# Jini 2.x 보안 프레임워크에서 프락시 준비의 개선 효과

김성기*, 민병준**
*선문대학교 교양대학 IT교육원, **인천대학교 컴퓨터공학과
e-mail: skkim@sunmoon.ac.kr, bjmin@incheon.ac.kr

# The Effects after Improvement in Proxy Preparation in the Jini 2.x Security Framework

Sung-Ki Kim*, Byoung-Joon Min**
*IT Education Center, Sun Moon University
**Dept. of Computer Science and Engineering, University of Incheon

요 약

Jini 2.x 표준의 보안 모델은 클라이언트와 서버간의 신뢰와 안전한 통신을 위해 크게 3 가지 순차적인 과정을 요구한다. 순서대로, 프락시 준비, 상호인증을 위한 자격증명과 신원의 확인, 세션의 암호화 연산이다. 첫 번째 프락시 준비 과정은 다운로드 서비스프락시가 올바른 출처에서 온 안전한 이동코드인가를 검증하는 과정을 요구하고, 보안 제약조건과 동적 권한부여 구성을 완료하는 과정을 포함한다. 나머지 두 과정은 기존 X.509 인증서 기반의 TLS/SSL 통신이 요구하던 과정과 유사하다. 프락시 준비 과정은 클라이언트-서버 양측에 안전하고 신뢰 할 수 있는 스텁(stub)을 준비하는 시간이기 때문에 실질적인 통신지연 요소에 들지 않는다. 그러나 나머지 과정들은 기존 SSL 통신이 갖는 오버헤드를 가지고 있다는 것을 본 논문의 실험에서 확인하였다. 본 논문에서는 프락시 준비 과정에서 상호 인증정보와 세션 키를 확보하기 위한 Jini 서비스 구조와 방법을 제시하고 서비스 연결 지연을 단축한 실험결과를 논한다.

## 1. 서론

Jini 2.x 표준에서 제시된 보안모델은 클라이언트와 서버간의 안전한 통신을 위해서 다음 3 가지 태스크를 순차적으로 요구한다. 첫 째 태스크인 프락시 준비는 다운로드 서비스프락시에 대해 신뢰를 검증(trust verification)하는 과정을 요구하고, 보안관련 제약사항(constraints)과 메서드 호출에 대한 동적 권한부여의 구성을 요구한다. 두 번째 태스크는 스텁을 실질적으로 이용하고 있는 주체에 대한 상호 인증을 위해 자격증명(credential)과 신원(subject)의 확인을 요구한다. 그리고 세 번째는 세션의 무결성과 기밀성 유지를 위한 암호 알고리즘 연산이다. 두 번째와 세 번째 태스크는 기존 X.509 인증서 기반의 TLS/SSL 통신이 요구하던 과정과 같다.

프락시 준비 과정은 클라이언트-서버 양측에 안전하고 신뢰할 수 있는 스텁을 준비하는 시간이기 때문에 실질적인 통신지연 요소에 들지 않는다. 그러나 나머지 태스크들은 기존 SSL 통신이 자격증명과 신원확인을 위해 소비하는 연결 지연 오버헤드를 가지고 있다는 것을 본 논문의 실험에서 확인하였다.

본 논문에서는 프락시 준비 과정에서 상호인증을 위한 인증정보와 세션 키를 미리 준비함으로써 실질적인 통신 지연을 줄이는 방안을 제시한다.

Jini 서비스 제공자(service provider)는 서비스프락시를 배출(exprot) 할 때, 인증서버가 발급한 자신의 공개키 인증서가 삽입되도록 하고 클라이언트는 ***bootstrap proxy*** 를 통해 ***trust verifier*** 를 서비스 제공자에게 요구할 때 자신의 공개키 인증서를 제공하는 방법으로 프락시 준비 과정에서 상호인증을 위한 인증정보와 세션 키를 준비한다. 준비된 인증정보는 클라이언트-서버 양측 스텁에 대한 실질적인 사용 주체를 인증할 수 있게 하며, 세션의 기밀성을 제공할 수 있는 암호 키 생성의 원천(seed)을 제공한다.

본 논문에서 제안한 프락시 준비 개선을 통해서 메시지 크기에 비례하여 약 2.6 배의 연결 지연을 단축하는 실험 결과를 얻었다.

## 2. 관련 연구

Jini 시스템의 보안성 향상을 위한 대표적인 연구들은 [1,2,3,4]의 연구가 있다. [1]에서는 Jini 연합의 중심에 인증과 권한부여를 담당하는 서비스 구현을 두고 이를 통해 Lookup 서버와 서비스프락시, 서버간의 신뢰확립과 접근제어 요구를 해결하는 Jini 서비스 구조를 제시하였다. [2]에서는 SPKI (Simple Public Key Infrastructure) 인증서를 이용하여 서버-서비스프락시-사용자를 연결하는 인증, 인가 체인을 형성할 수 있는 방안과 Jini 서비스 구조를 제시하였다. 인가 체인은 서비스를 구현한 서버에서 확인하고 인증 체인은 사용자가 확인할 수 있게 함으로써 상호 인증과 권한 인가 요구사항을 동시에 충족시키는 점이 특징이다. [3]에서는 [1]의 구조와 비슷하나 통신채널의 기밀성과 무결성을 제공하는 메카니즘이 다른 연구와 다르다. 또한 클라이언트 구현에 대한 보안투명성을 제공하기 위해서 보안 기능 수행의 상당부분을 서비스프락시가

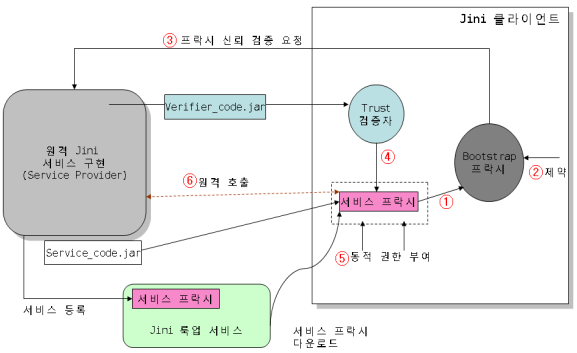
담당하는 구조이다. [1,2]의 연구는 보안성 제공이 응용 구현에 의존하며 클라이언트에서 보안투명성이 없다.

서비스프락시 코드의 안전 문제는 [1,2,3]의 연구 모두가 인증된 서비스프락시 코드에 대해서는 안전을 신뢰한다는 가정에 기초하고 있다.

메시지 교환에서의 보안성 제공은 [1]에서는 Java 오픈소스로 구현한 SSL 기반 RMI 통신 구현에 의존하고 있고, [2]에서는 Java 표준 JSSE(Java Socket Security Extension)[5]를 이용한 TLS 기반의 RMI 통신 구현에 의존한다. [3]에서는 서버와 서비스프락시 간에 연결마다 동적인 Diffie-Hallman 키교환 알고리즘으로 공유세션키를 생성한 다음 이를 통해 기밀성을 제공한다. 그리고 HMAC-MD5 알고리즘으로 메시지의 무결성을 보호한다.

## 3 Jini 2.x 보안모델에서 프락시 준비

### 3.1 서비스프락시 객체의 출처 검증




(그림 1) 프락시 신뢰 검증 과정

(그림 1)은 다운로드 서비스프락시가 보안상 안전하다는 신뢰 확립 과정을 보이고 있다. 클라이언트가 Lookup 서비스로부터 받은 Lookup 서비스프락시 객체는 서비스에 필요한 데이터 인스턴스와 관련 클래스 파일의 위치를 말해주는 URL이 포함되어 있다. 대게 서비스 제공자는 클래스 파일 또는 클래스 아카이브(jar) 파일을 HTTP 서버에 올려놓는다. 클라이언트가 이 서비스프락시 객체들을 HTTP 를 통해서 받아간다.

일단 필요한 서비스프락시 객체들을 받으면 이에 대해 보안관련 제약(constraint)들을 설정할 수 있지만, 실제로는 원격 서비스 구현과 통신을 수행하는 것은 서비스프락시 객체이다. 그러나 프락시코드가 인증되지 않은 소스로부터 유입될 수 있기 때문에 프락시 신뢰 검증은 두 가지로 측면으로 이루어진다. 하나는 서비스프락시 객체가 올바른 출처로부터 온 것인지를 확인하는 일이고, 다른 하나는 프락시 객체가 애초에 서비스 제공자가 만든(export) 프락시인지를 확인하는 일이다.

전자를 검증하기 위한 수단으로 Jini 2.x 시스템은 HTTPMD URL을 사용한다. Lookup 서비스프락시 객체에 표시되어 있는 URL 에 해시 값이 포함되어 있어서 클라이언트가 받은 클래스 파일에 대해 계산한 해시 값이 URL에 첨부된 해시 값과 동일하다면 프락시 객체를 신뢰한다.

### 3.2 서비스프락시 구현의 신뢰 검증

***bootstrap proxy*** 라고 하는 완전히 신뢰할 수 있는 로컬 코드로 이루어진 프락시 객체를 클라이언트가 다운받은 서비스프락시 객체로부터 얻는다. 이를 지원하기 위해서 서비스프락시는 자신의 서비스 인터페이스에 정의된 메서드들과는 별개로 다음과 같은 특별한 메서드 하나를 구현해야한다.

*ProxyTrustIterator getProxyTrustIterator();* (1)

클라이언트는 ***java.lang.Reflectio*** 클래스를 사용하여 프락시 객체 내에서 이 메서드를 발견한다. 클라이언트가 이 메서드들을 호출하면 ***ProxyTrustIterator*** 가 궁극적으로 ***bootstrap proxy*** 객체를 만들어 낸다.

***bootstrap proxy*** 는 다음 메서드를 호출한다.

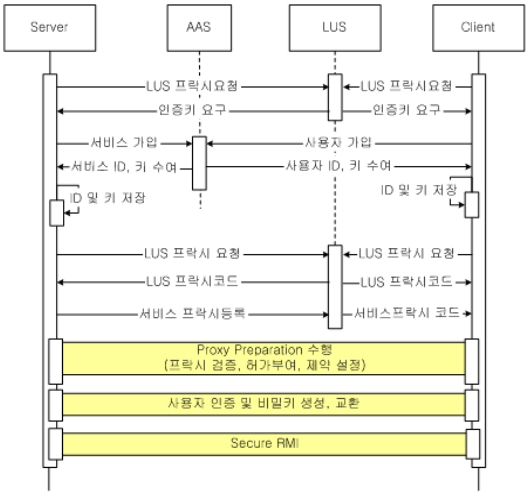
*TrustVerifier getVerifier() throws RemoteException* (2)

이 메서드는 서버에게 ***TrustVerifier*** 객체를 요청하는 원격 콜백을 수행한다. 반환되는 ***TrustVerifier*** 객체는 서비스 프락시를 검증하는 객체이다. ***TrustVerifier*** 는 클라이언트로 전달되는 지역객체이며, 다음 메서드의 입력 파라미터로 전달된 서비스프락시가 신뢰할 수 있는지를 판정한다.

*boolean isTrustedObject(Object jiniProxy,*
*TrustVerifier.Context context)* (3)

여기서 ***TrustVerifier.Context*** 파라미터는 ***TrustVerifier*** 객체에게 필요한 컨텍스트를 전달하기 위해 사용한다.

Jini 2.x 보안 프레임워크에서는 객체 구현 내용(semantic)의 동등성(equality)에 의존하는 프락시 신뢰 알고리즘을 제공한다. 즉, 서버가 ***bootstrap proxy***로 반환하는 ***TrustVerifier***는 하나의 규범적인 프락시 인스턴스를 내장하고 있는데, ***TrustVerifier*** 는 클라이언트가 받은 프락시 객체와 이 규범적인 프락시 인스턴스를 비교하여 일치여부를 판정한다. 만약 일치하면 다운로드 서비스프락시를 신뢰할 수 있다고 판정한다.




(그림 2) Jini 시스템 보안구조

## 4 사전인증을 위한 프락시 준비의 개선 방법

### 4.1 안전한 Jini 서비스 구조

프락시 준비 과정을 개선하고 안전한 통신을 지원하기 위해서 본 논문에서 제안한 안전한 Jini 시스템 구조는 위 (그림 2)와 같다.

클라이언트와 서버가 Lookup 서버에 접근하여 Lookup 서비스프락시를 얻기 위해서는 이에 접근할 수 있는 키를 얻어야 한다. 본 논문에서는 Jini 서비스 환경에서 Lookup 서버에 대한 접근을 인증, 인가하고 하나의 Jini 시스템 내에서 CA 역할을 수행하는 Jini 서비스를 설계하였다. (그림 2)에서 '응용 레벨 인증 서버(AAS)' 가 이를 수행하는 Jini 기반 인증서버이다.

사용자는 가입을 하고 Lookup 서버에 대한 접근키를 얻어 서비스 발견과 서비스 등록의 자유를 얻는다. 이때 Lookup 서버에 대한 접근키를 얻으면서 사용자와 서비스 제공자는 각각 자신의 전자서명에 대한 인증서도 얻는다. 이러한 접근키와 인증서는 서버와 클라이언트 노드에서 하나의 지역객체로 저장된다.

### 4.2 공개키 인증서 교환

프락시 준비 과정은 3.1절에서 논한바와 같이 클라이언트-서버를 안전하게 연결하는 스텁을 준비하는 과정이다. 이 과정이 끝나면 그 스텁을 이용하는 통신주체에 대한 인증과정이 뒤따른다. 이 인증과정은 X.509 인증서를 기반으로 하는 기존 TLS/SSL 통신과정과 같다. 본 논문에서는 클라이언트와 서버가 스텁을 준비하는 과정에서 각각 자신의 인증서를 교환하는 방법은 다음과 같다.

● 서비스 제공자 측

```
import java.rmi.*;
import net.jini.config.*;
import net.jini.export.*;
import java.io.*;
public class Serivce implements Remote
 public Service(String[] configArgs) {
   File certificateFile = null;
   Configuration config =
       ConfigurationProvider.getInstance(configArgs);
   exporter = BasicJeriExporter(TcpServerEndpoint.getInstance(0),
                                new BasicILFactory());
   Remote service = (Remote) config.getEntry("Service", "service",
                                              Remote.class);
   certificateFile = (File) config.getEntry("Service", "certificateFile",
                                             File.class);
   Remote proxy = exporter.export(service);
  }
}
```

(그림 3) 서비스 프락시의 배출(exporting)

```
import net.jini.jrmp.*;
import java.rmi.*;
import java.io.File;
Service {
 exporter = new BasicJeriExporter(TcpServerEndpoint.getInstance(0),
                                  new BasicILFactory());
 impl = (Remote) config.getEntry("Service", "service", Remote.class);
 certificateFile = new File("CertFile");  // 공개키 인증서 객체 파일
}
```

(그림 4) 서비스 프락시 구성(configuration)

서비스 제공자가 서비스프락시를 배출하기 전에, 자신의 공개키 인증서 객체가 함께 추가되도록 구성한다. 이러한 구성은 다음 (그림 3)과(그림 4)와 같이 Jini 구성(Configuration) 메커니즘을 이용한다.

● 클라이언트 측

클라이언트는 다운로드 서비스프락시로부터 ***bootstrap proxy*** 객체를 얻은 후, ***getVerifier()*** 를 통해 ***TrustVerifier*** 객체를 요구한다.

본 논문에서는 ***net.jini.security.proxytrust.ProxyTrust*** 인터페이스의 ***getVerifier()*** 구현에서 클라이언트의 공개키 인증서를 원격 서비스 구현에 전달하는 방법을 제안한다. 클라이언트는 4.2절에서 설명한 바와 같이 다운로드 서비스프락시로부터 인증서버가 서명한 서버의 공개키를 얻는다. 그리고 자신이 인증서버로부터 받은 자신의 공개키 인증서를 서버의 공개키로 암호화하여 ***context*** 객체를 얻는다.

$$\text{context} \leftarrow EK^{+}_{server}\left(K^{-}_{CA}\left(K^{+}_{client}\right),\ K^{+}_{CA},\ \text{nonce}\right)$$

***context*** 객체는 ***getVerifier(context)*** 호출로 원격 서비스 구현에 전달된다.

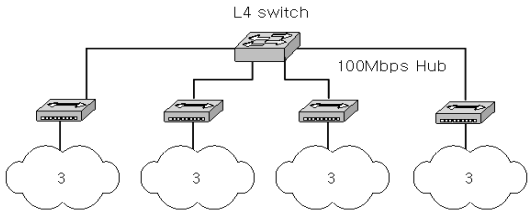
### 4.3 안전한 원격 통신

클라이언트가 서비스프락시의 신뢰를 검증하면서 얻은 서버의 공개키와 자신이 생성한 nonce 값 그리고 자신의 공개키 값을 서버의 공개키로 암호화해 보내고 서버 역시 클라이언트의 공개키로 자신이 확인한 nonce 값을 암호화해서 응답함으로서 양 측이 비밀키를 공유하도록 한다. 공유 비밀키에 의한 기밀성 제공은 향후 서버와 클라이언트를 잇는 엔드포인트에서 암복호화에 빠른 응답성을 보일 것이다. 메시지의 무결성 확인은 HMAC 알고리즘을 이용한다.

## 5. 실험 및 논의

### 5.1 실험환경과 방법

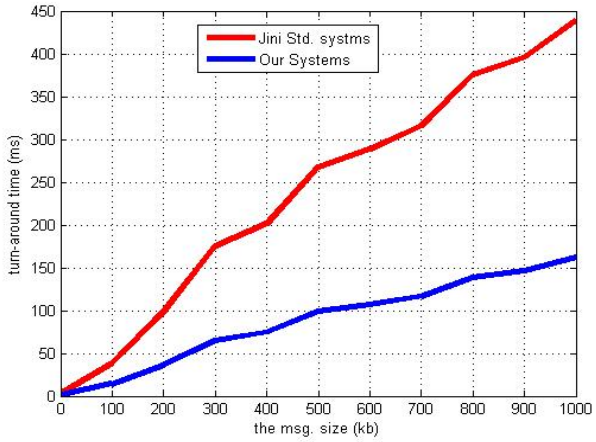
본 논문의 실험은 (그림 5)와 같이 총 12대의 PC(intel pentium4 CPU, 512MB 메모리)로 구성된 하나의 서브넷을 하나의 Jini 시스템으로 구성하였다. 각 PC는 실험에 따라서 Jini 서비스들을 호스팅하거나 클라이언트 소프트웨어를 호스팅 하는데 사용하였다. 소프트웨어 개발은 JDK1.6 버전과 Jini Starter kit 2.1을 사용하였다.




(그림 5) 실험 환경

실험은 간단한 에코 서비스를 구현하였다. 일정 크기의 메시지를 클라이언트가 전송했을 경우, turn-around time을 측정하였다. 이 때 메시지 크기별로 측정한 turn-around time은 독립적인 측정 결과이다. 즉 매 연결시도 마다 클라이언트가 최초 서비스 연결과 응답을 얻는 데 걸리는 시간을 측정한 것과 같은 과정을 반복한다.

### 5.2 실험 결과




(그림 6) 메시지 크기에 따른 응답 지연시간

클라이언트가 서버에 연결하고 응답을 얻기 까지 프락시 준비, 상호 통신주체에 대한 인증, 세션 암호화라는 3 단계의 시간 비용을 감수하다. (그림 6)은 본 논문이 제안한 방법대로 서비스프락시 준비 단계에서 통신주체에 대한 인증정보를 미리 교환한 덕에 메시지 크기에 비례하여 약 2.6 배의 지연시간 단축효과를 얻는 결과를 보이고 있다.

## 6. 결론

Jini 2.x 보안 프레임워크는 유비쿼터스 환경에서 취약할 수 있는 Jini 시스템의 보안요구 사항을 충족시키기 위해 기존 Java 보안 모델에서 이동코드에 대한 보안 메커니즘이 강화되었다.

본 연구는 새로운 Jini 2.x 보안 프레임워크에서 불필요한 연결 지연을 초래하는 오버헤드를 발견하고 이를 줄일 수 있는 구조적 해결 방안을 제시하였다.

## 참고문헌

[1] Peer Hasselmeyer, et al., "Trade-offs in a Secure Jini Service Architecture", LNCS 1890, Springer-Verlag Berlin, 2000.

[2] Pasi Eronen and Pekka Nikander. "Decentralized Jini security", In Proceedings of the Network and Distributed System Security Symposium (NDSS 2001), pages 161 - 172, San Diego, California, February 2001.

[3] Thomas Schoch, et al. "Making Jini Secure", Proc. 4th International Conference on Electronic Commerce Research, pages 276-286, Nov. 2001.

[4] Sun Microsystems, "Jini Technology Starter Kit Overview v2.0," Published Specification, http://java.sun.com/developer/products/jini/arch2_0.html, 2003.

[5] Sun Microsystems, "Java Secure Socket Extension(JSSE) Reference Guide for Java Platform Standard Edition 6", http://java.sun.com/javase/6/docs/tech-notes/guides/security/jsse/JSSERefGuide.html#Fe-atures.