JAR 파일 포맷 분석과 JAR 파일 동적 서명 방법

김정일[0] 노봉남
전남대학교 전산학과
(kci[0], bongnam)@chonnam.ac.kr

Analyzing the JAR File Format and Signing JAR Files Dynamically

Chong-IL Kim[0] Bong-Nam Noh
Dept. of Computer Science, Chonnam National University

요 약

자바 코드의 무결성(integrity)과 인증(authenticity)을 보장하기 위해, JAR 파일에 대해 전자서명하는 방법이 이용되고 있다. JAR 파일은 기본적으로 애플리케이션이나 애플릿의 효율적인 배포를 위해 이용된다. 즉, 애플리케이션이나 애플릿을 구성하는 모든 클래스 파일이나 데이터 파일을 묶어 하나의 JAR 파일에 포함시켜 온라인 상으로 클라이언트로 전송한다. 이러한 JAR 파일의 내용은 변경되기 쉽다. 즉, 기존 클래스 파일의 제거, 변경 또는 새로운 클래스 파일의 추가 등으로 인해 기존 JAR 파일의 내용이 변경되고, 또한 클라이언트의 요구에 맞추어 데이터나 코드를 즉시 생성하여 전송하여야 하는 경우, 새로운 JAR 파일이 생성되어야 한다. 따라서 애플리케이션 서버는 예기치 않은 JAR 파일의 변경에 맞추어 동적으로 JAR 파일에 대해 전자서명을 수행하는 기능이 포함되어야 한다.

이 논문에서는 JAR 파일의 포맷과 전자서명 절차를 분석하여, JAR 파일에 대한 전자서명의 특징과 문제점들을 제시하고, 애플리케이션 서버에서 발생 가능한 에러에 적절히 대처하면서 동적으로 JAR 파일들을 전자서명할 수 있는 프로그램 프로시저의 구현을 예시하였다.

1. 서 론

자바(java)는 클라이언트/서버 프로그래밍의 문제점들을 해결하기 위한 해결책으로 가장 폭넓게 주목받고 있다[1]. 이는 네트워크 지향 컴퓨팅과 분산 애플리케이션을 위한 유니버설(universal) 시스템을 지원하며, 특히 최근 수년 간에 '웹 기술과 결합하여 획기적인 웹 프로그래밍을 가능하게 하기 때문이다. 그러나 실행가능한 코드(executable code)를 네트워크 상으로 배포하는 자바의 능력은 보안의 측면에서 문제점들을 야기시켰는데, 그 중에서 중간자(man in the middle) 공격에 취약한, 자바 코드의 무결성과 인증을 보장하는 것은 해결하여야 할 가장 기본적인 문제 중의 하나이다[2, 3].

이러한 공격으로부터 자바 코드의 무결성과 인증을 보장하기 위해 자바 코드에 대해 전자서명을 덧붙이는 방법이 일반적으로 이용되고 있다. 즉, 자바 코드를 전자서명하기 위해, 먼저 모든 자바 클래스 파일과 관련 데이터 파일들을 묶어 하나의 JAR 파일을 생성하고, JAR 파일 내의 파일들에 대해 각각 전자서명을 생성하여 JAR 파일 내에 삽입한다[2, 4].

JAR 파일은 기본적으로 애플리케이션이나 애플릿을 효율적으로 배포하기 위해 이용된다. 예를 들면, 하나의 애플릿은 보통 여러 개의 클래스 파일과 데이터 파일로 구성되는데, 이 모든 파일들을 하나의 JAR 파일로 담게 된다. 이렇게 함으로써, 클라이언트는 여러 번의 HTTP 세션에 걸쳐서 온전한 애플릿을 다운로드 받기 보다는, 단 한번의 HTTP 세션으로 온전한 애플릿을 다운로드할 수 있게 된다.

이러한 JAR 파일의 내용은 변경되기 쉽다. 즉, 기존 클래스 파일의 제거, 변경 또는 새로운 클래스 파일의 추가 등으로 인해 기존 JAR 파일의 내용이 변경되기 쉽다. 또한, 클라이언트의 요구에 맞추어 데이터나 코드를 즉시 생성하여 전송하여야 하는 경우, 새로운 JAR 파일이 생성되어야 한다[6]. 따라서 자바 코드를 온라인 상으로 제공하는 애플리케이션 서버는 예기치 않은 JAR 파일의 변경에 맞추어 동적으로 JAR 파일에 대해 전자서명을 수행하는 기능이 포함되어야 한다.

간단한 해결책으로 JDK 1.2에서 제공하는 툴-jarsigner[8]-을 애플리케이션 서버에서 시스템 호출을 통해 수행하여 JAR 파일에 대해 전자서명할 수 있다. 그러나 이 방법은 전자서명하는 도중 일어나는 에러들을 텍스트 형태로 표준 출력에 출력하기 때문에, 모든 에러 메시지들을 정확히 파싱하여야 에러들을 정확히 식별할 수 있다. 그러나 에러 메시지들의 포맷에는 일관성이 있지 않기 때문에 파싱에 어려움이 있다. 따라서 에러들을 정확히 식별할 수 없어, 발생한 에러에 적절히 대처할 수 없다.

이 논문에서는 전자서명된 JAR 파일의 포맷을 분석하여 그 특징과 문제점들을 제시하고,애플리케이션 서버가 동적으로 JAR 파일들을 전자서명할 수 있고 도중에 발생하는 에러들을 정확히 식별할 수 있도록, JAR 파일을 전자서명하는 프로시저를 구현한다. 이를 위해 2장에서는 JAR파일에 대해 전자서명하는 절차를 분석, 제시한다. 그리고 3.1절에서는 구현시 고려사항들을 설명하고, 3.2절에서는 구현 프로시저의 프로토타입(prototype)을 예시한다. 3.3절에서는 구현 프로시저의 활용를 설명한다. 마지막으로 4장에서 결론 및 향후 연구방향을 제시한다.

2. JAR 파일에 대한 전자서명 과정

2.1 생성과 검증

JAR 파일에 대한 전자서명은 자바 코드의 무결성과 인증을 보장하는 수단이다. 전자서명은 PKI(public key infrastructure)와 관련되는데, 서명자의 비밀키로서 전자서명을 생성한다. 이때 서명자는 전형적으로 코드 개발자이거나 배포자이다. 서명된 JAR 파일의 검증은 서명자의 공개키를 이용하여 검증한다[4, 7].

전자서명 때문에 추가되는 정보들은 JAR 파일 내에 덧붙여진다.

2.2 JAR 파일 포맷(Format)

JAR 파일은 zip 파일 포맷에 기초하여 여러 개의 파일들을 한 개의 파일로 취합하여 압축하기 위한 포맷이다. JAR 파일의 기본적인 구성은 실제 포함되는 클래스/데이터 파일들의 데이터와 META-INF 디렉토리로 구성된다. META-INF 디렉토리는 JAR 파일을 다양한 목적으로 사용하기 위한 부가 정보들이 저장된다. 특히, "manifest" 파일(META-INF/MANIFEST.MF)은 여러 개의 엔트리들로 구성되는데 각 엔트리는 "NAME=VALUE"의 쌍으로 기술된다[4].

2.3 전자서명 후 JAR 파일의 변경 사항

JAR 파일에 대한 전자서명 방법의 주요 특징은 "multiple signature"와 "hash table"이다. 즉, 여러 명의 서명자가 동일한 JAR 파일에 대해 전자서명이 가능하고, 전자서명에 관련한 해쉬값은 JAR 파일에 테이블 형태로 삽입된다.

구체적으로 보자면, JAR 파일에 대해 전자서명을 하면 원래의 JAR 파일에 약간의 변화가 생긴다. META-INF 디렉토리 밑에 signature 파일과 signature block 파일이 추가로 생성되며, manifest 파일에 전자서명에 관련한 정보가 추가된다.

- Manifest 파일에는 각 클래스 파일 또는 데이터 파일들에 대한 해쉬값이 계산되어 기록된다. 이때 해쉬값 계산을 위해 SHA-1이 기본적으로 이용된다.
- Signature 파일에는 각 파일에 대해 manifest 파일에 기록된 해당 엔트리들에 대한 해쉬값이 저장된다.
- Signature block 파일에는 signature 파일에 대한 전자서명값과 서명자의 공개키 또는 키 체인이 저장된다[4,5].

3. 해결 프로시저의 구현

프로시저를 구현하기 위해서는 JAR 파일의 포맷을 적절히 처리하고, 해쉬값의 계산, 전자서명 생성 등의 기능이 구현되어야 한다. 자바에서는 이러한 기능을 구현한 적절한 패키지들을 제공하고 있다. 따라서 자바를 이용하여 프로시저를 구현하였다.

3.1 고려 사항

JAR 파일의 포맷을 처리하고, 해쉬값의 계산, 공개키를 이용한 전자서명을 위해 java.util.jar (JarFile, Manifest 등), java.security (Certificate, PrivateKey 등) 등을 이용할 수 있다. 그러나 이러한 자바 패키지들을 이용하여 JAR 파일에 대해 전자서명하는 도중 다양한 에러가 발생할 수 있다:

- InvalidKeyException: 전자서명시 사용되는 비밀에 대해서 형식이 잘못된 경우 발생함
- NoSuchAlgorithm: 해쉬값을 계산하기 위해 지정된 방법이 지원되지 않는 경우 발생
- SignatureException: 해쉬값을 계산하여 저장하는 버퍼의 용량 초과시 발생
- IOException: JAR 파일을 읽을 수 없거나 결과를 저장할 수 없을 때 발생함

구현 프로시저를 사용하는 애플리케이션 서버는 전자서명 도중 발생하는 에러들을 적절히 대처하기 위해서는 먼저 그 에러들을 정확히 식별할 수 있어야 한다. 따라서 구현 프로시저는 발생한 에러를 명확히 전달하여야 한다.

3.2 프로토타입

그림 1에서는 구현 프로시저의 프로토타입을 보여주고 있다. SingJarFile 클래스의 객체를 생성함으로써 특정 서명자에 대한 전자 서명 과정을 준비한다. 그 클래스의 메쏘드(method) 중 createSignedJarFile은 실제 해당 JAR 파일에 대해 전자서명을 수행하고, 전자서명에 관련한 정보들이 추가된 JAR 파일을 생성한다.

전자서명 수행 중 발생하는 에러들은 예외(exception)를 발생시켜 애플리케이션으로 전파한다.

3.3 활용 방안

JAR 파일의 내용은 변하기 쉽다. 즉, 기존 클래스 파일의 제거, 변경 또는 새로운 클래스 파일의 추가 등으로 인해 기존 JAR 파일의 내용이 변경된다.

더구나 클라이언트의 요구에 맞추어 데이터나 코드를 즉시 생성하여 전송하여야 하는 경우, 애플리케이션 서버가 동적으로 전자서명하여야 하므로 그 서버에 전자서명하는 기능이 포함되어야 한다[6]. 앞서 구현한 프로시저를 애플리케이션 서버에 탑재함으로써 이러한 문제를 해결할 수 있다. 그림 2에서는 구현한프로시저를 활용하는 애플리케이션 서버의 일부분이다.

그림 3에서는 Count.class로 구성되는 JAR 파일을, 앞서 구현한 프로시저를 이용하여 전자서명함으로써 생성된 코드(sCount.jar)가 클라이언트의 자바 실행 환경에서 이상없이 수행되는 것을 보여주고 있다. Count.class는 주어진 파일의 문자 개수를 출력하는 코드이다.

```
class signJarFile {
  public signJarFile(String alias, PrivateKey
privatekey, X509Certificate[] certChain) {..}
  public void createSinedJarFile(JarFile jarFile,
OutputStream outputStream) {
    // 주어진 JAR 파일의 Manifest 추출
    manifest = getManifest(jarFile);
    // 주어진 JAR 파일 내의 요소들에 대한 해쉬값
계산
    signatures=createFileSignatures(jarFile,hash
Function);
    // Manifest 에 해쉬값 기록
    manifest = updateManifest(manifest,
signatures);
    // manifest에 대한 해쉬값 계산
    signatureFile =
  createSignatureFile(manifest, hashFunction);
  // 전자서명 생성
    signatureBlock=createSignatureBlock(signat
ureFile, alias, privatekey, certChain);
    // signed JAR 파일의 생성
    insertSignaturesIntoJar(outputStream,
jarFile, signatures,
signatureFile,signatureBlock);
  }
  private Manifest getManifest(..) throws
IOException {..}
  private SignatureFile createFileSignatures(..)
throws NoSuchAlgorithm SignatureException
{..}
  private SignatureBlock createSignatureFile(..)
throws NoSuchAlgorithm SignatureException
{ }
```

그림 1

```
privateKey =keyStore.getKey(alias, password);
certChain=keyStore.getCertChain(alias);
try {
  signJarFile = new SignJarFile(alias,
privateKey, certChain);
  signJarFile.createSignedJarFile(jarFile,
signedJarFile);
} catch (Exception e) {...}
```

그림 2

```
$ java –cp Count sCount.jar  example
500 chars
```

그림 3

4. 결론 및 향후 연구방향

자바 코드의 무결성과 인증을 위해 사용되는 JAR 파일에 대한 전자서명 방법에 대해 그 절차와 특징을 분석하고, 이를 토대로 애플리케이션 서버에서 에러에 적절히 대처하면서 동적으로 JAR 파일들을 전자서명할 수 있도록 프로시저를 구현하였다.

한편, 자바 코드의 전자서명 방법은 상대적으로 용량이 부족한 클라이언트에 대한 고려가 없다[6]. 애플릿을 구성하는 클래스 파일이나 데이터 파일이 많은 경우 당연히 해쉬 테이블의 크기도 늘어나게 되는데, 이런 경우 상대적으로 용량이 부족한 클라이언트에게는 적절하지 못하다. 또한 모든 클래스 파일이나 데이터 파일에 대해 해쉬값을 구하여 전자서명하는데 드는 비용도 만만치 않을 뿐더러, 다운로드한 애플릿을 검증하기 위해 시간이 많이 들기 때문에 애플릿을 실제 수행하는 시점이 지체된다.

더 이상 참조되지 않는 클래스는 JAVA의 실행 환경에 보관(cache)되지만 클라이언트의 용량 초과시 삭제된다. 또다시 그 클래스를 다운로드하고자 할 때 그 클래스를 포함하는 JAR 파일을 전부 다운로드하여야 하는 문제가 있다.

향후 위와 같은 문제들을 고려한 자바 코드의 전자서명 방법을 연구해야 한다.

참고문헌

[1] A. Fyggetta and G. P. Picco and G. Vigna, " Understanding Code Mobility," IEEE Transactions on Software Engineering, 24(5), pp.342-361,1998

[2] J. S. Fritzinger and M. Mueller," Java Security," Sun Microsystems, 1995.

[3] Drew Dean, Edward W. Felten, and Dan S. Wallach, " Java Security From HotJava to Netscape and Beyond," Proceedings of the 1996 IEEE Symposium on Security and Privacy, pp.190-200, 1996

[4] Sun Microsystems, Signing and Verification JAR Files, http://developer.java.sun.com/developer/Books/JAR/sign/

[5] Sun Microsystems, JAR File Format, http://developer.java.sun.com/developer/Books/JAR

[6] R. Gennaro and R. Rohatgi, " How to Sign Digital Streams," Information and Computation, 165, pp. 100 - 116, 2001

[7] W.A. Jasen, " Countermeasures for mobile agent security," computer communications, 23, pp. 1667-1676, 2000

[8] Sun Microsystems, JAR Signing and Verification Tool, http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/jarsigner.html