SSL 컴포넌트의 설계 및 구현

조은애° 김영갑 문창주 박대하 백두권
고려대학교 컴퓨터학과 소프트웨어 시스템 연구실
{eacho°, yjkim, mcj}@software.korea.ac.kr, dhpark@stitec.com, baik@software.korea.ac.kr

Design and Implementation of a SSL Component

Eun-Ae Cho° Young-Gab Kim, Chang-Joo Moon, Dae-Ha Park, Doo-Kwon Baik
Software System Lab. Department of Computer Science and Engineering, Korea University

요 약

최근 괄목할만한 인터넷의 보급과 발전으로 인해, 온라인 상의 정보에 대한 보안의 중요성이 증대되고 있다. 현재 Netscape사의 SSL(secure socket layer)을 비롯하여 몇몇의 정보 보안에 관한 표준안 들이 나와 있지만, 개발자가 이를 사용하고자 하는데 있어서는 API에 대한 지식뿐만 아니라 이에 더불어 보안에 관한 전문 지식도 함께 요구된다. 그러므로 일반 개발자가 SSL 등의 API를 직접 이용하여 정보 보안에 관한 완벽한 개발을 하는 것은 힘든 과제이다. 대표적인 보안 통신 프로토콜인 SSL은 응용 계층(application layer)과 전송 계층(transport layer) 사이에서 동작하며, 안전한 보안 채널을 제공하지만, 각 표준안별 API간의 호환성이 적고, 세부 항목들에 적용하는 알고리즘에 대한 개발자의 선택이 불가능하며, 전송에 관한 모든 데이터를 암호화해야 하는 단점이 있다. 따라서 본 논문에서는 SSL 컴포넌트를 제안함으로써 위와 같은 단점을 해결하고자 하였으며 SSL 컴포넌트가 차후 컴포넌트 환경에서 비즈니스 컴포넌트로써 수행하게 될 역할을 설명한다.

1. 서 론

최근 네트워크 상의 정보량이 기하급수적으로 증대되면서, 개방형 시스템에 대한 보안 문제의 중요성이 대두되고 있다. 이에 따라 보안과 암호 API에 대한 전문 지식을 모두 갖춘 개발자의 수요가 증가하고, 분산 환경 하에서 애플리케이션에 표준화된 암호화 API 도입이 필요하게 되었다.

현재, 정보에 대한 보안 프로토콜(security protocol)로 Netscape 사의 SSL(secure socket layer)이 가장 일반적으로 이용되고 있다. 이것은 브라우저와 웹 서버 등에 포함되어 HTTPS(secure hypertext transfer protocol) 형식으로 사용된다. 또한, C언어로 구현된 Eric Young의 OpenSSL과 Sun Microsystems의 JSSE(JAVA secure socket extension) 역시 API로 표준화 되어 SSL 프로토콜의 기능을 제공하고 있다. 하지만, 이런 프로토콜들은 애플리케이션 내부에서 함수 형태로는 이용이 가능한 반면에, 컴포넌트 환경을 지원해 주기 위한 API 호환 기능을 충분히 가지고 있지는 않으며, 한번 설정된 SSL 연결에 대해서는 모든 데이터들을 암호화해야만 한다. 또, 모든 데이터들을 암호화하는데서 가져오는 높은 시스템 부하로 인해 오버헤드가 발생하는 단점을 가진다.

따라서 본 논문에서는 하부의 암호화 API의 종류에 관계없이 보안 요구사항을 컴포넌트 플랫폼에 독립적으로 지원할 수 있고, 컴포넌트 형식에 맞추어 실행할 수 있는 기능을 가진 비즈니스 컴포넌트 형식의 SSL 컴포넌트를 제안한다.

2. 연구 배경

2.1 SSL 프로토콜(SSL protocol)

SSL 프로토콜은 TCP 상에서 인증과 암호화를 통하여 신뢰성이 있는 종단간(end-to-end) 보안 서비스를 제공하기 위해 설계한 통신 프로토콜이다. 플랫폼과 애플리케이션에 독립적이며, 클라이언트와 서버 사이의 안전한 통신 채널을 관리한다. 대체로 온라인 상의 민감한 정보에 대한 처리를 목적으로 하고 있으며, 각 애플리케이션은 정보에 대한 트랜잭션의 중요도와 가치에 따라 비밀성, 무결성, 인증 등의 보안 요구사항 기능을 이용한다[1][2].

SSL 프로토콜은 그림 1에서 볼 수 있는 것과 같이, 하나의 프로토콜로 이루어진 것이 아니라, 두 계층의 프로토콜로 이루어진다[3].

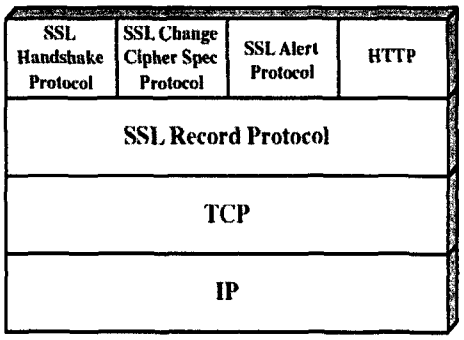

그림 1 SSL 프로토콜 스택

SSL 프로토콜의 구조는 크게 SSL 프로토콜의 동작에 대한 관리를 위해 사용되는 부분과 실질적인 보안 서비스를 제공하는 부분으로 나눌 수 있다.

(1) SSL 프로토콜의 상위 계층

SSL 프로토콜의 상위 계층은 동작에 대한 관리를 위해 사용되는 부분으로 SSL 핸드쉐이크 프로토콜(SSL handshake protocol), SSL 암호기 스펙 교환 프로토콜(SSL change cipher spec protocol), SSL 경고 프로토콜(SSL alert protocol)로 나누어진다. SSL 핸드쉐이크 프로토콜은 메시지 교환을 통해 한 세션 동안 서버와 클라이언트의 보안 서비스에서 사용되는 세션키, 암호 알고리즘 등과 같은 암호 매개변수를 협상하며, 암호기 스펙 교환 프로토콜은 기존의 암호기 스펙에 새로운 암호기 스펙을 복사하여 새로운 것으로 대체하는 일을 한다. SSL 경고 프로토콜은 오류를 심각성에 따라 분류하고 오류 관련 메시지를 전송한다.

(2) SSL 프로토콜의 하위 계층

SSL 프로토콜의 하위 계층은 TCP의 바로 윗단에 위치하며, 실질적인 보안 서비스를 제공하는 부분인 SSL 레코드 프로토콜(SSL record protocol)이다. SSL 레코드 프로토콜은 상위 계층의 프로토콜에서 생성된 세션 정보에 대한 기본적인 보안 서비스(security service)를 제공한다. 특히 웹 클라이언트/서버 상호작용에 대한 전송 서비스를 제공하는 HTTP는 SSL 프로토콜 위에서 동작할 수 있다.

2.2 SSL 프로토콜의 단점

원격 접속 단말기의 경우에 SSL 프로토콜 제어 기능을 가지고 있지 않은 경우도 있으므로, 클라이언트의 보안이 의심될 수 있다. 또한 암호화 하지 않은 데이터에 대해 초당 155.2개의 연결을 처리할 수 있는 e-비지니스 웹서버가 SSL 암호화 데이터에 대해 초당 41.6개의 연결만 처리할 수 있다는 사실[4]을 보면, SSL 프로토콜 방식의 암호 통신은 시스템에 높은 부하를 초래하여 암호화 트랜잭션의 성능과 서비스의 속도를 현격하게 저하시킴을 알 수 있다. 따라서 만족할 만한 성능(high performance)의 SSL 프로토콜의 기능 수행을 기대하기 위해서는 높은 성능의 CPU가 요구되며, 모든 메시지의 암호화에 일상적인 키 길이를 사용해야 한다. 또한, SSL 프로토콜 표준들이 서로 호환성을 가지고 사용되지 않고 있으며, 애플리케이션의 수준, 적용환경 그리고 프로그래밍 언어에 따라서 개발자가 선택적으로 API를 이용하여야 한다는 단점이 있다.

3. SSL 컴포넌트(SSL Component)

본 논문에서 제안하는 SSL 컴포넌트는 표준화된 분산 애플리케이션 환경에 적응하여 개발자에게 플러그앤플레이(plug-and-play)방식으로 보안 인터페이스에 대한 메커니즘을 제공한다. 또한, 현재 SSL 컴포넌트에 대한 표준이 제정되어 있지 않은 상태이므로 이에 대한 보안 컴포넌트 표준안을 제공하는 데에 목적을 둔다.

3.1 SSL 컴포넌트의 구조

다른 보안 컴포넌트(예 : KISA의 비밀성 컴포넌트 'Confidentiality Component'와 무결성 컴포넌트'Integrity Component'[5])와 연동이 가능한 SSL 컴포넌트는, 시퀀스 다이어그램(sequence diagram)[6]으로 그려보면 그림 2와 같이 나타낼 수 있다.

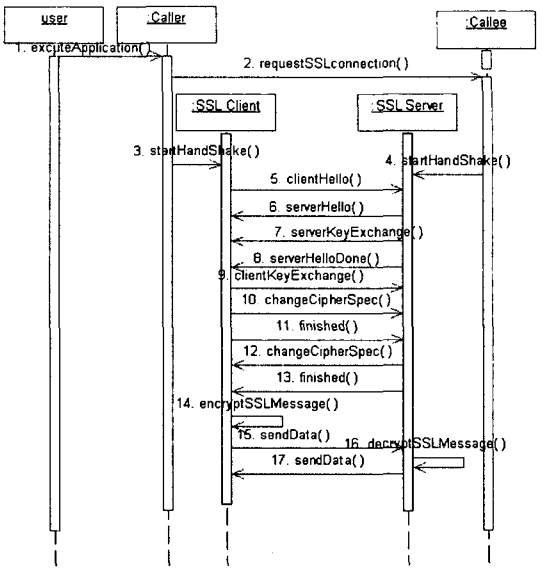

그림 2 SSL 컴포넌트 시퀀스 다이어그램

User가 SSL 컴포넌트인 Security Service를 실행시킨다. 그러면, SSL의 클라이언트 서비스를 원하는 쪽은 caller 컴포넌트가 되고, 서버 서비스를 원하는 쪽은 callee 컴포넌트가 된다. Caller가 Callee에 원격 메소드 호출(remote method invocation: RMI)로 SSL 연결을 요청하는 메시지를 보내면 SSL 서비스 시스템이 동작을 실행한다. 컴포넌트에서 이루어지는 SSL 서비스는 기존 SSL 프로토콜의 트랜잭션에 따라 핸드쉐이크 과정(Handshake Process), 레코드 과정(Record Process), 세션 관리(Session Management)를 기반으로 실행한다.

다음의 표 1은 SSL 컴포넌트의 세부 메소드를 나타낸 것이다.

표 1 SSL 컴포넌트 주요 메소드의 종류 및 설명

주요 메소드	설명
excuteApplication()	클라이언트를 실행하여 애플리케이션을 실행하는 메소드
requestSSLConnection()	SSL 연결을 형성하기 위해서 Caller가 Callee에게 RMI로 연결을 요청하는 메소드
startHandShake()	SSL 컴포넌트에서 사용할 세션키와 알고리즘을 교환하기 위한 핸드쉐이크 프로토콜을 시작하는 메소드
clientHello()	핸드쉐이크 프로토콜을 시작하기 위해 클라이언트가 서버를 호출하는 메소드

표 1 SSL 컴포넌트 주요 메소드의 종류 및 설명(계속)

serverHello()	핸드쉐이크 프로토콜을 시작하기 위해 서버가 클라이언트를 호출하는 메소드
serverKeyExchange()	키 교환을 위해 서버가 클라이언트에게 자신의 공개키를 전송하는 메소드
serverHelloDone()	서버로부터 serverKeyExchange가 발생한 후, 서버 쪽 Hello 메시지의 전송이 끝났음을 알리는 메소드
clientKeyExchange()	키 교환을 위해 클라이언트가 서버에게 자신의 공개키를 전송하는 메소드
changeCipherSpec()	현재 설정되어 가지고 있는 암호기 스펙을 보냄으로써 finished 메시지와 함께 보안 연결이 완성될 수 있도록 설정하는 메소드
finished()	프로토콜이 끝났음을 알리는 메소드
encryptSSLMessage()	핸드쉐이크 프로토콜이 종료된 후, 교환된 세션키를 이용하여 데이터를 암호화하는 메소드
decryptSSLMessage()	핸드쉐이크 프로토콜이 종료된 후, 교환된 세션키를 이용하여 데이터를 복호화하는 메소드
sendData()	해당하는 쪽에 암복호화된 데이터를 보내는 메소드

본 논문에서 제시하는 SSL 컴포넌트는 표준화된 보안 컴포넌트 인터페이스를 이용한 컴포넌트의 재사용성을 보장하고 시스템의 하부 암호화 API 종류에 상관없이 보안 요구사항을 SSL 컴포넌트 플랫폼에서 독립적으로 지원하며, 구체적인 메커니즘에 종속되지 않고 암호화 개념에 따라 적절한 보안 메커니즘을 제공한다. 또한, SSL 핸드쉐이크 프로토콜 등의 표준 사항을 따르며, SSL 레코드 프로토콜에 대한 부분은 표준화된 보안 컴포넌트 인터페이스들을 이용하여 MAC을 생성하거나 암복호화를 수행하도록 한다. 국내 표준 암호화 알고리즘인 SEED와 HAS-160을 지원할 수 있는 암호기 조합의 구현도 역시 제공한다.

SSL 컴포넌트는 종단간으로 접속되는 클라이언트와 서버 간에 세션 상태 매개변수와 접속 상태 매개변수를 유지 및 관리한다. 따라서 SSL 컴포넌트는 세션 상태와 접속 상태를 내부 상태로 갖는 세션 빈 형태의 EJB로 이루어지는 것이 좋다.

3.2 SSL 컴포넌트의 기능

본 논문에서 제안한 SSL 컴포넌트는 컴포넌트 기반의 소프트웨어이므로 Microsoft의 .NET 플랫폼이나 Sun Microsystems 또는 IBM을 중심으로 하는 J2EE 플랫폼에서 작동하는 비즈니스를 이용할 경우에 비즈니스 컴포넌트를 재사용할 수 있다. SSL 프로토콜의 보안 요구 사항을 충족하며, 이를 컴포넌트 플랫폼에 독립적으로 지원받을 수 있다. 또한 SSL 컴포넌트는 암복호화 알고리즘(SEED, DES, IDEA 등), 해쉬 알고리즘(HAS-160, MD5, SHA-1 등), 공개키 알고리즘(RSA, DSA, ElGamal 등) 등의 구체적인 메커니즘에 종속되지 않고 암호화 개념에 따라 적절한 보안 메커니즘을 제공할 수 있도록 표준화된 보안 인터페이스이다. 따라서 비즈니스 컴포넌트에서 발견과 연동이 쉽고, 플랫폼의 보안 정책에 따라 국산 암호화 알고리즘을 포함한 다양한 암호화 메커니즘의 설정이 가능하다.

4. 기존의 SSL 프로토콜과 SSL 컴포넌트와의 비교

기존 SSL 프로토콜의 단점은 2.2에서 기술한 바와 같이 암호화 과정에서 시스템 부하에 대하여 트랜잭션의 성능이 저하되는 현상이 나타난다는 점이다. 또한 한번 설정된 SSL 프로토콜의 네트워크 상에서는 모든 데이터를 SSL 프로세스에 따라 처리하고 암호화해야 하므로 시간에 따른 오버헤드가 발생하고 CPU의 성능이 높여야 데이터 운영의 속도를 높일 수 있다는 점도 단점이 된다.

그러나 SSL 컴포넌트를 이용하게 되면, 한번 설정한 연결을 계속 사용하는 기존 SSL 프로토콜과는 달리, 선택적으로 데이터에 SSL 서비스를 수행하게 된다. SSL 서비스를 할 때에도 역시 보안 요구사항들을 제공하기 위해서 쓰이는 알고리즘과 키를 개발자의 의도에 따라 신뢰성을 가지고 사용할 수 있다. 뿐만 아니라, 본 논문에서 제시한 SSL 컴포넌트는 다양한 보안 애플리케이션에서 간단하게 재사용될 수 있으며, 기존의 TCP/IP 상에서 전송 계층의 보안에 사용되는 SSL 프로토콜과 동일한 안전성을 보장하면서 컴포넌트의 메시지 전송방식(예: RMI)에 적합한 서비스의 제공이 가능하다는 장점을 갖는다. 이를 사용하여 비즈니스 컴포넌트(DCOM 또는 EJB)에서 표준화된 인터페이스를 이용할 경우, 비즈니스 컴포넌트의 재사용성을 보장할 수 있다.

5. 결론 및 향후 연구 과제

본 논문은 기존 SSL 프로토콜의 단점을 보완하여 암호화 프로토콜의 성능을 높이고, SSL 컴포넌트는 컴포넌트 인터페이스 표준의 활용에 따른 핵심 정보보호 서비스에 대한 호환성을 확보하여 재사용을 방지하기 위해 제안하였다. 특히 개념적으로 이해가 쉬운 보안 컴포넌트를 제안하므로 보안이 필요한 비즈니스 컴포넌트의 개발에 편의성을 제공할 것이다. 그러나 향후 여러 방법에 대한 인증부분의 제안과 설계가 필요하다.

참고 문헌

[1] Matt Bishop, "Computer Security Art and Science", Addison-Wesley, 2002.

[2] Ben Galbraith, et. al., "Professional Web Services Security", Wrox, 2002.

[3] William Stallings, "Cryptography and Network Security - Principles and Practices", Prentice Hall, 2003.

[4] "Scaling e-Commerce Application", NetWorkshop Report, 1999

[5] KISA 기술팀, "The Standardization of Confidentiality and Integrity Service Component Interface", KISA, 2003

[6] John Cheesman, John Daniels, "UML Components : A Simple Process for Specifying Component-Based Software (The Component Software Series)", Addison-Wesley, 2001.