정형명세 보안기능의 코드생성 도구 비교[1)]

유희준*0 김일곤* 최진영* 김상호** 노병규**
고려대학교 컴퓨터학과*
한국정보보호진흥원**
{hyoo, igkim, choi}@korea.ac.kr
{shkim, nono}@kisa.or.kr

Comparison of Code Generation Tools from Formal Specification of Security Function[1)]

Hee-Jun Yoo*0 Il-Gon Kim* Jin-Young Choi* Sang-Ho Kim** Byung-Gyu Nho**
Dept. of CSE Korea University*
Korea Information Security Agency**

요 약

최근 들어 보안에 대한 관심이 높아지고 있으며, 많은 보안 관련 프로그램이 사용되고 있는 상황이다. 하지만 안전성을 예측하기 어려운 보안 기능을 완벽하게 구현하기 위해서 현재의 테스팅 방법론만으로는 완전성을 보장할 수 없으므로 보다 안전한 보안 기능을 구현하기 위해서 정형명세 방법은 반드시 필요하다. 또한 실제로 시스템들이 개발되는 과정에 있어서 설계될 당시와는 달리 결함을 가질 수 있다. 시스템에 대한 명세가 정확하다 하더라도 개발자인 사람의 개입으로 인해서 개발하는 도중에 명세와 다른 결과를 초래하여 결함이 발생할 수 있다. 이러한 결함은 개발자나 사용자에 의해 쉽게 발견되기 어렵다. 만일 보안 시스템이 알려지지 않은 결함이 있다면 비밀 정보가 쉽게 누설될 수 있다는 큰 문제가 발생할 수 있다. 통신망을 사용하는 인구의 확대와 더불어 다양한 형태의 통신 서비스가 제공됨에 따라 시스템의 특성이 복잡해지고 불법적인 자료 접근의 위험도 많이 노출되고 있다. 따라서, 명확한 정형명세 보안기능으로부터 자동 코드 생성은 반드시 필요하다. 본 논문에서는 여러 정형 도구를 사용해서 보안기능에 대한 정형명세를 작성한 후에 소스코드 생성한 후, 정형명세를 이용하여 설계 및 구현한 프로그램에 대한 명세와 소스코드간 일치성 검토 방법을 개발하여, 정형명세를 통해 생성된 보안기능 소스코드의 보안성 측면과 성능적 측면의 Trade-Off 분석하는 것이다. 논문에서는 인증 관련 보안 기능인 S/KEY에 대해서 4개(STATEMATE MAGMUM, Rational Rose 98, SPEAR, VDM-SL)의 정형도구를 사용하여 명세 후, 소스코드를 생성하여 생성된 각 코드에 대하여 비교하였으며, 현재 상용화 되어있는 Bellcore 의 S/KEY 시스템과의 비교를 수행하였다. 분석에서는 제품의 효율성뿐만 아니라 보안성을 중요하게 생각하였으며, 앞으로 보안 관련 소프트웨어 개발에 사용될 수 있는 도구들의 가이드 라인에 대한 정보를 제공한다.

1. 서 론

보안 기능을 구현함에 있어서 개발 초기 단계에서부터 정형기법을 사용하는 이유는 명확한 정형화된 언어를 사용함으로써 요구 분석단계에서부터 구현에 이르기까지 발주자, 설계자와 개발자가 발생 가능한 모든 오류(misunderstanding)를 방지하며, 초기에 분석 설계된 시스템과 최대한 동일한 시스템을 개발할 수 있도록 도와준다.

실제로 시스템들이 개발되는 과정에 있어서 설계될 당시와는 달리 결함을 지닐 수 있으며, 이러한 결함은 개발자나 사용자에 의해 쉽게 발견되기 어렵다. 만일 보안 시스템이 알려지지 않은 결함이 있다면 비밀 정보가 쉽게 누설될 수 있다는 큰 문제가 발생할 수 있다. 또한 컴퓨터가 발전함에 따라 침입자들의 공격방법이 매우 다양해지고, 그 강도 역시 강해지고 있기 때문에 보다 안전한 프로토콜 및 시스템의 개발이 필수적이라고 할 수 있겠다.

하지만, 일반적으로 시스템 개발에 있어서 설계자와 개발자가 다른 경우에 개발 단계에서 개발자가 명세를 구현하는 동안 의식 중 혹은 무의식중에 명세와 다르게 구현될 가능성을 배제할 수 없다. 이 문제를 해결하기 위해서는 정형 명세를 수행한 설계에서 구현인 소스 코드를 인간의 개입을 최소한으로 줄이면서 자동적으로 생성할 수 있는 방법론이 절실하게 필요하다.

논문은 SKEY시스템[1]에 대해서 STATEMATE[2], Ratioanl Rose[3], SPEAR[4]와 같은 그래픽한 정형 명세 도구와 IFAD VDM-SL[5]과 같은 논리 기반 정형 명세 도구를 사용하여 명세로부터 코드를 자동으로 생성하는 방법론에 대해서 설명하고, 생성된 코드와 실제 사람이 작성된 코드의 일치성 분석 및 비교를 수행하였다. 2장에서는 도구를 사용한 명세와 생성된 코드의 수준을 설명한 후, 3장에서 도구를 비교하겠다. 마지막으로 4장에서는 결론을 맺겠다.

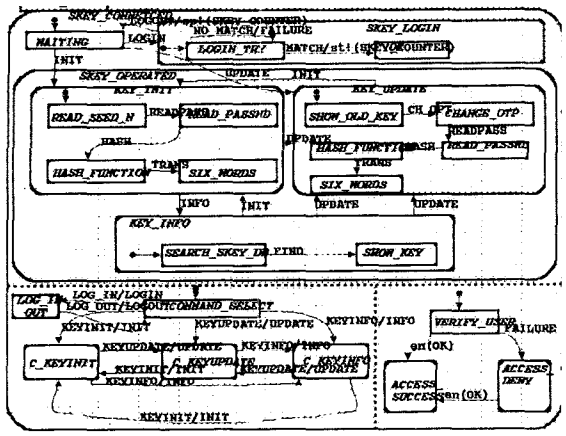
2. 도구를 사용한 명세와 코드 생성

2.1 STATEMATE MAGMUM

이 도구는 리엑티브 시스템에 대한 그래픽한 명세와 이에 대한 시뮬레이션을 수행하는 도구로 C 혹은 Ada 코드를 생성하여 준다. 다음은 시스템을 명세한 일부이다.

2.1.1 STATEMATE에서 생성된 코드

[1)] 본 연구는 2001년도 한국정보보호진흥원의 지원을 받은 것이다.



[그림 1] SKEY 동작 절차 및 인증

생성된 소스 파일은 모델의 로직과 스케줄링을 포함하고 있으며 모델의 가장 중요한 부분인 컨트롤 모듈, 외부 환경 또는 다름 사용자-추가 모듈과 함께 행동 모델을 인터페이스화 하기 위해 사용되어지는 후크(hook)와 프레임을 포함하는 사용자 추가 모듈, PGE와 디버거를 위한 인터페이스 모듈, 실제적으로 소스파일로부터 어플리케이션을 만드는 과정을 자동화하기 위해 사용되는 Makesfiles과 Compilation 스크립트 파일과 상호 래퍼런스 정보를 포함하는 Info 파일로 나누어진다.

2.2 Rational Rose 98

Rose 98은 Rational사가 개발한 UML 모델링의 자동화를 돕는 도구이다. Rose 98은 같은 모델에서 C++, Visual Basic, Java 같은 다양한 언어를 혼합하고 연결할 수 있다.

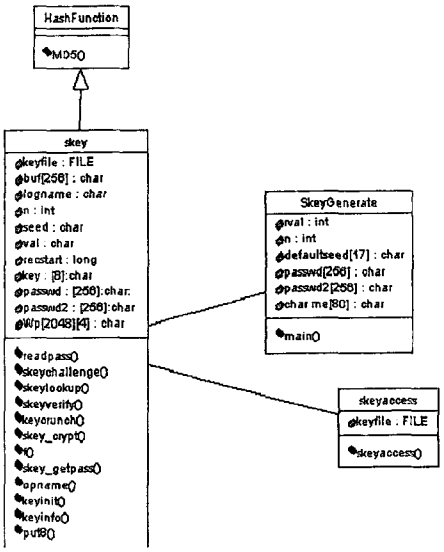
2.2.1 Rational Rose 98에서 생성된 코드

도구는 전형적으로 해당 언어로 코드 스켈레톤을 생성하며 그 모델의 정보를 해당 언어로 변형시킨다. 일반적으로 생성된 코드의 종류는 클래스 정의와 같이, 속성과 메소드 선언을 포함하는 정적인 정보의 형태를 지닌다. 비록 이론적으로는 동적인 모델 부분이 몸체의 코드로 변형되어지지만, 실질적인 코드를 포함하는 메소드의 몸체는 나중에 프로그래머가 입력하도록, 일반적으로 빈 공간으로 남겨져있다. 코드 생성은 사용자에 의해 행해지며, 도구들은 어떻게 소스코드가 생성되었는지 이에 대한 정보를 제공해주게 된다.

여러 종류의 해당 언어가 지원되게 된다. 일반적으로 객체지향형 언어인, C++이나 JAVA뿐만 아니라, SQL이나 IDL과 같은 언어도 지원한다.

2.3 SPEAR

SPEAR, the Security Protocol Engineering and Anaylsis Resource는 특히 보안 프로토콜을 주목해서 만들어진 프로토콜 엔지니어링 도구인데, 보안과 효율적인 디자인을 제공하기 위해, 그리고 프로덕션 과정을 지원하기 위한 목적으로 만들어졌다.

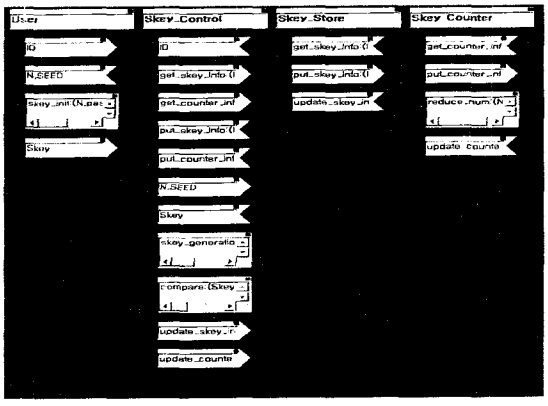


[그림 2] SKEY의 클래스 다이어그램

2.3.1 SPEAR에서 생성된 코드

코드 생성 모듈은 실제 클라이언트와 서버 소프트웨어로 사용되어질 수 있는 완전한 기능의 소스 코드를 생성해야만 한다. 이는 네트워크 프로그래밍을 작성을 보다 원활하게 해준다.

소스코드는 프로토콜들이 종종 이종 네트워크 상에서 구현되기 때문에 적용하기 유용해야 한다. 프레임워크를 구현하는 도구는 보안 프로토콜 기능과 알고리즘의 확장된 형태를 제공해야만 한다. 이 도구는 특히 DES, RSA, MD5, 키 생성, 난수 생성과 그외 잘 사용되어지는 기능의 구현을 보여주고 있다. 가능한 근접하게 GSS-API와 같은 라이브러리를 따르는 형태를 취하고 있다. 이러한 주 이유는 이미 보안 커뮤니티에서 허용되어온 기능들에 대한 표준 인터페이스를 제공하기 때문이다.



[그림 3]SPEAR에서 SKEY 명세

2.4 IFAD VDM-SL

IFAD에서 개발한 논리기반 언어인 VDM을 지원하는 도구이다. 함수, 변수, 불변자 등을 기술한다. C++코드를

생성한다. VDM 명세와 생성된 코드에 관한 비교는 2.4.1에서 설명하겠다.

2.4.1 VDM-SL에서 생성된 코드

모듈단위로 코드를 생성해 준다. 다음은 시스템의 명세에 대한 일부분과 그에 해당하는 원 코드에 대한 비교이다.

File 이라는 변수가 문자형 수열의 타입을 가진다는 것을 의미한다. 이 변수는 Skey structure 안에서 사용되는 하나의 타입으로 정의된다. 이 부분에 대한 소스 코드는 아래와 같다.

```
struct skey {
        FILE *keyfile;
        char buf[256];
        char *logname;
        int n;
        char *seed;
        char *val;
        long     recstart;
};
```

FILE은 문자형의 수열로 이루어진 자료형이기에 위와 같이 명세를 한 것이다. Skey 시스템에서 기본적인 구조로 사용되는 structure 이다. 이 코드를 VDM-SL 명세로 변환하면, 다음과 같이 표현할 수 있다.

```
Skey:: keyfile : File
       buffer : seq of char
       logname : char
       n : int
       seed : char
       val : char
       recstart : Long
  inv mk_Skey(-, buffer, -, -, -, -, -) == len buffer
= 256;
```

명세 자체가 원시 코드와 유사하게 이루어져 있으며, 이를 사용하여 생성된 코드는 inv 로 표시된 불변자가 만족되는 경우에 대해서만 코드가 호출되어 사용될 수 있도록 정의되어 있다. 생성된 코드를 살펴보면, 원 코드에 비해서 크기는 하지만, 반드시 만족해야될 조건을 검사하며, 실행하도록 되어있다.

3. 도구 비교

앞 절에서 사용한 도구를 비교해 보면, STATEMATE는 도구에서 생성해 주는 코드는 실제 시스템에서 사용할 수 있는 수준의 코드가 아니라 시뮬레이션을 하기 위한 코드를 생성하는 이유로 실제 시스템에 사용할 수 없는 이유로 시스템 개발에 바로 적용할 수 없으며, 생성되는 코드의 사이즈가 명세의 크기에 상관없이 매우 크다는 단점이 있다. UML은 코드 생성 부분에서는 9개의 다이어그램 중에서 클래스 다이어그램등 극히 일부 다이어 그램만 가지고 작업을 하는 이유로 상세한 코드를 생성하지는 못한다. 각각의 클래스, 메소드 등에 대한 skeleton 코드만을 생성하여 주기 때문에 실제 시스템 개발을 위해서는 개발자의 손이 많이 필요하게 된다. 따라서, 개발자의 개입에서 발생할 수 있는 문제가 여전히 남아있게 된다는 문제점이 있다. SPEAR는 java.security 라이브러리에 정의되어 있는 함수를 사용한다면, 도구는 링크된 함수의 코드를 생성되는 코드에 바로 삽입을 시켜준다. 명세를 보면 클라이언트/서버 구조와 같은 통신 프로토콜을 구현하는 경우에 적합하도록 구성되어 있다. 이런 이유로 생성된 코드의 구조는 프로토콜 각 참가자에 대한 코드를 생성하여 준다. 생성된 코드는 각 부분의 시스템에서 바로 실행 가능한 수준으로 생성된다. 따라서, java.security에서 정의된 함수만을 가지고 통신 프로토콜을 명세 하는 경우에 가장 좋은 수준의 코드를 생성하여 준다.

VDM-SL은 그래픽한 정형 명세 언어와 다르게 디자인하는 시스템에 대해서 보다 명확하고 상세한 명세를 할 수 있게 제공을 해준다. 이런 종류의 언어들은 일반적으로 시스템에 대한 제약조건을 명세에 완벽하게 명시할 수 있으며, 어떤 동작이 이루어지기 전에 조건이 만족하는지 여부를 먼저 검사한 후에 동작을 가능하게 해준다. 어떤 자료형, 동작에서 반드시 만족해야할 조건을 *불변자*(Invariant)라고 부르며, VDM-SL에서는 명세에서 "inv"는 명령어를 사용해서 명세에 명시를 해준다. 예를 들어, 최근 들어서 시스템의 해킹으로 가장 많이 사용되고 있는 방법이 Buffer Overflow 공격법을 방지하기 위해서는 매 동작마다 사용되는 버퍼의 크기와 현재의 위치와 같은 정보를 계속 체크해 주어야 하며, 이럴 경우에 시스템의 효율성은 현저히 떨어질 것이다.

생성된 코드와 원 소스코드간의 일치성을 검사하는 문제는 주어진 입력에 대해서 처리한 결과가 같은 지를 검사하는 것이 좋다. 하지만, 기본적으로 불확실성을 가지고 있는 소프트웨어에서 이러한 항목을 검사하는 것이 어려우며, 또한 검사를 위해 코드의 추가가 개발자에 의해서 이루어진다면, 수정된 코드의 보안성 문제가 다시 제기될 것이다.

4. 결과

본문에서 살펴보았듯이 코드를 자동으로 생성해 주는 도구들은 공통적으로 명세가 어느 정도로 상세히 이루어졌는지에 따라서 코드가 생성되는 수준의 차이를 나타낸다. 보안 기능을 강화한 코드를 생성하기 위해서는 불변자를 선언하여 코드 생성에 제약 조건으로 추가할 수 있는 능력을 가진 언어들이 보다 높은 보안성을 제공하는 코드를 생성할 수 있지만, 보안성을 높이게 되면, 그에 수반되는 불변자 만족성 검사를 수행하기 위한 추가적인 시간과 공간적인 문제로 인해서 시스템의 효율성이 감소하게 된다. 따라서, 명세 시에 시스템에서 반드시 필요한 부분 외에서는 불변자를 선언하지 않는 것이 좋다. 연구의 결과로 현재 사용된 도구 중에서 보안기능을 강화하기 위해서는 VDM-SL을 사용하는 것이 실 시스템에서 바로 사용할 수 있는 수준의 코드를 보안성 부분까지 고려하여 생성할 수 있었다. 다른, 그래픽한 명세 도구에서는 실 시스템에서 사용하기 위해서는 개발자가 개입을 해야만 하기 때문에 사람의 개입으로 인해서 발생할 수 있는 미세한 오류 문제가 여전히 남아있게 된다는 것을 알 수 있다.

5. 참고문헌

[1] Neil M. Haller, The S/Key™ One-Time password System, Bellcore, 1992

[2] i-Logix, STATEMATE MAGNUM Reference Manual ver1.0, 1996

[3] Rational Software Corporation, Rational Rose 98 : Using Rational Rose, Rational Software Corporation, 1998

[4] J. P. Bekmann, P. de Goede, Multi-Dimensional Security Protocol Engineering using SPEAR, 1997

[5] C. B. Jones, Systematic Softtware Development using VDM, 2nd ed. Prentice Hall, 1990