主題

자바 보안과 암호

한남대학교 조한진, 이희규, 김봉한, 이재광

차 례

요 약

인터넷의 발전과 함께 인터넷 환경에 적합하게 개발된 자바 언어는 독립적인 구조, 분산처리 능력, 그리고 이식성이 좋다는 특징을 가지고 있어, 국내외적으로 자바를 이용한 전자상거래 관련 기술의 개발이 활발하다. 그러나, 자바는 애플릿을 클라이언트에서 실행할 경우 발생할 수 있는 여러 가지 보안 문제에 노출되어 있기 때문에, 자바 개발자들은 JDK를 보안 문제에 중점을 두어 개발하고 있다. 한편, 보안 문제뿐만 아니라 암호에 관한 기술도 매우 중요시되고 있지만, 암호 및 인증 체계 기술은 미국에서 보유한 기술 중 인증기관 관련 기술로 미국 내에서 금수 기술로 분류되어, 미국과 캐나다를 제외한 세계 어느 지역에서도 기술 도입이 불가능한 상태이다. 이러한 이유로, 여러 기술 선진국에서는 Sun JCA의 구조에 바탕을 두고 Sun JCE와 호환되는 독자적인 JCE를 구현하기 위해 노력하고 있다. 본고에서는 JDK를 중심으로 자바 보안 모델과 암호 구조에 대하여 살펴본다. 2장에서는 자바의 보안 모델의 3요소에 대하여 알아보고, 3장에서는 여러 가지 보안 문제를 극복하려는 JDK 보안 구조의 변화에 대하여 살펴본다. 그리고 4장에서는 자바 보안 API에 대하여 자세히 살펴보며, 마지막으로 5장에서 결론을 맺는다.

Ⅰ. 서 론

정보통신 분야의 비약적인 발전으로 인하여 제한적이던 네트워크 환경이 인터넷과 같은 개방된 네트워크 환경으로 변하면서, 정보의 공유를 목적으로 한 학술·연구용 인터넷이 마케팅을 목적으로 하는 상업용 인터넷으로 바뀌어 가고 있으며, 웹 서비스를 이용하여 가상공간에서 상품을 구매하고 결제할 수 있는 전자상거래(Electronic Commerce) 서

비스가 활발히 개발되고 있다.

그러나, 인터넷의 표준 프로토콜인 TCP/IP (Transmission Control Protocol/ Internet Protocol)는 네트워크의 개방성을 위한 소스의 공개로 인하여 악의를 가진 자에 의한 중요 정보 자원의 오용 및 도용할 수 있는 위험성을 가지고 있으므로, 인터넷을 이용하는 대다수의 사람들은 보안 문제에 노출되어 있는 실정이다.

한편, 인터넷의 발전과 함께 인터넷 환경에 적합하게 Sun사에 개발된 자바(JAVA) 언어는 독립적인 구조를 가지고 있고, 애플릿(Applet)이라는 실행코드를 서버로부터 다운로드받아 클라이언트에서 실행함으로써 서버와 클라이언트가 작업 부하를 나누는 분산처리 능력과, 다양한 플랫폼에서 실행이 가능한 이식성이 좋다는 특징을 가지고 있다[1, 2, 3, 12].

그러나, 자바는 네트워크를 통하여 서버로부터 다운로드받은 애플릿을 클라이언트에서 바로 실행할 경우 발생할 수 있는 여러 가지 보안 문제에 노출되어 있기 때문에, Sun사의 개발자들은 보안 문제에 중점을 두어 JDK(Java Development Kit)를 개발하고 있다.

최근, 자바 보안에 관련된 기술이 활기를 띠고 있어, 국제적인 전자상거래 관련 암호 및 보안 프로토콜 기술들이 자바로 전개되고 있는 실정이다. 그러나, 이러한 필요성에도 불구하고 암호 및 인증 체계 기술은 미국에서 보유한 기술 중 인증기관 관련 기술로 미국 내에서 금수 기술로 분류되어, 미국과 캐나다를 제외한 세계 어느 지역에서도 기술 도입이 불가능한 상태이다. 그래서 Sun사에서는 JDK와 JCE(Java Cryptography Extension)를 분리하여 배포하고 있다[4].

이러한 이유로, 여러 기술 선진국에서는 Sun JCA(Java Cryptography Architecture)의 구조에 바탕을 두고 Sun JCE와 호환되는 독자적인 JCE를 구현하기 위해 노력하고 있다. 이러한 작업 결과로 JCP(영국), IAIK-JCE(오스트리아), JCSI(호주), FORGE(호주), ABA(호주), 그리고 Cryptix(다국적) 등이 있으며, 국내에서도 현재 CEAL98(장미디어 인터렉티브)과 J/LOCK (씨큐리티 테크놀로지스)의 제품이 개발되었다.

본 고에서는 JDK를 중심으로 자바 보안 모델과 자바의 암호 구조에 대하여 분석해 본다. 2장에서는 자바의 보안 모델의 3요소에 대하여 알아보고, 3장에서는 여러 가지 보안 문제를 극복하려는 JDK 보안 구조의 변화와 보호 메커니즘에 대하여 살펴본다. 그리고 4장에서는 자바 보안 API에 대하여 자세히 살펴보며, 마지막으로 5장에서 결론을 맺는다.

Ⅱ. 자바 보안 모델

자바에서 보안은, 언어-수준의 보안, 자바 가상 머신(Java Virtual Machine)-수준의 보안, 그리고 API(Application Programming Interface)-수준의 보안 메커니즘으로 제공된다. 첫째 자바 언어-수준의 보안은 기존의 언어인 C나 C++에 비하여 좀더 안전한 프로그램을 작성하도록, 사용자가 메모리를 직접 접근하는 것을 방지하기 위하여 포인터 연산을 배제하였고, 가비지 콜렉션(Garbage Collection) 기능을 제공하고 있다[1, 2]. 둘째, JVM-수준의 보안에서, 바이트코드 검사기(ByteCode Verifier)는 정당한 자바 코드만 실행되도록 보장하고, 클래스 로더(Class Loader)는 애플릿에 이름 영역(name space)을 할당하여 실행될 수 있는 환경을 제한하고 있다. 그리고 보안 관리자(Security Manager)는 시스템의 자원 접근을 통제하는 기능을 제공하고 있다. [그림 2-1]은 자바 어플리케이션의 구조를 나타내고 있다[3, 5].

1. 바이트코드 검사기(ByteCode Verifier)

바이트코드 검사기는 바이트코드를 통과시켜 타입-상태 정보를 구성하고, 모든 바이트코드 명령에 대한 파라미터의 타입을 검사한다. 다시 말해서, 바이트코드가 자바 규칙에 의거하여 작동하는지 또는 악성 컴파일러에 의해 작성된 바이트코드인지를 검사할 필요가 있다. [그림 2-2]는 자바 컴파일러를

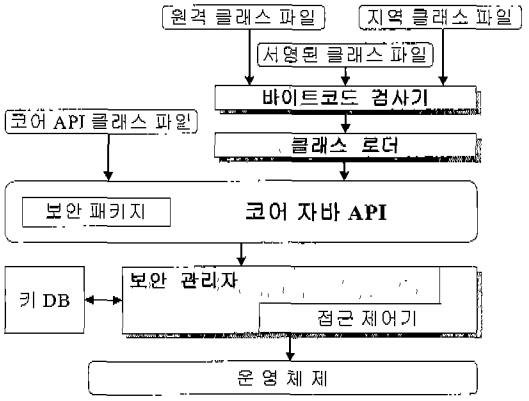

그림 2-1. 자바 어플리케이션의 구조

통하여 자바 언어 소스 코드로부터 바이트코드 검사기를 거쳐 자바 인터프리터에 이르기까지 데이터와 제어의 흐름을 보여준다[3, 5, 6, 7].

여기서 중요한 점은 자바 바이트코드 검사기와 바이트코드 로더가 첫 번째 바이트코드 스트림의 소스에 대한 고려를 하지 않는다는 것이다. 이 코드는 지역 시스템의 것일 수도 있고 네트워크를 통해 온 것일 수도 있다. 바이트코드 검사기는 일종의 문지기처럼 동작하고, 자바 인터프리터로 전달될 코드가 실행되기에 적절한 상태로 있는지를 보장하고, 자바 인터프리터의 중지를 걱정할 필요 없이 실행할 수 있다는 것을 보장한다. 일단 들어온 코드는 바이트 검사기의 테스트를 통과할 때까지 실행할 수 없다. 바이트코드 검사기가 작업을 완료했을 때, 비로소 다음과 같은 중요한 속성들을 알 수 있다.

- 피연산자 스택 오버플로우나 언더플로우가 없다.
- 모든 바이트코드 명령의 파라미터 형태는 정확하다.
- 객체 필드 접근은 합법적이다(예: private, public, 또는 protected).

이러한 검사가 너무 자세하게 보일 수도 있지만, 바이트코드 검사기가 이 작업을 끝냈을 때, 자바 인터프리터는 코드가 안전하게 실행될 수 있다는 것을 확인한 뒤에 동작을 시작한다. 이들 속성들은 자바

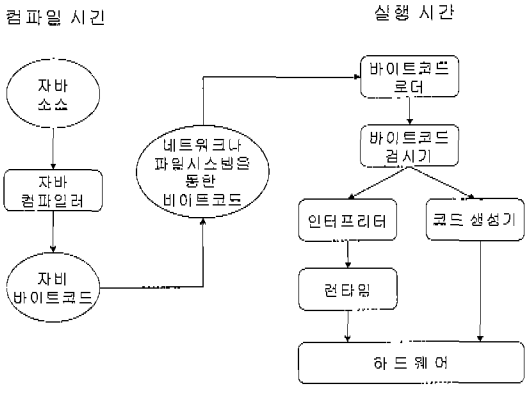

그림 2-2. 바이트코드 검사기

인터프리터를 매우 빠르게 동작하도록 만든다. 왜냐하면 이것은 아무 것도 검사할 필요가 없기 때문이다. 여기에서는 피연산자 형 검사나 스택 오버플로우 검사를 하지 않는다. 이와 같이, 인터프리터는 신뢰성을 잃지 않고 빠른 속도로 동작할 수 있다.

2. 클래스 로더(Class Loader)

클래스 로더는 실행 시간에 소프트웨어 컴포넌트를 설치할 수 있는 기능을 자바 플랫폼에 제공하기 때문에, 클래스 로더는 자바 가상 머신의 중요한 특징 중 하나이다. 클래스 로더는 여러 가지 특징을 가지고 있다. 첫째, 느린 로딩(lazy loading)은 클래스들의 요구가 있을 때만 로드된다. 둘째, 동적 클래스 로딩은 연결시간 검사를 추가함으로써 자바 가상 머신의 형(type) 안전성을 유지한다. 연결시간 검

사는 런타임 검사를 대신하고 단지 한번만 실행된다. 프로그래머는 로드되는 클래스들의 원격지의 위치를 명시하거나, 클래스 로더의 보안 속성을 적절히 지정함으로써 자신의 클래스 로더를 정의할 수 있다. 마지막으로 클래스 로더는 다양한 소프트웨어 컴포넌트의 독립된 이름 영역(name space)을 제공하기 위해서 사용할 수 있다[3, 5, 6, 7].

예를 들어, 브라우저는 독립된 클래스 로더를 사용하여 다른 웹 페이지로부터 애플릿을 로드할 수 있다. 이렇게 하여 애플릿 클래스 사이의 독립성이 유지된다. 실제로 이들 애플릿들은 같은 이름의 클래스에 포함될 수 있는데, 이들은 자바 가상 머신에 의해 다른 형태로 취급되어 진다.

클래스 로더의 원리는 자바 프로그래밍 언어의 동적인 특성에만 집중되지 않는다. 클래스 로더는 클래스 파일을 적절히 위치시키고 가져오며, 보안 정책을 고려하고, 적절한 접근 권한을 가지고 클래스 객체를 정의할 책임을 가지고 있기 때문에, 클래스 로더는 보안 기능을 제공하는데 중요한 역할을 수행한다.

3. 보안 관리자(Security Manager)

자바 어플리케이션은 보안 지침서로서 자신의 보안 관리자 객체를 가질 수 있다. java.lang 패키지 안의 SecurityManager 클래스는 모든 자바 보안 관리자에 대한 부분적인 구현과 프로그래밍 인터페이스를 제공하는 추상 클래스이다. 어플리케이션은 기본적으로 보안 관리자를 가질 수 없다. 즉, 자바 런타임 시스템은 자동적으로 자바 어플리케이션에 대한 보안 관리자를 생성하지 못한다. 따라서 어플리케이션은 주체가 되는 모든 연산들을 보안 제약에 할당한다[3, 5, 6, 7, 8].

이러한 기본적인 사항을 바꾸기 위해, 어플리케이션은 자신의 보안 관리자를 생성하고 설치해야 한다. SecurityManager를 작성하여 보안 관리자를 생성하고 설치하는 방법은 다음과 같다.

사용자는 시스템 클래스의 getSecurityManager() 메소드를 이용하여 어플리케이션에 대한 현재의 보안 관리자를 얻을 수 있다.

SecurityManager appsm = System.getSecurityManager();

getSecurityManager()는 null을 반환한다. 만약 현재 어플리케이션에 대한 보안 관리자가 없다면, 자신의 메소드를 호출하기 전에 정당한 객체를 가지고 있는지 조사해야만 한다.

일단 보안 관리자가 있다면, 연산을 허용(또는 불허)하기 위한 접근권한을 요청할 수 있다. 자바 패키지 내의 클래스들은 이러한 방식으로 동작한다. 예를 들어, 자바 인터프리터를 종료하는 System.exit() 메소드는 종료 연산을 허용하기 위해 보안 관리자의 checkExit() 메소드를 이용한다.

```
SecurityManager security = System.
getSecurityManager();
if (security = null) {
        security.checkExit(status);
}
. . .
//code continues here if checkedExit()
returns
```

만약 보안 관리자가 종료 연산을 허용한다면, checkExit()는 정상적으로 반환하고 그렇지 않다면, checkExit() 메소드는 SecurityException에 들어간다. 이런 방법으로, 보안 관리자는 잠재적인 위협이 있는 연산을 허용 또는 불허한다.

SecurityManager 클래스는 다른 종류의 연산을 검사하는데 사용하는 여러 메소드를 정의하고 있다. SecurityManager의 checkAccess() 메소드는 쓰레드 접근을 확인하고, checkProperty

Access()는 특정 속성에 대한 접근을 검사한다. 각 연산 또는 그룹 연산은 자신의 checkXXX() 메소드를 갖는다.

그리고, checkXXX() 메소드의 집합은 자바 패키지 클래스 내의 연산 집합을 나타내고, 보안 관리자를 보호하기 위한 주체인 자바 런타임을 나타낸다. 그래서, 일반적으로 코드는 Security Manager의 checkXXX() 메소드를 코드에 가져오지 않아도 된다. 그러나 자신의 보안 관리자를 작성할 때, 특수 연산자들에 대한 보안 정책을 변경하거나 강화하기 위해 SecurityManager의 checkXXX() 메소드를 오버라이드 할 필요가 있을 수 있다. 또는, 보안 관리자의 정밀 검사 후에 다른 종류의 연산자들을 자신의 연산에 넣을 수 있도록 추가할 필요가 있을 수 있다. SecurityManager 클래스 내에서 각 checkXXX() 메소드는 보호되도록 설계되어야 한다.

Ⅲ. 자바 보안 구조

Sun사의 자바 보안 개발자들은 JDK 1.0 버전부터 JDK 1.1과 JDK 1.2를 거쳐, 현재 저자가 본 고를 작성할 시점에는 JDK 1.3 베타버전에 이르기까지 자바 보안 구조에 대하여 지속적인 연구를 하고 있다. 본 고에서는 현재 정식 버전인 JDK 1.2를 기준으로 하여 기술하고자 한다[3, 4, 5, 6, 7].

1. JDK 1.0 보안 모델

접근제어는 이전 버전의 자바 플랫폼보다 더욱 정교하게 발전되었다. sandbox 모델이라고 알려진 자바 플랫폼에 의해서 제공된 원래 보안 모델은 개방된 네트워크로부터 획득한 신뢰되지 않은 코드를 매우 제한된 환경에 제공하기 위해 존재하였다[3, 5, 9, 10].

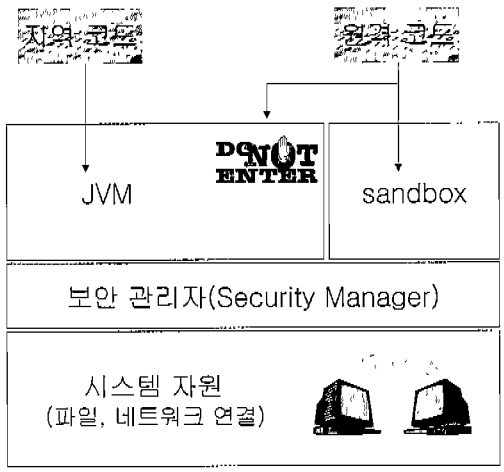

그림 3-1. JDK 1.0 보안 모델

[그림 3-1]에서 보이는 것처럼 이 sandbox 모델에서 지역 코드는 아주 중요한 시스템 자원에 대한 모든 접근을 가질 수 있도록 신뢰된다(예: 파일 시스템). 그러나 다운받은 원격 코드(애플릿)는 신뢰되지 않고, 단지 sandbox 안에서 제공되는 제한된 자원만을 접근할 수 있다. 이것은 보안 관리자의 책임이며, 아래의 [그림 3-1]은 자원 접근에 대하여 나타내고 있다.

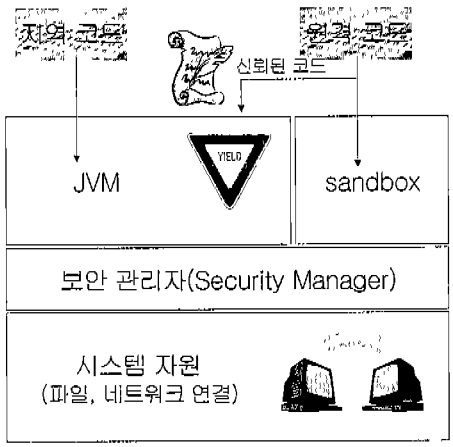

그림 3-2. JDK 1.1 보안 모델

2. JDK 1.1 보안 모델

[그림 3-2]에 나타난 것처럼, JDK 1.1은 신뢰된 서명 애플릿(trusted signed applet)의 개념을 도입하였다. 전자적으로 서명된 애플릿은 서명을 검증하기 위해 사용된 공개키가 신뢰된다면, 지역 코드처럼 신뢰되고 자원에 대한 모든 접근 권한을 가질 수 있다. 신뢰되지 못한 애플릿은 여전히 sandbox 내에서만 실행된다. 서명된 애플릿은 서명된 JAR(Java ARchive)파일에 각 서명들과 함께 전송된다.

3. JDK 1.2 보안 모델

JDK 1.2는 JDK 1.1에 비하여 많이 향상되었다. 지역코드나 원격코드는 보안 정책의 주체가 될 수 있다. 보안 정책은 다양한 서명자나 위치로부터 코드가 이용 가능한 접근 권한의 집합을 정의하고, 이것은 사용자와 시스템 관리자에 의해 구성될 수 있다[3, 5, 9, 10].

각 접근 권한은 특정한 파일이나 디렉토리에 대한 읽기나 쓰기 또는 지정된 호스트나 포트에 대한 연결 접근처럼, 특정한 자원에 대한 허용된 접근을 지정한다. 자바 런타임 시스템은 코드를 개별 도메인으로 구성한다. 도메인 각각은 같은 접근 권한 집합에 허용된 인스턴스 클래스 집합을 포함한다. 도메인은 sandbox 모델과 동일하게 구성될 수 있다. 그리고 애플릿은 사용자나 관리자가 이렇게 동작하도록 선택했다면, 여전히 제한된 환경에서만 실행될 수 있다. 응용 프로그램은 예전처럼 기본적으로 제한 없이 실행되지만, 선택적으로 보안 정책의 주체가 될 수 있다.

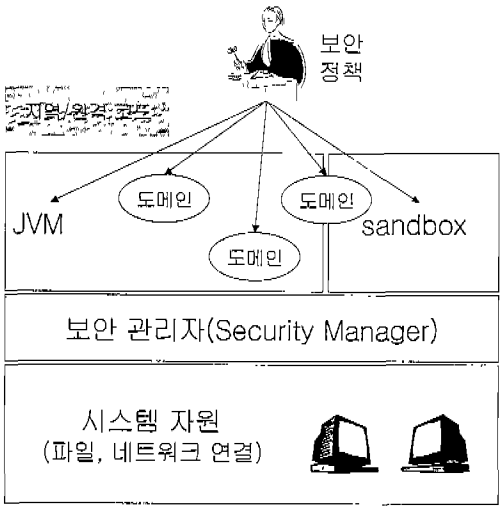

그림 3-3. JDK 1.2 보안 모델

JDK 1.2의 새로운 보안 모델이 [그림 3-3]에 나타나 있다. 왼쪽 끝의 화살표는 도메인을 의미한다. 이 도메인 코드는 자원에 대한 모든 접근이 허용된다. 오른쪽의 화살표는 이와 반대이다. 여기에서 도메인은 정확하게 sandbox에서처럼 제한된다. 이들 사이의 도메인은 완전한 접근 권한을 가지지는 못하지만 sandbox 보다는 많은 접근 권한을 가진다.

4. 보호 메커니즘(Protection Mechanism)

시스템 보안의 기본적인 개념과 원칙은 도메인(domain)이다. 도메인은 원칙에 의해 직접 접근할 수 있는 객체의 집합으로 한정될 수 있다. 여기에서 원칙은 접근 권한이 허가된 컴퓨터 시스템의 실체이다. JDK 1.0에서 이용된 sandbox는 고정된 한계를 가진 보호 도메인의 한 예라 할 수 있다[3, 11].

보호 도메인의 개념은 보호되는 단위 사이에서 그룹화하거나 고립시키기 위한 편리한 메커니즘을 제공한다. 보호 도메인은 두 개의 범주(시스템 도메인과 어플리케이션 도메인)로 나누어진다. 파일 시스템과 같은 모든 외부 보호 자원, 네트워크 장비, 그리고 스크린과 키보드는 시스템 도메인을 통해서만 접근되어질 수 있다. 아래 [그림 3-4]는 자바 어플리케이션 환경의 도메인 구성을 보여주고 있다.

도메인은 동일한 접근권한 집합으로 허가된 인스

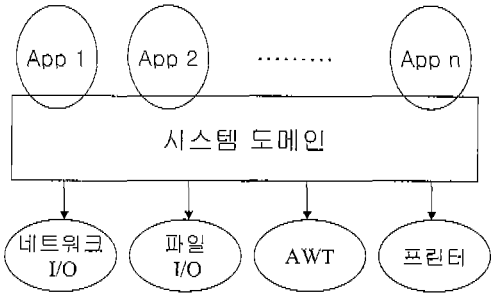

그림 3-4. 자바 어플리케이션 환경 도메인

턴스 클래스 집합을 개념적으로 수반한다. 일반적으로 보호 도메인은 현재 통용되는 정책에 의해 결정된다. [그림 3-5]의 자바 어플리케이션 환경은 코드(클래스와 인스턴스)의 보호 도메인 및 접근 권한에 대한 매핑을 보여주고 있다.

실행 쓰레드는 완전하게 단일 보호 도메인에서 실행되거나 어플리케이션이나 시스템 도메인에 속할 수도 있다. 예를 들어, 메시지를 출력하는 어플리케이션은 단지 출력 스트림에 대한 접근 지점이 있는 시스템 도메인하고만 상호 작용하여야 한다. 이 경우, 임의의 시간에 어플리케이션 도메인이 시스템 도메인을 호출함으로서 추가적인 접근권한을 얻지 못한다는 것은 중대한 일이다. 왜냐하면, 심각한 보안 관련 문제가 존재할 수도 있기 때문이다.

시스템 도메인의 메소드가 어플리케이션 도메인의 메소드를 호출하는 위에서와는 반대 상황에서(예, AWT 시스템 도메인이 애플릿을 보여주기 위해 애플릿의 paint 메소드를 호출할 때), 유효한 시스템 도메인에서의 접근 권한이 애플리케이션 도메인에서의 현재 권한과 동일할 때에는 또 다시 심각한 문제가 발생한다.

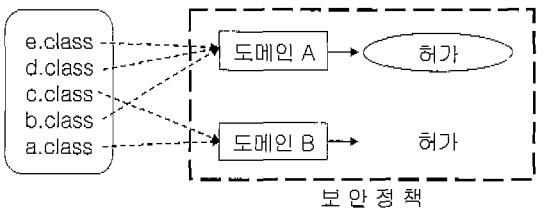

그림 3-5. 클래스의 도메인과 접근권한

두 보호 도메인에 모두 속하는 하나의 쓰레드에 대한 논의는 물론 다중 보호 도메인에 속하는 쓰레드에 대한 원칙을 제시하고 있다. 접근 권한을 산출해내는 중요한 법칙은 다음과 같다

■ 실행 쓰레드의 접근 권한 집합은 실행 쓰레드가 속하는 모든 보호 도메인의 접근 권한의 교차점으로 간주된다.

■ 어떤 코드가 doPrivileged 메소드를 호출할 때, 상술한 코드의 보호 도메인과 호출되거나 직·간접적으로 기입된 모든 보호 도메인에 의해 허용된다면, 실행 쓰레드의 접근 권한 집합은 접근 권한을 가지는 것으로 간주된다.

doPrivileged 메소드는, 일부 신뢰할 수 있는 코드가 이 코드를 일시적으로 호출한 어플리케이션에 직접 접근하는 것 이상의 더 많은 자원에 접근할 수 있도록 해 준다. 어떤 상황에서는 이 메소드가 필요할 수도 있다. 예를 들어, 어플리케이션은 폰트를 포함하는 파일에 대한 직접적인 접근이 허용되지 않을 수도 있지만, 문서를 보기 위해서는 시스템 유틸리티에서 그러한 폰트들을 사용자에게 제공하야 한다. 이 경우에 대처하기 위해 시스템 도메인을 위한 doPrvileged 메소드가 제공되며, 그 메소드는 모든 도메인에서 이용 가능하다.

실행 과정에서, 중요한 시스템 자원(파일 I/O와 네트워크 I/O와 같은)이 요청될 때, 자원-핸들링 코드는 요청을 평가해서 그 요청이 허용될지(또는 거부될지)를 결정하는 AccessController 클래스 메소드를 직·간접적으로 호출한다. 이러한 평가가 수행되어지는 실질적인 방법은 구현과정에서 변형될 수도 있다.

기본 원칙은 호출 내역과 관련된 보호 도메인에 허용된 접근권한들에 대해 조사하는 데, 이때 요청

이 허용되면 호출된 메소드에서 반환되며, 허용되지 않는다면 보안 예외처리 루틴을 실행한다.

마지막으로, 각각의 도메인(시스템 또는 어플리케이션)은 자신의 도메인 테두리 안에 있는 내부 자원들에 대해서도 보호할 수도 있다. 예를 들어, 은행 어플리케이션은 계좌 조회, 예금 그리고 예금 인출과 같은 내부적인 개념을 보호하고 지원할 필요가 있을 수 있다. 왜냐하면 이러한 보호의 의미는 JDK에 의해 구현되거나 선언될 수 없기 때문에, 이러한 단계의 보호 어플리케이션은 어플리케이션 개발자들이나 또는 시스템에 맡겨지는 것이 최선이다. 그럼에도 불구하고, 개발자들의 부담을 덜어주기 위한 SignedObject 클래스가 제공되어진다.

Ⅳ. 자바 보안 API

JCA(Java cryptography architecture)는 JDK 1.1의 JDK 보안 API의 첫 공개 버전에서 소개되었다. JCA는 자바 플랫폼에서의 암호 기능을 접근하고 개발하기 위한 구조이다. JCA는 다수의 상호-동작할 수 있는 암호 구현을 위한 프로바이더 구조를 포함하고 있다. 프로바이더, 다시 말해서 CSP (cryptographic service provider)는 JDK 보안 API의 암호 부분의 구체적인 구현을 제공하는 패키지이다[3, 4, 13, 14].

JDK 1.1에서, 프로바이더는 디지털 서명 알고리즘, 메시지 다이제스트 알고리즘, 그리고 키 생성 알고리즘의 구현을 포함하였다. 그 뒤로 JDK 1.2는 아래의 서비스 형태를 추가하였다.

- 키 스토어(Keystore) 생성과 관리
- 알고리즘 파라미터 생성
- 알고리즘 파라미터 관리
- 키 팩토리(Key factory)는 다른 키 표현들 사이의 변환을 가능하게 한다.
- 인증서 팩토리(Certificate factory)의 인코딩으로 인증서와 인증서 취소 리스트를 생성할 수 있다.

또한, JDK 1.2는 프로바이더가 RNG (random-number generation) 알고리즘을 제공한다. 자바 런타임 환경의 Sun 공개판은 SUN이라는 이름을 가진 기본 프로바이더를 가지고 있다. SUN 프로바이더 패키지는 다양한 DSA(Digital Signature Algorithm) 서비스, MD5(RFC 1321)와 SHA-1(NIST FIPS 180-1) 메시지 다이제스트 알고리즘의 구현, X.509 인증서를 위한 인증서 팩토리, 인증서 취소 리스트, PNG (pseudo-random-number generation) 알고리즘, 그리고 키 스토어의 구현을 포함하고 있다.

JCE는 암호, 키 교환, 그리고 MAC(message authentication code)을 위한 API를 포함하도록 JDK를 확장하고 있다.

JCE와 JDK의 암호 관점은 완벽하고 플랫폼 독립적인 암호 API를 제공한다는 것이다. JCE는 미국의 수출 규제법에 따라 JDK와 독립적으로 발표되었다. 다음의 [그림 4-1]은 다양한 JCA 모듈을 보여준다.

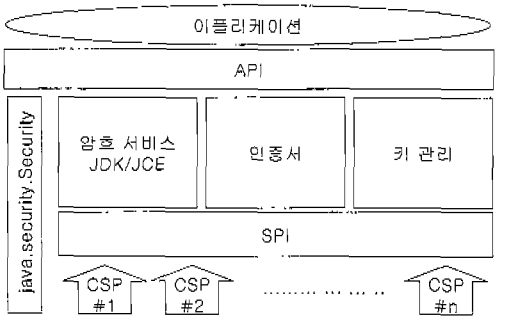

그림 4-1. JCA 모듈

1. 암호 서비스

JDK 1.2 내에는 새로운 engine 클래스들이 추가되었다. engine 클래스는 추상적인 스타일로 암호 서비스를 정의한다. engine 클래스는 응용 프로그램이 제공하는 특정한 형태의 암호 서비스에 접근할 수 있도록 API 메소드를 정의한다(예: 디지털 서명 알고리즘). 실제로 하나 이상의 프로바이더로부터의 구현은 특정 알고리즘을 위한 것들이다.

engine 클래스에 의해 제공된 API는 SPI(service provider interface)에 의해 구현된다. 즉, 각 engine 클래스는 CSP가 제공해야만 하는 SPI를 정의한 상응하는 추상 SPI 클래스를 가지고 있다. 예를 들어, API 클라이언트는 파일을 전자서명하고 전자서명 알고리즘의 기능에 접근하기 위해 Signature engine 클래스의 인스턴스를 요청하고 사용해야한다. SignatureSpi 서브 클래스에 제공된 실제 구현은 특정한 종류의 서명 알고리즘을 위한 것이다(예: SHA-1 with DSA 또는 MD5 with RSA).

각 engine 클래스의 인스턴스는 CSP에 의해 구현된 것처럼 상응하는 SPI 인스턴스를 캡슐화한다. 각 engine 클래스의 API 메소드는 캡슐화된 SPI 객체의 상응하는 SPI 메소드를 수행한다.

2. 인증서 클래스와 인터페이스

JDK 1.2는 인증서를 파싱하고 관리하기 위한 클래스 인터페이스와 클래스를 도입하였다. 그리고 인증서 인터페이스를 위한 X.509v3를 제공한다. 인증서는 기본적으로 하나의 실체(예: 사람, 회사)로부터 전자적으로 서명된 문장이다.

인증서와 관련된 클래스들은 아래와 같고, 이들 모두는 java.security.cert 패키지 안에 포함되어 있다.

- Certificate : 이 클래스는 다양한 형태를 가지지만 일반 사용자에게 중요한 인증서를 위한 추상형이다. 예를 들어, 다양한 형태의 인증서(예: X.509, PGP)는 일반적인 인증서 기능을 공유한다(예: 인코딩과 검사). 그리고 여러 형태의 정보(예: 공개키, X.509, PGP 그리고 SDSI 인증서)는 심지어 이것들이 다른 정보의 집합을 포함하고 다른 방법으로 정보를 저장하고 검색할지라도 모두 Certificate 클래스를 서브클래스화 함으로써 구현될 수 있다.

- CertificateFactory : 이 클래스는 인증서 팩토리에 대한 기능을 정의한다. 이것은 인코딩으로부터 인증서와 인증서 취소 리스트를 생성하기 위해 사용된다.

- X509Certificate : X.509 인증서를 위한 이 추상 클래스는 X.509 인증서의 모든 속성에 접근하기 위한 표준방법을 제공한다.

3. 키 관리 클래스와 인터페이스

JDK 1.1은 추상 Key 인터페이스를 도입하였고, JDK 1.2는 다음을 추가하였다.

- KeyStore class : keystore 의 정보에 접근하고 수정하기 위한 인터페이스를 제공하는 KeyStore 클래스(엔진 클래스). 다수의 다른 구체적인 구현이 가능하다. 여기에서 각 구현은 특별한 형태의 keystore이다. keystore 타입은 keystore 정보의 저장과 데이터 형태를 정의한다.

- default KeyStore : JKS라는 적절한 keystore 형태를 사용하여 파일로서

keystore를 구현하는 구현물이다. keystore 구현은 개별 패스워드를 사용하여 각 개인 키를 보호하고 또한 가능한 다른 패스워드를 사용하여 전체의 keystore의 무결성을 보호한다.

- Key specification interfaces : 키를 캡슐화 하는 키 재료의 투명한 표현을 위해 사용된다. 키를 위한 키 재료는 키 그 자체로 구성된다. 그리고 알고리즘 파라미터는 키 값을 캡슐화 하기 위해서 사용된다. 키의 투명한 표현은 개별적으로 각각의 키 재료에 접근할 수 있다는 것을 의미한다. 키와 인증서를 관리하기 위해 keytool을 사용할 수 있다.

V. 결 론

자바에 의해 제공되는 기능은 JDK에 의해 제공되는 일반적인 보안 기능과 JCE에 의해 제공되는 암호 기능으로 분류할 수 있다. JDK의 초기 버전에서는 많은 보안 문제점들이 발생하였다. 이러한 문제점들은 CERT의 CA-96.05와 CA-96.07에 발표되었으며, 미국의 프린스톤 대학과 워싱턴 대학에서도 자바의 보안 취약성에 대하여 많은 연구가 있다.

이러한 취약성으로 네트워크로부터 다운로드된 바이트코드를 클라이언트에서 실행시켜 하드디스크를 삭제하거나, 클라이언트에 연결된 네트워크를 위험에 빠뜨릴 수 있다. 다시 말해서, 자바는 다양한 플랫폼에서 실행될 수 있도록 개발되었기 때문에, 자바 개발자들과 웹 브라우저 개발자들은 특정 H/W나 O/S가 제공하는 보안 기능에 의존하지 않고 S/W로 보안 기능을 구현하여, 네트워크 보안에 중점을 두어 개발하였다. 현재 자바는 이러한 취약성을 극복하였기 때문에, 인터넷을 이용한 전자상거래와 같은 분야에 많이 응용되고 있다.

한편, 전자상거래와 같은 분야에 응용되려면 공개키, 전자서명, 인증서 및 키 관리와 관련된 기능을 제공하는 암호 API가 필요하다. 그래서 Sun사에서는 JCE라는 패키지를 별도로 제공하고 있다. 하지만 이것은 미국과 캐나다에서만 이용이 가능하기 때문에, 그 외의 국가에서는 기술종속의 이유로 이 분야에 대한 연구가 활발히 진행되고 있으며, 현재 구현된 암호 프로바이더로는 IAIK-JCE(오스트리아)[15], JCSI(호주)[16], FORGE(호주)[17], ABA(호주)[18], 그리고 Cryptix(다국적)[19] 등이 있으며, 국내에서도 현재 CEAL98(장미디어 인터렉티브)[20]과 J/LOCK(씨큐리티 테크놀로지스)[21] 등이 개발 중에 있다.

만약, 외국의 제품을 국내에 수입하여 사용하게 될 경우 발생할 수 있는 여러 가지 문제점(예; 트랩도어)으로, 이 분야에 대한 국내의 자체 연구가 필요하다. 자바 암호 및 보안 프로토콜 기술의 국내 보유는 국내·외 산업체와 연계된 기술 수요 및 다양한 응용 서비스에 대한 사용자의 폭발적인 증가를 보장하여, 국내 산업 경쟁력이 선진국 수준에 용이하게 도달하는 효과가 있을 것이다.

※ 참고문헌

[1] 이강수, "Java 환경에서의 보안 위협과 메커니즘", 정보과학회지, 제15권, 제7호, 1997년 7월.

[2] 이완석, 김홍근, "자바 보안 모델", 정보과학회지, 제16권, 제4호, 1998년 4월.

[3] Security Features Overview, http://java.sun.com/docs/books/tutorial/security1.2/overview/index.html

[4] Java Cryptography Extension 1.2, http://java.sun.com/security/

JCE1.2/spec/apidoc/index.html
[5] Li Gong, "Inside Java 2 Platform Security: Architecture, API Design, and Implementation" Addison-Wesley Pub Co., June 1999.
[6] Scott Oaks, "Java Security", O'Reilly & Associates, May 1998.
[7] IBM Redbook, "Java 2 Network Security", IBM, June 1999.
[8] L. Gong, "Secure Java Class Loading". IEEE Internet Computing, (2)6, November/December 1998.
[9] L. Gong, "Java Security: Present and Near Future". IEEE Micro, 17(3), May/June 1997.
[10] L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers, "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2", In Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California, December 1997, pp.103-112.
[11] L. Gong and R. Schemers, "Implementing Protection Domains in the Java Development Kit 1.2", In Proceedings of the Internet Society Symposium on Network and Distributed System Security, San Diego, California, March 1998, pp.125—134.
[12] J. Gosling, Bill Joy, and Guy Steele. "The Java Language Specification. Addison-Wesley", Menlo Park, California, August 1996
[13] D. Maughan, M. Schertler, M. Schneider and J. Turner, Internet Security Association and Key Management Protocol(ISAKMP), Internet-Draft draft-ietf-ipsec-isakmp-10.txt, work in progress, Internet engineering Task Force, July 1998.
[14] Jonathan Knudsen, "Java Cryptography", O'Reilly & Associates, May 1998.
[15] IAIK-JCE, http://jcewww.iaik.tu-graz.ac.at/jce/jce.htm
[16] Java Crypto and Security Implementation, http://security.dstc.edu.au/projects/java/jcsi.html
[17] Forge Security Provider, http://www.forge.com.au/products/crypto/index.html
[18] The Open JCE Project for Java (ABA), http://www.aba.net.au
[19] Cryptix, http://www.cryptix.org
[20] CEAL 98, http://crypto.jmi.co.kr/product/package/ceal/ceal.html
[21] J/LOCK, http://www.stitec.com/index.html

조 한 진

1997년 한남대학교 컴퓨터공학과(학사)
1999년 한남대학교 대학원 컴퓨터공학과(석사)
현재 한남대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 컴퓨터네트워크, 자바 보안, 전자상거래 보안

이 희 규

1998년 우송대학교 컴퓨터과학과(학사)
2000년 한남대학교 대학원 컴퓨터공학과(석사)
현재 한남대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 컴퓨터네트워크, 자바 보안, 전자상거래 보안

김 봉 한

1994년 청주대학교 전자계산학과(학사)
1996년 한남대학교 대학원 전자계산공학과(석사)
2000년 한남대학교 대학원 컴퓨터공학과(박사)
현재 한남대학교 강사
관심분야 : 컴퓨터네트워크, 멀티캐스트, 정보보호

이 재 광

1984년 광운대학교 전자계산학과(학사)
1986년 광운대학교 대학원 전자계산학과(석사)
1993년 광운대학교 대학원 전자계산학과(박사)
1986년~1993년 군산전문대학 전자계산학과 부교수
1997년~1998년 University of Alabama 객원교수
1993년~현재 한남대학교 컴퓨터공학과 부교수
관심분야 : 컴퓨터 네트워크, 정보통신 정보보호