보안 정형 검증 도구를 이용한 보안 S/W 검증 방안

김기환*, 장승주*, 박일환**
*동의대학교 컴퓨터공학과
**한국전자통신연구원 국가보안기술연구소
e-mail:{khkim,sjjang}@deu.ac.kr
ilhpark@etri.re.kr

Verification Methodology of Security S/W using Security Formal Verification Tool

Seung-Ju Jang*, Il-Hwan Prak**
*Dept. of Computer Engineering, Dong-Eui University
**ETRI National Security Research Institute

요 약

보안 소프트웨어 개발을 위한 정형 기법은 소프트웨어의 안정성과 신뢰성을 보장할 수 있는 기반을 마련해 준다. 정형화 기법에는 정형 명세와 정형 검증으로 분류할 수 있으며, 이를 위해 여러 도구가 제공되고 있다. 본 논문에서는 보안 소프트웨어 개발을 위한 RoZ 정형 명세 도구를 이용하여 ACS(Access Control System)의 UML 모델을 통한 Z 명세 자동 생성 과정을 살펴본다. 그리고, 정형 검증 도구인 Z/EVES를 이용하여 ACS 의 특정 기능의 명세에 대한 검증 과정을 수행함으로써, 소프트웨어 설계에 따른 보안 소프트웨어의 안정성을 보장할 수 있는 개발 방안을 제시하였다.

1. 서론

보안 소프트웨어 개발 방안에서 소프트웨어의 정형화된 명세 규격을 정의하는 정형 기법은 명세 과정을 위하여 수학적 표기법이 제공된다. 이러한 정형 기법은 명세에 대한 정확한 의미를 가지고 명세의 모호성을 제거함으로써 시스템 명세로부터의 불일치성을 검사하며 구현하고자하는 시스템의 안정성을 보장할 수 있다. 정형 기법에 사용되는 도구들은 추론 과정에서 요구하는 의미론을 분석하며 효과적인 시스템 구현을 도와줄 수 있다. 명세 과정만으로 시스템 수행 과정의 안정성을 보장할 수 없는 부분에 대해서는 시스템의 검증 단계를 수행함으로써 정형화된 시스템에 대한 안정성을 함께 보장할 수 있다. 검증 단계에서 지원하는 여러 도구들은 SPIN, SLAM, SMV, Z/EVES[3] 등과 같은 검증 도구들을 이용할 수 있다.

본 논문에서는 접근 제어 시스템을 구현해 봄으로써 소프트웨어의 정형 명세 및 개발 방안을 제시하였다. 소프트웨어의 명세 과정은 Z 언어를 이용하여 ACS(Access Control System)에 대한 명세 과정을 수행하게 되는데 Z 언어를 사용하기 위해 제공하고 있는 도구가 RoZ이다. RoZ[1]는 Rational Rose 도구에서 제공할 수 있는 확장 기능으로 UML 모델로부터 Z 명세를 생성하는데 필요한 명세 도구이다. 명세 과정에서는 UML 모델의 각 클래스에 대한 Z 표기법을 정의가 이루어져야 한다. 정의된 각 클래스의 Z 표기법 정의는 RoZ를 통하여 현재 개발하고자하는 ACS에 대한 Z 명세를 자동적으로 생성하는 기능을 가지고 있다. 다음으로, RoZ 명세 도구로부터 Z 명세를 생성하고 난 후 Z/EVES를 통하여 명세화된 ACS로부터 정형 검증 절차 수행과 의미론적 결과를 분석함으로써 보안 소프트웨어 개발 방안을 제시하였다.

본 논문의 구성은 2장에서는 정형 기법에 대한 관련 연구를 설명하고, 3장에서는 소프트웨어 검증을 지원하는 도구에 대한 설명을 하며, 4장에서는 정형 검증 도구를 이용한 실험 방법에 대해 설명하며, 5장에서 결론을 맺는다.

2. 관련 연구

정형 기법은 수학과 논리학에 기반을 둔 방법으로 하드웨어 시스템이나 정형 소프트웨어 시스템을 명세하거나 검증하는 방법론이다. 수학적 기호를 사용하여 시스템을 명세하고 검증할 특성 또한 논리식을 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.

정형 기법은 정형 명세(formal specification)와 정형 검증(formal verification)으로 두 가지를 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.

3. 보안 소프트웨어 검증을 지원하는 도구

소프트웨어 정형 검증 도구로는 여러 가지 있다. 이 중 다음과 같은 소프트웨어 검증을 지원하는 도구들을 몇 가지 소개한다.

SPIN은 기존의 모델 체킹 방법을 개선하기 위해 On-the-Fly 방식을 사용하여 메모리의 효율적인 사용을 가능하도록 하였다. SPIN은 통신 프로토콜과 같은 분산 환경의 소프트웨어 시스템의 정확성 및 안정성을 증명하기 위해 개발된 모델검증이다. SPIN은 명세하고자 하는 시스템의 동작을 오토마타와 같은 형식으로 모델링한다. 이 때 사용하는 언어가 Promela(PROtocol MEta LAnaguage)이다. Promela는 시스템을 프로세스의 형태로 추상화 하고, 프로세스들이 병렬적으로 수행하면서 채널을 통해 서로 간에 동기화 및 통신을 수행하게 된다.

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

Statechart는 리엑티브(*rective*) 시스템의 명세에 적합한 그래픽 언어이다. 그 근본 구조는 상태 전이도(*state transition diagram*)에 계층(*hierarchy*), 동시성(*concurrency*), 통신(*communication*)의 개념을 도입한 것이다. 계층은 트리에서 서브-트리가 파생되듯이, 하나의 스테이트가 서브 스테이트를 가질 수 있게 하는 것으로 계층적 구조를 표현한다. 동시성은 스테이트들 간에 병렬성을 제공하는 것인데, 두 개 또는 그 이상의 병렬 컴포넌트가 모여서 하나의 스테이트를 구성하게 된다. 통신은 시스템의 컴포넌트들 간에 방송(broadcasting)을 통해 정보를 주고받으면서 시스템이 진행하는 것을 의미한다.

4. 정형 검증 도구를 이용한 보안 S/W 검증

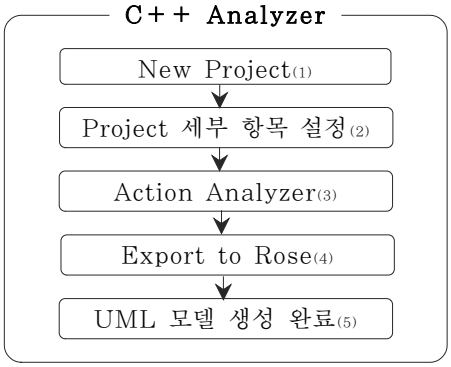
본 논문의 실험에서는 Z/EVES 정형 검증 도구를 이용하여 ACS(접근제어시스템)에 대한 검증 과정을 수행해 보았다. 정형화 기법을 이용한 보안 소스 코드 명세 및 검증 과정은 다음 세 가지 과정을 통하여 완료할 수 있다.

(1) 소스 코드로부터 UML 모델 생성 과정
(2) UML 모델로부터 Z 명세 변환 과정
(3) Z 명세에 대한 정형 검증 과정

4.1 소스코드로부터 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. Rational Rose C++ Analyzer는 C++ 소스 코드로부터 디자인 정보를 추출하고, 소스코드의 논리적이고 물리적인 구조를 표현하는 모델을 생성하기 위해 사용된다. 4.1에서 수행하는 과정은 ACS의 소스코드

를 이용한 것이 아니라 Rose C++ Analyzer를 이용해 C++ 소스코드로부터 UML 모델을 생성하는 과정에 대해서만 단계별(5단계)로 설명을 한 것이다.



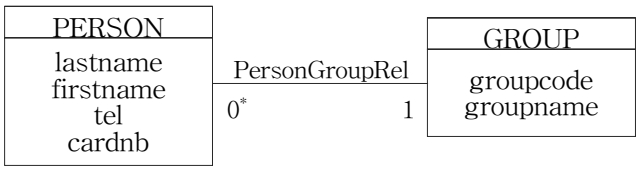
[그림 1] C++ Analyzer를 이용한 C++코드로부터 UML 모델 생성 과정

[그림 1]에서 보여주는 과정은 Rose 4.0에서 제공하는 역공학 과정을 설명한 것이다. UML 모델 생성을 위하여 C++ Analyzer에서 새로운 프로젝트를 생성하고 난 후 [그림 1]-(2)에서와 같이 프로젝트 세부 항목을 설정해 준다. 세부항목 설정은 현재 UML모델로 변환하고자 하는 C++코드의 위치를 선택하고 UML 모델이 생성되는 위치를 설정하는 역할을 수행한다. [그림 1]-(3)은 C++코드를 분석하여 UML 모델을 생성하기 위한 주된 역할을 수행하는데, C++코드 오류의 문제점이 발생할 때에는 UML 모델을 생성할 수가 없으므로 C++ 소스코드의 UML 변환을 위한 코드 구성이 제대로 이루어져 있어야만 UML 모델로의 변환이 가능하다. [그림 1]-(4)는 C++ 코드 분석이 제대로 완료되었다면, C++코드에 대한 UML모델을 생성하게 된다. 이 생성된 UML 모델은 Rose 4.0을 이용하여 살펴볼 수 있다.

4.2 UML 모델로부터 Z명세 변환 과정

본 절에서는 ACS의 UML 모델을 이용하여 Z 명세를 하는 과정에 대해 설명을 한 것이다. 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.

ACS 시스템은 다음과 같은 UML 모델 구조를 가진다.

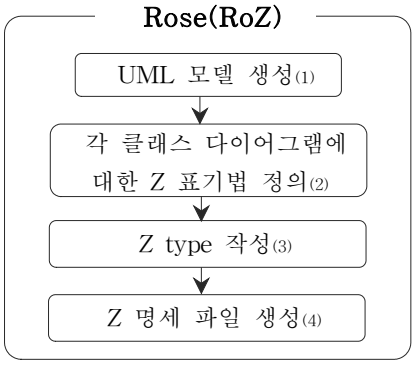


[그림 2] Access Control System 구현을 위한 UML 모델

[그림 2]에서 PersonGroupRel 연관에서 한 PERSON은 단지 한 GROUP과 연관관계를 가지면서 PERSON과 GROUP 클래스의 관계를 유지한다. 다음은 [그림 2]의 다이어그램에서 추가적인 3가지 규제을 정의한다.

① 모든 사람은 적어도 하나의 전화번호를 가지고 있다.
② 카드 번호는 그 사람의 키이다.
③ 주어진 그룹 멤버의 전화번호는 같은 접두사를 가지고 있다

ACS 시스템이 가지는 기본 모델을 위와 같은 사항으로 참고하여 다음 단계별(4단계)로 Z 명세 과정을 수행한다.



[그림 3] UML 모델로부터 Z 명세 파일 생성 과정

[그림 3]의 과정은 [그림 2]에서 보여주는 ACS에 대한 기본적인 UML 모델의 생성 후 각 클래스 다이어그램마다 Z latex type의 정의가 필요하고, ACS 시스템의 3가지 규제에 적용되는 Z type 파일을 작성한 후 Rose의 RoZ를 이용한 Z 명세 자동생성 과정을 수행하는 과정을 나타낸다.

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

증명하고자하는 어떤 법칙을 보고자 한다면, Z/EVES commend 상에서 다음 [그림 4]과 같은 명령를 실행한다.

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

Untried goals: PERSON\$domainCheck, PersonGroupRelRel1\$domainCheck,
PersonGroupRelRel\$domainCheck, PERSONChangeLastname\_Pre,
PERSONChangeFirstname\_Pre, PERSONChangeTel\_Pre, PERSONChangeCardnb\_Pre,
AddPerson\_Pre, RemovePerson\_Pre, GROUPChangeGroupcode\_Pre,
GROUPChangeGroupname\_Pre, AddGroup\_Pre, RemoveGroup\_Pre
=>
```

[그림 4] print status 결과 화면

Z/EVES에서 디스플레이 정보를 보여주는 출력 관련 명령어 중 printf status는 모든 확립된 증명 목적의 이름들을 보여준다. [그림 13]에서 보여주는 내용들은 현재 증명 목적을 보여주는 리스트가 "No current goal"이라고 하여 현재 증명 중인 리스트가 없음을 나타내며, print status 에서 나타나는 증명되지 않은 목록 중 PERSONChangeTel_Pre를 증명하기 위해 다음 [그림 5]와 같이 try명령을 이용한다.

```
=> try lemma PERSONChangeTel\_Pre;
Beginning proof of PERSONChangeTel\_Pre ...

            PERSON \\
        \land newtel? \in \finset TEL \\
        \land newtel? \neq \emptyset \\
\implies
 (\exists cardnb': \num; firstname': NAME; lastname': NAME; tel': \power TEL
  @ PERSONChangeTel)
=>
```

[그림 5] PERSONChangeTel_pre 증명 시도

다음으로 [그림 5]의 명제를 검증 수행하기 위하여 마지막 증명 과정으로 prove 명령어를 이용하여 PERSONChangeTe_Pre에 대한 명제를 증명한다.

```
=> prove by reduce;
Which simplifies
with invocation of \Delta PERSON, PERSONChangeTel, DIGIT8, PERSON
when rewriting with `[internal items]`, nullSubset, weakening, notEqRule,
emptyDefinition, inRange
forward chaining using Delta\$PERSON\$declarationPart,
PERSONChangeTel\$declarationPart, KnownMember\$declarationPart, knownMember,
PERSON\$declarationPart, `[internal items]`
with the assumptions `&neq$declaration`, select\_2\_1, select\_2\_2,
`&upto$declaration`, finset\_type, `[internal items]`
with the instantiations
tel' = newtel?, lastname' = lastname, firstname' = firstname, cardnb' = cardnb
to ...
true
Proving gives ...
true
=>
```

[그림 6] prove by reduce 결과 화면

[그림 6]의 "*prove by reduce*" 명령어는 현재 목적에서 감소가 어떠한 효과를 가지지 않을 때까지 반복되는 감소를 실행하며, [그림 5]의 해당 오퍼레이션인 PERSONChangeTel_Pre 명제 조건의 검증이 모두 이상없는 명제임을 증명할 때 true 메시지를 출력하면서 검증 과정을 마친다.

V. 결론

본 논문에서는 UML과 정형 명세 언어인 Z와의 조합을 이용함으로써 명세를 더욱 더 향상 시킬 수 있는 방법을 제시하였다. 이를 이용하여 Z/EVES 정형 검증 도구를 통해 보안 소프트웨어의 안정성 및 신뢰성을 보장할 수 있는지에 대한 검증 절차를 수행하였다. UML 클래스 다이어그램과 그에 대한 Z 정형 명세 표현에 기반한 모델을 생성한 후 보안 소프트웨어의 안정성을 보장할 수 있는 모델을 생성하는 과정은 RoZ 정형 명세 도구를 이용하여 명세 과정을 수행하였다. 정형 명세를 위해 사용된 예제는 ACS(접근 제어 시스템)를 이용하여 UML 모델로 표현한 다음, 시스템의 명세를 위하여 각 클래스 다이어그램을 Z 언어로 정의함으로써 정형화된 시스템을 구현했다. 명세화된 시스템을 통하여 완전히 안정된 소프트웨어라고 보장할 수는 없다 이러한 이유 때문에 정형 검증 과정을 통하여 설계된 소프트웨어의 각 항목에 대한 검증 절차가 필요하다. 검증 절차를 수행하기 위해 사용되는 도구에는 여러 가지 있지만, 본 논문에서는 Z/EVES 보안 정형 검증 도구를 이용함으로써, 소프트웨어 설계에 따른 UML 모델을 통해 생성된 Z 명세의 각 오퍼레이션에 대한 전제 조건들을 검사함으로써 소프트웨어 설계시 제시한 규제 조건을 만족할 수 있도록 보장한다.

참고 문헌

[1] Sophie Dupuy, "RoZ version 0.3 an environment for the integration of UML and Z", Laboratoire Logiciels, Systems et Reseaux-IMAG, BP 72-38402 Saint-Martin d'Heres Cedex

[2] Y.Ledru, Identifying pre-conditions with the Z/EVES theorem prover, Dans Int. IEEE Conference on Automated Software Engineering '98, October 1998

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

[4] Irwin Meisels , Mark Saaltink, "The Z/EVES Reference Manual", TR-97-5493-03d, ORA Canada, September 1997