□ 기술해설 □

자바 보안 모델

한국정보보호센터 이완석·김홍근

1. 서 론

인터넷의 발전·보편화와 함께 인터넷 환경에 적합하게 개발된 자바 언어는 매우 폭 넓은 분야에서 다양하게 사용되고 있다. 특히 자바 언어로 작성된 애플릿(applet)이라는 작은 실행 코드를 서버로부터 다운로드 받아 클라이언트에서 실행함으로써, 서버와 클라이언트가 작업 부하를 나눈다는 점과 다양한 플랫폼에서 실행될 수 있다는 점은 특히 많은 관심을 받고 있다[5, 12, 14,17]. 하지만 네트워크에서 다운로드 받은 프로그램을 클라이언트에서 실행할 경우 다운로드된 프로그램이 클라이언트의 하드디스크를 삭제한다거나, 클라이언트에 연결된 내부 네트워크를 위험에 빠뜨릴 수 있다[10, 14]. 따라서 자바 언어로 작성된 애플릿과 같은 모빌(mobile) 코드를 사용할 경우 보안에 많은 관심을 가져야 한다. 자바는 개발 단계에서부터 네트워크와 보안에 중점을 두어 개발되었다. 즉, 다양한 플랫폼에서 실행될 수 있도록 플랫폼 독립성을 목표로 하였기 때문에 특정 하드웨어나 운영체제가 제공하는 보안 기능에 의존하지 않고 소프트웨어적으로 보안 기능을 구현하였다. 소프트웨어로 구현된 보안 기능을 사용할 경우, 이식성이 뛰어나며 성능 또한 우수하다는 시험 결과가 발표된 바 있다[8]. 지금까지 웹 브라우저나 자바가능 응용 프로그램(java enabled application)을 실행할 안전한 플랫폼 제공에 관한 노력은 주로 자바 개발 툴킷 JDK를 통하여 이루어져 오고 있다. 본 고에서는 JDK를 중심으로 이루어 지고 있는 보안 구조의 개념과 변화를 고찰해 보고자 한다. 2장에서는 자바 보안 모델에 대해 알아보고, 3장에서는 JDK에 구현한 보안 구조의 변화를 1.0, 1.1 그리고 1.2 베타 버전 각각에 대하여 기술한다. 마지막으로 4장에서 결론을 맺는다.

2. 자바 보안 모델

자바에서 보안은 여러가지 보안 메커니즘의 제공으로 이루어지고 있다. 첫째는 자바 언어 측면에서 기존의 C나 C++에 비하여 프로그래머가 좀더 안전한 프로그램을 작성할 수 있도록 자동 메모리 관리, 가비지 콜렉션, 스트링·배열의 범위 점검 등의 기능을 제공한다. 이러한 언어 측면에서의 자바 안정성(safety) 연구도 일부 이루어지고 있으며[3, 5, 12], 본 고에서 이 주제는 생략하기로 한다. 바이트코드 검사기(bytecode verifier)는 합법적인 자바 코드만 실행되도록 보장하며, 클래스 로더(class loader)는 애플릿에 name space를 할당하여 실행 환경을 제한한다. 마지막으로 보안 관리기(security manager)는 시스템 자원 접근 통제 기능을 제공한다. 이들은 각각의 맡은 바

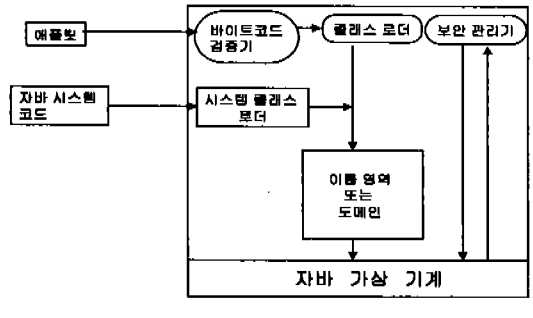

그림 1 자바 보안 모델의 3요소

임무를 수행함으로서 전체 자바 보안 기능을 제공한다.

2.1 클래스 로더

자바 애플릿 클래스 로더는 현재 실행되고 있는 자바 환경에 언제 어떻게 애플릿이 새로운 클래스를 로드할 수 있는가를 결정한다. 애플릿 클래스 로더는 자바 실행환경의 주요한 시스템 코드가 애플릿이 로드하는 코드와 대체되지 않도록 통제하는 역할을 수행한다. 다운로드된 바이트코드를 로드하기 위해 자바 런타임 시스템은 클래스 로더 슈퍼 클래스의 서브 클래스로 구현된 클래스 로더를 호출한다. 다시 말해, 네트워크로부터 애플릿을 다운로드 할 때, 자바 브라우져는 애플릿 클래스 로더를 호출한다. 클래스 로더는 네크워크로부터 애플릿 실행 코드를 가져오는 역할뿐만 아니라 이름 영역 계층구조(name space hierarchy)를 다운로드 받은 코드에게 할당하는 역할을 수행한다. 이름영역은 애플릿이 자바 가상기계의 자원들, 특히 시스템 보안에 관련된 자원들에 대한 접근을 제어한다.

브라우져와 같은 자바 응용 프로그램에서는 JDK 클래스 로더 클래스의 서브 클래스를 생성하여 클래스를 로드하거나 다운로드 받은 코드에 이름 영역을 할당하는데 사용한다. 만약 애플릿이 클래스 로더를 생성할 수 있다면 애플릿이 자체의 이름 영역을 만들 수 있다는 것을 의미한다. 이러한 상황이 발생한다면 애플릿이 시스템의 완전한 통제권을 행사할 수 있다.

각각의 애플릿이 자신의 이름 영역을 가지고 있도록 하는 기법에는 두 가지의 장점이 있다. 첫째 애플릿에서 서로 같은 이름의 변수를 사용한다 하더라도 문제가 발생하지 않도록 하며, 둘째는 애플릿간의 통신을 어렵도록 한다. 자바 가상 기계에서는 로드된 각각의 클래스마다 이름 영역에 대한 정보가 붙어, 클래스가 다른 클래스를 호출하고자 한다면 정해진 순서를 따라 클래스를 찾아가도록 하였다. 즉, 내장 클래스(built-in 클래스로서 JDK에서 기본적으로 제공하는 클래스)를 먼저 찾아가도록 하였으므로 다운로드 된 클래스가 내장 클래스인 것처럼 행동하지 못한다. 이러한 정책은 애플릿이 입출력 작업을 통하여 권한이 없는 파일 시스템을 사용하지 못하도록 한다.

클래스 로더가 바이트코드를 로드하면 클래스 로더는 바이트코드 검사기를 실행하여 바이트코드에 관한 보안 검사를 수행하도록 한다.

2.2 바이트코드 검사기

자바 프로그램이 자바 컴파일러에 의해 컴파일 되면 바이트코드라는 .class파일이 생성된다. 자바 컴파일러에서는 강력한 타입 점검을 하지만 바이트코드가 자바의 규칙에 의거 작동하는지 또는 악성 컴파일러에 의해 작성된 바이트코드인지 검사할 필요가 있다. 바이트코드 검사기는 실행시간에 이루어져야 할 각각의 명령에 대한 점검을 대신 해주기 때문에 인터프리터의 성능을 향상시킨다. 인터프리터는 위의 사항들에 대해서는 이미 점검이 이루어진 것으로 간주하여 실행시간에는 이러한 점검을 하지 않으므로 더 빨리 수행될 수 있다.

2.3 보안 관리기

자바 브라우져의 보안 구성 요소 중 보안 측면에서 볼 때 보안 관리기의 구현이 다른 어느 것보다 중요하다. 이것은 애플릿의 실행환경에 대해 샌드박스(sandbox)라는 보안 울타리를 치는 것이 바로 이 보안 관리기이기 때문이다. 샌드박스는 보안 관리기에 설정되어 있는 보안 정책에 의해 애플릿의 모든 활동을 통제함으로서 생성되는 논리적인 울타리이다.

어떤 애플릿이 로컬 시스템의 자원을 사용하려 하거나 이로부터 정보를 얻으려고 한 경우, 자바 가상기계는 우선 보안 관리기에게 이러한 작업이 수행되어도 좋은지를 문의한다. 만약 보안 관리기가 작업을 승인하면-예를 들어 로컬 디스크로부터의 애플릿이 디스크를 읽으려 하거나, 네트워크를 통하여 다운로드된 애플릿이 원래 자신이 저장되어있던 서버와 연결을 맺으려하는 등의 경우-가상기계는 해당 동작을 수행한다. 만약 보안 관리기에 의해 승인되지 않으면 가상기계는 보안 예외상황을 발생시키며 자바 콘솔에 오류 메시지를 출력한다.

응용 프로그램이나 자바 브라우져는 단 하나

만의 보안 관리기를 가질 수 있다. 이렇게 함으로써 모든 접근에 대한 검사가 하나의 보안 정책을 따르도록 요구하는 유일한 보안 관리기에 의해 이루어지도록 보장할 수 있다. 보안 관리기는 로드된 후에 확장되거나, 대체되거나, 오버라이트 될 수 없다. 물론 당연한 이야기이겠지만, 애플릿은 자신의 보안 관리기를 생성할 수 없다. 보안 관리기는 보안 관리기라는 클래스를 서브클래스화 하여 구현한다. 보안 관리기는 모든 보안 기능을 위해 응용 프로그램 인터페이스(API)를 제공하며, checkRead (파일에 대한 읽기 권한을 검사), checkCreateClassLoader 메쏘드 등이 이 클래스에 포함되어 있다.

3. 자바 보안 구조의 변화

자바 개발자들은 JDK 1.0으로 출발하여 1.1 그리고 1.2 베타 버전에 이르기까지 자바 보안 구조를 계속해서 진화시켜오고 있다. JDK를 보다 보안성이 강화된 안전한 플랫폼으로 만들려는 노력은 컴퓨터 보안(computer security)의 이론적 연구 측면보다는 이미 잘 알려져 있는 보안 원리(security principles)를 공학적 실제로 구현하려는 측면에서 이루어지고 있다. 본 장에서는 JDK 1.0에서 1.1, 1.2 베타 버전까지의 자바 보안 구조에서의 변화를 고찰해 보기로 한다.

3.1 JDK 1.0 버전의 샌드박스 모델

JDK 1.0은 앞에서 언급된 클래스 로더와 보안 관리기를 사용하여 샌드박스 모델을 구현하였다. 클래스 로더는 외부에서 다운로드된 코드와 로컬 코드에 서로 다른 권한을 부여하여 실행될 수 있도록 이름 영역을 할당하였으며, 보안 관리기는 다운로드받은 코드가 주어진 이름 영역 내에서만 활동할 수 있도록 감시하며 통제하는 역할을 수행한다. 샌드박스 모델은 인터넷이나 다른 시스템에서 다운로드된 프로그램은 모두 외부 프로그램으로 간주하여 제한된 권한으로 실행되도록 하였으며, 로컬시스템에 저장된 자바 시스템 프로그램을 포함한 모든 프로그램은 신뢰할 수 있는 프로그램으로 간주하여 모든 권한을 기지고 실행되도록 하였다. 외부 프로그램이 실행할 수 없는 기능은 로컬 시스템의 파일에 대한 삭제 및 열람, 프로그램이 다운로드된 시스템 외의 컴퓨터와 통신연결, sun.* 계층에 있는 패키지로의 접근, java.* 계층에 새로운 클래스 생성, Runtime.exe()를 사용한 클라이언트 시스템내의 프로그램 실행 등이 있다.

JDK 1.0의 샌드박스에서 다운로드된 코드가 시스템 보안에 관련된 모든 기능들 즉, 파일의 쓰기, 읽기, 삭제 및 네트워크 기능 등을 사용하지 못하도록 하였음에도 불구하고 JDK 1.0에서는 많은 취약점이 발견되었다. 예를 들면, 악성 코드를 사운드나 그래픽 파일로 위장하여 temporary 디렉토리나 cache 디렉토리에 다운로드받아 저장하게 한 후 다운로드된 또다른 코드에 의해 실행되게 함으로써 로컬 코드의 권한을 가지도록 하였다. 또 다른 예는 두 개의 다른 시스템이 같은 URL 주소를 가질 수 있다는 점을 이용하여 실제로 다운로드된 코드가 저장되어있지 않았던 공격자의 시스템과 통신 연결이 가능하였다[5].

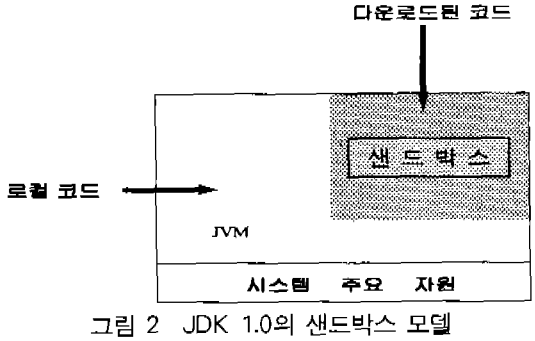

그림 2 JDK 1.0의 샌드박스 모델

3.2 JDK 1.1 버전의 전자 서명 (signed applet) 모델

JDK 1.1 버전에 구현된 전자 서명 모델은 1.0 버전의 샌드 박스 모델의 인터넷에서 다운로드 받은 애플릿의 실행 환경을 다시 두가지로 구분하여 약간의 융통성을 부여한 것에 불과하다. JDK 1.0의 경우 애플릿이 저장되어 있는 위치(local 또는 remote)에 따라 애플릿 실행 환경을 샌드 박스로 제한할 것인가 말 것인가를 결정한다. 이 방식은 무조건 원격 애플릿의

실행 환경을 제한하였기 때문에 애플릿을 많은 응용에 적용하기 어렵게 만들었다. 따라서 원격 애플릿도 다시 구분하여 신뢰성을 부여하자는 것이 전자 서명 모델이다. 작성된 프로그램과 서명은 JAR 파일 포맷으로 다운로드되며, 다운로드된 파일은 브라우저나 자바 실행 프로그램에 의해 프로그램과 서명으로 분류된다. 브라우저는 서명이 신뢰된 자의 것이라고 인정되면 로컬 시스템의 프로그램과 같은 권한으로 실행한다. 그렇지 않으면 JDK 1.0에서와 같이 샌드박스 내에서만 실행된다.

서명을 사용하는 방법 또한 취약점이 발견되었다[5, 17]. 코드 작성자의 서명을 인증하기 위해서는 신뢰된 작성자들의 서명키를 저장 관리하는 데이타베이스가 클라이언트에 존재해야 한다.

문제는 서명키 데이타베이스가 시스템 코드 외에는 접근할 수 없어야 하지만 네트워크 상에서 다운로드 받아 샌드박스에서 실행되는 코드까지도 서명키 데이타베이스에 접근하여 서

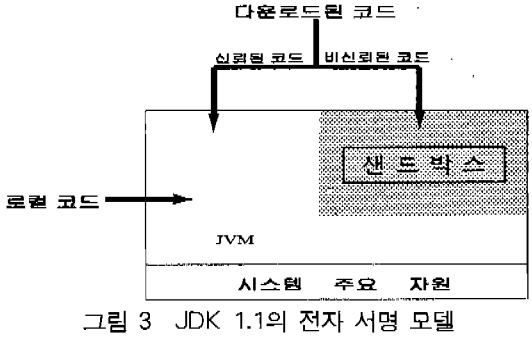

그림 3 JDK 1.1의 전자 서명 모델

명키를 열람할 수 있다는 것이다. 이에 따라 공격자는 서명키 데이타베이스에 저장된 정보를 전송받아 공격자가 작성한 코드에 서명함으로써 클라이언트의 모든 권한을 가지고 실행될 수 있었다.

또한 서명 여부에 따라 신뢰할 수 없다고 분류된 프로그램은 여전히 샌드박스 내의 제한된 권한으로만 실행되므로, 프로그램이 가능한 한 많은 권한을 가지고 실행되기를 바라는 사용자들의 욕구를 완전히 만족시키지 못하였다. 이러한 점들을 보완하기 위하여 개발된 것이 fine-grained 접근 통제 모델이다.

3.3 JDK 1.2 버전의 fine-grained 접근 통제 모델

이전의 두 버전에서 보여 주었던 보안 구조에 비교하여 진일보한 느낌을 주는 JDK 1.2는 아직 베타 버전만이 배포되고 있다[13]. 자바의 보안 개발자가 JDK 1.2에서 제공하는자 하는 보안 구조의 목표는 다음의 4가지로 요약될 수 있다[3].

• JDK 1.1에서의 접근 제어 방식도 애플릿의 다양한 응용을 여전히 제한하고 있어, JDK 차원에서 좀더 정교하고 세분화된 접근 제어 기법이 도입될 필요가 있다. 물론 JDK 1.0(1)에서 JDK가 제공하는 접근 제어 수준을 벗어나는 좀더 정교한 접근 제어를 구사할 수 있다. 이를 위해 SecurityManager와 ClassLoader 클래스를 서브클래스화 하고, 요구되는 보안 기능을 코드화하는 상당한 수준의 구현 작업이 요구되었다. 이러한 프로그래밍 구현 작업은 아주 깊은 컴퓨터 보안 지식을 수반하고 구현자에 따라 보안에 아주 민감한 영향을 미친다. 이를 해결하기 위해 JDK 1.2에서는 프로그래머가 좀더 안전하고(secure) 간단하게 정교하고 세분화된 접근 제어를 구사할 수 있도록 한다.

• 보안 기능을 수행하는 코드를 작성하는 것이 그리 간단치 않기 때문에 프로그램으로 보안 요구조건을 표현하는 것보다 보안 정책(security policy)을 구성(configuration)하도록 하는 것이 더 바람직하며, 이러한 보안 정책은 상황에 따라 융통성 있게 조정할 수 있어야 한다.

• 접근 제어를 위해 객체가 갖는 접근 권한을 유형화하고 이들의 처리를 자동화 한다. 따라서 JDK 1.1에서 보였던 필요할 때 마다 새로운 접근 권한을 위한 메쏘드를 SecurityManager 클래스에 추가하는 일은 없도록 한다.

• 모든 자바 코드에 대한 보안성 검토가 동일하게 이루어진다. 다시 말해 로컬 코드라 해서 무조건(built-in) 신뢰하는 개념에서 벗어나 모든 코드에 대해 동일한 보안 통제가 이루어진다. 물론 로컬 코드에 보다 높은 신뢰를 부여하도록 보안 정책을 수립하므로써 보다 많은

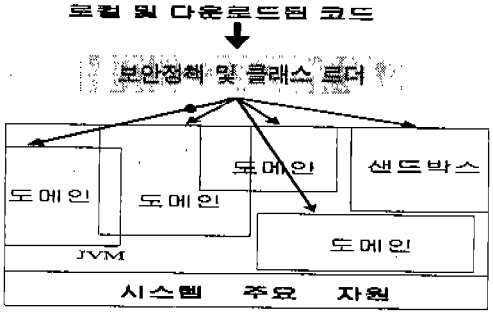

그림 4 JDK 1.2의 fine-grained 접근 통제 모델

일을 로컬 코드에 부여할 수 있다.

JDK 1.2 베타 버전에서의 보안 구조에 대해 정확히 이해하고자 한다면 어떠한 권한이 어떠한 방법으로 부여되는지, 실제로 부여된 권한의 접근 통제가 어떻게 이루어지는지에 대해 이해하여야 한다.

3.3.1 보안 정책

보안 정책은 조직의 시스템 관리자에 의해 정의되는 것으로 시스템 보안 측면에서 어떠한 규칙을 적용하여 시스템의 자원을 관리하며 통제할 것인가를 명시한다. 시스템(JDK를 포함하는 웹 브라우저와 같은 응용 프로그램) 보안 정책은 JDK가 시동될 때 java.security.Policy 클래스의 오브젝트로 객체화 된다. 시스템 보안 정책은 아스키 정책 설정 파일(configuration file)에 기록되며, 이에 대한 접근은 프로그래밍 API를 통해 이루어진다. 각 엔트리는 코드 소스에 대한 정보와 접근 권한(permission)의 쌍으로 구성된다. 다음은 JDK 1.2 베타 버전의 보안 정책의 한 예이다.

```
grant CodeBase "http : //www.kisa.or.kr/bin",
        SignedBy '*' {
        Permission java.io.FilePermission
            "read,write", '/tmp/*";
        Permission java.net.SocketPermission
            "connect", '*.sun.com";
};
grant CodeBase "/Program Files/bin",
        SignedBy "self" {
        permission java.io.FilePermission
            "read,write,delete", "/temp/-";
};
```

첫번째 블록은 http://www.kisa.or.kr/bin/에서 다운로드되는 클래스는 누구에 의해 작성되었는지와 상관없이 /tmp 디렉토리 및 그 하위 디렉토리에 쓰고 읽을 수 있는 권한과 sun.com 도메인에 접속할 수 있는 권한을 정의한다. 두번째 블록은 /Program Files/bin에 저장되어 있는 self에 의해 서명된 클래스는 /temp 디렉토리에 위치한 파일을 읽고, 쓰며 삭제할 수 있는 권한을 정의한다.

3.3.2 접근 권한 및 도메인

앞에서 설명된 보안 정책을 예로 들어 어떠한 방법으로 도메인이 만들어지는지 살펴보자. 클래스가 다운로드 되면 클래스의 URL 또는 로컬 시스템에 저장되어 있던 패스와 서명키를 사용하여 어떠한 보안 정책이 적용되는가를 결정한 후 도메인이 생성된다. 예를 들어 /http://www.kisa.or.kr/bin/index.class가 다운로드 되면 상기 예로 든 보안 정책의 첫번째 블록에 해당하는 /tmp/*에 읽고 쓸 수 있는 권한의 도메인이 생성되어 객체에게 할당된다. 그 후 첫번째 블록에 해당하는 다른 클래스가 다운로드 되면 앞에서 생성된 도메인이 할당된다. 만약 다운로드 받은 코드의 서명키나 패스가 다르지만 /tmp/*에 읽고 쓸 수 있는 권한이 부여될 경우가 발생할 수 있다. 이러한 경우에는 앞에서 생성된 도메인이 아닌 다른 도메인이 생성되어 다운로드된 코드에 할당된다. 또한 하나의 클래스가 여러 개의 보안 정책을 충족시킨다면 각 보안 정책에서 부여하는 권한 모두를 가진 도메인이 생성된다. 예를 들어, A에 의해 서명된 코드 a가 m의 권한을 부여받고 B에 의해 서명된 코드 b가 n의 권한을 부여받는다면 A와 B가 같이 서명한 코드 c는 m과 n 권한의 도메인이 생성되어 코드 c에게 할당된다.

3.3.3 접근 통제

JDK 1.2에서는 implied 메쏘드를 사용한 접근 통제 기법을 사용한다. 사용자는 필요로 하는 접근 통제에 관한 모든 내용을 세세히 명시할 필요가 없다. 다만 명시한 접근 권한이 필요로 하는 권한을 포함하고 있으면 된다. 예를

들어 프로그램이 /tmp/*에 대한 read, write 권한을 가지고 있다면 /tmp/index.txt의 read, write 권한을 가진다. 다음은 접근통제 권한을 예로 보도록 한다. 여기서 p1.implies (p2)는 true이지만 p1.implies(p3)와 p1.implies(p4)는 false이다.

```
FilePermission p1=new FilePermission
                    ("/tmp/*", "read");
FilePermission p2=new FilePermission
                    ("/tmp/x.txt", "read");
FilePermission p3=new FilePermission
                    ("/usr/bin", "read");
SocketPermission p4=new SocketPermission
                    ("*", "accept");
```

쓰레드가 항상 주어진 권한만을 사용하여 실행되란 법은 없으며 상황에 따라 부여되지 않은 권한의 작업을 실행해야 하는 경우가 발생할 수 있다. 애플릿이 시스템의 자원을 사용하여야 하는 예외 상황이 발생할 수 있으며 이를 위하여 beginPrivileged와 endPrivileged라는 메쏘드를 제공한다. 이 메쏘드들은 그 쓰레드에게 잠시나마 필요한 권한을 부여하였다가 다시 회수해 가는 수단을 제공한다.

또한 쓰레드 하나가 여러 개의 도메인을 사용하는 경우가 있다. 예를 들어 쓰레드가 FileInputStream을 사용하고자 할 경우 시스템 도메인을 사용하여야 한다. 이러한 경우에 쓰레드는 시스템 도메인이 가지고 있는 모든 권한을 가져서는 안된다. 이렇게 여러 개의 도메인이 사용될 경우, JDK 1.2 이전 버전에서는 단순히 콜을 하는 도메인에 대한 정보만 있으면 되었으나 JDK 1.2에서는 실제로 콜이 발생한 쓰레드의 권한에 대한 정보를 가지고 있어야 한다. 이는 모든 콜에 대한 접근통제 권한을 기록하고 있어 첫번째 콜을 한 쓰레드의 권한보다 더 많은 권한을 가지지 못하도록 하고 있다(least privilege principle). 따라서, 자바 가상 기계는 항상 쓰레드의 권한을 추적해야 하는데 두 가지 방법을 사용할 수 있다. 첫째는 쓰레드가 도메인을 들어가거나 떠날 때마다 부여된 권한들을 새로이 갱신하는 방법(eager evalution)이 있으며, 나머지 하나는 자바에서 채택한 방법으로 쓰레드의 권한을 추적해 달라는 요청이 있을 때 권한들을 갱신하는 방법(lazy evalution)이 있다. 후자의 경우 권한을 추적하는 데에는 약간의 시간이 더 소요되나, 좀더 안전한 환경을 제공하고 시스템에 저장되어 있는 정보를 안전하게 보호하기 위해서 이다.

4. 결 론

자바가 제공하고 있는 보안 기능은 주로 JDK에 구현된 소프트웨어 방식이다. 초기 버전에서 고안되었던 보안 구조는 해커들과 실험실의 연구자들에게 좋은 공격 대상으로 여겨졌다. 자바와 관련하여 두가지 CERT 권고문 CA-96.05, CA-96.07이 발표되었으며[10, 11], 프린스톤 대학의 연구자들을 중심[14, 15]으로 많은 보안 취약성들이 발견되었다. 여기에는 로컬 시스템의 파일 시스템을 삭제하는 등의 치명적인 피해로부터 시스템 자원을 소모시켜 시스템 자원을 다운시키거나 성인용 이미지를 화면에 디스플레이 하는 등의 피해까지 다양하다. 이러한 취약성의 노출은 설계상 또는 구현상의 오류로 자바 개발자와 웹 브라우저 제작자들의 협력으로 곧 바로 버전업 되어 왔다. 한편 인터넷 전자 상거래와 같은 응용을 자바로 구현하기 위하여 공개키 생성과 전자 서명, 인증서와 키의 관리 등을 지원하는 암호 API가 새로운 버전에 자연스럽게 추가되었다. 실제 자바 응용이 인터넷을 기반으로 이루어지고, 보안 취약성 문제가 자바에서 중요하게 다루어 짐에 따라 자바 개발 진영에서는 JDK 버전 1.0과 1.1에서 보여준 기능에서 한 차원 높은 보안 기능을 1.2에 설계하였다. 새로운 암호 알고리즘이 발표되면 곧 바로 수 많은 암호 분석자들에 의해 깨어지듯이(이는 보안 분야의 속성인 듯 하다) JDK 1.2 베타 버전과 정식 버전(아직 발표되지 않았다)도 좋은 공격 대상이 될 것이며, 머지 않아 보안 취약성 발견에 대한 기사를 보게될 것이다. 기본적으로 소프트웨어에서 복잡도가 증가하면 설계상 구현상의 버그 발생이 그만큼 증가할 것이다. 물론 모든 버그가 보안 취약성으로 연결되지는 않겠지만 JDK 1.2 베타 버전에서 보안 취약성을 얼마나

잡아 낼지 모르겠지만 가능성이 높으므로 앞으로 어떠한 보안 문제가 발생하는지 예의 주시하여야 할 것이다. 미래에는 멀티 유저 환경에 걸맞게 여러 사용자에게 각각 다른 권한을 할당하여 프로그램을 실행할 수 있도록 변화하여야 할 것이다. 이는 JDK 1.2 버전의 도메인 개념을 더욱 더 확장한 것으로서 호스트의 사용자 인증을 통하여 사용자에 따라 각기 다른 보안 정책을 구현할 수 있도록 개발되어야 한다. 또한 자바 가상 기계 위에 탑재되는 자바 제품 JavaEnterprise, JavaCommerce, JavaServlet, JavaManagement 등이 현재 개발되고 있거나 일부 초기 버전이 개발되었다. 현재 안전성이 입증되지 않은 JDK 상에 구현된 이들 자바 제품도 그 안전성을 입증할 수 없다. 또한 이들은 JDK에서 제공하는 보안 기능 외에도 자체의 응용 요구조건에 따른 추가적인 보안 기능을 필요로 한다.

자바에서 객체가 생성될 때 그 상위 객체의 특성을 상속받는다. 이러한 상속 개념을 활용하고 보다 안전한 응용을 구현하기 위하여 루트 객체 자체에 보안 특성을 부여한다면 생성되는 그 하위 객체가 상위 객체의 보안 특성까지 상속받게 될 것이다. 이렇게 각각의 객체에 보안 기능을 추가할 경우 현재까지 존재하던 많은 보안상의 어려움들을 해결할 수 있을 것이며, 객체에 어떤 보안 특성을 어떻게 구현할 것인가에 대한 연구 결과가 기대된다.

참고문헌

[1] Balfanz and Edward W. Felten, *A Java Filter*, TR567-97, Princeton University, 1997.

[2] Li Gong, "New Security Architectural Directions for Java", Proceedings of IEEE COMPCON, pp. 97~102, 1997.

[3] Li Gong, Marianne Mueller, Hemma Prafullchandra, and Roland Schemers, "Going Beyond the Sandbox : An Overview of the New Security Architecture in the Java™ Development Kit 1.2", Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1997.

[4] Li Gong and Roland Schemers, "Implementing Protection Domains in the Java™ Development Kit 1.2, Proceedings of the Interent Society Symp. on Network and Distributed System Security, 1998.

[5] Gary McGraw and Edward Felten, *Java Security : Hostile Applets, Holes, and Antidotes*, John Wiley & Sons, 1997.

[6] 이강수, "Java 환경에서의 보안 위협과 메커니즘", 정보과학회지 제15권 제7호, pp. 48~56, 1997.

[7] Java Security API Overview, http://java.sun.com/products/jdk/...ecurity/JavaSecurityOverview.html, Febrary 1997.

[8] Dan S. Wallach, Dirk Balfanz, Drew Dean & Edward W. Felten, 'Extensible Security Architecture for Java", Proceding of 16th Symposium on Operating Systems Principles, Oct. 1997.

[9] JDK 1.2 New Feature Summary, http://java.sun.com/products/jdk/1.2/docs/relnotes/features.html.

[10] CA-96-05 Java Implementations Can Allow Connections to an Arbitrary Host, CERT Coordination Center, http : //www.cert.org.

[11] CA-96-07 Weaknesses in Java Bytecode Verifier, CERT Coordination Center, http : //www.cert.org.

[12] Gary Cornell and Cay S. Horstmann, Core Java, SunSoft Press, A prentice Hall Title, 1996.

[13] 자바 홈페이지, http://java.sun.com/.

[14] 프린스톤 대학 Secure Internet Programming Team, http://www.cs.princeton.edu/sip/.

[15] 워싱톤 대학 Kimera Group, http://www.kimera.cs.washington.edu/.

이 완 석

1990 Virginia Polytechnic Institute and State University(Va. Tech) 전산학과(학사)
1994～1996 현대정보기술
1996～현재 한국정보보호센터 주임연구원
관심분야 : 컴퓨터 보안, 분산 처리, 정보전

김 홍 근

1985 서울대학교 컴퓨터공학과(공학사)
1987 서울대학교 대학원 컴퓨터공학과(공학석사)
1987 서울대학교 대학원 컴퓨터공학과(공학박사)
1994～1996 한국전산원 전산망 보안실 선임연구원
1996～현재 한국정보보호센터 책임연구원
관심분야 : 컴퓨터 보안, 컴퓨터 알고리즘, 병렬 및 분산 처리

● '98 통신정보합동학술대회(JCCI '98) ●

- 일 자 : 1998년 4월 22일(수)～24일(금)
- 장 소 : 전주 리베라호텔
- 주 최 : 정보통신연구회
- 내 용 : 논문발표 등

● 제25회 임시총회 및 춘계학술발표회 ●

- 일 자 : 1998년 4월 24일(금)～25일(토)
- 장 소 : 충남대학교
- 주 최 : 한국정보과학회
- 내 용 : 초청강연, 튜토리얼, 논문발표 등

● '98 데이타베이스 춘계튜토리얼 ●

- 일 자 : 1998년 5월 21일(목)～22일(금)
- 주 제 : 데이타마이닝
- 주 최 : 데이타베이스연구회
- 문 의 처 : 박종수 교수(02-920-7179)