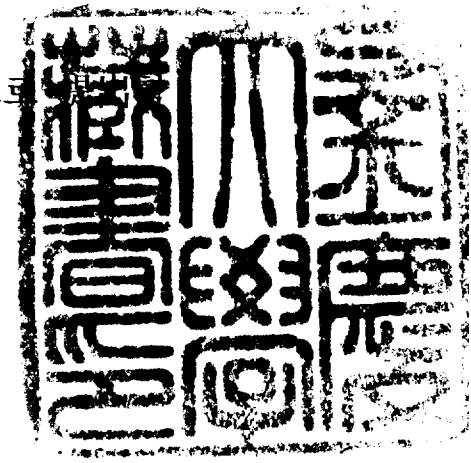
理學碩士 學位論文

웹 서버와 안전한 통신을 위한 SSL 기반의 Web Tunneling 설계 및 구현

指導敎授 金 昌 壽

이 論文을 理學碩士 學位論文으로 제출함



2002年 2月

釜慶大學校 大學院

電 子 計 算 學 科

梁 棟 守

양동수의 이학석사 학위논문을 인준함

2001년 12월 26일

주 심 이학박사 윤 성 대 

위 원 공학박사 정 순 호 (인) 

위 원 공학박사 김 창 수 (인) 

< 차 례 >

< 표 차 례 >

< 그 림 차 례 >

Design and Implementation of Web Tunneling Based on SSL for Secure Communication with Web Server

Dong-Soo Yang

Dept. of Computer Science Graduate School of
Pukyong National University

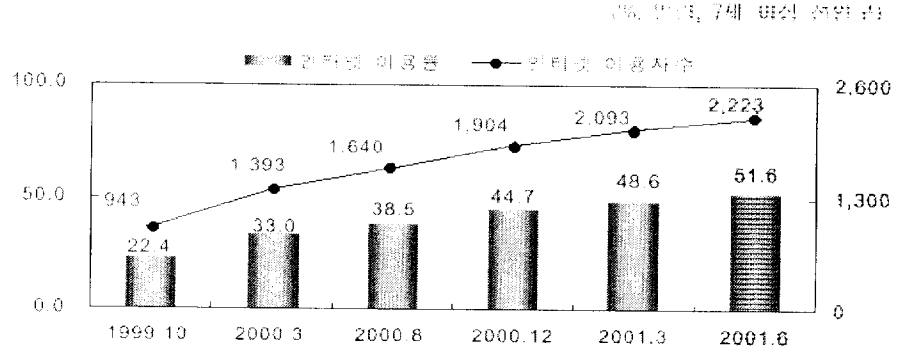
Abstract

As the rapid growth of Internet, the areas of Web-based Internet Banking and Electronic Commerce have been expanding world-widely. Many people want on-line service like to sell something exhibits and sells a variety of goods on the Internet. A well-known fact is that TCP/IP protocol does not provide any security function. So, many researcher have been researched methods to provide security about end-to-end communication based on TCP/IP. SSL(Security Socket Layer) is one of internet security protocols. The security technique in application layer is generally used SSL algorithm can run under application protocols such as HTTP, FTP and TELNET etc.

In this thesis, we design and implement Web Tunneling security module, which can be used on variety operating systems, for secure communication between clients using Web Tunneling technique and server using Apache.

1. 서 론

멀티미디어 정보 처리 기술의 발달과 인터넷 사용 저변이 확대됨에 따라 정보 통신뿐만 아니라 사회 전반에 걸쳐 웹을 이용한 정보 서비스 시스템 구축 및 운영이 활발히 이루어지고 있다. 그림 1은 한국 인터넷 정보 센터(KRNIC)에서 발표한 "2001년 6월말 기준 인터넷 이용자수 설문조사 결과보고서"로서 우리나라 7세 이상 국민의 인터넷 이용률이 50%를 넘으며, 그 수는 2,223만 명에 이르는 것으로 나타났다[5].



시점 / 항목	1999.10	2000. 3	2000. 8	2000.12	2001.3	2001.6	2001.3 대비 증가
이용률	22.4%	33.0%	38.5%	44.7%	48.6%	51.6%	3.0%p
이용자 수	943만 명	1,393만 명	1,640만 명	1,904만 명	2,093만 명	2,223만 명	130만 명

[그림 1] 국내 인터넷 이용률과 인터넷 사용자 수

웹은 일반적으로 분산되어 있는 하이퍼미디어 정보를 검색하기 위한 것으로 인터넷을 이용하여 전 세계에 분산되어 있는 정보를 동적으로 검색할 수 있다. 그러므로 인터넷은 기본적으로 정보를 공유하기 위하여 서로 공개되어 있고, 공개된 정보를 검색하기 위한 것으로 처음에는 보

안에 대한 문제에 대해 고려하지 않았다. 그러나 점차적으로 인터넷의 편리함과 효율성으로 인터넷 뱅킹, 전자우편, 전자게시판, 전자상거래 등 광범위한 분야로 이용이 확대되고 있다. 웹을 통해서 온라인 업무의 편리함을 제공하지만 그 역기능으로 인터넷을 이용한 웹은 제 3자에 의한 중요한 정보의 도청과 변조 등과 같이 도처에 산재한 위험 요소로 인해 경제적인 피해가 속출하고 있다. 그래서 현실적으로 전자상거래를 구현하는데 많은 어려움에 직면하게 되었고, 인터넷을 이용한 전자상거래를 실현하기 위해 보안의 필요성이 대두되었고, 이러한 보안에 대한 연구들이 활발히 이루어지고 있다.

본 논문에서는 서버와 클라이언트 사이에 안전한 통신을 위한 프로그램 구현을 위해 TCP/IP 기반으로 하는 IPSec, SHTTP, SSL등의 인터넷 보안 프로토콜을 분석하고, 현재 대표적인 Web Server로 이용되는 UNIX 기반의 시스템에 대해 알아본다. 그리고 기존의 Web Server와 본 논문에서 구현한 Web Tunneling을 통하여 서로 안전하게 통신을 할 수 있도록 모듈을 작성하였고, Web browser에 Proxy Server 설정 모듈을 작성함으로써 Web Tunneling이 동작할 수 있도록 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서는 Apache Web-Server와 TCP/IP 기반의 인터넷 보안 프로토콜에 대해서 기술하고, 3장에서는 본 논문에서 적용하고자 하는 보안 프로토콜에 대해서 동작 방법, 라이브러리 등을 기술한다. 4장에서는 Apache Web-Server와의 안전하게 통신을 할 수 있는 Web browser와 Web Tunneling 모듈에 대한 구현 환경, 통신 형태, 전체 구성도 등에 대하여 기술하고, 구현 결과를 보여 준다. 마지막으로 5장에서는 결론과 향후 연구 과제를 기술한다.

2. 관련 연구

2.1. Apache Web-Server

Apache는 1995년 NCSA HTTPD 1.3 버전을 기반으로 탄생하였고, 그 후 NCSA 서버에 더욱 향상된 기능들을 탑재하여, Apache Web-Server를 발표하였다. 인터넷의 대중화가 이루어지며 빠른 속도로 증가하고 있는 소프트웨어중의 하나로 가장 많이 사용하는 Web Server이다. 국내에서는 초고속 인터넷 망의 보급화로 많은 개인 사용자들이 이미 자기 자신만의 영역을 사이버 공간에서 만들어 나아가고 있다. 전 세계적으로 많이 사용되어지고 있는 Web Server인 Apache는 사용자들에게 그 기능 및 성능을 이미 인정받아 지금까지 가장 인기 있는 Web Server 소프트웨어로 군림해 오고 있다. 현재 Apache 1.3.X 의 뒤를 이을 Apache 2.0 이 설계 및 개발되어져 왔으며, Apache에 대한 관심은 나날이 높아져 가고 있다.

또한, APR(Apache Portable Run-time)의 기능으로 Apache Web-Server가 좀더 많은 플랫폼에서 쉽게 포팅 되어 질 수 있고, 더불어 관리까지 쉽게 이루어질 수 있도록 도와준다. 그러므로, 현재는 Windows용 플랫폼뿐만 아니라 Mac이나 BeOS 등도 지원하고 있다[2].

2.1.1. Apache-SSL

Apache에서 Terisa사가 개발해서 Netscape에 의해 일반화된 SSL을

이용하기 위해 만들어진 Apache 변형 서버이다. SSLeay라는 UNIX 기반의 SSL 라이브러리를 이용하면서 이 라이브러리와 Apache를 연결해 주는 Apache-SSL interface로 제공되어 왔으며, SSLeay의 단점을 수정 보완한 OpenSSL이라는 package가 등장, Apache-SSL의 근래 모습으로 굳어지게 되었다. 또한, Apache Server와 Apache-SSL interface, SSLeay의 개선 라이브러리인 OpenSSL을 이용한 복잡한 설치과정을 대폭 간소화하여, 최근에 각광받고 있다[2][17].

2.1.2. OpenSSL 라이브러리

OpenSSL은 Eric A. Young에 의해 SSLeay에서 유래한 SSL을 구현한 공개 소프트웨어/라이브러리로써 SSLeay는 SSLv2와 SSLv3을 구현하였고, OpenSSL은 SSLv2와 SSLv3뿐만 아니라 TLSv1까지 구현되어 있다. OpenSSL은 다양한 운영체제(Unix, DOS, Windows 등)에 포팅하기가 쉬우며, SSLeay는 원래 SSL을 구현하기 위한 목적으로 설계되었으므로 일반적인 암호화 기능 구현에는 잘 맞지 않는 경향이 있었으나, 관련 OpenSSL 그룹 등에서의 지속적인 버전 향상에 의해서 많은 기능이 추가되었다. CA 관련 프로그램은 계속적으로 개발함에 따라 다양한 형태의 인증서(PEM, DER, PKCS12, PKCS7 등)를 다룰 수 있으며, smime utility와 인증서 검증 기능 등이 지속적으로 개발되어 추가되고 있다[18].

2.2. HTTP 프로토콜

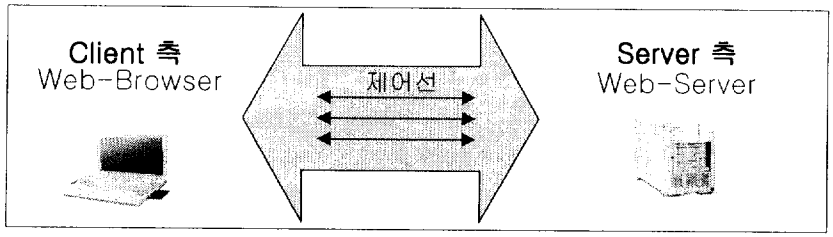
2.2.1. HTTP 프로토콜의 개요

HTTP 프로토콜은 1999년 7월 7일 IETF에 의해 HTTP/1.1 버전이 공식 발표되었다. HTTP는 현재 인터넷 상에서 가장 많이 사용되어 지고 있는 프로토콜 중의 하나이며, 처음 CERN에 근무하고 있었던 Tim Berners-Lee에 의해 제안되었다. 그 후 개발된 HTTP/1.0은 인터넷 상에서 지금까지 널리 사용되어 왔으나, 기하급수적으로 인터넷의 사용이 늘어나면서 웹의 속도 문제가 제기되어 왔다. 그래서 Roy Eielding에 의해 처음 HTTP/1.1이 제안되어졌으며, 이것은 End User에게 좀더 빠른 속도를 제공하며, persistent connections, pipelining, caching, IP address preservation 등을 이용한 무결성, 안정성 등을 제공할 수 있게 되었다. 또 한가지 중요한 점은 HTTP Digest Authentication 기법을 이용하여 HTTP 상에서의 사용자 패스워드가 외부의 누군가에게 노출되는 것 없이 사용자를 인증해 줄 수 있는 방법 등이 정의되어 있으며, 웹 상에서의 보안적인 측면을 강화하였다는 점에서 지금과 같은 전자상거래 시대에서는 중요하지 않을 수가 없다.

2.2.2. HTTP 프로토콜의 동작

HTTP 프로토콜은 전형적인 요구/응답(Request/Response) 방식에 의해 동작한다. 예를 들어 그림 2와 같이 요구(GET, HEAD, POST)에 대해,

서비스 요구를 하면 데이터 송수신을 위한 TCP 연결이 만들어지고, 서버가 응답을 보내 데이터 전송이 끝나면 자동으로 연결이 끊어지는 방식인 것이다[8].



[그림 2] HTTP/1.1 프로토콜 동작 모델

HTTP/1.1 프로토콜 수준에서 하나의 연결에 session이라 불리는 다수의 가상 연결을 지원하는 방식을 도입하게 되었다

2.3. 인터넷 보안 프로토콜

TCP/IP 기반의 안전한 통신을 하기 위한 인터넷 보안 프로토콜로써 IPSec, SHTTP, SSL 등이 있다. IPSec은 일종의 네트워크상의 주소인 IP로서 IP계층의 정보를 유폐하기 위한 보안 프로토콜이며, SHTTP는 EIT(Enterprise Integration Technologies)에서 제안한 HTTP의 Security 확장판으로서 HTTP 세션으로 주고받는 자료를 암호화, 전자서명 등을 지원하는 메커니즘이다. 그리고 전자상거래 보안을 위해 가장 많이 사용하고 있는 SSL은 TCP/IP 계층과 HTTP, NNTP, FTP, TELNET 등의 응용계층 사이에 위치하는 프로토콜로서 다양한 응용 프로그램들이 SSL

를 이용하여 보안기능을 수행할 수 있게 된다[15].

2.3.1. IPSec 프로토콜

IPSec(Internet Protocol Security)은 IETF에 의해서 IP계층 보안을 위한 개방형 구조로 설계되고 있다. 이러한 IPSec은 네트워크 계층의 보안에 대해서 안정적이고 영구적인 기초를 제공한다. IPSec은 오늘날의 암호화 알고리즘을 수용할 수 있을 뿐만 아니라 새로운 알고리즘을 수용할 수 있다. 차세대 인터넷 프로토콜인 IPv6에서는 IPSec을 포함하고 있다. 또한 현재 사용되고 있는 IPv4에서도 IPSec을 사용하는 것이 보안을 위해서 필요하다. IPSec은 인터넷의 기초적인 보안을 튼튼하게 할 수 있는 방법을 제공하며 이 기술은 현재 방화벽이나 VPN 등에 응용되고 있다.

IPsec은 네트워크 계층의 VPN 구현을 위해서 무결성과 인증을 보장하는 AH(Authentication Header)와 암호화된 캡슐화로 기밀성을 보장하는 ESP(Encapsulation Security Payload)를 사용한다. AH는 패킷 안의 데이터를 검증 가능한 서명과 결합시켜 데이터 송신자의 신원을 확인하고 데이터가 변조되지 않았음을 검증할 수 있으며, ESP는 각 payload를 불법적으로 해독할 수 없도록 기밀성을 제공한다. 또한 키의 안전한 관리 메커니즘으로 IEFT의 제안에서는 ISAKMP를 사용한다[6][7][16].

2.3.2. SHTTP 프로토콜

SHTTP(Secure HTTP)는 EIT(Enterprise Integration Technology) 에서

개발한 HTTP 프로토콜의 보안 확장판으로 기존의 HTTP 프로토콜에 보안기능을 갖게 하기 위하여 몇 가지의 헤더를 추가한 것이다. SHTTP는 기본적으로 클라이언트/서버 모델을 바탕으로 트랜잭션 단위로 통신을 하게 되며 종단간에 보안 서비스를 제공하도록 설계되었고, 메시지 무결성(Integrity), 트랜잭션 기밀성(Confidentiality), 발신자 인증(Authentication)과 부인 봉쇄(Non-repudiation) 서비스를 제공한다. SHTTP가 지원하는 메시지 포맷은 PKCS-7, PEM(Privacy Enhanced Mail), PGP(Pretty Good Privacy) 등이 있으며, 암호화 방식으로는 RSA, RC2, ,MD4등의 여러 방식을 이용할 수 있도록 설계되어 있다. 또한 SHTTP는 기본적인 형식은 HTTP를 따르면서 request에 대해서는 Secure HTTP 즉, shttp라는 새로운 형식의 URL이 사용된다. 이와 같이 다양한 암호화 알고리즘, 메시지 포맷, 키 관리 메커니즘 등을 수용할 수 있도록 하는 협상 메커니즘을 지원하지만, SHTTP는 그 용도가 HTTP에 제한되어 있으므로 다양한 종류의 TCP/IP 기반 통신에 적용할 수 없으며, 현재는 거의 사용하지 않고 있다[9].

2.3.3. SSL 프로토콜

SSL(Secure Socket Layer)은 1994년 Netscape사에 의해서 Netscape 웹 브라우저를 위한 보안 프로토콜로 개발한 것으로써 Netscape Navigator에 SSL 2.0이 포함되었다. 그러나 SSL 2.0은 보안 측면에서 많은 약점들을 보완한 SSL 3.0을 Internet Draft로 제안하였다. 이후에 SSL 3.0을 수정 보완하여 표준화하는 작업이 진행되어서 1999년

TLS(Transport Layer Security)로 이름을 바꾸어 표준화하였다[11].

SSL은 클라이언트와 서버의 환경으로 동작하는 Web browser와 Web Server가 각각 클라이언트와 서버의 역할을 한다. 클라이언트와 서버간의 교환되는 데이터를 안전하게 보호하기 위하여 데이터의 암호화 및 서버 인증, 메시지 무결성을 제공하며 선택적으로 클라이언트에 대한 인증도 지원한다[4].

SSL은 하나의 프로토콜로 이루어진 것이 아니라, 두 계층으로 나누어져 있다. 먼저 클라이언트와 서버가 SSL를 이용해 연결을 할 경우 Handshake Protocol을 수행하여 한 세션동안 보안 서비스 제공에 사용되는 세션 키, 암호 알고리즘 등과 같은 암호 매개변수를 서로 공유하게 됨으로써 상대방을 인증하고, MD5 SHA-1 등과 같은 해쉬 함수를 이용하여 생성한 MAC(Message Authentication Code)으로 메시지 무결성을 검사한다. 여기서 생성된 세션정보는 Record Protocol에서 보안 서비스를 제공하는데 이용된다. 그러므로 SSL은 TCP/IP 기반으로 HTTP 프로토콜뿐만 아니라 다양한 응용 계층의 프로그램들에 보안 기능을 제공할 수 있으며, 선택적으로 클라이언트에 대한 인증기능을 제공하고 있으므로 전자상거래에 필요한 부인방지 기능을 제공할 수 있다[15]. 그러나 인터넷을 통한 전자상거래에서는 필수적으로 보안이 필요하지만, 일부 인터넷 쇼핑몰의 경우 회원정보와 신용카드 번호조차 보호되지 않는 등 해킹에 취약한 것으로 드러났다. 그러므로 개인정보가 쉽게 유출될 수 있어 보안 시스템의 당위성을 느끼게 한다. 그래서 쇼핑몰 업체들이 필수적으로 갖추는 보안 시스템이 SSL 프로토콜이다. 그러나 현재 SSL의 경우 암호화 기법 중 40bits와 128bits 암호화 기법이 사용되고 있으나

40비트 암호화 기술은 암호 해독이 가능한 것으로 알려져 있어 이보다 안전한 128비트의 SSL 시스템을 구축해야만 한다. 표 1은 경향신문과 한국 전자 거래 진흥원에서 공동 조사한 "인터넷 쇼핑몰 체험진단"에 의한 쇼핑몰 주요 보안점검 사항을 나타낸 것이다.

[표 1] 쇼핑몰 주요 보안점검 사항

국내 인터넷 쇼핑몰	사용 인증서 시스템
LG 홈쇼핑	40bits
롯데닷컴	40bits
마트24	40bits
바이엔조이	40bits
삼성몰	40bits
인터파크	40bits
한솔CS클럽	128bits
해피투바이	128bits

쇼핑몰의 보안 점검의 결과 중에서 한솔CS클럽와 해피투바이 2곳만이 128비트의 SSL을 구동하고 있었다. 이 결과 개인정보, 신용정보 등을 입력하거나 전송하는 곳에서는 반드시 128bits SSL 시스템을 사용해야 한다[1].

본 논문에서는 Apache Web-Server와의 안전한 통신을 위해 TCP/IP 환경에서 가장 많이 이용하고, 폭넓은 보안 기능을 제공하는 SSL 프로토콜을 이용한다. 따라서 웹 서버와 안전한 통신을 위한 보안 프로토콜 기반으로 SSL 프로토콜을 이용하여 Web Tunneling을 구현하였다.

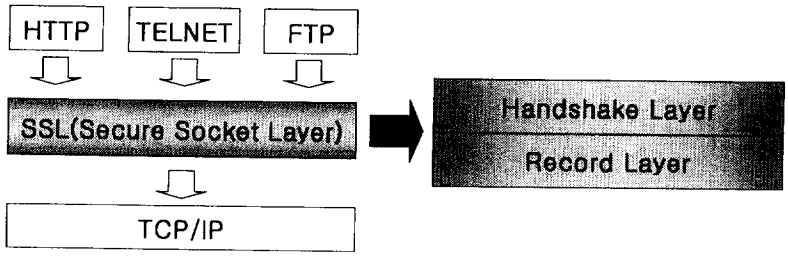
3. SSL 프로토콜 분석

SSL(Secure Socket Layer) 프로토콜은 Netscape사에서 발표된 이후 전송 계층에서의 안전한 통신을 위한 사실상의 표준 프로토콜로서 자리를 잡고 있다. 1996년 3월 SSL Version 3을 발표한 보안 프로토콜로 통신을 하는 양측에 인증 및 기밀성을 제공하였고, 현재 IETF(Internet Engineering Task Force)에서 TLS(Transport Layer Security)로 개선되어 표준화되었다. SSL 프로토콜은 클라이언트와 서버의 환경으로 동작하는 프로토콜로서 대표적인 인터넷 보안 프로토콜로 인식되고 있다. 그리고 Web browser인 인터넷 익스플로러와 넷스케이프에 기본으로 포함되어 현재 보안이 필요한 전자상거래, 인터넷 뱅킹 등에서 가장 많이 사용하고 있는 인터넷 보안 프로토콜이다.

SSL 프로토콜은 클라이언트와 서버사이에 교환되는 데이터를 안전하게 보호하기 위하여 인증을 하고, 비밀키 암호(DES, 3DES, RC4)를 사용하여 데이터를 보호한다. 이때, 사용되는 비밀키는 키 교환 알고리즘으로 교환된 secret로 유도되고, 데이터의 위/변조를 막기 위하여 MAC (Message Authentication Code)를 사용한다[10][11].

그림 3에서 SSL은 TCP/IP 계층과 HTTP, TELNET, FTP 등과 같은 어플리케이션 계층 사이에 위치하며, 클라이언트와 서버간의 종단간 보안 서비스를 제공한다. 세부적으로는 Handshake 계층과 Record 계층으로 구성되어 있다[3]. Handshake 계층은 두 계층 중 상위에 위치하는 계층으로 서버와 클라이언트간 통신을 수행하기 전에 상호 인증하고, 암호화된 통신을 위해 키 교환을 수행한다. Record 계층은 Handshake 계층

의 하부에 위치하며 Handshake 계층의 데이터를 받아서 캡슐화(encapsulation) 즉, 데이터를 암호화하는 기능을 담당한다[16]. SSL의 특징은 어플리케이션 프로토콜에 독립적으로 수행함으로서 기존의 프로토콜에 대한 영향을 최소한으로 할 수 있다.



[그림 3] SSL 계층 구조

SSL은 기밀성, 무결성, 사용자 인증, 부인봉쇄의 보안서비스를 다음과 같이 제공한다[11].

① 두 응용간의 기밀성 서비스

DES, RC4등과 같은 대칭키 암호화 알고리즘을 사용하여 제공되며, 이때 사용되는 비밀키는 Handshake 계층에서 생성된다.

② 메시지 무결성 서비스

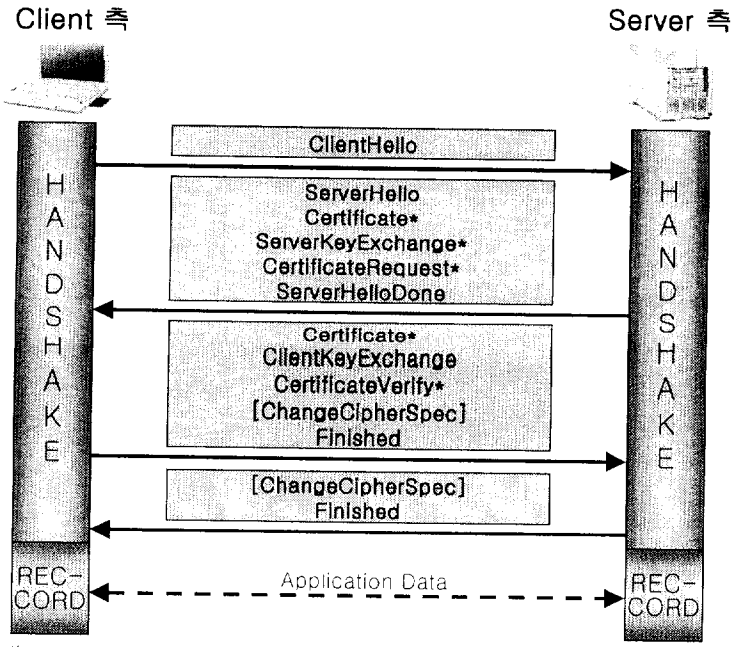
MAC(Message Authentication Code) 기법을 이용하여 데이터의 무결성을 제공함으로서 데이터 변조 여부를 확인할 수 있다.

③ 클라이언트와 서버의 상호 인증

연결 설정 과정에서 서로간에 신뢰할 수 있도록 클라이언트와 서버간에 인증을 할 수 있도록 하는데, RSA와 같은 비대칭키 암호 알고리즘, DSS와 같은 서명 알고리즘 등을 사용된다.

3.1. Handshake 프로토콜

SSL Handshake 계층은 한 세션 동안 이용되는 암호 매개변수는 SSL Handshake Protocol에 의해서 생성되는데, 한 세션에서 사용되는 비밀 정보를 공유하기 위해 이용된다. SSL 클라이언트와 서버가 처음으로 통신을 시작할 때 서로 프로토콜 버전을 확인하고, 암호 알고리즘을 선택하고, 선택적으로 서로를 인증하고, 공유할 비밀 정보를 생성하기 위해 공개키 암호 기법을 사용한다[13]. 이러한 처리는 그림 4와 같은 Handshake Protocol에서 수행되고, 표 2는 Handshake 계층과 Record 계층에서 사용되는 기본적인 암호화 알고리즘의 조합이다[11][13].



[그림 4] Handshake 프로토콜 수행과정

[표 2] SSL/TLS Cipher Suites

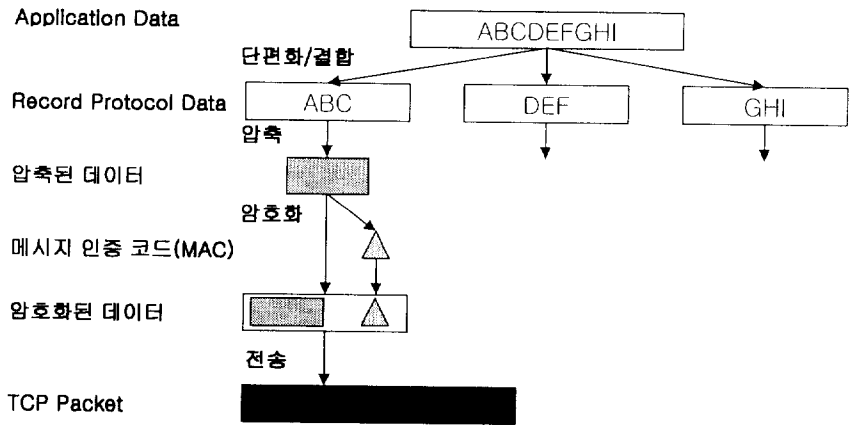
CipherSuite	Key Exchange	Encryption	Digest
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5	RSA_EXPORT	RC4_40	MD5
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA_EXPORT	RC2_CBC_40	MD5
TLS_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA_CBC	SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA_EXPORT	DES40_CBC	SHA
TLS_RSA_WITH_DES_CBC_SHA	RSA	DES_CBC	SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	DH_DSS_EXPORT	DES40_CBC	SHA
TLS_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES_CBC	SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	DES_EDE_CBC	SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	DH_RSA_EXPORT	DES40_CBC	SHA
TLS_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES_CBC	SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	EDE_CBC	SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DHE_DSS_EXPORT	DES40_CBC	SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES_CBC	SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE_CBC	SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	DHE_RSA_EXPORT	DES40_CBC	SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES_CBC	SHA
TLS_DHE_RSA_WITH_C_SHA	DHE_RSA	3DES_EDE_CBC	SHA
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5	DH_anon_EXPORT	RC4_40	MD5
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA	DH_anon_EXPORT	DES40_CBC	SHA
TLS_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES_CBC	SHA
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA

Handshake의 동작에서 클라이언트가 ClientHello Message를 서버에게 보내면 서버는 반드시 Response를 해야 하고 Response를 하지 않을 경우 fatal error를 일으켜 서버와 접속을 할 수 없다. 클라이언트와 서버의 Hello Message에 포함되어 있는 내용은 Protocol Version, Session ID, Cipher Suites, Compression Method 등으로 구성되어 있다. 그리고 인증서 및 키 교환은 클라이언트와 서버의 Hello Message가 교환된 후, 서

버의 인증이 필요하면 서버의 인증서가 클라이언트에 보내지고, 클라이언트는 비밀키를 생성하여 서버의 공개키를 사용해서 암호화하여 서버로 전송한다. 서버의 인증서를 보낸 경우에 서버는 클라이언트에게 인증서를 요청할 수 있다. 그러면 서버는 ServerHelloDone을 보내어 ServerHello Message가 끝났음을 알리고 클라이언트의 응답을 기다린다. 서버가 클라이언트의 인증서를 요청하였으면 클라이언트는 반드시 Certificate를 전송하여야 한다. 그리고 암호화 알고리즘의 선택과 인증서 및 키 교환이 모두 끝나면 클라이언트와 서버는 ChangeCipherSpec 프로토콜을 수행한 뒤 Finished를 보내고, 서버와 클라이언트간의 암호화된 데이터를 전송하기 위해서 Record 계층으로 전달된다[10][12][14].

3.2. Record 프로토콜

Handshake 계층 단계에서 암호 알고리즘, 키 등의 정보를 협상에 의해서 클라이언트와 서버가 공유한 후, 한 세션동안 그 정보는 Record Layer에서 계속적으로 유지된다. 상위 계층으로부터 Record 계층으로 전달된 가변적인 크기의 데이터를 SSL에서 처리할 수 있는 데이터 블록 크기만큼씩 단편화(Fragmentation)하고, 단편화된 데이터를 경우에 따라서 압축을 한 후 MAC와 암호화를 계산하여 암호화된 데이터를 TCP 계층으로 전달된다. 이러한 데이터를 수신하면 복호화하고, MAC에 대한 무결성을 검증한 후 압축을 해제하고, 단편화된 데이터 조각을 모은다. 그리고 이렇게 처리된 평문 데이터를 응용 계층으로 전달하게 된다. 그림 5는 일련의 Record Protocol 처리 과정을 나타낸 것이다[10][15].



[그림 5] Record Protocol

3.3. Change Cipher Spec 프로토콜

Change Cipher Spec 메시지는 값이 1인 1바이트로 이루어져 있으며, Record에 대해 CipherSpec에 정의된 알고리즘과 키를 이용해 보호될 수 있도록 수신측에 알리기 위해 클라이언트와 서버 모두가 보낼 수 있다. 예기치 않은 Change Cipher Spec 메시지에 대해서 unexpected_message alert 메시지를 보낸다.

3.4. Alert 프로토콜

Alert 메시지는 압축 및 암호화 오류, MAC 오류, Handshake Protocol 실패, 인증서 오류 등에 대한 메시지와 설명을 전송하는데 이용된다.

3.5. SSL 프로토콜을 이용한 기존의 보안제품

지금까지 대부분의 전자상거래는 적용하기 쉬운 SSL 방식을 채용하여 고객의 개인정보나 신용정보를 안전하게 전송하고 있다. SSL 방식의 보안 제품으로서는 상거래 서비스를 제공하는 기관에서 운영하는 서버 버전과 사용자가 설치, 운영하는 클라이언트 버전이 있다. 그리고 클라이언트 측면에서 보면 대부분의 기존 보안제품이 Windows 환경에서 제작되어 있다. 다음은 기존의 SSL을 이용한 보안 제품들이다.

① J/SSLcok

시큐리티 테크놀로지사의 J/SSLock은 Netscape사의 SSLv3.0에 따라 구현되어 인터넷의 통신채널에 보안성을 제공해 주는 소프트웨어로서 100% 자바로 구현되어 Cross-Platform을 지원하는 J/SSLock을 사용하여 강력한 128bits의 SSL통신이 가능하다[19].

② XecureProxy

소프트포럼사의 XecureProxy는 SSL Proxy Tunneling 방식을 활용하여 웹상의 컨텐츠를 암호화하여 안전하게 전달하는 PKI 암호화 솔루션으로 보다 폭넓은 범위에서 안전한 e-business 환경을 제공한다[20].

③ INIssl

이니텍사의 INIssl은 Client PC상의 Web브라우저와 Web서버간에 전송되는 데이터를 암호화하는 보안솔루션으로 INIssl-128은 한국 표준 암호알고리즘 SEED등 국산암호기술을 지원하는 K-SSL 지원방식의 암호화 모듈 제품으로 Web Sever와 PC간의 정보를 안전하게 교환한다[21].

4. SSL 기반의 Web Tunneling 설계 및 구현

4.1. 구현환경

본 논문에서 제안한 Web Tunneling 프로그램은 Apache Web-Server와 TCP/IP 기반의 HTTP 통신과 HTTPS 통신을 하기 위해 OS 플랫폼과는 독립적으로 수행할 수 있도록 설계 및 구현하였다. Web Tunneling 모듈을 동작하기 위해 시스템 환경은 크게 서버 측과 클라이언트 측으로 나눌 수 있다. 그리하여 서버 측은 기존에 사용하고 있는 Apache Web-Server에 OpenSSL Library를 패치한 Apache-SSL를 구축하여 안전한 128bits 통신을 할 수 있도록 설정하고, 클라이언트 측은 기존의 Web browser에서 제공하는 Proxy Server 설정을 하기 위해 MFC를 이용하여 Web browser를 구현하고, Proxy Server 설정 모듈을 Web browser 프로그램 내에 추가함으로서 Web Tunneling 프로그램이 동작할 수 있도록 되어 있다. Web Tunneling이 동작하기 위한 시스템 환경은 다음과 같다.

① OS Platform : Windows 98, Windows NT, Linux, Unix
② SSL Library : OpenSSL Library
③ Tool : Perl 5, Visual C++, C Language, gcc(Linux)

그리하여 서버와 클라이언트 사이에 안전한 128bits 통신을 할 수 있도록 SSL 프로토콜을 이용한 Web Tunneling 환경이 구축된다.

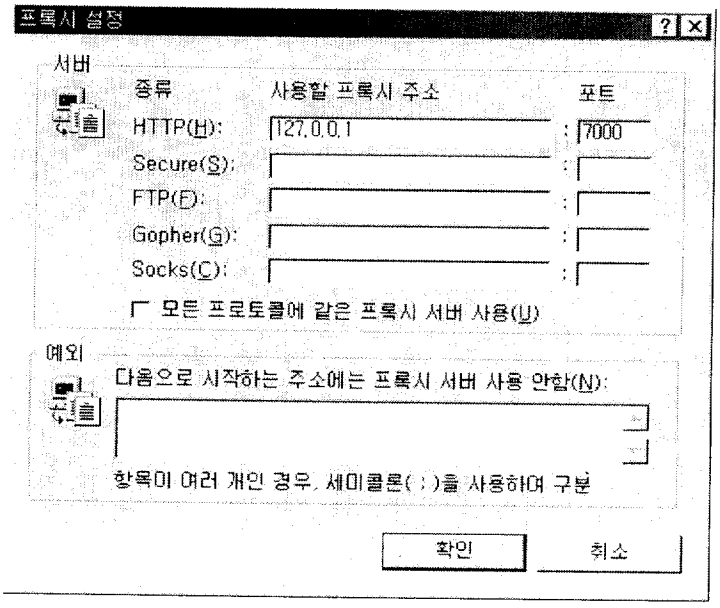
4.2. Web browser의 Proxy Server 설정

본 논문에서 구현한 Web Tunneling을 실행하기 위해 클라이언트 측은 Proxy Server를 설정하는 모듈을 포함한 Web browser를 구현하였고, 이를 실행시킴으로서 HTTP와 HTTPS 통신이 동작하도록 연동시켜 준다. 그림 6은 Windows 환경 하에서 Web browser를 동작시켰을 때 레지스트리의 Proxy Server 부분을 설정 및 해제하는 루틴이다.

```
// Proxy Server 설정 루틴
RegSetValueEx (hKey,
               "ProxyServer",
               0,
               REG_SZ,
               (CONST BYTE *)"http=127.0.0.1:7000",
               20);
int binary = 16;
RegSetValueEx (hKey,
               "ProxyEnable",
               0,
               REG_BINARY,
               (CONST BYTE *)&binary, 4);
RegCloseKey(hKey);
                              ...
// Proxy Server 해제 루틴
int binary = 0;
RegSetValueEx (hKey,
               "ProxyEnable",
               0,
               REG_BINARY,
               (CONST BYTE *)&binary, 4);
RegCloseKey(hKey);
```

[그림 6] Proxy Server 설정 및 해제 루틴

그림 7은 그림 6에서 나타난 Web browser의 Proxy Server 설정을 했을 경우 다음과 같이 설정이 된다.



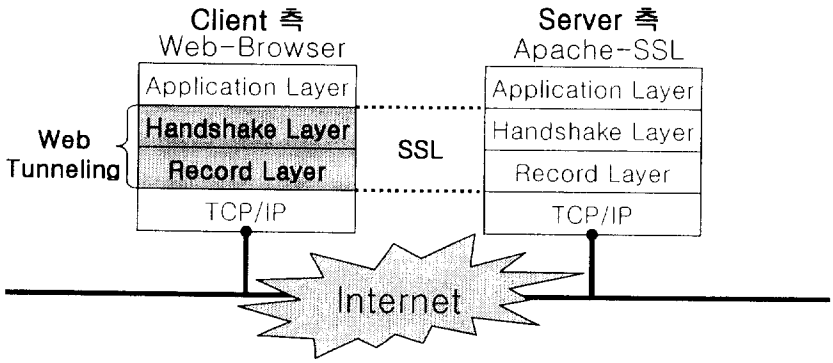
[그림 7] Proxy Server 설정

4.3. SSL 기반의 Web Tunneling의 설계

4.3.1. Web Tunneling 구성

본 논문에서는 SSL 기반의 Handshake 계층과 Record 계층을 구현한 Web Tunneling 프로그램은 클라이언트 측에 설치하여 서버 측인 Apache-SSL과 안전한 통신이 가능하도록 구성되어 있다. 그림 8은 본 논문에서 구현한 클라이언트 측의 Web Tunneling과 서버 측의

Apache-SSL과 통신하는 구성도이다.



[그림 8] Web Tunneling 구성도

그림 8의 구성도에서 클라이언트 측의 응용 계층과 TCP/IP 계층 사이에 위치한 SSL의 Handshake 계층과 Record 계층을 구현한 Web Tunneling 프로그램이 추가되어 있다. SSL 프로토콜 기반의 Web Tunneling 프로그램이 통신하기 위해서 첫째, Handshake 계층에서는 서버와 클라이언트간 통신을 수행하기 전에 상호 인증하고, 암호화된 통신을 위해 키 교환을 수행한다. 둘째, Handshake 계층에서 전달된 데이터를 암호화하고 무결성 검증 코드를 추가하여 TCP/IP 계층으로 전달한다. 그리하여 Web Tunneling을 포함한 클라이언트와 Apache Web-Server에 OpenSSL을 패치한 Apache-SSL 서버간의 통신이 실제적으로 네트워크 상에서 이루어진다.

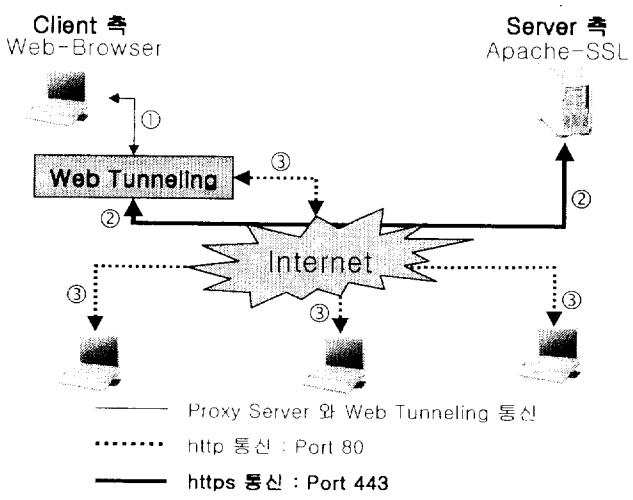
4.3.2. Web Tunneling의 HTTP 통신과 HTTPS 통신 형태

일반적인 인터넷에서 가장 많이 사용하는 프로토콜로서 TCP/IP 기반의 HTTP 프로토콜이 대표적이다. HTTP 프로토콜에 OpenSSL Library가 적용된 Web Tunneling의 프로그램을 추가함으로서 well-known port 80을 사용하는 HTTP 통신과 port 443을 사용하는 HTTPS 통신이 가능하도록 구성되어 있다. 본 논문에서는 인터넷 익스플로러의 Proxy Server 설정을 하기 위해 MFC를 이용하여 구현한 Web browser를 클라이언트에 사용하였고, 서버 측은 OpenSSL 0.9.6과 Apache 1.3.12를 패치함으로서 Apache-SSL을 구축하여 사용하였다.

일반적으로 Desktop에서 가장 많이 사용하는 Web browser인 MS사의 인터넷 익스플로러 경우 보안 강도가 약해 해킹의 우려가 높은 40bits를 아직 많이 사용하고 있어 문제시되고 있으며, Apache-SSL을 패치한 서버와 128bits 통신을 하기가 어렵다. 이러한 문제점을 해결하기 위해 본 논문에서 제안한 SSL 프로토콜 기반의 Web Tunneling 프로그램을 설치함에 따라 Web browser 자체의 보안 강도와는 상관없이 독립적으로 강력한 128bits의 암호 기능을 제공함으로서 클라이언트와 서버간의 안전한 통신을 할 수 있다. 그림 9는 Web Tunneling 프로그램을 설치하여 HTTP 통신과 HTTPS 통신 형태를 나타낸 것이다.

그림 9에서 클라이언트 측의 Web browser에서 서버 측과 port 443을 사용하여 HTTPS 통신을 하거나 다른 인터넷 사이트와 port 80을 사용하여 통신을 하기 위해서 항상 Web Tunneling을 통과하도록 되어 있다. 클라이언트의 Web browser의 Proxy Server를 세팅함으로서 address는 127.0.0.1, port는 7000으로 설정하여 Web Tunneling과 통신을 하게 되

고, 이러한 Web Tunneling을 통하여 HTTP 통신 또는 HTTPS 통신을 하게 된다.



[그림 9] HTTP 통신과 HTTPS 통신 형태

(1) HTTP 통신

TCP/IP 기반의 HTTP 프로토콜의 통신을 하기 위해서 클라이언트 측 Web browser의 URL 주소에 http://~로 접속 요청을 하게 된다. 그림 9에서 Web browser↔①↔Web Tunneling↔③↔Internet의 경로를 통해 다른 인터넷 사이트에 접속을 할 수 있다. 그리하여 인터넷 상에는 보안 기능이 제공되지 않는 port 80을 사용하여 통신을 하게 된다. port는 7000(①), 80(③)번을 사용하게 된다. 그러므로 정보를 공유하기 위해 개방되어 있는 인터넷 상에 클라이언트와 서버 사이의 해킹 툴인 sniffer를 사용할 경우 서로 간의 통신 정보를 쉽게 얻을 수 있으며, 이렇게 알게 된 개인 정보는 또 다른 범죄를 유발하게 된다.

(2) HTTPS 통신

일반적으로 HTTPS 통신을 하기 위해서 URL 주소를 https://~로 접속을 요청을 하게 된다. 그러나 본 논문에서는 Web browser에서 http://~로 접속하더라도 address를 Web Tunneling에서 식별하여 해당 Apache-SSL과 안전한 SSL 통신이 가능하도록 되어 있다. 그림 9에서 Web browser↔①↔Web Tunneling↔②↔Internet↔②↔Apache-SSL의 경로로 통신을 하게 된다. ②에서 사용되는 port는 SSL 통신을 할 수 있는 443번을 사용한다. 그리하여 클라이언트 측의 Web Tunneling과 서버 측의 Apache-SSL은 인터넷을 통하여 SSL2, SSL3, TLS1 프로토콜의 사용으로 암호화된 통신을 하게 된다. 이러한 통신을 하기 위하여 Web Tunneling에서 설정된 address와 Web browser의 입력 URL address를 비교할 수 있도록 HTTP header의 reassemble 과정이 필요하며, Web Tunneling에서는 address를 식별을 할 수 있는 모듈이 필요하게 된다. 이렇게 암호화된 통신을 할 경우에는 클라이언트와 서버간의 도청을 하더라도 패킷 내용은 식별할 수가 없다.

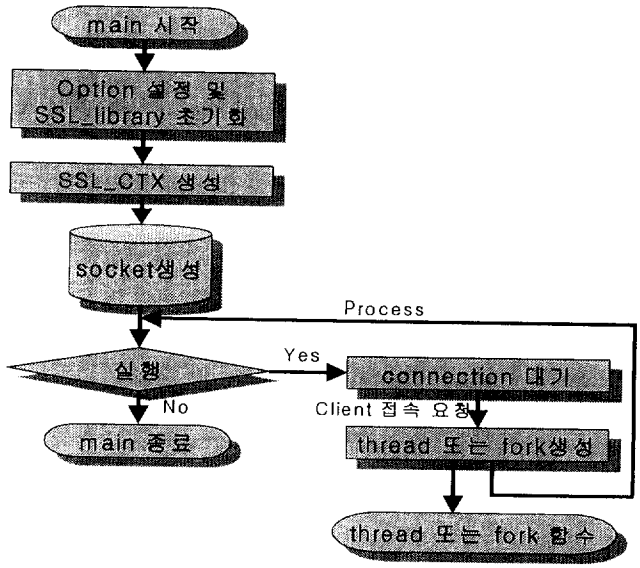
4.4. SSL 기반의 Web Tunneling 구현

본 논문에서 구현한 SSL 기반의 Web Tunneling은 OpenSSL 라이브러리를 사용하여 암호화된 통신을 제공하고 있다. Web Tunneling을 동작을 하기 위해서 우선 OpenSSL Library를 컴파일 하여 현재의 구현되어 있는 OS 플랫폼에 맞게 포팅을 하게 된다. 클라이언트 측의 Web Tunneling을 통해 서버 측인 Apache-SSL과 통신을 할 경우 OpenSSL

Library를 호출하여 사용하게 된다. Web Tunneling의 구현된 프로그램은 OpenSSL에서 제공하는 SSL Library, Crypto Library 등을 사용하였다.

4.4.1. Web Tunneling의 전체 구조

Web Tunneling 프로그램은 Web browser와 통신을 하기 위해서 소켓을 생성하여 연결을 설정한 후, Web Tunneling에서 thread 함수 또는 fork 함수를 이용하여 다중의 접속을 처리하기 위하여 구현되어 있다. 그리하여 다중 접속을 요청할 때마다 소켓에서 thread 또는 fork를 생성하여 통신이 가능하게 되어 있다. 실제로 SSL 기반의 통신을 하기 위해서는 main 함수를 작성하여 세부적으로 SSL 관련 함수를 이용하여 초기화, Handshake 처리, Record 처리 등을 구현하였다.



[그림 10] main 함수의 전체 구성도

그림 10은 main 함수를 토대로 하여 Web Tunneling이 Apache-SSL과 통신을 하기 위한 전체적인 구조를 나타낸 것이다.

main 함수가 시작하면 가장 먼저 SSL 기반의 통신을 하도록 SSL_library를 초기화시키고 SSL connection을 가능하게 하기 위해 SSL_CTX 함수를 생성하게 된다. 그리고 통신을 하기 위해 소켓을 생성하여 다른 다중 작업의 요청이 있으면 연결 대기 상태에서 thread 또는 fork를 생성하여 실행할 수 있도록 되어 있다. 이렇게 생성된 thread 또는 fork는 한번의 접속 요청을 처리하고 read data가 없을 경우 바로 소멸된다. 그리고 Web Tunneling을 실행하는데 있어 SSL을 사용하기 위해 명령어 프롬프트 상에서 입력되는 실행 명령어와 여러 옵션들을 분리하여 client address, client port, server address, server port, protocol 등 저장할 수 있는 부분이 포함되어 있다.

4.4.2. SSL Library 초기화 및 Option 설정 과정

SSL 기반의 Web Tunneling을 사용하기 위해서는 OpenSSL Library의 초기화가 반드시 필요하므로 OpenSSL에서 제공하는 기본적인 함수를 사용해야 한다. 그리고 Web Tunneling이 동작하기 위해 client address, client port, server address, server port 등의 기본적인 옵션 설정이 필요하다. 그림 11은 OpenSSL Library를 제공하기 위해 기본적인 함수를 이용한 초기화를 나타내고, Web Tunneling에 사용될 기본적인 옵션들을 나타낸 것이다.

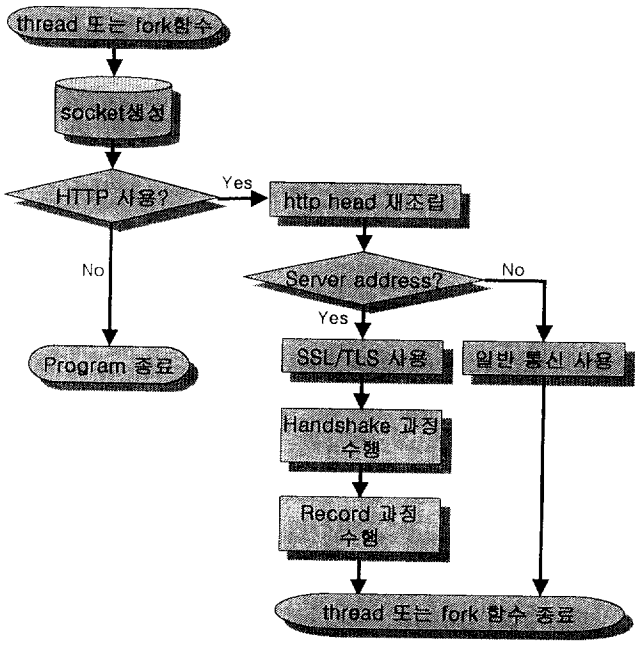
```
/* OpenSSL Library를 사용하기 위한 기본적인 함수 */
SSL_load_error_strings();
SSL_library_init();
SSLeay_add_ssl_algorithms();
                                ...
/* Web Tunneling에 사용될 기본적인 옵션들 */
struct option{
                unsigned clientIp;
                int clientPort;
                unsigned serverIp;
                int serverPort;
                int http;
};
```

[그림 11] Library 초기화 및 Web Tunneling 기본 옵션들

그림 11에서 나타난 OpenSSL를 사용하기 위한 기본적인 함수 중에서 SSL_library_init()를 이용하여 SSL 통신을 하기 전에 초기화하여야 하고, SSL_load_error_strings()를 이용하여 libcrypto, libssl 함수 등의 error string들을 기록한다. 기본 옵션들 중에서 clientIp와 clientPort의 경우에는 Web Tunneling과 연결하기 위해 사용되는 Web browser의 Proxy Server에 설정된 address와 port로 내부적인 통신을 하기 위해 사용되고, serverIp와 serverPort는 접속할 server address와 server port를 나타낸 것이다. 그리고, http 옵션 설정은 일반 HTTP 통신을 할 것인지 SSL 기반의 암호화된 HTTPS 통신을 할 것인지 결정하기 위해서 사용되어진다.

4.4.3. HTTP 통신과 HTTPS 통신의 처리 과정

그림 12는 본 논문에서 구현한 Web Tunneling을 통하여 HTTP 통신할 것인지 또는 HTTPS 통신을 할 것인지를 결정하는 전체 처리 과정을 나타낸 것이다.



[그림 12] 다중 작업의 전체 처리 과정

main 함수에서 다중 작업의 필요할 경우 thread 또는 fork함수를 호출하면 socket 번호와 해당 정보를 전달하게 되고, thread 또는 fork 함수는 서버와 통신을 할 수 있는 또 다른 socket을 생성하게 된다. HTTP header 재조립은 Web Tunneling을 통해서 내려오는 Web browser의

Request를 분석하여 접속할 호스트의 address와 port를 알아내고 Web browser에서 내려온 HTTP header의 구조를 접속할 Web server가 인식할 수 있는 형태로 수정하는 역할을 수행한다.

그림 13은 다중 작업을 처리하기 위해 사용되는 thread와 fork 함수를 처리하는 과정을 나타낸 것이다.

```
#ifdef USE_WINDOWS    // Windows 환경에서 다중작업 처리
  threadParameter = (struct thread_parameter*) malloc
                                (sizeof(struct  thread_parameter));
  threadParameter->clientFd = clientFd;
  threadParameter->option = optionInit;
  CreateThread(0, 0, client_thread, threadParameter, 0, &dummy);
#else                         // Unix, Linux 환경에서 다중작업 처리
  if((child_pid=fork()) == 0){        // 자식 프로세스의 처리과정
      if( fork() != 0 ) exit(0);
        if(setpgid( getpid(), getpid() ) == -1)
           exit(1);
        close(sockFd);
        client(clientFd, optionInit);
        exit(0);
    }else{                            // 부모 프로세스의 처리과정
        close(clientFd);
        waitpid(child_pid, NULL, 0);
    }
#endif
```

[그림 13] 다중작업 처리를 위한 함수 처리 과정

4.4.3.1. HTTP header의 재조립 과정

본 논문에서 구현한 Web Tunneling 프로그램은 Web browser에서 전달된 HTTP header의 내용을 Web server가 인식할 수 있는 형태로 바꾸기 위해 HTTP header의 재조립의 과정이 필요하다. 이러한 처리를 하기 위해서 Web Tunneling으로 내려온 HTTP header의 내용을 버퍼에 저장하게 된다. 이렇게 저장된 HTTP header의 내용을 재조립하는 함수인 http_Reassemble 함수로 인자를 넘겨주게 되면 HTTP header의 명령(PUT, GET 등), address, port 등으로 분류하여 저장하게 된다. 다음의 그림 14에서 http_Reassemble 함수는 함수의 인자로 header의 내용과 리턴 값을 포인터로 전달받아 header의 내용을 분석하여 특정한 기준으로 HTTP header의 명령어와 address, port 등으로 구분하여 저장하게 된다. 그리고 http://~일 경우에만 수행이 가능하도록 되어 있다. 그 결과 구분된 address와 port를 인자로 반환하면 반환된 address와 Web Tunneling에 설정한 address를 비교하여 port 443 통신과 port 80 통신으로 구분하게 되어 있다.

```
int http_Reassemble(char *string_Total, char *address, int *port)
{
        char buffer[2048], command[512], protocol[128];
        char *address_Point, *blank_Point;
        char string_Tail[2048], string_Middle[256];
                                ...
        blank_Point = string_Total;
/* HTTP header 분류 */
        for(count=0; count<2 ;){
                if(*blank_Point == ' '){
                        count++;
                        if(count == 1)
                                temporary = blank_Point;
                }
                blank_Point++;
        }
                                ...
/* http://~ 주소를 address_Point에 저장 */
        address_Point = strstr(buffer, HTTP);
        if(address_Point)               /* http://~일 때 수행 */
        {
                address_Point += strlen(HTTP);
                for(k=0; k<strlen(address_Point); k++)
                        if(*(address_Point+k) == '/') break;
                strcpy(string_Middle, address_Point+k);
                                ...
```

[그림 14] http_Reassemble 함수

4.4.3.2. 통신 형태를 구분하는 과정

http_Reassemble의 처리 과정에서 얻어진 address와 port를 이용하여 HTTP 통신과 HTTPS 통신을 구분하는 과정이다. 그림 15에서는 Web browser에서 입력한 address와 Web Tunneling에서 설정된 address가 같은지를 비교하여 Port로서 통신 형태를 구분할 수 있다.

```
/* HTTP 통신과 HTTPS 통신을 구분하기 위한 모듈로서
   Web browser address와 Web Tunneling address 비교 */

/* HTTPS 통신 */
   if(httpString.serverad.sin_addr.s_addr ==
                                    THREADP option->serverIp){
      sockAddr.sin_family = AF_INET;
      sockAddr.sin_addr.s_addr=httpString.serverad.sin_addr.s_addr;
      sockAddr.sin_port=
                    htons((short) THREADP option->serverPort);
   }
/* HTTP 통신 */
   else{
      sockAddr.sin_family = AF_INET;
      sockAddr.sin_addr.s_addr=httpString.serverad.sin_addr.s_addr;
      sockAddr.sin_port = htons(80);
   }
}
```

[그림 15] HTTP 통신과 HTTPS 통신 구분하는 과정

4.4.4. SSL 기반의 Handshake 및 Record 처리 과정

그림 16의 Handshake 과정에서는 암호화된 통신을 하기 위해 SSL 기반의 통신을 하게될 서버 측에 connection을 요청하여 클라이언트와 서버 사이에 상호 인증하고, 서로 간의 협상된 암호화 형태를 보여 주게 된다. SSL_new 함수는 연결하기 위해 새로운 SSL 구조체를 생성하는 함수이며, SSL_connect 함수에서는 접속할 Server와 Handshake 하기 위해 시작하는 함수이다.

```
if((ssl = SSL_new(sslContext)) == NULL){
        error = ERR_get_error();
        fprintf(stderr, "Error Allocating Handle: %s\n",
                                ERR_error_string(error, NULL));
        return -1;
}
        SSL_set_fd(ssl, descript->fd);
        fprintf(stderr, "Handshake Processing Start !!!\n");
if(SSL_connect(ssl) <= 0){
        error = ERR_get_error();
        fprintf(stderr, "Error Connecting Socket: %s\n",
                                ERR_error_string(error, NULL));
        return -1;
}
      fprintf(stderr, "Negotiated Cipher Type: %s\n",
                                SSL_get_cipher(ssl));
```

[그림 16] Handshake 계층의 처리과정

그림 17의 Record 과정은 Handshake 과정에서 협상된 암호 형태를 이용하여 안전한 통신을 하게 된다. data_Interface 함수는 SSL_write 함수와 SSL_read 함수를 이용하여 버퍼에 있는 데이터를 암호화하여 통신을 하게 되고, 그렇지 않을 경우는 일반 통신을 하게 된다.

```
/* Web browser에서 Web Tunneling으로 암호화된 데이터 전송 */
  if(serverDesc.ssl != NULL) {
    fprintf(stderr, "Record Processing Start !!!\n");
    writeData = SSL_write(serverDesc.ssl, httpString.buffer,
                                                httpString.len);
  }else{
    writeData = send(serverFd, httpString.buffer, httpString.len, 0);
  }
                              ...
int data_Interface(file_Descriptor *serverFd, file_Descriptor *clientFd) {
  if(serverFd->ssl != NULL){
    length = SSL_read(serverFd->ssl, buffer, sizeof(buffer));
  }else{
    length = recv(serverFd->fd, buffer, sizeof(buffer), 0);
  }
  for(written = 0; written < length; written += writeValue){
    if(clientFd->ssl != NULL){
      writeValue = SSL_write(clientFd->ssl, buffer + written,
                                              length - written);
    }else{
      writeValue = send(clientFd->fd, buffer + written,
                                            length - written, 0);
  }
}
```

[그림 17] Record 계층의 처리과정

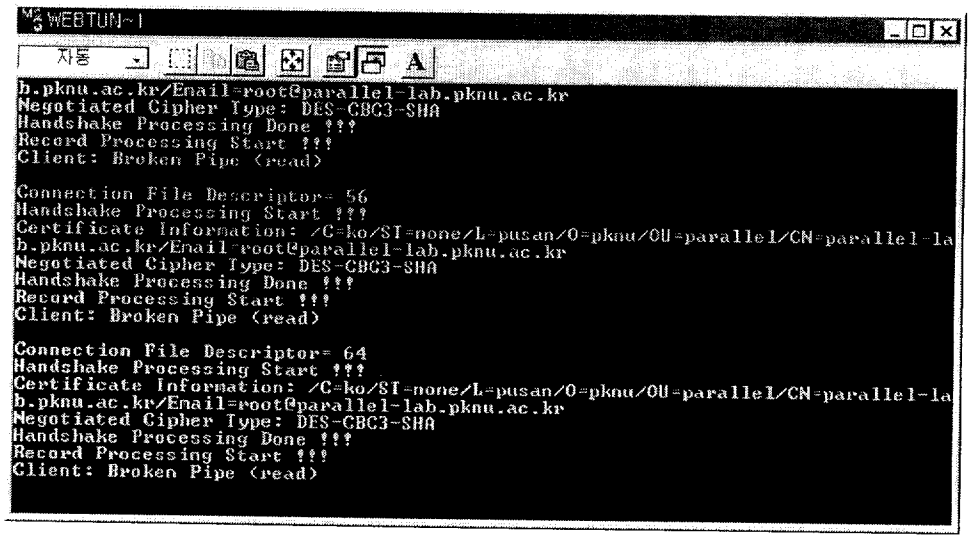
4.5. 구현 결과

본 논문에서 구현한 Web Tunneling은 Windows 계열과 Linux 플랫폼에서 Apache Web-Server와 SSL 기반의 안전한 통신을 하기 위한 것이다. 우선 Web Tunneling을 동작하기 위해서 Web Tunneling과 Web browser를 연동시킬 수 있는 역할로서 Proxy Server 설정이 필요하다. 본 논문에서 구현한 Windows 기반의 Web browser는 Proxy Sever 설정 모듈을 추가시킴으로서 browser가 동작하면 자동으로 설정하게 되어 있고, browser를 닫을 경우 Proxy Server 설정이 해제된다. 구현된 프로그램은 TCP/IP 기반의 가장 대표적인 인터넷 프로토콜인 HTTP와 HTTPS에서 테스트하였다. 표 3은 SSL 기반으로 하는 기존의 제품들과의 비교를 나타낸 것이다.

[표 3] 기존 제품과의 비교

제품 / 항목	기존의 제품 [19]	기존의 제품 [20]	기존의 제품 [21]	본 논문
OS 플랫폼	Windows계열	Windows계열	Windows계열	**Windows계열 Linux**
암호 강도	128 (bits)	128 (bits)	128 (bits)	**128 (bits)**
암호 라이브러리	J/LOCK Library(자체)	Xecure Library(자체)	INICrypto Library(자체)	**OpenSSL Library**
인터페이스	GUI	GUI	GUI	**GUI, Console**
설치 범위	Server, Client	Server, Client	Client	**Client**

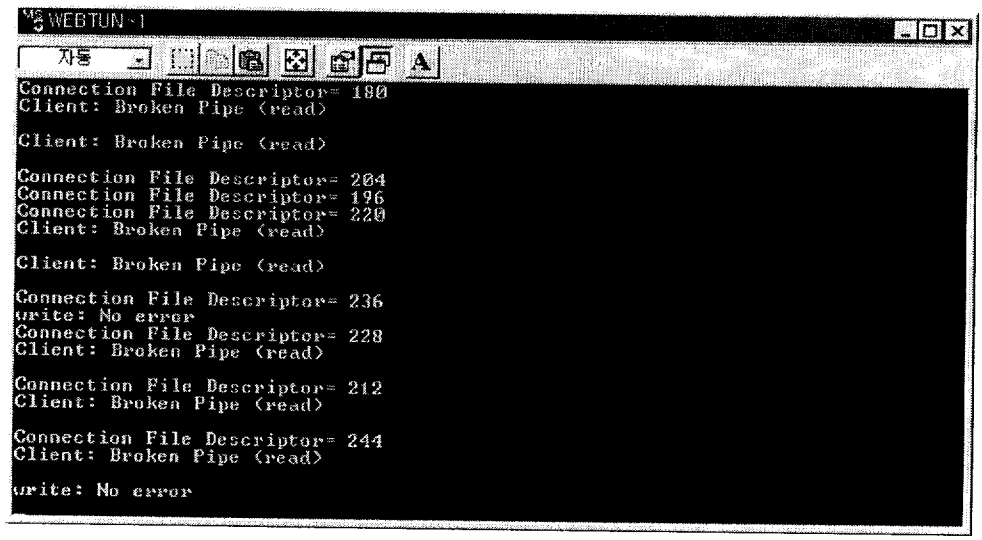
그림 18에서 Web Tunneling은 TCP/IP 기반으로 하는 HTTP 통신과 HTTPS 통신으로 구분하여 통신하게 되어 있다. 클라이언트 측의 Web Tunneling에 설정되어 있는 Web Server와 동작 형태이다.



[그림 18] HTTPS 통신일 때의 동작 형태

그림 18은 연결된 소켓 번호, 클라이언트와 서버간의 설정된 암호 알고리즘과 해쉬 알고리즘 등을 볼 수 있다. 본 논문에서 구현한 Web Tunneling에서 협상된 암호 형태는 DES-CBC3-SHA으로 나타났다. 암호 알고리즘으로 DES-CBC3과 SHA의 해쉬 알고리즘으로 협상되었고, 현재 DES-CBC3-SHA의 DES3 암호 강도는 168-bits key를 사용하여 매우 강력하고, 거의 깨어지기가 힘들다.

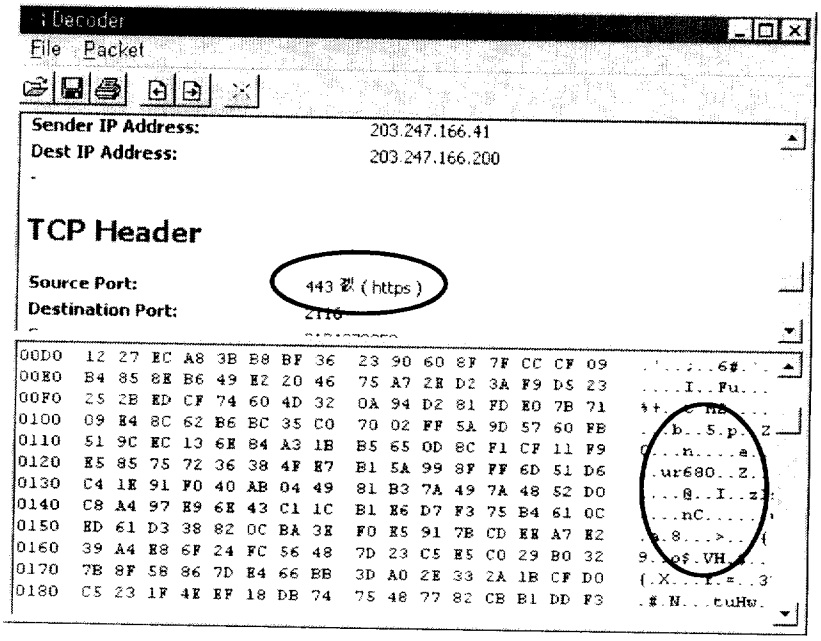
그림 19는 SSL 기반의 Handshake와 Record 과정을 거치지 않고 일반 HTTP 통신을 하는 동작 형태이다.



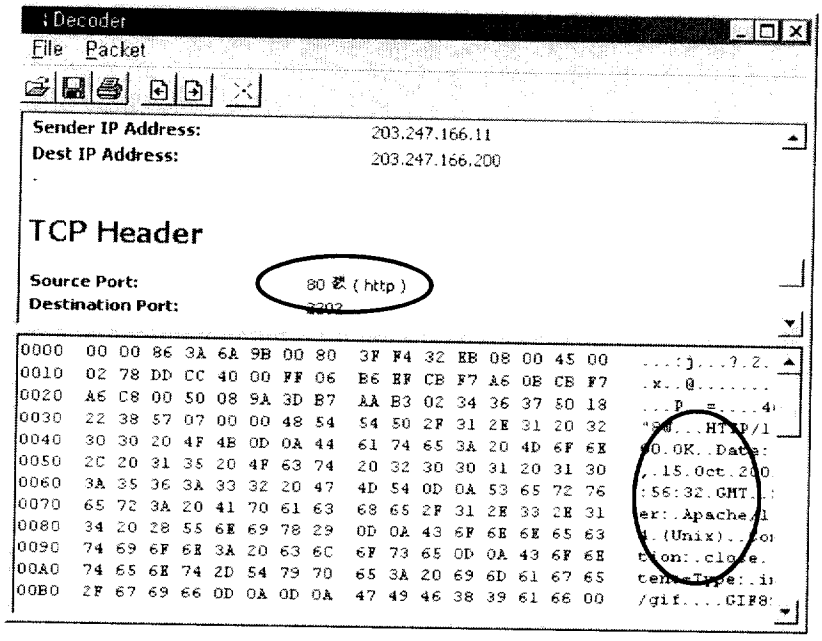
[그림 19] HTTP 통신일 때의 동작 형태

일반 통신인 그림 19에서는 암호 알고리즘과 해쉬 알고리즘을 적용하지 않고 일반적으로 통신하는 형태인 소켓을 open하여 클라이언트와 서버간의 통신을 하게 되어 있다. 그러므로 Web Tunneling에서 설정된 서버와는 SSL 기반으로 암호화된 데이터를 주고받으며, Web Tunneling에 설정되어 있지 않은 서버와는 암호화되지 않은 일반 통신을 한다.

그림 20과 그림 21은 HTTPS 통신과 HTTP 통신을 했을 때의 패킷을 캡처한 것이다. 그림 20에서는 클라이언트와 서버간에 port 443으로 HTTPS 통신을 하는 것이며, Sniffing Tool로서 패킷을 캡처한 결과 데이터의 내용이 암호화되어 알아볼 수가 없다. 한편 그림 21에서는 port 80의 HTTP 통신을 했을 경우를 나타낸 것이며, 패킷의 데이터가 일반 사용자의 눈으로 식별이 가능하여 중요한 개인 정보 등이 유출될 수 있다.



[그림 20] HTTPS 통신으로 암호화된 데이터



[그림 21] HTTP 통신으로 암호화되지 않은 데이터

5. 결론 및 향후연구과제

전세계적으로 인터넷 사용은 기하 급수적으로 증가하고 있으나, 인터넷의 특징인 정보 공유, 개방형 등을 역이용하여 불법적인 행위나 해킹, 개인 정보 유출 등의 피해 사고가 급격히 증가하고 있다. 이러한 인터넷을 통해 전자상거래를 사용하는데 있어 보안 문제에 대해 크게 대두되고 있다. 이러한 문제를 해결하기 위해 인터넷 보안 프로토콜로서 SSL 기반의 안전한 통신을 하도록 Web Tunneling을 설계하고 구현하였다. 본 논문에서의 Web Tunneling은 전자상거래에 있어 가장 많이 사용하는 인터넷 보안 프로토콜인 SSL을 이용하기 위해 OpenSSL Library를 이용하여 구현하였으며, 이를 기존의 Web Server와 통신을 하기 위해 적용하였다. 그러므로 Web Tunneling은 address, port 그리고 옵션 등을 설정하여 동작시키면 Web browser의 버전과 상관없이 안전한 통신을 할 수 있도록 구현되었다.

향후 연구 과제는 본 논문에서 구현한 Web Tunneling이 TCP/IP 기반의 HTTP, HTTPS 통신만 가능하므로 향후에 FTP, TELNET 등의 다양한 어플리케이션에서 동작하도록 추가작업이 필요하다. 현재 Web browser와 Web Tunneling 등 두 개의 프로그램을 실행시켜야 하는 단점이 있으므로 독립적으로 동작하는 형태를 보완하여 Web Tunneling을 Web browser에 내장하는 형태로 구현할 필요성이 있다. 그리고 현재 국제 표준화를 추진하는 국내 암호 알고리즘인 SEED 알고리즘 등의 추가가 필요하다.

참 고 문 헌

[1] 경향신문-한국전자거래진흥원 공동조사, "인터넷 쇼핑몰 체험진단", 2001. 8.

[2] 김재환, "인터넷 보안 설정 기술에 대한 보고서", 1998. 8.

[3] 이동훈, 임채훈 "SSL 3.0과 TLS 1.0의 비교 분석", http://www.future.co.kr, 2000년 11월.

[4] 정보통신연구진흥원 연구 개발 결과 보고서, "웹 환경 구축 및 운영을 위한 보안 기술 연구", 1997.12.

[5] 한국 인터넷 정보센터, "인터넷 이용자수 및 이용행태에 관한 설문조사 결과 보고서", 2001. 7.

[6] [AH] Kent, S., and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[7] [ESP] Kent, S., and R. Atkinson, "IP Encapsulating Security Payload(ESP)", RFC 2406, November 1998.

[8] [HTTP] HyperText Transfer Protocol -- HTTP/1.1. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee., RFC 2616, June 1999.

[9] [SHTTP] The Secure HyperText Transfer Protocol. E. Rescorla, A. Scruffman., RFC 2660, August 1999.

[10] [SSL] A. Frier, P. Kearton, and P. Rocher, "The SSL 3.0 Protocol version 3.0", Internet Draft, http://home.netscape.com /eng/ssl3/draft302.txt, Nov, 1996.

[11] [TLS] T. Dierks, C. Allen., The TLS Protocol Version 1.0., RFC 2246 January 1999.

[12] Eric Rescorla, "SSL and TLS Designing and Building Secure Systems", January 2001.

[13] Eric S. Ridgway , SSL Handshake, WWW Security Using SSL, http://www.ececs.uc.edu/~iouaiss/web_prog/misc/security/H_2.htm

[14] Jeremy Bradley and Neil Davies, The SSLP Reference Implementation Project, University of Bristol, 15.3.1998, http://www.cs.bris.ac.uk/~bradley/publish/SSLP/chapter4.html.

[15] Liisa Erkomaa, "Secure Socket Layer and Transport Layer Security", Faculty of Computer Science Helsinki University of Technology, 3. 1998.

[16] Naganand Doraswamy, Dan Harkins, "IPSec, The New Security Standard for the Internet, Intranets, and Virtual Private Networks", Prentice Hall PTR.

[17] The Apache Project Group, http://www.apache-ssl.org

[18] The OpenSSL Project Group, http://www.openssl.org

[19] STI, "J/SSLock", http://www.stitec.com

[20] SOFTFORUM, "XecureProxy", http://www.softforum.co.kr

[21] Initiative Technology of Internet Security, "INIssl", http://www.initech.com

[22] SafePassage, "Web Proxy version 2.0 User's Guide", 1999

감사의 글

본 논문이 나오기까지 여러 방면에서 많은 도움을 주신 저를 아는 모든 분들께 고마운 마음을 전하고자 합니다.

먼저 저를 대학원 2년 동안 학문적으로 많은 가르침과 올바른 길로 인도해 주신 김창수 교수님께 깊은 감사를 드립니다. 또한 바쁘신 와중에서도 논문 심사를 위해서 세심한 관심과 격려를 아끼지 않으셨던 윤성대 교수님과 정순호 교수님 그리고 박만곤 교수님, 여정모 교수님, 박승섭 교수님, 박지환 교수님, 박홍복 교수님, 이경현 교수님, 김영봉 교수님께도 감사 드립니다. 그리고 학과에서 늘 수고하시고 많은 도움을 주신 김미은 조교님, 신재원 조교님, 안성림 조교님께도 감사 드립니다.

항상 도움을 아끼지 않은 시스템 소프트웨어 및 이동통신 연구실 선후배 님들께 감사한 마음을 전합니다. 항상 따뜻한 위로의 말을 해주신 고신대학교 강병식 교수님, 연구실 선배로 저에게 많은 도움을 주신 박미정 선배, 전태건 선배, 정정훈 선배 그리고 대학원 동기로 옆에서 챙겨준 태호씨, 대학원 후배인 종우, 덕현이, 기욱이, 학부생으로 많은 일을 하는 남진이, 진수, 미영이, 선아, 경순이, 그리고 입학 동기이자 방장으로 연구실을 위해 애써주신 정동호 선생님, 장용호 선생님, 박철동 선생님, 변옥남 선생님, 현영숙 선생님 그리고 산대원과 교대원 선생님께도 깊은 감사를 드립니다. 그리고 이번에 졸업하는 동기들과 학과 선후배 님들에게도 감사 드립니다.

마지막으로 항상 저를 귀엽게만 봐주시는 할머니, 저를 믿고 묵묵히 뒷바라지 해주신 아버지, 항상 가족을 위해 기도하시는 어머니, 그리고 여동생에게도 감사를 드리며 이 논문을 바칩니다.