이동코드에 관련한 보안문제 및 방안기법 연구

김준회, 조충호(고려대학교 전산학과 데이터통신 및 네트워크연구실)

요 약

이동코드는 자바 애플릿(applet)이나 스크립트와 같이 원격지에서 실행가능한 코드로서 현재 웹 브라우저를 통하여 쉽게 수행 가능하다. 이러한 프로그램은 누구나 작성할 수 있고 브라우저를 수행할 수 있는 어떤 컴퓨터에서도 수행 가능하다. 즉, 자바 애플릿과 같이 운영체제나 하드웨어에 관계없이 어떤 플랫폼에도 동일 코드가 수행될 수 있다. (일반적으로 에이전트도 이동 코드라고 부르지만 여기서는 포함시키지 않는다.)

인터넷에서 어떤 컴퓨터에서도 공통적인 스크립트를 수행할 수 있다는 것은 편리함, 기능성에서 많은 장점을 가지고 있지만 보안 관점에서 보면 이러한 공통의 스크립트를 수행할 수 있는 인터프리터는 매우 위험하다. 또한 이러한 인터프리터가 브라우저의 한 부분이기 때문에 위험은 더욱 증가한다.

이동 코드 인터프리터에서 어떤 버그가 존재할 경우 이것을 이용한 악성 사용자가 프로그램을 특정 컴퓨터에서 수행시켜 접근 권한을 쉽게 얻거나 시스템을 파괴할 수 있다. 일반 사용자들이 주로 사용하는 윈도95 같은 운영체제에서는 이러한 공격을 막을 보호대책이 없고 심지어 UNIX에서도 사용자의 권한을 가지고 이동 코드가 수행되기 때문에 사용자의 파일을 조작하거나 정보가 유출될 수 있다.

또한, 이동코드가 서로 다른 수행환경을 이동할 경우, 악성 이동코드로부터 영향을 받을 수 있는 수행환경의 보호와 악성 호스트 및 수행환경에 의해 이동코드가 파괴되는 경우도 있다.

위와 같은 이동 코드의 위험으로부터 발생할 수 있는 보안문제점들의 실제 피해 사례 및 시스템을 보호하기 위해 사용되어온 몇 가지 기법을 제시하였다

본 문

1. Malicious mobile code

트로이안 목마, 컴퓨터바이러스 등의 악성 이동코드로 인해 발생할 수 있는 실행환경을 보호하기 위한 방법

- Sandboxing : 네트웍을 통하여 받은 신뢰할 수 없는 프로그램들은 Sandbox에 의해 제한된 권한으로 실행

- Digital shrink-wrap" : 이동코드에 그에 상응하는 인정서와 함께 전자서명을 하게 함으로써 악성코드를 가지고 있는지 없는지 몰라도 요청이 되었던 소스인지 인증 가능

- Proof carrying code : 호스트 시스템이 신뢰성을 검증할 수 없는 외부에서 제공되는 프로그램을 안전하게 수행할 수 있는가를 결정할 수 있는 방법으로 증명을 포함하게 함.

코드를 작성하는 사람은 반드시 그 코드가 미리 정의된 안전성 정책(safety proof)을 함께 제공해야 한다.

2. Malicious hosts and environment executions

데이터의 흐름제어, 코드조작, 호스트 및 실행환경의 변화등의 악성환경에 의해 이동코드가 파괴는 경우를 보호하기 위한 방법

- Limited blackbox security : 변형 조작이 불가능한 blackbox spec으로부터 실행 가능한 코드를 생성토록 함
- Computing with encrypted functions : 암호화된 함수를 이용하여 원래의 이동코드임을 확인함
- Cryptographic traces : agent가 실행하는 동안 traces의 데이터를 분석함으로써 확인함

3. Sandbox model

몇 개의 연산들만으로 실행할 수 있는 권한을 제한하는 것으로서 sandbox 내에서만 이동 코드가 수행되어 실행 환경에 어떤 피해를 줄 수 없다. 이동 코드가 임의적으로 파일 시스템 접근이나 네트웍 연결을 수행할 수 없게 된다. 이것은 자바 인터프리터에서 사용되는 방식으로 JDK 1.0, 1.1에서 구현되었다.

인터프리터의 구현은 각각의 보안 정책에 따르게 되는데 일반적으로 애플릿은 시스템의 파일을 접근할 수 없고 시스템의 정보를 가질 수 없다. 자바 인터프리터를 안전하게 하는 주요 구성요소는 Classloader, Verifier, Security Manager로 이것은 JVM(Java Virtual Machine)내에 있다.

Classloader : 원격지의 바이트코드를 자바 클래스로 바꾸어 주는 역할을 한다. 이것을 통해서만 원격지의 클래스들이 시스템의 클래스에 부가될 수 있다.

Verifier : 로드되기 전에 static checking을 수행한다. 정당한 코드 여부, 스택 오버플로우, 부적절한 레지스터의 사용, 데이터 타입의 무단 변경 등을 체크한다.

Security Manager : 시스템 자원에 접근을 통제한다. 시스템 관리자나 브라우저 개발자 에 의해 자원에 대한 접근 권한을 변경할 수 있다.

이러한 sandbox 모델의 가장 큰 단점은 구성 요소들의 에러나 구현시 버그가 위험을 초래할 수 있다는 것이다. 실제적으로 네트스케이프나 익스플로러의 에러로 인한 많은 위험이 보고되어 있다. 또한 심각한 피해를 일으키지는 않지만 반복적으로 자원을 소모시키는 등의 malicious 공격에 대해서는 취약하다.

4. Code Signing

클라이언트가 신뢰할 수 있는 대상들의 리스트를 작성해 놓고 실제적으로 코드가 수행될 때 코드와 이 리스트의 서명을 검증한다. 대표적인 예는 Microsoft의 ActiveX에서 사용된다. 사용자는 모든 ActiveX 내용을 받아들이기 때문에 공격자는 한 번 허가를 얻으면 보안 리스트를 손쉽게 변경할 수 있다.

5. 서명과 sandbox의 결합

sandbox 모델과 코드 서명 기법의 장점을 결합해서 사용한다. JDK 1.1에서는 서명이 검증된 애플릿만 받아서 sandbox 모델에 적용시킨다.

6. Firewall

이것은 일반적인 침입차단시스템과 마찬가지로 클라이언트가 속한 도메인에 이동 코드가 들어오는 지점에서 코드를 수행할 것인지 여부를 결정한다. 이러한 기법은 Finjan, Security 7 같은 몇

몇 회사들에서 개발하고 있는데 정확히 어떤 방식으로 사전에 체크하는지는 알려지지 않았다. 이와 유사한 방법으로 playground 방식이 있다. 이것은 다음 그림에서 보여지는 것처럼 proxy 서버에서 애플릿을 변경시켜 playground라는 공간에서 애플릿이 대신 실행되도록 하고 I/O 결과만 사용자에게 보여지게 한다.