通信情報保護學會誌
第6卷 第3號, 1996. 9

Java 환경의 보안문제에 관한 고찰

이 강 수*, 최 성 자*

요 약

미국 썬.마이크로시스템즈사의 인터넷 통합 솔루션인 자바(Java)환경은 새로운 아이디어는 아니지만, 95년도의 인터넷 부움을 타고 마케팅에 성공하고 있으며, 프로그래밍 언어, 컴파일러, 브라우저, 인터프리터, 개발도구, OS, DB, 전용칩 및 이를 모두 실장한 자바 터미널까지 제공하고 있는 통합 솔루션이다. 자바환경은 인터넷을 하부구조로하여 구성되므로 보안문제의 해결없이는 그 활용성이 의문시된다. 자바환경의 보안문제는 자바의 개발자들이 초기부터 염두해두고 해결해 오고 있다. 본 연구에서는 자바환경에서 발생가능한 보안위협, 필요한 보안서비스 및 자바환경에서 제시한 보안 메커니즘들을 조사 및 평가한다.

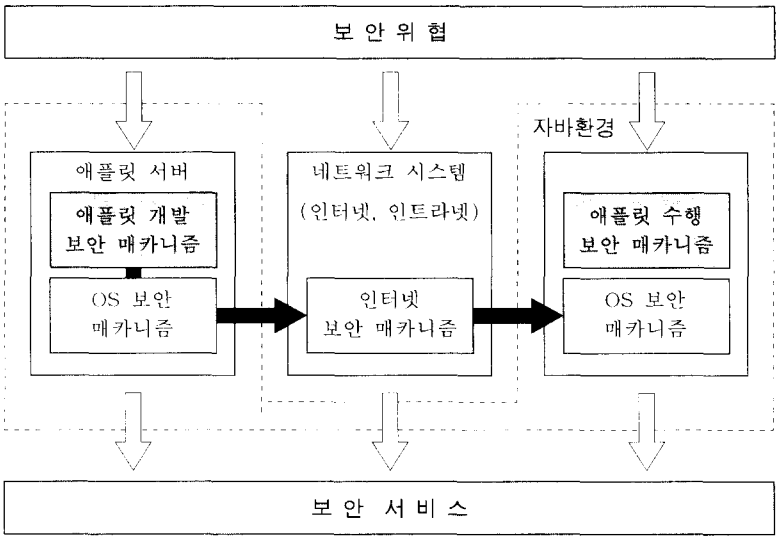
1. 서 론

컴퓨터 네트워크를 하부구조로하는 정보통신 시스템에서 보안문제(security)는 대수학의 공리들과 같으며 정보화 사회의 성패를 좌우한다. 따라서, 정보통신 시스템들에서는 보안문제를 가장 중요하게 고려하고 있지만, 아직도 완전히 해결되고 있지 못하다. 사실, 인간사회에서 범죄가 없어지는 날 보안문제는 완전히 해결될 것이다. 자바환경(Java environment)도 썬.마이크로시스템즈사에서 제시한 "정보통신 네트워크(특히, 인터넷) 환경에서 프로그램의 개발, 통신 및 수행을 지원하는 통합 솔루션"일 뿐이므로[1], 그 또한 보안문제를 피할 수는 없다. 다행히, 자바환경의 개발자들은 개발 초기부터 보안문제를 인식하여 보안문제를 다소나마 해결할 수 있는 메커니즘들을 개발하여 자바환경내에 포함시켰으며 계속해서 보안 메커니즘을 확장 개발하고 있다[1-11].

본 글에서는 자바환경의 일반적인 사항[1]을 생략하며, 자바환경에서 발생할 수 있는 "보안위협"과 필요한 "보안 서비스" 및 그 서비스를 제공하기 위한 "보안 메커니즘"들을 고찰한다. 그림 1에서 자바환경은 네트워크의 응용 시스템에 해당하므로, 네트워크 시스템 부분을 자바환경에 포함시키지는 않았으며, 본 글에서는 편의상 네트워크 시스템을 "인터넷"으로 칭할 것이다.

본 글의 2장에서는 자바환경의 개요를 설명하며, 3장부터 5장까지는 각각 자바환경에 가해질 수 있는 보안위협(또는, 보안공격), 요구되는 보안 서비스 및 자바환경에서의 보안문제 해결 메커니즘을 설명하고 6장에서 평가와 결론을 맺는다.

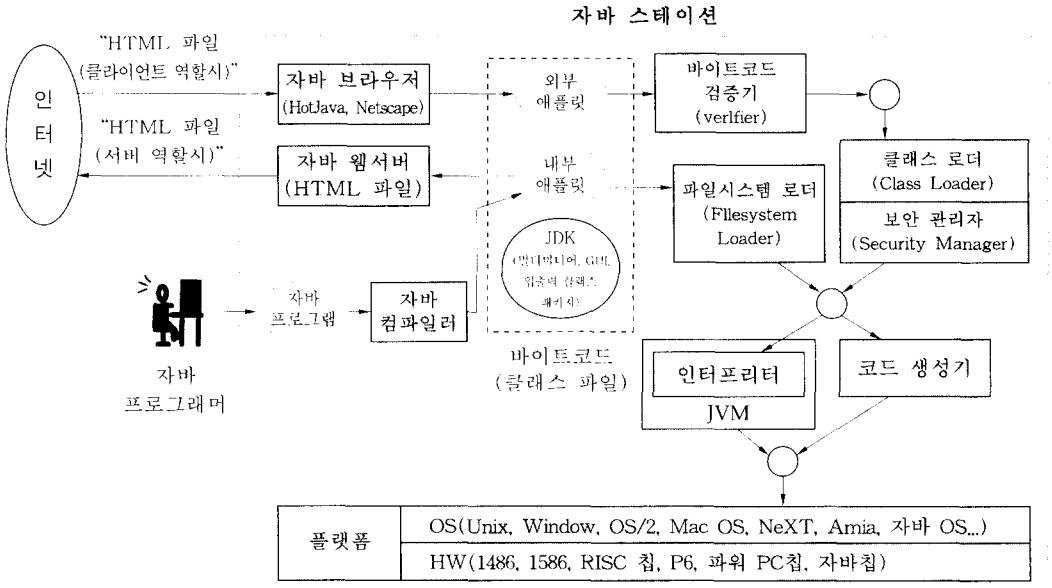
* 한남대학교 전자계산공학과



(그림 1) 자바환경의 보안 위협, 메커니즘 및 서비스 구조

2. 자바환경 개요

자바환경은 그림 2와 같은 요소로 구성되며 현재 계획되어 있거나 개발중인 것들도 있다. 자바환경은 자바 칩부터 GUI개발도구 및 웹 브라우저에 이르기까지 통합적인 솔루션을 제공해주는 개발 및 운영 환경이다[1-10].



(그림 2) 자바 환경의 구조

2.1 자바 스테이션

자바 스테이션(또는, 자바 터미널)은 크게 자바 프로그래밍 요소, 개발도구 및 개발환경 요소, 수행환경 요소로 구성된다.

2.1.1 프로그래밍 요소

(1) 자바 언어[9,10] : 새로운 병행(concurrent) 및 객체지향 프로그래밍 언어로서, C, C++, Objective-C, Modula, Mesa, Modula-3 및 LISP 언어들의 특성들을 이용하여 정의하였다. Netscape 2.0 이상의 버전에서는 웹 구축시 HTML내에서 태그할 수 있는 자바언어를 "자바스크립트"라 부른다.

(2) 에플릿(applet) : 바이트코드(byte-code)로 구성된 단위 프로그램 컴포넌트(또는 모듈, 객체)이다. 자바 프로그램을 작성하여 이를 컴파일하므로서 얻거나, 인터넷상의 다른 자바 웹 싸이트로부터 HotJava나 Netscape2.0과 같은 브라우져를 통해 다운로드 받을 수 있다. GUI, 멀티미디어, 입출력 등 자주 쓰이는 에플릿들은 에플릿 패키지 내에서 '상속' 받을 수 있다.

2.1.2 개발도구 및 개발환경 요소

(1) Java Development Kit(JDK) : 자바 응용 프로그램 인터페이스(API)라고도 하며 RAD(rapid application development) 개념을 통해 자바환경의 API 및 GUI를 쉽게 개발할 수 있도록 제공된 에플릿 패키지이며 클래스 계층을 이루고 있다. 자바 프로그램 내에서 각 에플릿들을 선언 및 호출하면 수행시간에 클래스 로더에 의해 해당 에플릿이 삽입(즉, 링크)된다. 이 개념을 '동적 바인딩'이라 하며 '개방형 서브루틴' 또는 '매크로' 개념과 유사하다. JDK에는 자바 언어 컴파일 지원(java.lan.*), 에플릿 생성지원(java.applet.*), GUI지원(java.awt.*), 유틸리티(java.util.*), 멀티미디어(java.awt.image.*), 입출력(java.io.*), 통신(java.net.*) 등 멀티미디어 정보통신 및 GUI개발에 필요한 대부분의 에플릿들이 있고 계속 그 수가 증가되고 있다.

(2) Object Java 및 OLE Java : 자바환경은 기존의 IBM이나 마이크로소프트 환경과 인터페이스 하기 위한 것으로 현재 개발중에 있다. Object Java는 에플릿이 무리없이 IBM의 분산 OpenDoc 요소와 통합되어 조작할 수 있는 기술이며, OLE Java는 마이크로소프트의 OLE를 지원하며 차기 버전의 윈도우즈-95에 추가될 것이다.

(3) Java Workshop[16] : 96년 5월 24일 발표되었으며, 자바를 이용한 웹페이지의 제작 및 응용 시스템의 개발을(분석, 설계, 구현, 시험, 디버그) 지원하는 도구이다. 웹 중심의 시각적 개발도구이며, 자바언어 자체로 개발되었고 32비트 자바 개발환경을 제공한다.

2.1.3 수행 환경 요소

(1) 컴파일러와 인터프리터 : 자바 언어를 어셈블리언어 형태의 중간 코드인 바이트코드로 번역하며(컴파일 기능) 바이트코드를 실행한다(인터프리터 기능). 이 개념은 파스칼의 P-code개념으로부터 유래되었고 인터넷 상에서 다양한 컴퓨터 플랫폼간의 호환성 문제를 해결할 수 있다. 즉, 각 컴퓨터는 자바 언어 또는 C 언어로 프로그램된 자바 인터프리터를 가져다 자체의 자

바 컴파일러 또는 C 컴파일러로 컴파일하여 수행가능 코드를 설치하기만 하면 이기종간의 호환성 문제(즉, 아키텍쳐 중립성)가 다소 해결된다.

(2) 가상 기계(Java Virtual Machine, JVM)[3] : 자바의 바이트 코드가 수행(인터프리트)되는 가상적인 기계로서 일종의 스택 머신 형태이다. 즉, JVM의 어셈블리 코드가 바이트 코드인 셈이다. 또한, 바이트 코드의 집합은 에플릿이 된다. 그림 2-(b)는 바이트 코드의 예를 보인다.

(3) 파일시스템 로더 : 자바스테이션 내부 또는 인트라넷 내부에 존재하는 파일들을 로딩 및 링크 한다. 이 파일들은 안전하다고 가정하므로, 바이트코드 검증기를 통과할 필요가 없다.

(4) 자바 DB 및 자바 OS : 현재 개발중인 시스템으로서 자바 DB는 자바환경과 기존의 DB간을 연결하므로서, 클라이언트/서버 데이터베이스를 접근하도록 하는 것이며, 인트라넷 시스템 개발시 유용할 것이다. 또한, 자바 OS는 자바 터미널을 위한 운영체제로서 간단한 에플릿 관리, 자바터미널 내의 메모리관리, 멀티미디어 입·출력, 에플릿 다운로딩, 보안문제 처리 등의 기능을 수행할 것으로 판단된다.

(5) 자바 스테이션 : 오라클사에서 제시했던 인터넷 PC 또는 인터넷 컴퓨터(NC)와 유사한 개념이다. 자바 터미널에는 자바 인터프리터과 HotJava 등과 같이 최소한의 자바환경을 유지하고 전용 자바 칩을 이용하므로서, PC보다 싼 500$ 정도의 컴퓨터를 구성한다는 개념이다. 이미 토스터 크기만한 자바스테이션 시제품이 개발되었다.

2.2 인터넷과 보안 시스템

자바환경과 인터넷 환경간의 인터페이스 요소에 해당한다. 인터넷상에 존재하는 다른 자바(에플릿) 웹서버로부터 에플릿이 포함된 웹 페이지(HTML로 작성되고 HTTP프로토콜로 디코드함)를 브라우징하고 에플릿을 분리하며 보안성을 검증한다.

(1) HotJava : Netscape과 유사한 웹 브라우져로서 기존의 HTML문서(문자 및 정지화상)뿐 아니라, 에플릿들도 검색 및 다운로드 받을 수 있다.

(2) 바이트코드 검증기(verifier)[3,4] : HotJava에 의해 인터넷상의 에플릿 서버로부터 다운로드된 에플릿의 보안성을 검증한다.

(3) 클래스로더(ClassLoader) : 외부 에플릿을 내부 바이트코드로 변환하며 바이트코드를 위한 네임스페이스의 구성 및 접근제어를 수행한다.

(4) 보안 관리자(SecurityManager) : 외부 에플릿에 대한 보안통제를 위한 것으로서, 자바스테이션의 보안통제 방법을 구현한 것으로서 이에 필요한 클래스들이 제공되어있다.

3. 자바환경의 보안 위협

기존의 웹 환경에서는 HTML로 작성되고 인코드된 웹 페이지를 Netscape2.0과 같은 웹 브라우져와 인터넷을 이용하여 다운로드하고 Netscape에 내장 HTTP프로토콜을 통해 디코드하므로서 원격정보를 볼 수 있다. 그러나, 웹 환경에서는 정적인 문서나 이미지와 같은 '수동적인'(즉, 비활성적, 정적인) 정보만을

송, 수신할 수 있다. 웹 환경과는 달리 자바환경에서는 기존의 웹 환경을 이용하여 '능동적인' 정보(즉, 수행 가능한 프로그램 코드)도 송·수신할 수 있다. 이 개념을 "수행가능 내용(executable contents)"이라 하며 수행가능 코드(프로그램)를 웹을 통해 다운로드하여 자체 컴퓨터에서 수행한다는 개념이다. 이 개념은 인터넷이나 PC 통신망을 통해 공개 소프트웨어를 다운로드하는 개념과 유사하며, 자바뿐 아니라 Safe-Tcl 프로그램을 e-mail에 포함시킨 "Enabled Mail" 개념 및 General Magic사의 Telecsript 개념에서도 이용되고 있다[6].

수행가능 내용 개념을 사용하는 자바환경에서는 심각한 컴퓨터 보안문제를 안고 있으며 "바이러스 에플릿"의 활동무대가 될 수 있다. 예컨대, 기존의 웹 환경에서는 죽은 사람을 고용하는 것인데 비해, 자바환경에서는 낯선 사람을 자신의 회사나 집에서 고용하여 같이 일하는 것과 같다. 정체 모를 웹 싸이트로부터 다운로드한 에플릿이 다음과 같다고 가정하자.

```
public class Suicide{
    public static void main(){
        delete *.*
}
}
```

이 에플릿을 자신의 컴퓨터에서 수행했다면 어떻게 될 것인가.

3.1 에플릿 개발시

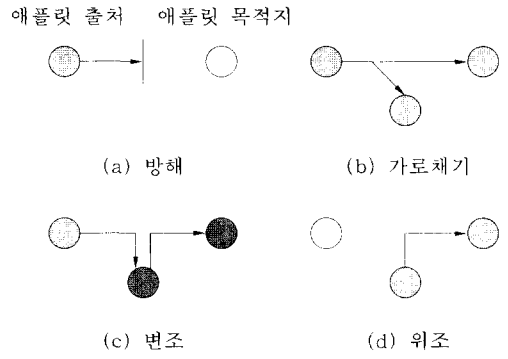
에플릿의 개발시 에플릿 개발자에 의해 발생할 수 있는 보안위협을 의미한다.

- 에플릿에 불법 기능 추가 및 트로이목마 삽입
- 에플릿에 불법 자원접근 기능 삽입 : 시스템의 자원인 파일, 망, 메모리, 출력장치, 입력장치, 프로세스 컨트롤, 쓰레드, 사용자환경, 시스템 콜 등의 불법접근, 변경 및 삭제.
- 에플릿의 수행 방해 기능삽입[5] : 윈도우 대량생성 및 고 우선 순위 프로세스/쓰레드 생성 등
- 불쾌감 유발(annoyance attack)[5] : 불쾌감을 유발하는 화면 및 사운드 생성 기능의 불법 추가

3.2 에플릿 전송시

인터넷에 존재하는 에플릿 서버(예: 에플릿 판매업체)로부터 에플릿을 다운로드하는 동안 발생할 수 있는 보안 위협(또는, 보안 공격)을 의미한다. 에플릿 서버의 입장에서 볼 때, 방화벽(firewall) 외부의 클라이언트로 에플릿을 송신할 경우의 발생할 수 있는 보안 위협을 의미한다. 이러한 위협은 자바환경 뿐아니라 정보통신시의 일반적인 보안 위협[17,18]이기도 하다.

- 방해(interruption) : 에플릿 전송 방해, 전송 에플릿 가로채기
- 가로채기(interception) : 전송 에플릿 도청 및 불법 복사
- 변조(modification) : 전송 에플릿의 변조
- 위조(fabrication) : 에플릿의 위장 송신
- 트래픽 분석 : 에플릿 전송특성 분석
- 에플릿 사용자 신분위장(masquerade) : 에플릿 불법 수신
- 재전송(replay) : 에플릿 전송의 재현, 마피아 공격
- 에플릿 송신 및 수신 부인(repudiation)
- 에플릿 배달 부인



(그림 3) 에플릿 전송시의 보안 위협 유형

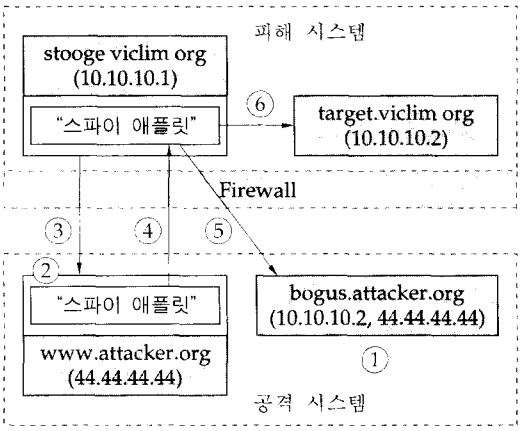
3.3 에플릿 수행시

에플릿 클라이언트내의 사용자들에 의한 에플릿 수행시의 보안 공격을 의미한다.

- 에플릿 내용 폭로(disclosure) : 자바 시스템 사용자, 에플릿 또는 호스트 정보(예를 들어, 파일, 망, 메모리, 입력장치, 사용자환경 및 시스템 콜 정보 등)의 불법 유출
- 에플릿 불법 수정 : 다운로드된 에플릿을 불법 수정하여 사용
- 인터넷 불법 접근 : 수행중 내부자 및 외부자에의한 인터넷 불법 접근
- 에플릿 불법 복제

3.4 자바 보안위협 사례: DNS공격[14]

그림 4는 자바환경과 같은 인터넷을 기반으로 하는 시스템에서 발생할 수 있는 공격을 보인다. 이 공격은 기존의 DNS(Domain Name System)의 문제점(즉, 한 노드에 다수의 IP주소를 부여할 수 있음.)을 이용한 것으로서 피해 시스템내의 방화벽을 피할 수 있다. 즉, "스파이 에플릿"을 피해 시스템에 잠입시켜 스파이 에플릿이 다른 노드를 공격하는 것이다. 이 경우 "stooge노드"는 스파이 에플릿이 내부에서 수행되므로, 자신도 모르게 범죄에 말려들게 되며 법적인 책임을 져야할 경우가 발생한다. DNS공격은 비록 자바환경에서만의 보안문제는 아니지만, 자바환경도 에플릿의 전송을 위해서는 인터넷을 기반으로 하며 DNS서비스를 받으므로, 이런 보안문제는 피할 수 없다.



① 가짜 노드(bogus)와 IP 주소 (10.10.10.2, 44.44.44.44) 생성
② 스파이 애플릿을 웹에 삽입
③ 브라우징
④ 스파이 애플릿 다운로드
⑤ 가짜 노드와 연결
⑥ 실제로는 목표 노드와 연결

(그림 4) 자바환경에서 DNS 공격의 예

4. 자바환경의 보안 서비스

3장에서 제시한 보안 위협들을 제거하기 위해서 자바환경에서는 다음과 같은 보안서비스를 제공해야한다. 자바환경에서의 서비스는 크게 에플릿 개발시, 전송시 및 수행시 제공되어야하며, 특히 에플릿 전송시의 보안서비스는 인터넷 환경과 연계되므로 기존의 인터넷 보안서비스(예: 전자우편 및 메시지 보안서비스 등)에 의해 제공받을 수 있다.

4.1 에플릿 개발시

- 에플릿 내용 무결성(integrity) : 정당한 에플릿 개발
- 에플릿 내용 기밀성(secrecy) : 개발된 자바 프로그램 및 그 에플릿의 기밀성
- 에플릿 개발의 부인봉쇄(non-repudiation) : 에플릿 개발 사실의 부인봉쇄

4.2 에플릿 전송시

전자우편 보안서비스[17,18]인 SECURE/32, MailSafe, PEM(Privacy Enhanced Mail), PGP(Pretty Good Privacy), MIME(Multimedia E-mail) 및 메시지 처리시스템 보안서비스인 MHS(X.400)에 의해 제공받을 수 있다.

- 기밀성(secrecy) : 도청방지, 암호화 기술, 에플릿 흐름 기밀성, 기밀성 수준 유지, 에플릿 사용 통계자료 기밀유지, 에플릿 송·수신자 비밀유지
- 인증 : 에플릿 발신자 및 수신자 인증, 에플릿 자체의 인증, 양방향 인증 필요, 디지털서명 기술
- 무결성 : 에플릿 수정방지, 해쉬 기술, 메시지 순서 무결성
- 부인봉쇄 및 증명 : 에플릿 송신, 수신, 전달 부인봉쇄 및 증명
- 기타 : 접근제어, 보안 감사, 프라이버시, 인터넷 운영내용의 보안, 수신후 파기 기능 등

4.3 에플릿 수행시

- 인증 : 에플릿 수행자(사용자) 인증, 디지털 서명
- 무결성 : 에플릿 무결성 및 에플릿 클래스 구조 무결성
- 기밀성 : 에플릿 내용 기밀성, 수행결과의 기밀성
- 부인봉쇄 : 에플릿 수행의 부인봉쇄

5. 자바환경의 보안 메커니즘

자바환경은 에플릿 서버로부터 인터넷를 통해 수행 가능한 에플릿(바이트코드)을 다운로드하여 자신(클라이언트)의 시스템에서 수행해야한다. 따라서, 에플릿의 클라이언트 입장에서는 안전(secure)한 곳으로부터 생성된 안전한 에플릿이며, 인터넷를 통해 안전하게 수신된 에플릿만을 수행해야한다(즉, "무결성"). 또한, 에플릿 서버의 입장에서는 상품이라 할 수 있는 에플릿을 정당한 클라이언트에 안전하게 송신해야하고(즉, "인증성" 및 "기밀성") 클라이언트의 "부인봉쇄"(즉, 에플릿 수신 사실 부인)가 이루어져야한다.

자바환경 개발자들은 앞에서 제시한 보안문제(위협 및 필요한 서비스)들을 인식하여 이를 해결할 수 있는 다단계 메커니즘들을 구성하였다(그림 4). 이들 메커니즘들은 주로 Discretionary Access Control (DAC) 수준의 접근제어를 위한 것이며, 암호화, 디지털 서명 및 인증 메커니즘은 현재 자바환경에서 직접 지원하지는 않고 기존의 인터넷 보안 메커니즘을 이용해야 한다.

그림 5에서는 에플릿이 수행되기 위해 서버(개발자)와 클라이언트(사용자)간의 대응관계를 보이며 다음과 같은 특성을 갖는다.

- Function to Function : 자바환경의 보안 메커니즘의 최종 목표이며, 자바 프로그램의 수행결과에 대한 보안성(즉, 정당성, 무결성 및 기밀성 등)을 보장해야 한다.
- Source to Source : 자바(또는, 자바 스크립트) 프로그램간의 보안성(즉, 기밀성, 무결성 및 인증성)을 보장해야한다. 클라이언트의 입장에서, 수행할 에플릿이 안전한 컴파일러로부터 생성된 에플

릿인가(즉, 보안규칙을 준수한 컴파일러인가)를 검증하는 것으로서, Verifier와 ClassLoader에 의해 보장된다.

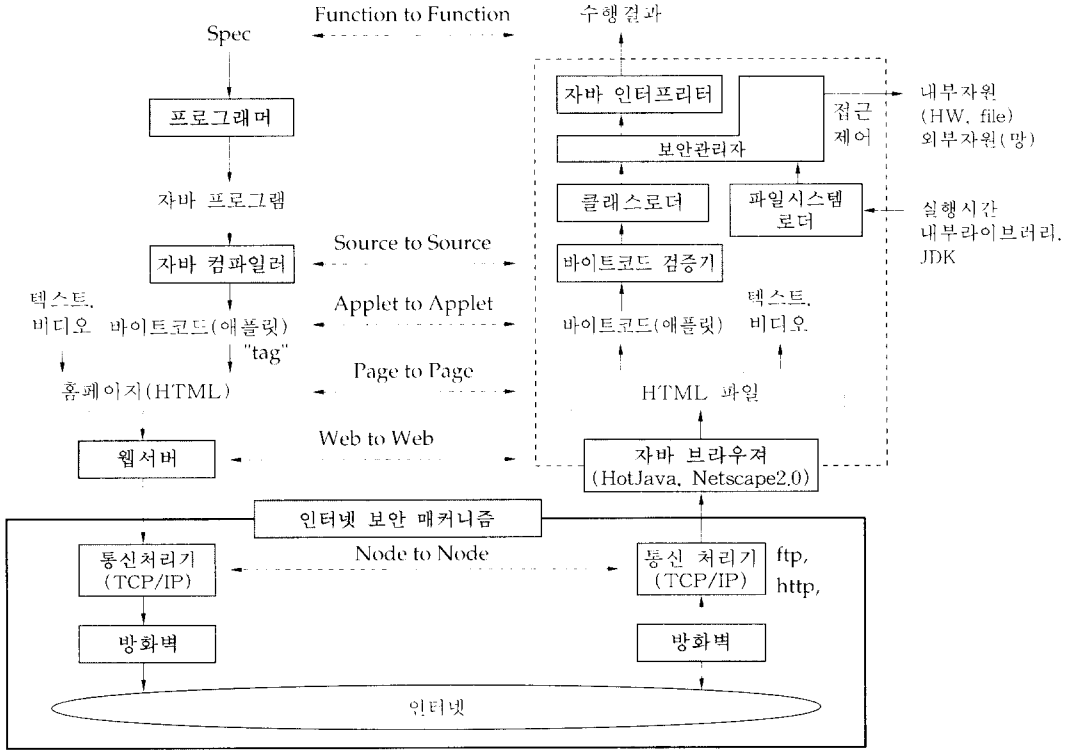
- Applet to Applet : 에플릿(바이트코드) 코드 자체의 보안성(무결성, 인증성, 기밀성)을 보장해야한다.
- Page to Page : 에플릿이 포함된 웹 페이지(예를 들어, HTML파일)간의 보안성(무결성, 인증성, 기밀성)을 보장해야한다.
- Web to Web : 웹서버와 클라이언트간의 인증성을 보장해야한다.
- Node to Node : 웹서버 사이트(노드)와 클라이언트 사이트간의 보안성을 보장해야하며, 기존의 인터넷 보안 메커니즘[17,18]을 이용한다. OSI 7층 구조에서 응용 층을 제외한 모든 층들이 이용된다.

5.1 자바 언어내의 메커니즘

자바언어는 통신망상의 보안을 지원하기 위해 언어자체의 설계시에 보안문제를 고려하여 설계하였다. 자바언어의 설계자들은 보안성 향상을 위해 기존 언어의 기능을 삭제하거나 새로운 개념을 추가하므로서 보안성 위협을 예방 및 방지할 수 있게 하였다[5,9].

5.1.1 기능 삭제

(1) Typedefs, Defines 및 Preprocessor : 언어의 가독성을 높이기 위해 typedefs가 없고, 프로그램의 분석을 시작하기 전에 모든 관련된 typedefs와 관련된 모든 #defines를 읽어야 한다.



(그림 5) 자바환경의 보안 매커니즘

(2) 자동 강제(coercions) : 정확한 손실의 결과인 하나의 데이터요소에서 다른 데이터를 명시적으로 사용하기 위해 Cast를 사용한다.

(3) 포인터 : 사용자가 메모리 영역의 접근을 직접적으로 하는 것을 방지하기 위해 포인터를 배제하였다. 즉 객체 인스턴스 변수를 사용하여 이루어지고 이것은 Private 데이터를 접근하는 트로이 목마를 가진 프로그래머의 내부 보안 위협에 대한 보안 프로텍션이다. 또한, over-run과 under-run의 에러를 예방하고, 부정확한 오프셋 연산을 예방한다. 포인터의 오용 가능성을 제거하므로서 객체에 대한 위장접근 가능성을 차단하였다.

(4) 기타 사항 : 전체 프로그램의 에러를 발생시킬 수 있는 전역변수, Goto문장, 데이터 캡슐화의 결함을 야기하는 Friend Function 등을 배제하고 Package는 단독 컨테이너에 관련된 클래스를 구축하도록 하고 "friendly" 인스턴스 변수와 메소드는 같은 Package 내에서 호출한 모든 것이 활용 가능하도록 한다. 이상으로 기존 언어의 보안상의 문제점을 야기시킬 수 있는 부분을 배제함으로써 자바언어는 설계되었다.

5.1.2 기능 추가

(1) 객체의 접근제어(access control) : public-method는 누구나 호출가능하며, private-method는 객체 메소드에 의해서만 호출가능하고, public 읽기호출시 보안 체크후 private를 읽기호출하므로서, 프로그래머는 접근제어를 올바르게 기술할 수 있고, 자바 언어 자체에 의해 안전성이 보장된 라이브러리를 작성 할 수 있다.

(2) final method/class선언 : 중요한 라이브러리 클래스를 서브클래스하거나 메소드를 오버라이딩하는 것을 방지하며 메소드의 무결성을 보장한다.

(3) 강력한 자료 형 선언 : 컴파일타임 타입과 런타임 타입간의 호환성을 유지하고 cast 연산자를 컴파일시 또는 수행시 체크하므로서, 객체에 대한 위장접근 방법에 의한 우회접근을 방지한다. 또한, 명시적이며 강력한 형을 제시함으로써, 버그를 초기에 발견할 수 있다.

(4) 자동 가비지 컬렉션 : 버그 및 보안 홀을 제거하며, 우회에 의한 불법 형변환(casting) 가능성을 제거한다. 즉, 사용이 끝난 널 포인터를 이용한 불법접근 및 불법코드의 삽입 가능성을 없앴다.

(5) 패키지 : 패키지는 객체 및 메소드 이름의 공간을 캡슐화하며, 내부(로컬)의 에플릿 코드와 다운로드한 에플릿 코드간의 구별을 용이하게 한다. 로컬 에플릿의 이름공간을 검색한 후 다운로드한 에플릿의 이름공간을 검색하므로서, 보안성이 검증되지 않은 에플릿을 참조할 수 없게 한다.

(6) True Arrays and Strings : 자바언어 인터프리터는 배열과 스트링 인덱스를 조사할 수 있음으로 상세한 설계가 이루어 지도록 한다.

5.2 바이트코드 검증기

바이트코드 검증기(Bytecode Verifier)는 생성된 소스코드가 신뢰할 수 있는 컴파일러에서 생성된 소스코드인지 또는 공격의 가능성을 가진 컴파일러에서 생성된 소스코드인지

확인하기위해서 사용되며[3,4], 자바 클라이언트의 수문장의 역할을 한다. 바이트코드 검증기는 첫째, 외부로부터 다운로드한 모든 에플릿의 보안성을 검증하고 보안 위협을 발견한다(어떤 기관의 출입시 신원조회 과정에 비유된다). 이로서 클라이언트내의 자바 인터프리터에게 "안전한" 에플릿(바이트코드)을 제공한다. 둘째, 다운로드한 에플릿이 에플릿서버내에서 안전한 자바 컴파일러로 컴파일하였고 안전하게(즉, 무결성) 전송되었는지 검증한다. 셋째, 단일 레벨(즉, 허가/불허만 통제) 정보흐름 모델(information flow model)[10]을 자바 프로그램의 변수수준에 적용한 것이다.

바이트코드의 검증은 다음과 같은 과정으로 구성된다[11].

- 검증과정 : 바이트 코드의 형식을 검사한다.
- 정리 증명(theorem prover)과정 : 바이트코드의 구조적 조건의 만족여부를 검증하고 포인터의 오용여부와 접근제어의 위배 여부를 검사한다. 또한, 올바른 객체의 접근여부, 매소드의 올바른 호출(예: 파라메타 타입 등) 및 스택 오버플로우의 여부를 검사한다.

5.3 보안관리자

보안관리자(SecurityManager)[5]는 자바환경의 보안 담당자가 시스템의 보안 정책(주로 접근제어 정책)을 고려하여 구축한 보안관리 시스템으로서, 주로 시스템의 "자원 객체 수준"의 접근제어이며 보안 위협의 "예방 및 방지"를 목적으로 한다. 여기서 시스템의 "자원 객체"란, 시스템내의 파일, 인터넷, 메모리, 입출력 장치, 프로세스 콘트롤, 사용자 환경 및 시스템 콜 등을 의미한다. 보안관리자는 단일 레벨 접근제어 모델(즉, 허가/불허만 통제) 중 "모니터 모델"에 해당하며, DAC 수준으로 이루어지며, Access Control List(또는, Table of Triples, Capabilities, Lock/Unlock 등)메커니즘[17]을 이용한다.

자바환경의 보안 담당자는 표 1에서 제시된 16개의 보안관리 퍼블릭 매소드로 구성된 클래스를 이용하여 보안관리자를 구현할 수 있다.

표 1 자바 보안관리자의 퍼블릭 메소드[5]

메 소 드	기 능
getInCheck	보안검사 수행여부 결정
checkCreateCklassKoader	클래스로더의 설치 여부 검사
checkAccess	쓰레드가 다른 쓰레드의 수정 여부 검사
checkExit	Exit명령 수행 여부 검사
checkExec	시스템 컴맨드 수행 여부 검사
checkLink	동적 라이브러리 링크 여부 검사
checkRead	파일의 읽기 여부 검사
checkWrite	파일의 수정 여부 검사
checkConnect	네트워크 접속 여부 검사
checkListen	네트워크 포트의 도청 여부 검사
checkAccept	네트워크 접속 수락 여부 검사
checkProperties	시스템 속성자료의 접근여부 검사
checkTopLevelWindow	윈도우가 특수한 경고를 소유해야만 하는지 여부 검사
checkPackageAccess	패키지 접근 여부 검사
checkPackageDefinition	패키지에 신규 클래스의 추가여부 검사
checkSetFactory	에플릿이 네트워크관련 객체 디렉토리의 세팅 여부 검사

5.4 클래스로더

클래스로더(ClassLoader)는 바이트 코드의 적합성을 체크하고 다운로드된 코드의 "네임스페이스(또는, logical working space)"를 생성하는 역할을 하며, 보안 위협의 "예방 및 방지"와 "변수수준의 접근제어"를 제공한다. 로컬 호스트의 파일에서 클래스를 로드시에는 클래스 로더를 포함하지 않는다[6]. 각 인터넷 소스에 대해 유일한 네임스페이스이어야 하고, 클래스가 다른 클래스 참조시에 내장된 네임스페이스를 먼저 살피고 참조 클래스 네임스페이스를 명시적으로 조사한다[3,4,5].

클래스로더는 외부 애플릿(바이트코드)를 내부 바이트코드로 변환한다. 로드된 코드(애플릿)을 위해 격리된 네임스페이스를 구성하고 네임스페이스는 우연적인 혹은 고의적인 네임 불일치를 예방하기위해 각 자바 바이트코드 소스의 분리를 위해 격리된 네임스페이스로 로드된다. 또한, 협력하는 소프트웨어 모듈사이의 fault isolation의 기능과 신뢰할 수 없는 객체의 캡슐화를 제공하며 logical memory protection(IBM에서의 base register, protection bit등 이용)과 유사하다.

5.6 인터넷상의 메커니즘

자바환경은 EDI시스템처럼 인터넷 시스템의 응용수준(application layer)에 위치하므로, 애플릿의 실제 전송시의 보안문제(즉, 애플릿의 기밀성, 인증성, 무결성)는 기존의 인터넷 보안 메커니즘 및 방화벽 시스템들[17]을 활용한다. 따라서, 자바환경에서는 별도의 인터넷상의 보안 메커니즘은 없지만, 기존의 인터넷 보안 메커니즘들을 그대로 활용할 수 있다.

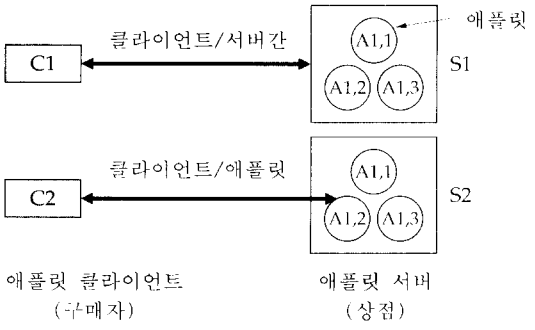
(1) 접근제어 메커니즘 : 인터넷에서의 접근제어 시스템이라 할수있는 "방화벽(firewall) 시스템"이나 "프록시 서버(Proxy server)"를 이용하므로서, 에플릿이 방화벽 내부(즉, 인트라넷)에서 다운로드한 것인지 또는 외부(즉, 인터넷)에서 다운로드한 것인지 조사할 수 있다. 여기서, 프록시 서버란 방화벽상에서 실행되는 HTTP서버로서 인터넷(즉, 방화벽외부)과 인트라넷(즉, 방화벽 내부)간의 인터페이스용 서버이다.[6]. 일반적으로, 방화벽 시스템에서는 인터넷상의 주체(즉, 외부 에플릿)와 객체(즉, 외부 에플릿이 접근할 수 있는 파일 등)간의 접근제어 메커니즘으로서[3,4] ACL(Access control list)를 이용한다.

(2) 무결성 메커니즘 : 무결성 메커니즘은 에플릿 서버로부터 애플릿을 다운로드 할 경우 애플릿의 내용과 순서가 변경되지 않았음을 보장해준다. 이때에 적용되는 기법으로는 해쉉기법과 RSA(Rivest-Shamir-Adelman) 기법이 주로 이용된다. 즉, 에플릿 서버가 에플릿 바이트코드의 해쉉값을 구하여 바이트코드와 함께 RSA로 암호화(또는 디지털 서명)하여 에플릿 클라이언트에게 보내면, 클라이언트는 복호화하여 에플릿과 해쉉값을 분리하고 에플릿을 해쉉하여 수신된 해쉉값과 비교한다.

(3) 기밀성 메커니즘 : 에플릿 전송시의 에플릿이 인터넷상에서 노출되므로 에플릿의 변조와 위조를 방지하기 위해 에플릿을 암호화 해야하며 RSA기법을 이용한다[3].

(4) 인증 메커니즘 : 자바환경을 이용하여 에플릿 통신판매 서비스를 구축한다면, 대금청구, 사용권부여 및 부인봉쇄 등을 위해 에플릿 상점(즉, 에플릿 서버와 서버내의 에플릿 상품)과 구매자(즉, 클라이언트)간의 인증이 필요하다. 즉, 그림 6처럼 "양방향" 인증이 필요하며(즉, 정당한 클라이언트인가, 정당한 서버이고 에플릿인가), 에플릿의

인증(즉, 올바른 에플릿인가)과 클라이언트의 인증(즉, 올바른 사용자인가)이 각각 필요하다. 인증은 기존의 "디지털 서명" 메커니즘을 이용할 수 있으며 특히, 에플릿 서버가 클라이언트(즉, 사용자)를 인증할 때 Kerberos 인증서버를 이용할 수 있다[6].



(그림 6) 자바환경에서 인증의 종류

6. 평가 및 결론

본 글에서는 표 3에서 요약 한바와 같이 자바환경에서의 가능한 보안 위협, 필요한 보안서비스 및 자바에서의 보안 메커니즘들을 다루었다.

지금까지 살펴본 자바 언어, 바이트코드 검증기, 클래스로더 및 보안관리자와 같은 자바환경의 보안 메커니즘들은 인터넷이 안전(secure)하다는 전제하에 "Discretionary Access Control (DAC)" 수준의 접근제어를 위한 것이며, 보안 위협의 "예방 및 방지"를 목표로 하고 있다. 특히, 바이트코드 검증기는 프로그램 변수수준의 정보흐름 모델[19]을 이용하여 개발된 것으로 판단되며 그 아이디어를 활용할 가치가 크다.

표 2 자바환경의 보안 위협, 서비스 및 메커니즘 요약

	가능한 위협	필요한 보안서비스	가능한 위협
애플릿 개발 (컴파일 타임)	- 불법 기능 추가, 트로이 목마 삽입 - 애플릿에 불법 자원접근 기능 삽입 - 애플릿의 수행 방해 기능 삽입 - 불쾌감 유발 생성기능 불법 추가	- 애플릿 개발사실의 부인봉쇄 - 내용 무결성 - 내용 기밀성	- 자바언어 (예방 및 방지)
애플릿 전송 (네트워크 보안)	- 방해, 가로채기, 변조, 위조 - 트래픽 분석 - 애플릿 사용자 신분 위장 - 재전송 - 애플릿 송신, 수신 및 배달 부인	- 애플릿 기밀성, 무결성 - 통신 애플릿 및 통신자 인증 - 애플릿 송·수신 부인봉쇄 및 증명 - 통신망 접근제어, 가용성, 보안감사 - 통신자 프라이버시, 네트워크 운영내용의 보안, 애플릿 수신후 파기 기능	- 바이트코드 검증기(발견) (무결성, 정보흐름 해결) - 네트워크 보안 메커니즘 (암호화, 디지털 서명, 접근제어, 데이터 무결성, 인증 교환, 통신량 삽입, 경로제어 선택, 공증)
애플릿 수행 (런타임)	- 애플릿 내용폭로 - 애플릿 불법복제 및 수행 - 애플릿 불법수정 - 수행 애플릿이 네트워크 불법접근	- 수행자 인증 - 애플릿 무결성 - 수행 기밀성 - 수행 부인봉쇄	- 바이트코드 검증기(발견) - 클래스로더(예방 및 방지) - 보안관리자(예방 및 방지)

자바환경에서는 필수적인 보안 매커니즘인 암호화, 디지털 서명 및 인증 메커니즘을 직접 지원하지는 않고 기존의 인터넷 보안 메커니즘을 이용하고 있다. 따라서, 자바환경의 보안성은 자바환경의 하위 층이라 할 수 있는 인터넷 보안 서비스 및 메커니즘에 의해 좌우된다. 그러나, 인터넷 보안 서비스는 아직 완벽하지 못하므로, 자바환경과 같이 인터넷의 보안 메커니즘에 의존해야하는 정보통신 응용시스템의 실용화를 어렵게 하고 있다. 이런 문제는 인터넷의 응용시스템이라할 수 있는 전자자료 교환(EDI), 그룹웨어 및 전자화폐(EC) 시스템에서도 마찬가지이다.

다운로드한 에플릿의 기능은 인터넷의 보안성 수준에 비례하므로, 인터넷의 보안성 수준이 높아질때까지는 외부(즉, 인터넷)로부터 다운로드한 에플릿은 현재 자바환경처럼 극히 제한된 기능만을 수행(예를 들어, 내부 자원의 접근 금지 등)할 수 있도록 해야하며, 인트라넷내(즉, 방화벽 내부)의 에플릿만을 자유롭게 수행할 수 있다. 따라서, 널리 홍보된 자바환경의 장점은 적어진다. 그러나, 자바환경은 완성된 것이 아니라 계속 진화되고 있고 인터넷의 보안성 수준도 높아지고 있다.

결론적으로, 인터넷의 보안문제가 해결되는 날, 자바환경은 EDI나 전자화폐 시스템 처럼 안전하게 사용될수 있을 것이며, 이 날을 앞당기기위해 특히, 인터넷의 보안성을 높이기 위한 연구가 계속되어야 한다.

참 고 문 헌

[1] 이강수, 조윤희, "자바환경에 대한 고찰", 한국정보과학회지, 1996년 5월, pp.75~84.

[2] 임영주, "자바 시큐리티", 월간 프로그램세계, 1996년 4월, pp.163~177.

[3] "HotJava(tm): The Security Story", http://java.sun.com/1.0alpha3/doc/security/security.html, 1996.

[4] F. Yellin, "Low Level Security in Java", http://www.w3.org/Conferences/WWW4/Papers/197/40.html, 1996.

[5] J. A.Bank, "Java Security", http://www-swiss.ai.mit.edu/~jbank/javapaper/javapaper.html, Dec. 1995.

[6] W. Wang, Yi Yan and L. Zang, "Security: How is it implemented in the Java language?", http://www.cs.colorado.edu/~yiyan/javaproj.html, 1996.

[7] "Frequently Asked Question-Applect Security", JAVASoft, http://java.sun.com, 1996.

[8] "CIAC NOTES", http://aurora.phys.utk.edu~kfg/javasecurity/caic96, March 1996.

[9] "Java White Paper", http://java.sun.com/whitePaper/java~whitepaper-8.html, 1995.

[10] E. Anuff, *Java Source Book*, 1996, p.56

[11] A. V. Hoff, et al., *Hooked on Java*, 1996, pp.17~18, pp. 33~34.

[12] B. Leach, "The Java Security Mechanism Include", http://compsoc.lat.oz.au/~leachbj/java/security.html, 1995.

[13] Tova Fliegel, "Bug in Java Security Enabled Malicious Applet to Attack", http://www.macworld.com/@440808rhxbzn/exclusive/javasecurity.html, Feb. 1996.

[14] D. Dean et al., "Java Security : DNS Attack Scenario", http://www.cs.princeton.edu/~ddean/java/DNS~ scenario. html, Feb. 1996.

[15] "Java Security: Our Story vs. Sun's Story", http://www.cs.princeton.edu/~ddean/java/our~reply. Feb. 1996.

[16] "Java WorkShop 1.0 Early Access Try and Buy-Download Information", http://www.sun.com/sunsoft/ Developer-products/java/tnb/index.html, 1996.5.24.

[17] W. Stalling, *Network and Internetwork Security*, Prentice-Hall, 1995.

[18] C. Kaufman, R. Perlman, M. Speciner, *Network Security: Private Communication in a Public World*, Prentice-Hall, 1995.

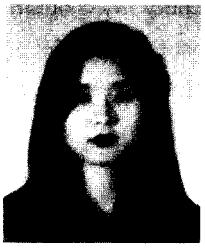
[19] D. Denning, *Cryptography and Data Security*, Addison-Weslay, 1983.

□ 著者紹介

이 강 수

1981년 홍익대학교 전자계산학과 졸업
1983년 서울대학교 계산통계학과 이학석사
1989년 서울대학교 계산통계학과 이학박사
1985년 ~ 1987년 대전산업대학교 전자계산학과 전임강사
1987년 ~ 현재 한남대학교 전자계산공학과 부교수
1992년 ~ 1993년 미국 일리노이대학교 객원교수
1995년 한국전자통신연구소 초빙 연구원

※ 관심 분야 : 소프트웨어공학(신뢰도, 프로젝트 관리, 분산 시스템 모델링), 패트리넷(응용), 멀티미디어(동기 모형), 보안 시스템(프로토콜 모델링 및 보안 감사), 자바 환경 등



최 성 자

1991년 한남대학교 전자계산공학과 졸업
1995년 ~ 현재 한남대학교 전자계산공학과 석사과정 재학중

※ 관심 분야 : 소프트웨어공학, 패트리넷, 보안평가 EDI 보안, 자바, 워크플로우 모델링 등