, "Computer Security," Addison-Wesley, 2002.

[5] K.L. McMillan, "SMV system," Last updated: November 6, 2000.

[6] David Harel and Amnon Naamad, "The STATEMATE Semantics of Starecharts," ACM Trans. Soft. Method, 1996.

[7] The HOL System Tutorial, version3. "http://www.cl.cam.ac.uk/users/mn200/hol-tutorial/index.html"

[8] A Quick Introductin to SPIN, "http://netlib.bell-labs.com/netlib/spin/whatispin.html"

[9] Gerard J. Holzmann, "The Model Checker Spin," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. VOL 23. NO. 5. MAY 1997.

[10] T. Ball and S. K. Rajamani, "SLIC: A Specification Language for Interface Checking (of C)," MSR-TR-2001-2.

[11] J. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, Robby, S. Laubach, H. Zheng, "Bandera : Extracting Finite-state Models from Java Source Code," Proceedings of the 22th ICSE, 2000.

[12] Crow J, Owre S, Rushby J, Shankar N, Srivas M. A tutorial introduction to PVS. *Workshop on Industrial-Strength Formal Specification Techniques (WIFT '95)*, 1995; 1-112.

[13] Owre S, Shankar N, Rushby J, Stringer-Calvert D. *PVS System Guide Version 2.3.* Computer Science Laboratory, SRI International, 1999; 1-88.

[14] C. Meadows, "The NRL Protocol Analyzer: An Overview," Journal of Logic Programming 24(2), pp. 113~131. 1996.

[16] T. Ball and S. K. Rajamani, "The SLAM Toolkit", proceeding of CAV 2001, 2000.

[17] C.Elks, "Issue in Formal Methods for the Analysis and Description of Security of Protocols," http://www.ee.virginia.deu/~rdw/EE68601/tps.psf

[18] M. Saaltink "The Z/EVES User's Guide," TR-97-5493-06, ORA Canada, 1997.

[19] 한국 소프트웨어 진흥원, "소프트웨어 모델링 및 분석 기법", 2003.

장 승 주

1985년 부산대학교 계산통계학과(전산학) 학사. 1991년 부산대학교 계산통계학과(전산학) 석사. 1996년 부산대학교 컴퓨터공학과 박사. 1987년~1996년 한국전자통신연구원 시스템 S/W연구실. 1993년~1996년 부산대학교 시간강사. 2000년~2002년 University of Missouri at Kansas City, visiting professor. 1996년~현재 동의대학교 컴퓨터공학과 부교수. 관심분야는 운영체제, 임베디드 운영체제, 분산시스템, 시스템 보안