이학석사 학위논문

SSL/TLS 취약성 평가도구와 안전한 SSLProxy 설계 및 구현

지도교수 이 경 현

이 논문을 [illegible] 학위논문으로 제출함

2003년 2월

부경대학교 대학원

정보시스템협동

박 지 철

이학석사 학위논문

SSL/TLS 취약성 평가도구와 안전한 SSLProxy 설계 및 구현

지도교수 이 경 현

이 논문을 이학석사 학위논문으로 제출함

2003년 2월

부경대학교 대학원

정보시스템협동

박 지 철

박지철의 이학석사 학위논문을 인준함

2002년 12월 26일

주 심 이학박사 박 만 곤 

위 원 공학박사 조 경 연 

위 원 공학박사 김 영 봉 

<차 례>

<표 차례>

<그림차례>

Design and Implementation of Evaluation Tool for SSL/TLS vulnerability and Secure SSLProxy

Ji-Cheol Park

Dept. of Information System, Graduate School,
Pukyong National University

Abstract

The Secure Sockets Layer(SSL) and Transport Layer Security(TLS) which have been taking Standard for a secure communication of Transport layer have an inborn vulnerability.

In this paper, We develop the evaluation tool which can evaluate existing SSL/TLS peculiar vulnerabilities, and develop the secure SSL/TLS Proxy which can provide advanced security against the founded vulnerabilities.

In addition, We add SEED, HAS-160, and KCDSA(Korean Certificate-based Digital Signature Algorithm) suitable for the domestic actual condition.

1. 서론

컴퓨터 기술, 통신망 기술 및 전송 기술의 발달로 인하여 정보통신 분야는 큰 발전을 이루었으며, 이러한 기술의 발달은 인터넷 분야의 눈부신 발전을 가져왔다. 현재 인터넷에 연결된 세계 여러 나라의 네트워크, 이들 네트워크에 연결된 호스트의 수 및 사용자의 수는 수 없이 증가하고 있다.

사용자 인터페이스를 보다 편하게 설계하고 여러 가지 형태의 자료를 손쉽게 보여주는 브라우저의 출현으로 일반인들의 인터넷 사용이 폭발적으로 증가하였고, 결과적으로 인터넷 전체의 크기를 증가시켰다. 인터넷에는 수많은 정보들이 저장되어 있고 이들 중에는 기밀성을 요구하는 정보들도 상당수 존재한다. 이러한 정보들은 어느 특정한 집단(또는 사용자) 만이 접근을 허용하는 정보의 접근제어 메커니즘이 필요하게 되었고, 편리한 사용자 인터페이스를 가지는 WWW의 등장으로 선별적인 정보의 공유뿐만 아니라 안전한 정보의 전송이 필요하게 되었다.

인터넷의 근간인 TCP/IP는 본래 개방형 시스템을 기반으로 설계되었기 때문에 네트워크를 통한 정보의 기밀성을 제공하지 못하고 있다. 또한 사용자에 대한 인증도 제공하지 않고 있다. 그래서 네트워크에 전송되는 데이터의 기밀성과 무결성을 제공하고 사용자 인증을 제공하기 위하여 1994년 Netscape사에 의하여 Netscape 웹 브라우저를 위한 보안 프로토콜로서 처음으로 제안되어서 Netscape Navigator에 SSL 2.0이 포함되었다. 그러나 SSL 2.0의 보안 측면에서 많은 취약점이 발견되어 1996년 Internet Engincering Task Force(IETF)에서는 SSL 2.0의 여러 가지 취약점들은 보완한 SSL 3.0을 Internet Draft로 제안하였다[1]. 이후에 SSL 3.0을 수정 보완하여 표준화하는 작업이 진행되어서 1999년 Transport Layer Security(TLS)로 이름을 바꾸어 RFC 2246(TLS 1.0)으로 표준화하였다[2].

SSL/TLS는 클라이언트와 서버 사이에 교환되는 데이터를 안전하게 보호하기 위하여 공개키 인증서를 통한 사용자 인증, 비밀키 암호 시스템을 통한 데이터 기밀성을

제공한다. 이때, 비밀키 암호화에서 사용되는 비밀키는 키 교환 알고리즘으로 교환된 비밀 값으로 유도된다. 그리고 데이터의 위/변조를 막기 위하여 메시지 인증 코드(MAC : Message Authentication Code)를 사용한다.

본 논문에서는 이러한 보안 프로토콜인 SSL의 고유의 취약성을 살펴보고 이 취약성에 대한 대응책을 가지고 있는지에 대한 평가를 할 수 있는 평가 도구를 설계 및 구현한다. 또한 이러한 보안 취약성을 보완한 안전한 SSL/TLS Proxy(이하 SSLProxy)를 설계 및 구현한다. 안전한 SSLProxy는 국내 실정에 맞는 국내 표준 블록 암호알고리즘인 SEED, 해쉬알고리즘 HAS-160, 전자서명 알고리즘 KCDSA(Korean Certificate-based Digital Signature Algorithm)를 추가한다. 본 논문의 구성은 2장에서 관련 연구로서 SSL/TLS의 개요와 SSL/TLS의 보안 취약성을 살펴보고, SSLProxy에 추가 할 SEED, HAS-160, KCDSA의 개요를 살펴본다. 3장에서는 앞에서 살펴본 내용을 바탕으로 SSL/TLS 안전성 평가 도구 및 안전한 SSLProxy를 설계 및 구현한다. 4장에서는 향상된 SSLProxy의 안전성 평가 및 응용을 살펴보고 마지막으로 5장에서는 결론 및 향후과제에 대해 논의한다.

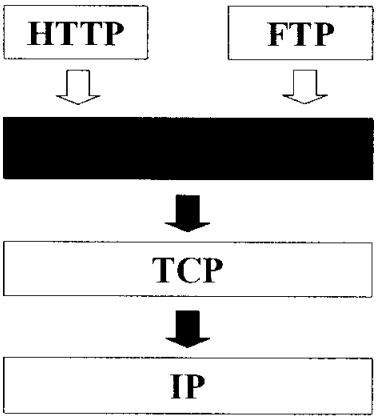
2. 관련 연구

2.1. SSL/TLS의 분석

2.1.1. SSL/TLS의 개요

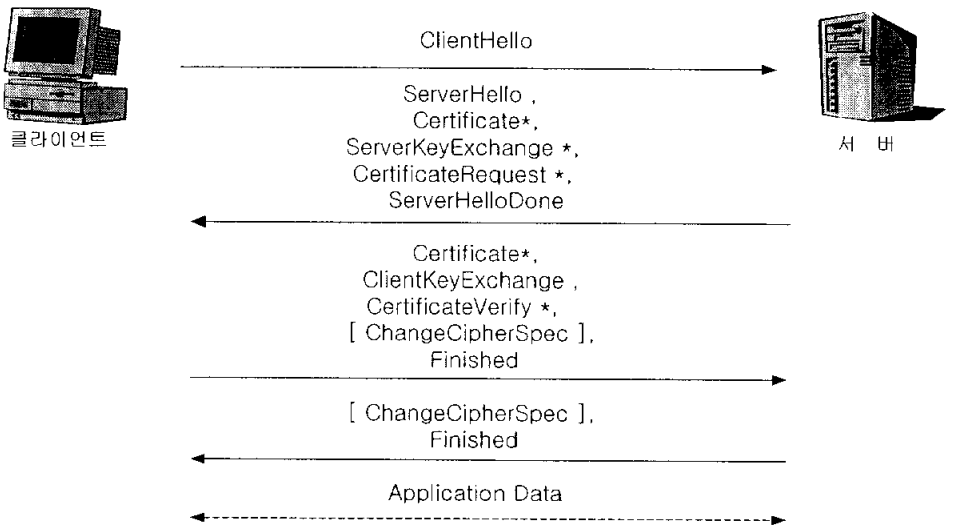
SSL/TLS(이하 SSL)는 클라이언트와 서버의 환경으로 동작하는 프로토콜로 웹 브라우저와 웹 서버가 각각 클라이언트와 서버의 역할을 한다. SSL은 클라이언트와 서버 사이에 교환되는 데이터를 안전하게 보호하기 위하여 사용자 인증을 하고, 비밀키 암호를 사용하여 데이터를 보호한다. 이때 사용되는 비밀키는 키 교환 알고리즘으로 교환된 비밀값으로 유도되며 데이터의 위/변조를 막기 위하여 메시지 인증 코드(MAC)를 사용한다.

SSL은 TCP/IP 계층과 HTTP, FTP, TELNET등의 응용 계층 사이에 위치하는 프로토콜이다. 계층 사이에 위치하는 프로토콜이므로 기존의 프로토콜에 대한 영향을 최소화 할 수 있게 된다. 또한 다양한 응용 계층의 프로그램들이 SSL을 이용하여 보안 기능을 수행할 수 있게 된다. 그리고 SSL은 웹 보안을 염두에 두고 설계되었으므로 TCP 위에서만 동작한다. SSL은 다음 [그림 1]과 같은 계층구조를 갖는다.



[그림1] SSL/TLS의 구조

클라이언트는 Handshake 프로토콜로 안전한 통신을 서버에게 요청을 하면서 필요한 파라미터들을 서버에게 제안한다. 서버는 클라이언트의 요청에 응답하여 통신에 필요한 파라미터들은 설정한다. Handshake에 의하여 설정된 파라미터들은 Change Cipher Spec 프로토콜에 의해 사용할 수 있도록 활성화되고 데이터들은 SSL에 의해 안전하게 보호되어 전송된다. 통신 과정에서 발생한 오류들은 Alert 프로토콜로 처리된다. 각 프로토콜의 모든 데이터는 Record Layer를 통하여 전송된다. Record Layer에서 사용되는 파라미터들은 Handshake 프로토콜에서 다름 [그림 2]와 같이 설정된다.



[그림 2] Full Handshake 프로토콜

클라이언트가 ClientHello를 서버에게 보내면 서버는 반드시 응답을 해야하고 그렇지 않으면 fatal error를 일으켜 서버와 접속을 할 수 없다. 클라이언트와 서버는 Hello 메시지에서 프로토콜 버전, Session ID, Cipher Suite, Compression Method

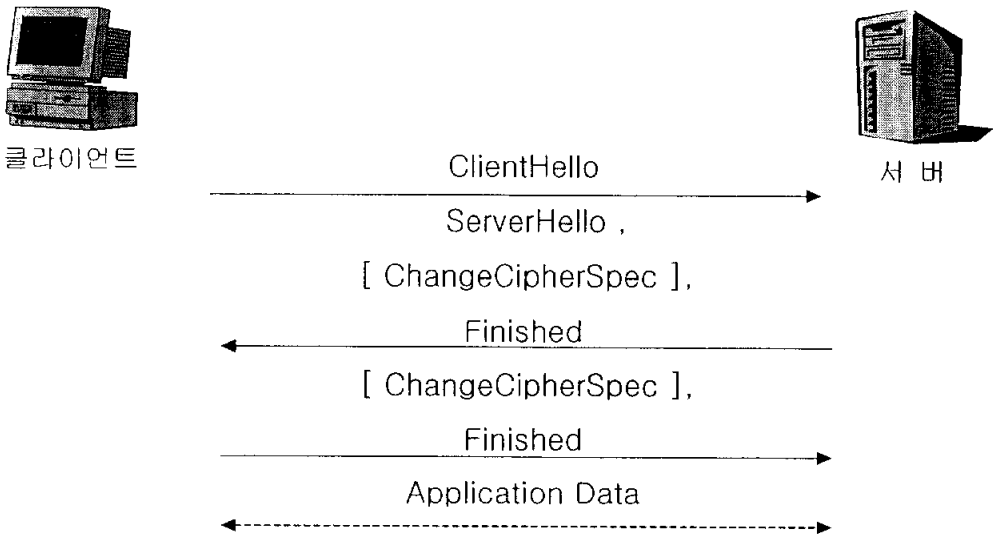
를 설정하고, 서로 난수를 각각 생성하여 교환한다.

서버의 인증서(Certificate), ServerKeyExchange, 클라이언트의 인증서, ClientKeyExchange를 통해 키 생성에 필요한 pre_master_secret과 master_secret를 서로 공유하게 된다. 서버의 인증이 필요하면 서버의 인증서를 보내고 만일 서버의 인증을 하지 않거나 혹은 서버가 서명용 인증서만을 보낸 경우에는 ServerKeyExchange를 통해 서버의 공개키를 전송하게 된다. 서버의 인증서를 보낸 경우에 서버는 클라이언트에게 인증서를 요청할 수도 있다.(CertificateRequest) 그리고 서버는 ServerHelloDone을 보내어 ServerHello 메시지가 끝났음을 알리고 클라이언트의 응답을 기다린다.

서버가 클라이언트의 인증서를 요청하였다면 클라이언트는 반드시 인증서를 전송하여야 한다. ClientKeyExchange 메시지에서는 앞에서 선택된 키 교환 알고리즘에 따라서 pre_master_secret에 대한 정보를 서버에게 보낸다. 클라이언트가 서명 가능한 인증서를 보낸 경우, 서명을 생성하여 보낸다(CertificateVerify).

그러면 클라이언트는 설정된 파라미터를 Change Cipher Spec 메시지를 전송한 후 Finished 메시지를 보낸다. 서버도 같은 방법으로 Change Cipher Spec 메시지를 전송한 뒤 Finished 메시지를 보낸다.

클라이언트가 새로운 Session을 생성하려면 비어있는 Session ID를 서버에 보내면 되고 이미 존재하는 Session을 사용하여 새로운 연결(connection)을 생성하려고 할 때는 클라이언트는 ClientHello에서 재사용하기를 원하는 Session ID를 보낸다. 서버는 클라이언트가 보낸 Session ID를 자신의 Session Cache에서 일치하는 것이 있는지 확인하여 일치하는 Session ID를 찾으면 [그림 3]과 같이 간단하게 Abbreviate Handshake 프로토콜이 진행된다. 일치하는 Session ID를 찾을 수 없으면 서버는 새로운 Session ID를 보내거나 비어있는 Session ID와 함께 Alert를 보내게 된다.



[그림 3] Abbreviate Handshake

2.1.2. SSL/TLS의 Handshake 프로토콜 취약성 분석

가. Ciphersuite Rollbake 공격

SSL 2.0 키 교환 프로토콜은 심각한 결점을 가지고 있다. 즉 Finished 메시지에서 cipher spec에 대한 인증이 이루어지지 않는다. 이러한 허점을 이용하여 서버와 클라이언트가 모두 강한 암호 알고리즘을 지원 가능함에도 불구하고 능동적인 공격자가 ClientHello에서 강한 암호 알고리즘을 삭제함으로써 40비트의 약한 암호 알고리즘을 사용하도록 만드는 공격이 가능하다.

이러한 공격을 "Ciphersuite Rollback 공격"이라고 하며, ClientHello 메시지 내에 전송되는 지원 가능한 CipherSuite의 평문 리스트를 편집함으로 가능한 공격이다.

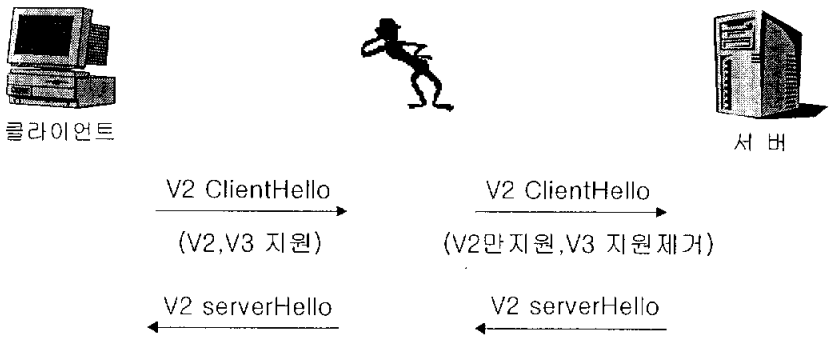
SSL 3.0은 Finished에서 master_secret를 키로 하여 Hello 메시지를 포함한 모든

Handshake 프로토콜 메시지에 대한 해쉬값을 만들어서 이전에 교환된 Handshake 메시지가 변조되었는지를 검증함으로서 이러한 공격을 막을 수 있고 필요하다면 세션이 종료된다.

나. Version Rollback 공격

SSL 2.0, SSL 3.0, TLS 1.0 모두 인터넷 프로토콜에서 같은 포트를 사용한다.(https로는 모두 443번 포트를 사용한다.) 따라서 클라이언트와 서버가 여러 버전을 동시에 지원하는 경우 어떤 버전을 사용할 것인지 결정한다.

서버와 클라이언트 모두가 SSL 2.0과 3.0을 지원하는 경우 클라이언트는 자신이 지원하는 가장 낮은 버전의 ClientHello로 시작해야함으로 [그림 4]와 같은 Version Rollback Attack에 의해서 SSL 2.0으로 통신이 이루어질 수가 있고 SSL 2.0의 취약성을 이용하는 공격이 가능하다.



[그림 4] Version Rollback Attack

이를 막기 위한 방법으로 SSL 2.0이 RSA 키 교환만 하는 것을 이용한다. SSL 3.0을 지원하는 클라이언트가 [그림 4]와 같은 공격으로 SSL 2.0의 Full Handshake를 진행한다고 가정해 보면, 클라이언트는 ClientKeyExchange에서

per_master_secret을 생성할 때, 첫 2바이트에 자신이 지원하는 가장 높은 버전을 포함한다. 또한 이것을 서버의 공개키로 암호화 할 때는 PKCS#1v1.5의 형식을 사용하게되는데 인코딩에서 paddingstring의 마지막 8개의 바이트를 0x03으로 만들면 서버는 이것을 보고 공격을 당했는지의 여부를 확인할 수 있다.

그러나 공격자가 처음에 SSL 3.0에 대한 세션을 만든 뒤 다시 2.0으로 그 세션을 resume하게 되면 위와 같은 방법으로는 막을 수가 없다. 따라서 같은 세션 안에서 버전을 혼용하는 것을 막아야 한다. 실제로 SSL 3.0/TLS 1.0 모두 resume 할 때, ClientHello의 버전은 원래 세션의 버전을 사용해야 하므로 resume할 때 버전이 바뀌지 않는다.

다. Dropping the Change Cipher Spec message

Change Cipher Spec 메시지는 Handshake 프로토콜에 포함되지 않으므로 Finished에서 해쉬의 입력에 포함되지 않는다. 그러므로 공격자가 change cipher spec 메시지를 중간에 가로채어도 Finished에서는 알 수가 없다. 특히 협상된 cipher suite가 암호화 없는 인증만 할 때는 SSL 3.0은 구현에 주의를 해야 한다. 이것으로 가능한 공격은 pending state와 current state가 각각 read state와 write state로 분리되어 Change Cipher Spec 프로토콜에서 pending state가 current state로 옮겨지는 것을 이용하여 중간에 Change Cipher Spec을 가로챔으로써 current read state를 NULL state로 만드는 공격 방법이다.

인증만을 하지 않고 암호화도 함께 하는 경우는 공격자가 Finished 메시지를 암호화한 것을 해독해야하므로 위 공격을 막을 수 있다. 그러나 40비트 비밀키를 사용하는 수출 가능한 약한 알고리즘을 사용하는 것은 안전하다고 보기 어렵다. 이러한 공격을 근본적으로 막기 위해서는 Finished 메시지를 검증하기 전에 Change Cipher Spec 메시지를 수신하여 pending read state를 current read state로 바꾸도록 SSL을 구

현해야 한다. TLS 1.0에서는 이렇게 구현하도록 명시하였다.

라. 선택적 암호문 공격

Bleichenbacher의 선택적 암호문 공격[3]은 RSA 암호문중에서 PKCS#1 v1.5의 형식으로 암호화된 것에 대한 공격이다. 이 공격에 의하면 PKCS#1 v1.5의 형식으로 암호화된 RSA 암호문은 2^{-40}의 확률로 해독된다.

SSL에서 cipher suite로 RSA 키 교환 알고리즘을 사용하는 경우 ClientKeyExchange에서 클라이언트가 pre_master_secret를 PKCS#1 v1.5의 형식으로 암호화하여 전송한다. 여기서 Bleichenbacher의 공격이 적용될 수 있다. 공격자는 클라이언트를 대신하여 ClientKeyExchange에서 임의로 암호문을 선택하여 서버로 전송하여 이것을 복호화하게 한다. 그러면 서버는 복호화된 평문이 PKCS#1 v1.5의 형식으로 올바르게 인코딩 되었는지 검사하고 올바르지 않으면 에러를 발생시킨다. 이와 같은 방법으로 공격자는 임의로 생성한 암호문이 올바른 암호문인지 아닌지를 구별할 수 있는 oracle을 갖게 되는데, 충분한 수의 조작된 암호문을 가지고 있으면 클라이언트가 원래의 pre_master_secret가 암호화되어 있는 암호문을 해독할 수 있다.

SSL 3.0에서는 올바른 인코딩이 아닌 암호문을 받았을 때 에러를 발생시켜서 공격자의 oracle로 사용될 수 있지만, TLS 1.0에서는 에러를 발생시키지 않고 서버가 임의로 48바이트의 난수를 생성하여 pre_master_secret로 간주하고 Handshake를 진행하여 공격자가 올바른 암호문인지 아닌지를 판단할 수 없도록 하여 선택적 암호문 공격을 막게 된다.

마. master_secret 보호

SSL 3.0은 세션을 처음 생성할 때 키 교환 알고리즘을 통해서 pre_master_secret

를 서로 교환하고 이것을 이용하여 master_secret를 생성한다. master_secret를 생성하면 pre_master_secret는 바로 메모리에서 소거해야 한다. 그러나 master_secret는 하나의 세션 내에서 재 접속하는 경우 같은 master_secret로부터 세션 키를 유도하게되어 계속 보관하여야 한다.

SSL의 모든 세션 키는 각 session state에서 유지하고 있는 master_secret에 의해 유도되므로 SSL의 안정성은 master_secret의 보호에 매우 크게 의존한다. 따라서 꼭 필요한 경우만 master_secret를 사용하는 것이 바람직하다. 다음은 master_secret이 쓰이는 곳이다.

- SSL

  CertificateVerify, Finished와 key_block의 생성.

- TLS

  Finished와 key_block의 생성.

2.1.3. SSL/TLS 안전성의 고려 사항

가. 인증서

SSL/TLS의 안전성에 있어서 인증서에 의존하는 부분이 매우 크다. 따라서 SSL/TLS를 구현하는데 있어서 인증서를 검증하는 부분은 매우 중요하다. 서버나 클라이언트의 인증서를 검증하기 위해서는 무엇보다도 인증서를 발행한 CA를 믿을 수 있다는 보장을 받아야 한다. 즉, CA의 인증서를 믿을 수 있는 곳으로부터 발행을 받는지 혹은 사용자 내부에 믿을 수 있는 CA와 CA의 인증서들의 목록을 관리하고 있어야 한다. 상대방으로부터 CA의 인증서를 받는 것은 공격자에 의해 변조될 가능성이 있으므로 바람직하지 않다.

상대방의 인증서를 받으면 각 인증서의 항목들(유효기간, 상대방의 이름(subject), 인증서에 대한 서명 등)이 유효한지 확인해야하고 또한 유효한 인증서라고 할지라도 인증서 폐기 목록(Certification Revocation List : CRL)에 포함되어 있지 않는지도 확인해야 한다.

나. RSA 키의 크기

현재 사용되고 있는 대부분의 인증서는 RSA 타입이므로 RSA로 키 교환이 빈번히 이루어진다. 즉, ClinetKeyExchange에서 pre_master_sercret를 클라이언트가 생성하여 서버의 RSA 공개키로 암호화하여 보낸다. 여기서 SSL/TLS의 모든 키들은 pre_master_sercret에 의해 생성된 master_secret으로부터 유도되므로 실제 RSA 비밀키가 노출되면 SSL/TLS의 모든 메시지들이 노출되는 것과 같다.

현재 응용 프로그램들은 512, 768, 1024비트 등의 합성수들을 많이 사용하고 있는데, 512비트의 합성수는 1999년 8월에 인수분해 결과[4]등으로 볼 때 인수분해의 위험성이 있으므로 이전 버전과의 호환성을 위해서만 사용하며, 이후의 응용에서는 1024비트를 표준으로 사용하는 것이 바람직하다.

다. 난수 발생

고정된 키를 사용하는 DH 알고리즘 이외에는 모두 pre_master_secret를 생성하기 위해서 클라이언트는 임의의 난수를 발생시켜서 얻어야 한다. 실제로 보안제품의 암호분석을 위해서는 개별적인 암호알고리즘을 분석하여 여기에 사용되는 비밀키를 찾아내는 것보다 이 비밀키를 생성하는 의사 난수 생성기를 분석하는 것이 훨씬 더 효율적인 공격이 될 수 있다.

일반적으로 난수는 의사 난수 생성기로 생성하는데, 안전한 의사 난수 생성기의 사용은 SSL/TLS뿐만 아니라 모든 보안제품에 있어서 가장 중요한 요소이다. 또한 간

과하기 쉬운 부분중 하나는 의사 난수 생성 알고리즘에 입력되는 seed 또한 반복되지 않은 값으로 예측 불가능한 값이어야 안전한 의사 난수를 생성할 수 있다.

2.2. 국내 표준 알고리즘의 개요

2.2.1. 128비트 블록암호알고리즘(SEED)의 개요

가. 개요

암호알고리즘은 암·복호화에 사용되는 키의 특성에 따라 암·복호화 키가 같은 대칭키 암호알고리즘과 암·복호화 키가 서로 다른 공개키 암호알고리즘으로 크게 구분할 수 있으며, 대칭키 암호알고리즘은 데이터 처리 형식에 따라 스트림 암호알고리즘과 블록 암호알고리즘으로 나눌 수 있다.

SEED는 대칭키 암호알고리즘으로, 블록 단위로 메시지를 처리하는 블록 암호알고리즘이다.

대칭키 블록 암호알고리즘은 비밀성을 제공하는 암호시스템의 중요 요소이다. n비트 블록 암호알고리즘이란 고정된 n비트 평문을 같은 길이의 n비트 암호문으로 바꾸는 함수를 말한다(n비트 : 블록 크기). 이러한 변형 과정에 암·복호키가 작용하여 암호화와 복호화를 수행한다.

블록 암호알고리즘은 대부분 Feistel 구조로 설계된다(예 : DES, FEAL, LOKI, MISTY, Blowfish, CAST, Twofish 등). Feistel 구조란 각각 t비트인 L_0, R_0 블록으로 이루어진 2t비트 평문 블록 (L_0, R_0)이 r라운드($r \geq 1$)를 거쳐 암호문 (L_r, R_r)으로 변환되는 반복 구조를 말한다. 반복 구조란 평문 블록이 여러 라운드를 거쳐 암호화되는 과정을 말한다(라운드 함수 $i\,(1 \leq i \leq r)$란 암호키 K로부터 유도된 각 서브키 K_i (또는, 라운드 키라 불림)를 중요 입력으로 하여

$L_i = R_{i-1}$, $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$를 통해 $(L_{i-1}, R_{i-1}) \xrightarrow{K_i} (L_i, R_i)$로 바꾸어 주는 함수를 말한다). 또한, 전체 알고리즘의 라운드 수는 요구되는 비도와 수행 효율성의 상호 절충적 관계에 의해 결정된다. 보통 Feistel 구조는 3라운드 이상이며, 짝수 라운드로 구성된다. 이러한 Feistel 구조는 라운드 함수에 관계없이 역변환이 가능하며(즉, 암·복호화 과정이 같음), 두 번의 수행으로 블록간의 완전한 diffusion이 이루어진다. 또한 알고리즘의 수행속도가 빠르고, H/W 및 S/W로 구현이 용이하고, 아직 구조상의 문제점이 발견되고 있지 않다는 장점을 지니고 있다.

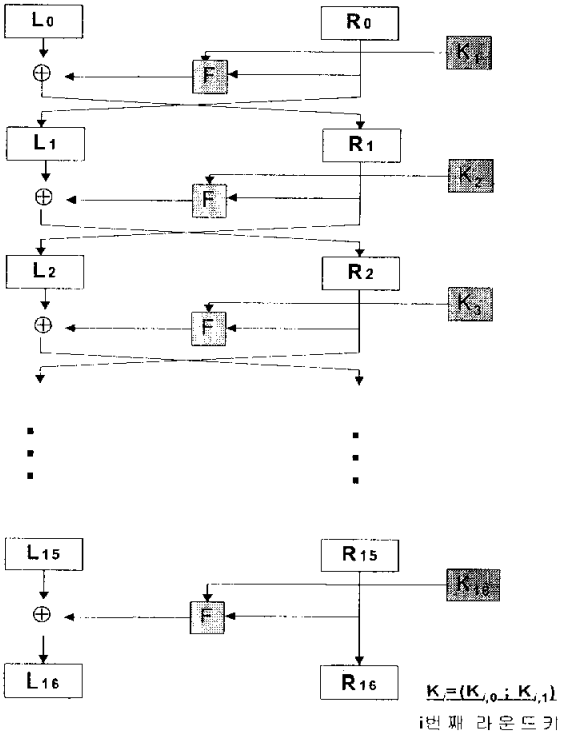
나. 기호와 표기

- a $\oplus$ b : 'a' bit-wise Exclusive-Or 'b'
- a $\boxplus$ b : (a + b) mod 2^{32}
- a & b : 'a' bit-wise AND 'b'
- $X^{\ll s}$: X를 s비트 만큼 왼쪽으로 순환 이동하는 연산
- $X^{\gg s}$: X를 s비트 만큼 오른쪽으로 순환 이동하는 연산
- L_i : i 라운드에서 출력된 왼쪽 메시지 블록(64비트)
- R_i : i 라운드에서 출력된 오른쪽 메시지 블록(64비트)
- $K_i = (K_{i,0}, K_{i,1})$: i 라운드의 라운드키(64비트)
- $K_{i,0}$: i 라운드 F 함수의 오른쪽 입력키(32비트)
- $K_{i,1}$: i 라운드 F 함수의 왼쪽 입력키(32비트)
- $X = (X_3 \| X_2 \| X_1 \| X_0)$: G함수의 입력값(32비트)
- $Y = (Y_3 \| Y_2 \| Y_1 \| Y_0)$: G함수에서 S-Box(S_1, S_2)의 출력값(32비트)
- $Z = (Z_3 \| Z_2 \| Z_1 \| Z_0)$: G함수의 출력값(32비트)

- m_i : 상수
- KCi : 라운드 키 생성과정에서 사용되는 i+1 라운드 상수

다. 알고리즘 전체 구성도

알고리즘의 전체 구조는 Feistel 구조로 이루어져 있으며, 128비트의 평문 블록단위당 128비트 키로부터 생성된 16개의 64비트 라운드 키를 입력으로 사용하여 총 16라운드를 거쳐 128비트 암호문 블록을 출력한다. 다음 [그림 5]는 SEED 알고리즘의 전체구조를 도식화한 것이다.

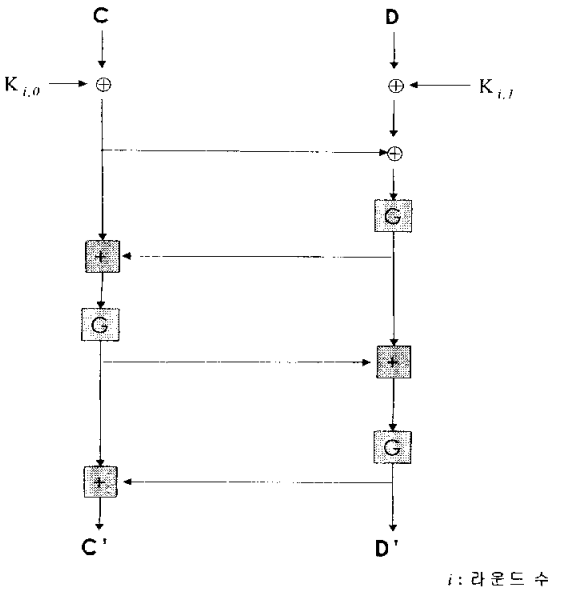


[그림 5] SEED 전체 구조도

$(L_{16}(64), R_{16}(64))$을 출력한다.

라. F 함수

Feistel 구조를 갖는 블록 암호알고리즘은 F 함수의 특성에 따라 구분될 수 있다. SEED의 F 함수는 수정된 64비트 Feistel 형태로 구성되며 F 함수는 각 32비트 블록 2개(C, D)를 입력으로 받아, 32비트 블록 2개(C′, D′)를 출력한다. 즉, 암호화 과정에서 64비트 블록(C, D)와 64비트 라운드 키 $K_i = (K_{i,0}; K_{i,1})$ 를 F 함수의 입력으로 처리하여 64비트 블록(C′, D′)을 출력한다.



[그림 6] F-함수 구조도

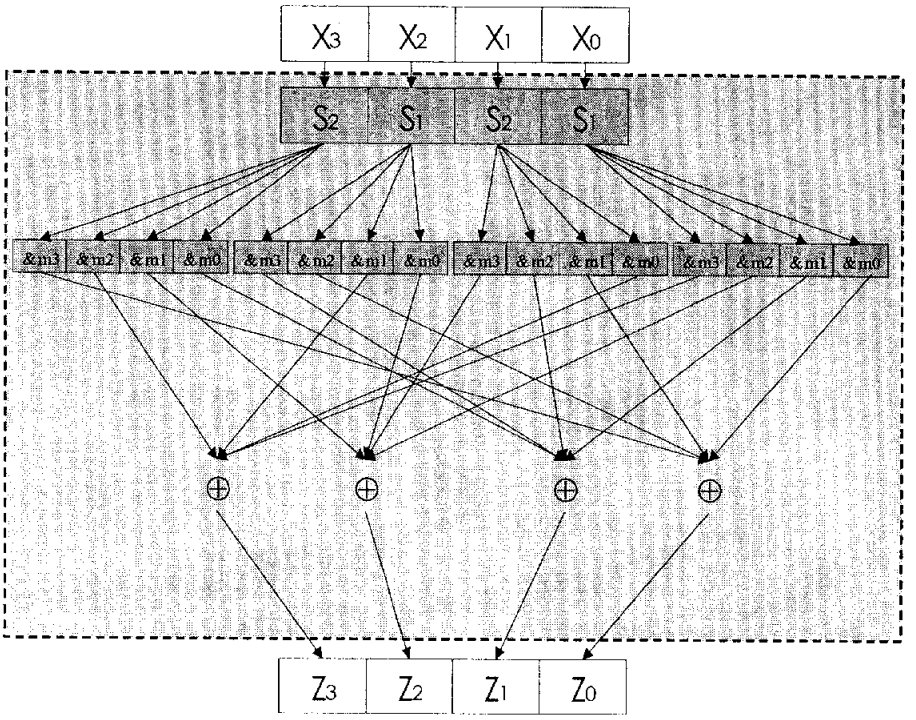
마. G 함수

전체 G 함수는 다음과 같다:

$Y_3 = S_2(X_3),\ \ Y_2 = S_1(X_2),\ \ Y_1 = S_2(X_1),\ \ Y_0 = S_1(X_0),$

$$Z_3 = (Y_0 \& m_3) \oplus (Y_1 \& m_0) \oplus (Y_2 \& m_1) \oplus (Y_3 \& m_2)$$
$$Z_2 = (Y_0 \& m_2) \oplus (Y_1 \& m_3) \oplus (Y_2 \& m_0) \oplus (Y_3 \& m_1)$$
$$Z_1 = (Y_0 \& m_1) \oplus (Y_1 \& m_2) \oplus (Y_2 \& m_3) \oplus (Y_3 \& m_0)$$
$$Z_0 = (Y_0 \& m_0) \oplus (Y_1 \& m_1) \oplus (Y_2 \& m_2) \oplus (Y_3 \& m_3)$$

$(m_0 = 0\text{xfc},\ \ m_1 = 0\text{xf3},\ \ m_2 = 0\text{xcf},\ \ m_3 = 0\text{x3f})$



[그림 7] G 함수

(단, $m_0 = 0xfc$, $m_1 = 0xf3$, $m_2 = 0xcf$, $m_3 = 0x3f$ 이고,

&는 bit-wise AND 연산을 의미한다)

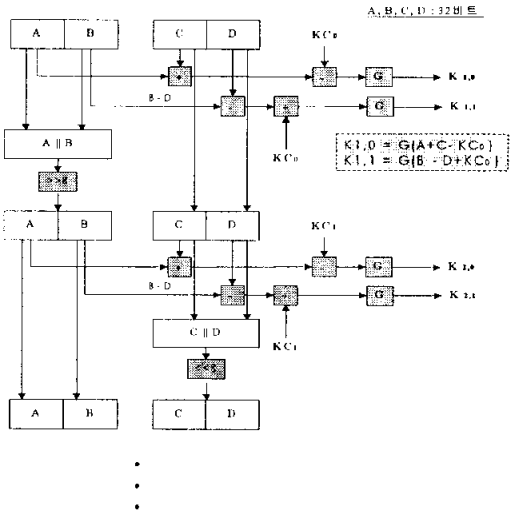
바. S-Box

알고리즘(SEED)의 S-Box는 8비트 입력(즉, 0~255)을 받아 8비트 출력(즉, 0~

255)을 내는 함수로써, 예를 들면, S_1-box의 경우, 입력이 '21'이면 출력은 '226'이 된다.

사. 라운드 키 생성과정

SEED의 라운드 키 생성과정은 128비트 암호키를 64비트씩 좌우로 나누어 이들을 교대로 8비트씩 좌/우로 회전 이동한 후, 결과의 4워드들에 대한 간단한 산술연산과 G 함수를 적용하여 라운드 키를 생성한다. 라운드 키 생성과정은 기본적으로 하드웨어나 (모든 라운드 키를 저장할 수 없는) 제한된 자원을 갖는 스마트카드와 같은 응용에서의 효율성을 위하여, 암호화나 복호화시 암호키로부터 필요한 라운드 키를 간단히 계산할 수 있다.



[그림 8] 라운드 키 생성과정 구조도

2.2.2. 해쉬함수 알고리즘(HAS-160)의 개요

가. 개요

해쉬알고리즘은 임의의 길이의 비트 열을 고정된 길이의 출력값인 해쉬코드로 압축시키는 알고리즘으로써 다음의 성질을 만족한다.

- 주어진 출력에 대하여 입력값을 구하는 것이 계산상 불가능하다.
- 주어진 입력에 대하여 같은 출력을 내는 또 다른 입력을 찾아내는 것이 계산상 불가능하다.

암호학적 응용에 사용되는 대부분의 해쉬알고리즘은 위의 두 성질뿐만 아니라 이보다 강한 충돌저항성을 지닐 것이 요구된다.

- 같은 출력을 내는 임의의 서로 다른 두 입력 메시지를 찾는 것이 계산상 불가능하다.(collision-resistance)

암호학적 해쉬알고리즘의 충돌 저항성은 전자서명에서 송신자 외의 제 3 자에 의한 문서위조를 방지하는 부인방지 서비스를 제공하기 위한 필수적인 요구조건이 된다.

해쉬알고리즘은 크게 DES와 같은 블록 암호알고리즘에 기초한 해쉬알고리즘과 전용 해쉬알고리즘으로 나눌 수 있다. 블록 암호알고리즘을 이용한 해쉬알고리즘은 이미 구현되어 사용되고 있는 블록 암호알고리즘을 사용할 수 있다는 이점이 있으나, 대부분의 블록 암호알고리즘의 속도가 그리 빠르지 않을뿐더러 이를 기본함수로 이용한 해쉬알고리즘의 경우 블록 암호알고리즘보다도 훨씬 더 속도가 떨어지므로 현재는 대부분의 응용에서 전용 해쉬알고리즘이 주로 이용된다.

대부분의 전용 해쉬알고리즘은 덧붙이기, 분할, 반복연산의 과정을 거쳐 계산된다. 임의의 길이의 메시지 X를 입력단위의 배수가 되도록 덧붙이기 하여 t개의 입력 블록($X_1, \cdots, X_t$)으로 분할한다. 해쉬코드는 각 블록 X_i에 대한 연쇄변수를 주어진 초기

값(IV)으로 초기화한 후 압축함수를 반복적으로 적용하여 계산되며, 그 처리과정은 다음과 같이 기술할 수 있다.

$H_0 = IV,$

$H_i = f(H_{i-1}, X_i), 1 \leq i \leq t,$

$h(X) = H_t$

여기서 f는 h의 압축함수이며, H_i는 단계 $i-1$과 단계 i의 중간 계산값이다.

2.2.3. 전자서명 알고리즘(KCDSA)의 개요

가. 개요

디지털 통신에서 메시지의 수신자는 수신된 메시지가 전송 도중 내용이 바뀌지 않았는지, 또는 제 3 자가 송신자를 가장하여 메시지를 보내지 않았는지를 확인할 수 있기를 바란다. 전자서명은 이러한 서비스를 제공해 주는 중요한 도구이다. 전자서명은 수신자나 제 3 자에게 서명자를 확인할 수 있도록 한다는 점에서 손으로 하는 수기 서명과 유사하며, 또한 장기간 보존되는 데이터 등이 보관 중에 내용이 변화되었는지를 확인할 수 있도록 하는 무결성 서비스도 제공한다.

전자서명 알고리즘은 서명자로 하여금 전자서명을 생성할 수 있도록 하고, 검증자에게는 서명의 진위여부를 확인할 수 있게 해준다. 모든 서명자는 자신만이 알고있는 비공개 서명키와 모든 사용자에게 공개되는 공개 검증키를 가지는데, 비공개 서명키는 서명 생성 과정에서 사용되고, 공개 검증키는 서명 검증 과정에서 사용된다. 이 두 과정 중에 서명될 메시지는 충돌저항성의 해쉬 함수에 의해 압축되어 사용된다. 특정 서명자의 비공개 서명키를 모르는 타인이 서명자의 서명을 위조하는 것은 계산상 불가

능하나, 누구나 공개 검증키를 이용해서 서명자의 서명을 실시간 내에 검증할 수 있다.

서명 검증 과정에서 검증자는 서명자의 올바른 공개 검증키를 얻어야 하는데, 일반적으로 모든 사람이 신뢰하는 제 3 자, 즉 인증 기관(CA : Certification Authority)이 각 개인의 ID와 그의 공개 검증키 등을 결합한 인증 정보를 CA의 비공개 서명키로 서명한 인증서를 배포함으로써 공개 검증키의 소유자를 보증한다. 이와 같이 CA가 발급하는 공개 검증키에 대한 인증서를 바탕으로 서명자의 비공개 서명키에 대응하는 공개 검증키를 확인하고 이를 이용하여 서명의 진위 여부를 판단하는 서명방식이 인증서 기반 전자서명 방식이다.

메시지에 대한 전자서명은 누구나 그 메시지의 출처를 확인할 수 있게 해주는 인증 서비스, 획득한 메시지가 도중에 변경되었는지의 여부를 알 수 있게 해 주는 무결성 서비스 및 메시지 서명을 했다는 사실을 부인할 수 없게 하는 부인방지 서비스 등 정보화 사회에서 필요한 다양한 보안 서비스를 제공하는 데 사용될 수 있다.

국내에서는 표준은 임의의 길이를 갖는 메시지에 대한 인증서 기반 부가형 전자서명 생성과 검증을 위한 알고리즘인 KCDSA(Korean Certificate-based Digital Signature Algorithm)를 규정하고 있다.

나. 적용범위

인증서 기반 전자서명 알고리즘은 정보 처리 시스템 또는 정보통신망 환경에서 인증, 무결성 및 부인방지 등의 정보보호 서비스를 제공하기 위하여 서명자가 전자서명을 생성할 때와 검증자가 서명된 메시지를 검증할 때 적용될 수 있다. 또한 정보통신 기기 및 관련 응용 개발 시 구현 결과의 구현적합성 여부를 확인하는데 이용될 수 있다.

3. SSL/TLS 안전성 평가 도구 및 안전한 SSLProxy의 설계 및 구현

본 장에서는 2장에서 소개된 SSL/TLS Handshake 프로토콜의 취약성들을 평가하기 위한 도구에 대해 논의한다. 2장에서 제시된 SSL/TLS Handshake 프로토콜의 취약성들은 대부분이 SSL/TLS Specification의 분석을 통한 것이며, 이러한 취약성들은 실제로 구현하는 벤더나 프로그래머에 따라 각각의 취약성들이 다르게 표출될 수 있다.

본 연구에서는 SSL/TLS Handshake 프로토콜에서 취약하다고 알려진 문제점들을 근간으로 해서 실제로 SSL/TLS를 구현한 제품들이 이러한 취약점에 대한 대응책을 마련하고 있는 가를 평가하기 위한 "SSL/TLS 안전성 평가 도구"와 이런 한 취약점들을 보안한 "안전한 SSLProxy"를 설계하고 구현한다.

3.1. SSL/TLS 안전성 평가 도구

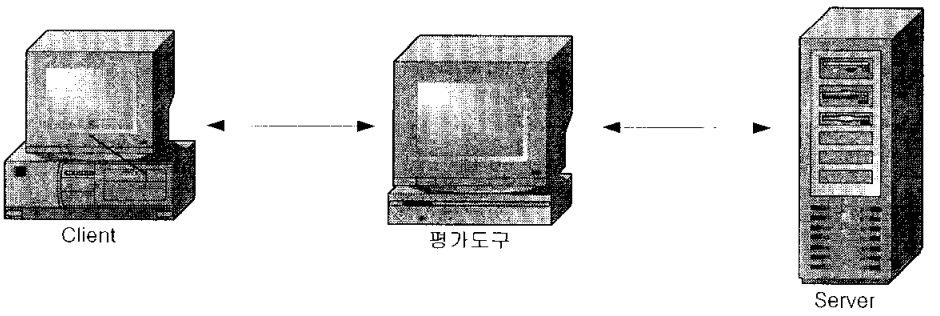
3.1.1. SSL/TLS 안정성 평가도구 개발 환경

본 논문에서 구현한 안전성 평가도구는 PureTLS 0.9b2[5]를 사용하여 SSL 기능을 제공한다. PureTLS는 자바 기반의 Cryptix 라이브러리[6]를 기반으로 SSL 기능을 제공하기 위한 암호, 해쉬 함수 및 Handshake를 위한 함수들을 제공하고 있다. 평가도구의 평가 항목의 적절한 평가를 위해 PureTLS의 상당부분을 수정하여 사용하였다.

본 논문에서 클라이언트와 서버는 SSL/TLS 프로토콜을 통한 통신을 하고, [그림 9]와 같이 중간에 평가도구를 통한 패킷분석 및 공격 테스트에 요구되는 데이터의 변

경을 가정한다.

하지만, 실질적으로 패킷 분석을 통해서 테스트에 요구되는 데이터 변경은 어렵기 때문에 클라이언트 측면의 평가도구를 개발한다.



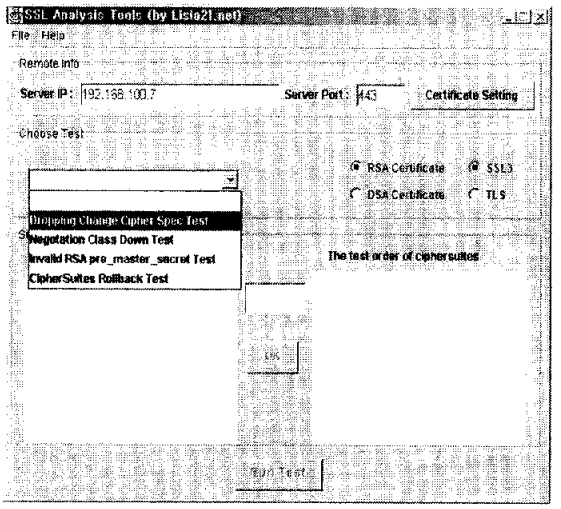
[그림 9] 평가도구 구성도

안전성 평가도구의 개발 환경으로는 [표 1]과 같은 환경으로 구현하였다.

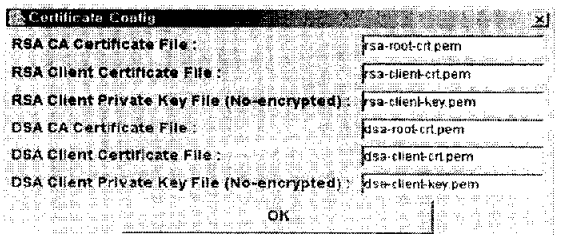
[표 1] 평가도구 구현환경

	구현환경
서 버	**Apache-Mod_SSL**
평가 도구 개발환경	JDK 1.3.1_03, PureTLS 0.9b2, Cryptix 3.2, Cryptix ASN.1 kit, Java-getopt-1.0.9

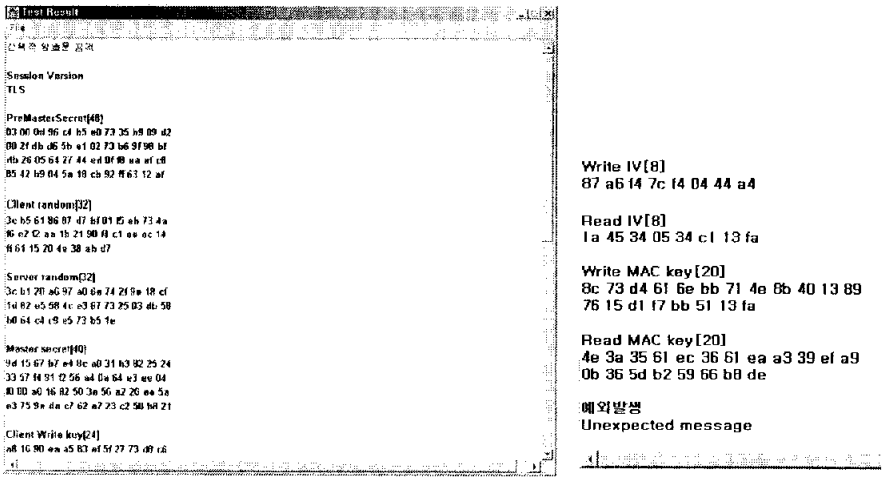
본 안전성 평가 도구는 [그림 10, 11]과 같이 GUI(Graphics Users Interface) 방식의 사용자 인터페이스를 도입하여 평가 대상 서버, 사용자 인증서, 평가 항목, 인증서 및 버전을 선택하고 변경할 수 있도록 하였으며, 2장에서 살펴본 SSL/TLS Handshake 프로토콜의 보안 취약성을 평가 할 수 있도록 하였다.



[그림 10] 평가 도구 매개변수 변경 화면



[그림 11] 사용자 인증서 설정



[그림 12] 테스트 결과

평가 결과에 대한 결과분석을 위하여 테스트명, 세션 버전, 세션 파라미터, 에러 종류와 같은 결과 값을 [그림 12]에서와 같이 출력하고 저장 할 수 있도록 구현하였다.

출력 결과는 SSL/TLS 프로토콜 분석 도구인 SSLDump 0.9b3[7]를 서버 측에서 운용하여 확인할 수 있다.

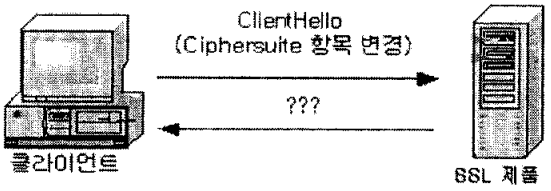
3.1.2. SSL/TLS 안정성 평가도구의 구현

SSL의 안전성 평가 도구로서 취약성 분석에서 분석된 자료를 바탕으로 구현 가능한 공격 유형을 살펴보고 그에 해당하는 공격을 시도한다. 기존의 SSL 구현 제품들의 안전성을 평가할 수 있을 것이다.

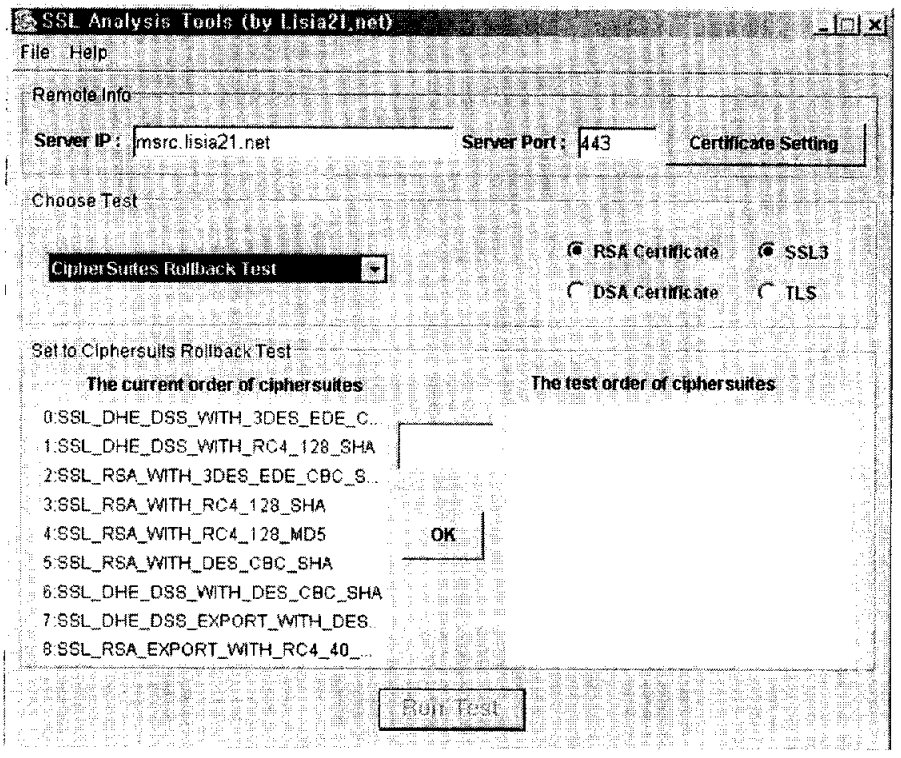
가. Ciphersuite Rollback 공격 테스트

서버와 클라이언트 모두 강한 알고리즘을 지원 가능함에도 불구하고 공격자가 [그림 14]와 같이 ClientHello 메시지 내에 포함되는 Ciphersuite의 평문 리스트를 재구성하여 서버에게 전송함으로써 서버가 수출 모드의 약한 암호 알고리즘과 같은 취약한 알고리즘을 결정하도록 [그림 15]와 같이 구현하였다.

Ciphersutie Rollback 공격 평가는 SSL/TLS 평가 제품 모두 Finished에서 master_secret를 키로 하여 Hello 메시지를 포함한 모든 Handshake 프로토콜 메시지에 대한 해쉬값을 만들어 이전에 교환된 Handshake 메시지가 변조되었는지 검증하는 가를 평가한다.



[그림 14] Ciphersutie Rollback 공격

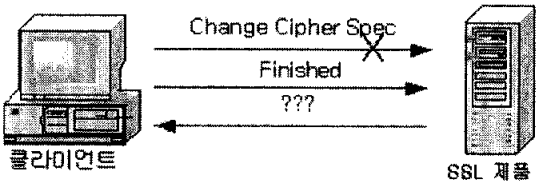


[그림 15] CipherSuite Rollback 공격 테스트 구현

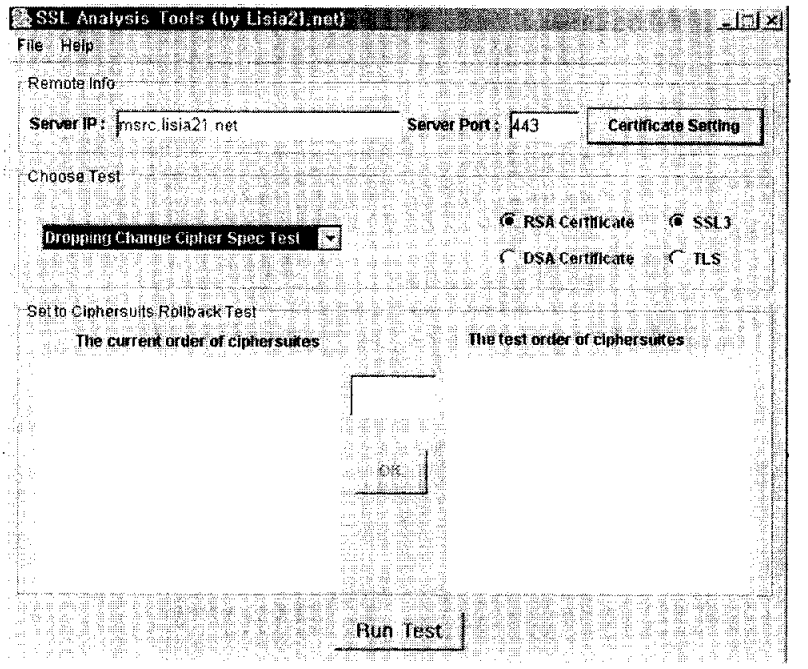
나. Change Cipher Spec 프로토콜 평가

SSL/TLS 프로토콜에서는 Handshake 세션의 상태가 pending state와 current state로 분리되어 있으며, Change Cipher Spec 프로토콜에 의해서 pending state에서 current state로 옮겨진다.

또한, Change Cipher Spec 메시지가 Handshake 프로토콜에 포함되지 않으므로 Finished에서 해쉬의 입력에 포함되지 않는다. 그러므로 공격자가 Change Cipher Spec 메시지를 중간에 가로채어도 Finished에서는 알 수가 없게 된다.



[그림 16] Change Cipher Spec 프로토콜



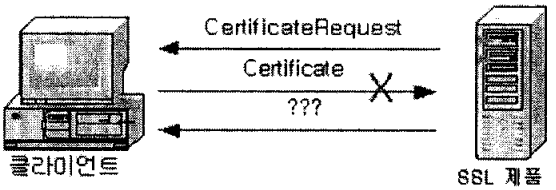
[그림 17] Change Cipher Spec 프로토콜 평가 구현

Change Cipher Spec 프로토콜 평가는 Change Cipher Spec 프로토콜에 의해서 pending state가 current state로 옮겨지는 것을 이용한다. 즉, [그림 16]과 같이 클라

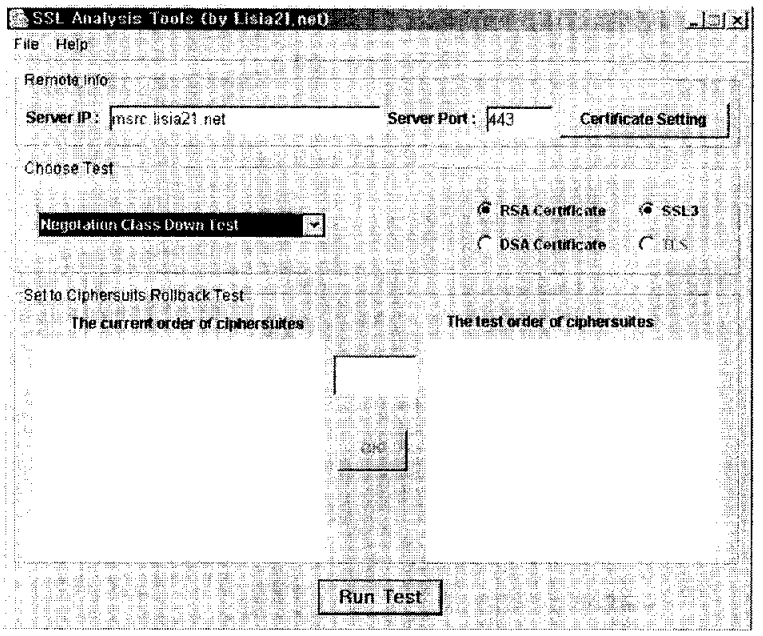
이언트에서 서버로의 Change Cipher Spec 메시지를 유실시킴으로써 current state를 NULL state로 만드는 공격에 대한 평가를 수행하도록 [그림 17]과 같이 구현하였다.

이러한 공격을 막기 위하여 SSL에서는 Finished 메시지를 검증하기 전에 Change Cipher Spec 메시지를 받게 하여 pending state에서 current state로 바꾸도록 권고하고 있으며, TLS에서는 이렇게 구현하도록 명시하였다.

다. 협정 Class 변경 평가



[그림 18] 협정 Class 변경 평가



[그림 19] 협정 Class 변경 평가 구현

협정 Class 변경 평가는 [그림 18]에서처럼 Handshake 프로토콜 수행 시 서버가 세션 설정을 위해서 클라이언트의 인증서를 요구했을 때, 클라이언트가 서버에게 인증서를 전송하지 않는 경우, SSL/TLS 서버에서 세션 설정을 유지하기 위해 클라이언트의 인증서 요구를 철회하고 세션 설정을 하는가를 평가하기 위해 [그림 19]와 같이 구현하였다.

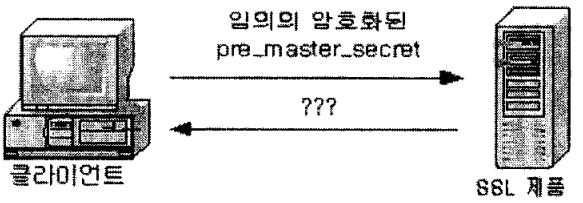
라. 선택적 암호문 공격

Bleichenbacher의 선택적 암호문 공격은 RSA 암호문중에서 PKCS#1 v1.5의 형식으로 암호화된 것에 대한 공격이다. 이 공격에 의하면 PKCS#1 v1.5의 형식으로 암호화된 RSA 암호문은 2^{-40}의 확률로 해독될 수 있다.

SSL/TLS에서는 Ciphersuite로 RSA 키 교환 알고리즘을 사용하는 경우, ClientKeyExchange에서 클라이언트가 pre_master_secret를 PKCS#1 v1.5의 형식

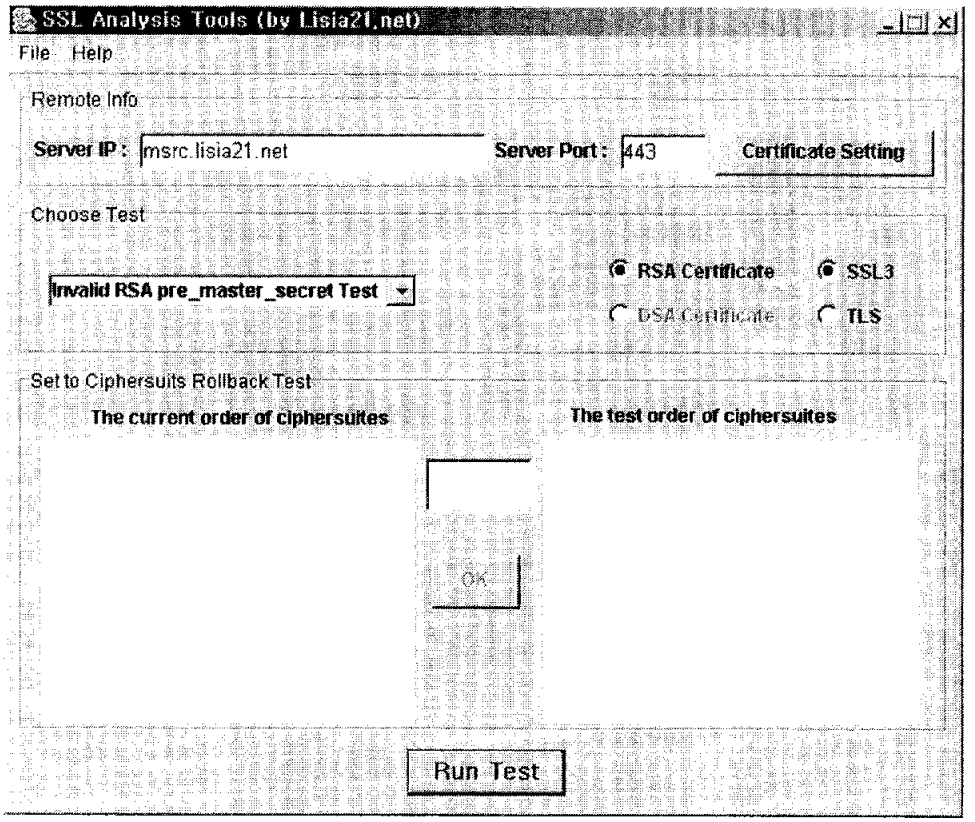
으로 RSA로 암호화하여 전송한다. 이 경우, 암호화된 pre_master_secret에 Bleichenbacher의 공격이 적용될 수 있다.

[그림 20]과 같이 임의로 잘못된 형식의 암호문을 서버로 전송하였을 때, 해당 서버가 선택적 암호문 공격에 적절히 대응하는가를 평가하도록 [그림 21]과 같이 구현하였다.



[그림 20] 선택적 암호문 공격

SSL에서는 올바른 인코딩이 아닌 암호문을 받았을 때 에러를 발생 시켜 알려줌으로써 Bleichenbacher의 선택적 암호문 공격의 가능성이 존재하며, TLS에서는 SSL의 Bleichenbacher의 선택적 암호문 공격의 가능성을 배제하기 위하여 에러를 발생시키지 않고 서버가 임의로 48바이트의 난수를 pre_master_secret로 하여 Handshake를 진행하여 공격자가 올바른 암호문인지 아닌지를 판단할 수 없도록 함으로써 선택적 암호문 공격을 방지하도록 명시하고 있다.



[그림 21] 선택적 암호문 공격 구현

3.2. Java를 이용한 안전한 SSLProxy

3.2.1. 소개

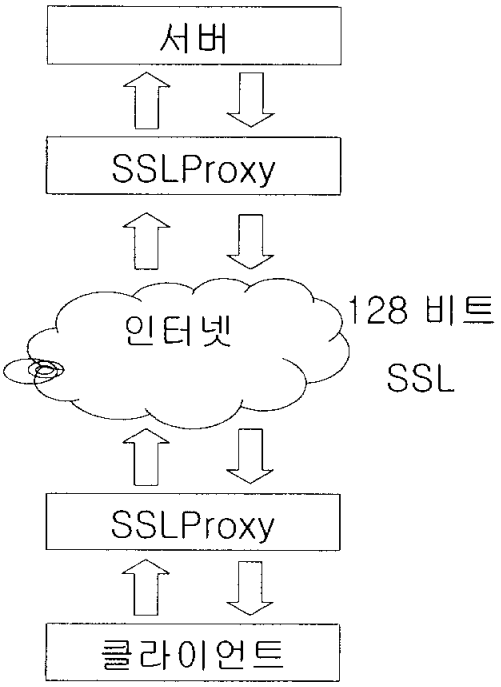
최근 기밀성, 무결성, 사용자 인증을 요구하는 정보들이 증가함에 따라서 클라이언트-서버 프로그램은 기존 비 보안 프로토콜인 TCP/IP를 이용하지 않고 SSL 프로토콜을 사용하는 경우가 많아지고 있다. 이러한 SSL 프로토콜은 기본적으로 클라이언트와 서버 사이에서 주고 받는 데이터를 암호화하여 전송할 수 있다. 하지만 일반적으로 널리 사용되어지는 대부분의 클라이언트-서버 프로그램들(TELNET, FTP등)은 근본적으로 문제점을 가지고 있다. 따라서 이러한 보안적 취약점을 원천적으로 해결하기 위해서는 수정된 SSL 통신을 할 수 있는 클라이언트-서버 프로그램을 개발해야 하겠지만, 기존에 사용되어 지고 있는 수많은 클라이언트-서버 프로그램들을 모두 포용하기 위해서는 기존의 시스템을 그대로 가지고 가면서 동시에 높은 비도수의 안전한 SSL통신을 할 수 있는 방법이 필요하다.

SSLProxy는 그러한 문제점들과 요구사항을 만족시키는 솔루션으로써 SSL을 지원하지 않는 클라이언트와 서버사이에 설치되어서 아래 [그림 22]과 같이 서로간에 128비트의 SSL통신을 가능하도록 해 준다.

많은 응용 프로그램이 C와 C++로 개발되어 보안에 적용되었으며 이에 적용되는 C와 C++ 기반의 암호라이브러리가 필요했다. 이것은 대부분의 서버 시스템들이 유닉스였기 때문에 가장 일반적인 프로그래밍 환경이 C, C++이었다. 암호 로직은 컴퓨팅 파워를 많이 요구하기 때문에 다른 언어보다 비교적 빠르다고 평가 받아 왔던 C 기반의 암호라이브러리를 사용한 순수 C 프로그램의 사용은 어쩌면 당연한 일로 여겨져 왔던 것이다.

요즘은 모바일, IC 카드, 홈네트워킹에 이르기까지 많은 신기술에도 보안이 적용되며 그 분야가 점차 넓어지고 있다. 지금의 C, C++ 기반의 응용 프로그램은 보안이 적

용되는 그 넓은 부분을 모두 수용하기에는 한계가 있으나 자바 기반의 응용 프로그램은 자바가 가지는 여러 가지 장점으로 인해 이런 신기술들의 구현 기술로 많이 채택되고 있다. 자바는 보안 개발 환경을 많이 고려하고 있어 사용하기 상당히 좋은 편이다.



[그림 22] SSLProxy 동작

요즘 서버 기술로 웹 애플리케이션 서버가 각광을 받고 있다. 이것은 말 그대로 웹상에서 애플리케이션을 실행 시켜줄 수 있는 서버로 자바를 근간으로 하고 있다. 또한 가트너 그룹에서는 엔터프라이즈 환경에 자바의 사용이 증가 할 것이며, 이러한 시장이 B2B 시장을 한동안 지원하게 되리라는 예상을 견주어 보면, 분명 본 논문에서 제안하는 자바 기반의 "안전한 SSLProxy"는 자바 진영에서 보안을 적용시키고자 할 때 큰 힘을 발휘할 것이다.

본 논문에서는 C 기반의 응용 프로그램이 자바 기반의 응용 프로그램보다 성능이

좋다거나 나쁘다거나 하는 그런 비교를 한 것이 아니다. 각 환경에 알맞은 응용 프로그램의 필요성과 요즘 많은 신기술들이 자바로 되어 있기 때문에 자바 기반의 "안전한 SSLProxy"도 그 만큼의 수요가 증가 할 것이다.

안전한 SSLProxy의 특징을 살펴보면,

- 다양한 암호화 알고리즘 지원

  국내 블록암호 알고리즘 SEED, 해쉬알고리즘 HAS-160, 전자서명 알고리즘 KCDSA 지원한다.

- 기존의 SSL 지원하지 않는 클라이언트 / 서버와 호환(Proxy 지원 가능)

  기존에 사용하던 응용 프로그램을 그대로 사용할 수 있다. SSLProxy로 클라이언트와 서버 간에 통신을 가로채어 안전한 보안 강도의 SSL통신을 제공해 주므로 소프트웨어에 Proxy 서버 설정만 가능하면 SSLProxy를 이용할 수 있다.

- 객체지향적 언어인 JAVA로 개발

  자바로 개발되어 플랫폼에 독립적이며 객체지향적 설계와 구현으로 소프트웨어의 유지보수가 쉽다. 따라서 새로운 기술과 기능을 보다 빠르게 적용시킬 수 있다.

- SSL v3.0 / TLS v1.0 지원

  SSL v2.0은 취약점이 많이 연구되고 발표되어 안전하지 않다. SSLProxy는 이러한 취약점을 보완한 SSL v3.0 / TLS v1.0을 지원한다.

- 기존 SSL/TLS의 개선(기존 취약성 개선)

  기존의 알려진 취약성들에 대해 수정 보완하였다.

3.2.2. 안전한 SSLProxy 개발 환경

본 논문에서 구현한 안전한 SSLProxy는 PureTLS 0.9b2를 사용하여 SSL 기능을 제공하고 있으며 자바 기반의 암호 라이브러리인 Cryptix, JSec[8]을 기반으로 SSL 기능을 제공하기 위한 암호, 해쉬 함수 및 Handshake를 위한 함수들을 제공하고 있다. 또한 본 논문의 구현은 순수 자바로만 이루어져 있으므로 자바의 언어적 특징인 플래폼 독립성을 그대로 가지고 있다. 평가도구의 평가 항목의 적절한 평가를 위해 PureTLS의 상당부분을 수정하여 사용하였다.

안전한 SSL Proxy의 개발 환경으로는 [표 2]와 같은 환경으로 구현하였다.

[표 2] SSLProxy 구현환경

	구현환경
서 버. 클라이언트	SSL을 지원하지 않는 서버, 클라이언트
개발환경	JDK 1.3.1_03, PureTLS 0.9b2, Cryptix 3.2, JSec, Cryptix ASN.1 kit, Java-getopt-1.0.9

본 SSLProxy는 다중 접속 처리를 위해 멀티스레드 구조를 사용하여 작성되어 있으며, 사용자 편의를 위해 GUI(Graphics Users Interface) 방식의 사용자 인터페이스를 도입하였다.

2장에서 살펴본 SSL/TLS Handshake 프로토콜의 보안 취약성을 보완하여 안전한 SSLProxy를 구현하였다.

자바 기반의 암호 라이브러리 프로바이더(CPS:Cryptographic Service Provider)를 [JAVA_HOME]/lib/security 또는 [JAVA_HOME]/jre/lib/security에 등록하고, [JAVA_HOME]/lib/ext 또는 [JAVA_HOME]/jre/lib/ext 디렉토리에 암호 라이브러리 프로바이더의 jar파일을 복사하거나 CLASSPATH에 등

록하여야 한다. 즉, SSLProxy에서 필요한 자바 기반의 암호 라이브러리 cryptix32.jar, jsec.jar, cryptix-asn1.jar, java-getopt-1.0.8.jar를 [JAVA_HOME]/lib/ext 또는 [JAVA_HOME]/jre/lib/ext 디렉토리에 추가한다.

SSLProxy에 대한 분석을 위하여 앞에서 구현한 평가 도구를 이용하여 다음 장에 그 타당성을 살펴 볼 것이다.

3.2.3. 설계 및 구현

가. 취약성 개선

앞에서 SSL/TLS의 Handshake 프로토콜 취약성 분석을 통해 살펴본 알려진 취약성을 분석하여 본 논문의 SSLProxy는 이를 보완하였다.

(1) Ciphersuite Rollbake 공격

SSLProxy는 Finished에서 master_secret를 키로 하여 Hello 메시지를 포함한 모든 Handshak 프로토콜 메시지에 대한 해쉬값을 만들어서 이전에 교환된 Handshake 메시지가 변조가 되었는지를 검증하도록 구현한다.

(2) Change Cipher Spec 프로토콜 평가

Change Cipher Spec 프로토콜의 특징을 이용하여 클라이언트에서 서로간의 Change Cipher Spec 메시지를 유실시킴으로써 current state NULL state로 만들게 된다.

이를 막기 위해 SSLProxy는 Finished 메시지를 검증하기 전에 Change Cipher Spec 메시지를 받게 하여 pending state에서 current state로 변경이 되도록 구현한

다.

(3) 협정 Class 변경 평가

Handshake 프로토콜 수행 시 SSLProxy가 세션 설정을 위해서 클라이언트에게 인증서요청 메시지를 보내어 인증서를 요구했을 때, 클라이언트가 SSLProxy에게 인증서를 전송하지 않은 경우 SSLProxy에서는 세션 설정을 유지하기 위해 클라이언트의 인증서 요구를 철회하고 기존 세션을 유지하도록 한다. 단, 본 SSLProxy에서는 변수 값을 변경함으로 협정 Class 변경 평가의 가능여부를 결정할 수 있다.

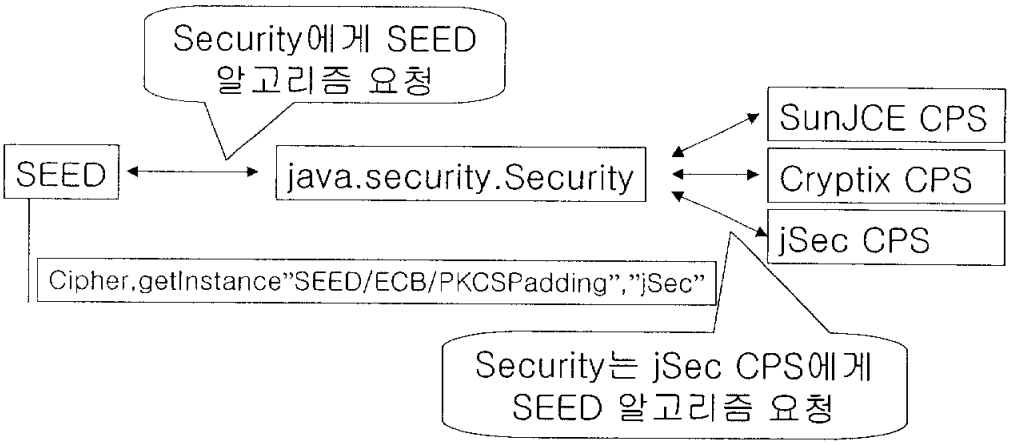
(4) 선택적 암호문 공격

SSLProxy는 선택적 암호문 공격을 이용하여 임의의 잘못된 형식의 암호문을 수신하였을 때, SSLProxy가 이 공격을 배제하기 위하여 에러를 발생시키지 않고 SSLProxy가 임의로 48바이트의 난수를 생성하여 이것을 pre_master_secret로 하여 Handshake를 진행하여 공격자가 올바른 암호문인지를 판단하지 못하도록 한다.

나. SEED 암호 알고리즘 구현

본 논문의 SSLProxy는 국내 실정에 맞는 국내 표준 알고리즘을 추가하였다.

본 논문에 기존 서비스 프로바이더(CPS)에서 지원하지 않는 국내 표준 암호 알고리즘 SEED를 지원하기 위해 JSec을 사용하였다. 다음 [그림 23]과 같이 Security에게 SEED 알고리즘을 요청하면 Security는 JSec 프로바이더에게 SEED 알고리즘을 요청하게 되고 최종적으로 JSec으로부터 SEED 알고리즘의 Cipher 인스턴스를 얻게 된다.

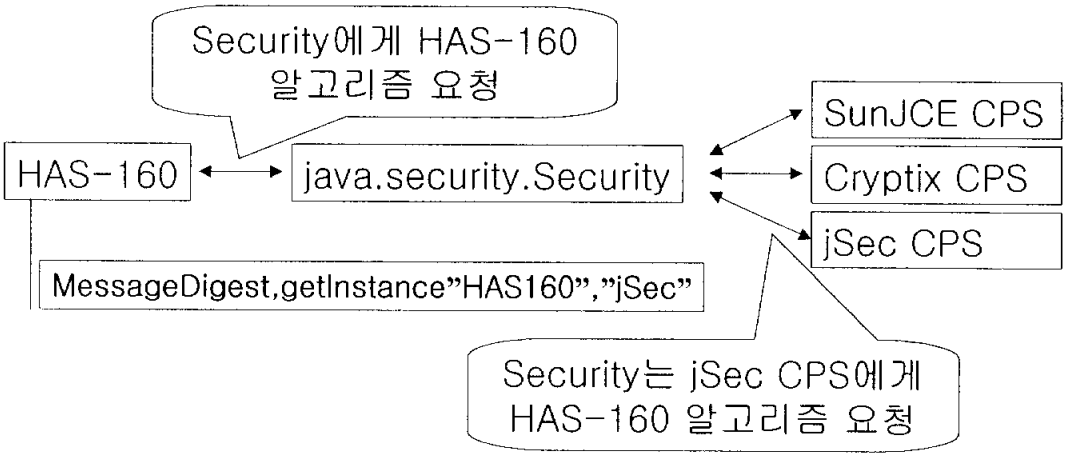


[그림 23] jSec으로부터 SEED 알고리즘의 Cipher 인스턴스 얻기

다. 국내 해쉬 알고리즘 구현

해쉬 알고리즘은 MessageDigestSpi 클래스를 상속받아 구현하였다. 위의 SEED와 동일하게 JSec 서비스 프로바이더로부터 HAS-160 알고리즘의 Message Digest 인스턴스를 얻는다.

[그림 24]는 메소드 구현을 위한 HAS-160의 전체 로직을 나타낸다.



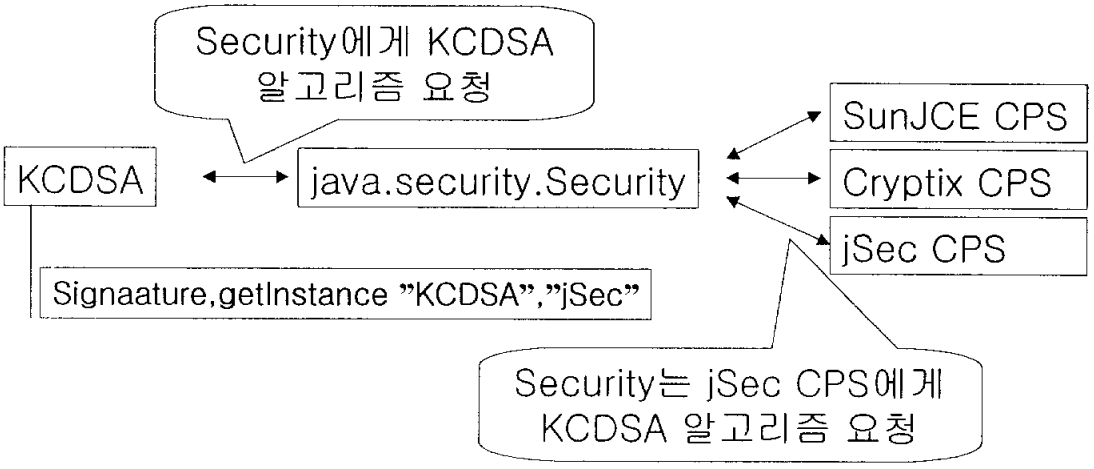
[그림 24] jSec CPS로부터 HAS-160 알고리즘의 MessageDigest 인스턴스 얻기

라. 국내 KCDSA 서명/검증 알고리즘 구현

디지털 서명 알고리즘은 SignatureSpi 클래스를 상속받아 구현하였다.

이것 또한 위의 SEED와 동일하게 JSec 서비스 프로바이더로부터 KCDSA 알고리즘의 Signature 인스턴스를 얻는다.

[그림 25]는 메소드 구현을 위한 KCDSA의 전체 로직을 나타낸다.



[그림 25] jSec CPS로부터 KCDSA 알고리즘의 Signature 인스턴스 얻기

4. 향상된 SSLProxy의 안전성 평가 및 응용

4.1. 안전성 평가도구를 이용한 평가

본 논문의 안전성 평가 도구를 이용하여 SSLProxy에 대한 기본적인 평가를 실시한다.

다음 [표 3]은 평가 결과이다.

[표 3] 평가도구를 이용한 평가 결과

평가 대상 / 평가 항목	Apache-Mod_SSL		제안 SSLProxy	
	SSL	TLS	SSL	TLS
Ciphersuite Rollback 공격 테스트	O	O	O	O
Change Cipher Spec 프로토콜	O	O	O	O
협정 Class 변경 평가	O	·	O	·
선택적 암호문 공격	X	X	O	O

※ O : 공격에 대한 정상적인 대응, X : 공격에 대한 비정상적인 대응, · : 미평가

평가 결과에 대해 분석을 해보면 기존 Apache은 기존에 알려진 취약성들에 대한 공격을 거의 대부분을 통과하고 있으며, TLS에서 선택적 암호문 공격에 대해서 스팩과 상이한 대응을 보이고 있다. 2장에서도 살펴보았듯이 Bleichenbacher의 선택적 암호문 공격의 가능성을 배제하기 위하여 에러를 발생시키지 않고 서버가 임의로 48바이트의 난수를 pre_master_secret로 하여 Handshake를 진행하여 공격자가 올바른 암호문인지 아닌지를 판단할 수 없도록 함으로써 선택적 암호문 공격을 방지하도록 명시하였으나 Apache에서는 이를 실행하지 않고 있다. 따라서 선택적 암호문 공격의 가능성을 가지고 있으며, 40비트의 낮은 키를 사용할 경우 더욱 위험해 진다.

본 논문에서 제안한 SSLProxy는 기존의 알려진 취약성들을 보완하여 다양한 공

적에도 적절히 대응하도록 구현을 하였다. 또한 국내 실정에 맞는 국내 표준 암호 알고리즘을 추가함으로 보다 다양한 성능을 지원한다.

4.2. 응용

4.2.1. SSL/TLS 안정성 평가도구

SSL/TLS Handshake 프로토콜 고유의 취약성과는 달리 구현 제품들에서의 문제점으로 인하여 보안성이 결여될 수 있다. 현재 SSL/TLS를 구현한 제품들이 다양하게 구현되어 있으나 구현과정에서 벤더나 프로그래머에 따라 SSL/TLS Handshake 프로토콜 고유의 취약성들이 다르게 표출될 수 있으므로 본 논문의 안전성 평가 도구는 이들 구현제품들의 문제점으로 인한 보안성 결여를 평가하기 위한 도구로서 응용될 수 있다.

본 논문의 평가도구는 SSLProxy 외에 다른 SSL/TLS 제품들의 보안기능에 대한 기본적인 평가를 할 수 있다. 이러한 평가 결과를 통해 전반적인 평가가 될 수는 없지만 제품에 대한 기본적인 보안성을 평가하기 위한 척도로 사용될 수 있을 것이다.

4.2.2. 안전한 SSLProxy

본 논문의 SSLProxy는 일반적인 클라이언트-서버 기반의 응용 프로그램(Proxy 지원)에 모두 응용할 수 있으며, 기존 클라이언트 및 서버의 프로그램 변경이 필요하지 않다. 또한 필요에 따라 클라이언트나 서버 한 곳에만 설치하여 보안 통신(SSL 3.0, TLS 1.0)을 수행 할 수 있다.

이러한 방법은 기존 대부분의 클라이언트-서버 기반의 응용 프로그램을 포용할 것이라 기대 된다.

5. 결론 및 향후과제

본 논문에서는 보안 프로토콜인 SSL/TLS를 구현한 제품들이 현재까지 알려진 취약성들에 대해서 어떻게 동작하는지를 평가하는 도구를 설계 및 구현하였다. 구현한 평가 도구로 SSL/TLS 제품의 보안기능에 대한 전반적인 평가가 될 수는 없지만, 기본적인 SSL/TLS 제품에 대한 보안성을 평가하기 위한 척도로 사용될 수 있을 것이다.

또한, 알려진 취약성들을 개선한 안전한 SSLProxy를 구현하였다.

향후 이러한 기본 평가 도구를 바탕으로 보다 다양한 평가 요소들을 도출하고 발생 가능한 여러 취약성들에 대한 평가 방안에 대해서도 연구되어야 할 것이다. 프로토콜 상의 취약성뿐만 아니라 해당 보안 제품이 제공하게 될 사용자의 신분확인 기능이나 무결성 기능에 대한 평가도 함께 수행되어야 할 것이다.

또한, 자바의 언어적 특성을 이용하여 모바일에 SSLProxy를 통한 보안 통신을 지원할 수 있을 것이다.

참 고 문 헌

[1] A. Freier, P.Karlton and P. Kocher, "The SSL Protocol version 3.0", Nov. 1996, InternetDraft, http://home.netscape.com/eng/ssl3/draft302.txt.

[2] T. Dierks and C. Allen, "The TLS Protocol version 1.0", Jan. 1999, RFC 2246, http://www.cis.ohio-state.edu/rfc/rfc2246.txt.

[3] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1", Advances in Cryptology-Crypto 98, LNCS 1462, Springer-Verlag, 1998, pp. 1-12.

[4] S. Cavalla, B. Dodson, A.K. Lensta, W. Loien, P.L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, P. Leyland, J. Marchand, F. Morain, A. Mu.ett, C. Putnam, C. Putnam and P. Zimmermann, "Factorization of a 512-bit RSAmodulus", Advances in Cryptology-Eurocrypt 2000, LNCS 1807, Springer-Verlag, 2000, pp. 1-18.

[5] Eric Rescorla for Claymore Systems, Inc, "PureTLS", http://www.rtfm.com/puretls/.

[6] Cryptix Development Team, "Cryptix", http://www.cryptix.org/

[7] Eric Rescorla, "SSLDump" , http://www.rtfm.com/ssldump/

[8] Open Source 그룹, "JSec", http://osg.javaland.co.kr/

감사의 글

어느덧 이곳에서의 2년이라는 시간이 훌쩍 지나 버렸네요..

그 2년 동안이 저에게는 소중히 간직해야할 많은 추억과 지식, 또 말로 다 표현하지 못할 만큼 값진 시간이었습니다.

이렇게 논문을 마무리를 하다보니 처음 제가 연구실에 들어온 날이 기억납니다.

낯선 공간, 낯선 사람들 사이에서 책상에 앉아 컴퓨터만 보던 어색함...

그렇게 컴퓨터만 마주하고 지내던 몇일간의 생활을 바꿀 수 있었던 현호선배의 한마디!!

"자 우리 커피한잔 하러갑시다"

라는 말과 함께 처음으로 연구실 식구들과 커피를 마시던 생각이 나네요..

그 이후로 거의 하루 평균 커피를 7잔씩이나 마셔댔지만 말이죠...

어색했던 시간들.. 그리고 연구실 사람들의 마음씀씀이가 느껴지는 많은 배려들, 많은 다짐을 하고 또 했던 나의 모습....

지난 2년 동안 있었던 행복했던 일과 힘들었던 많은 일들은 저의 마음을 가득 채워주는 소중한 기억이 될 것입니다.

오랜 시간이 지나 많은 기억들이 바래지는 시간이 오더라도 소중한 추억으로 따뜻해진 마음만은 변함없이 기억될 것입니다

제가 가고 나면 새로운 사람이 그 자리를 저보다 더 훌륭히 메워주겠죠.

지금 까지 많은 도움을 주신 모든 분들에게 감사의 마음을 전합니다.

많이 부족한 제자를 따뜻한 마음과 가르침으로 이끌어주신 이경현 교수님, 연구실의 작은 거인 신원선배, 연구실 짱(^^) 준석선배, 연구실생활과 가정 일까지 바쁜 수정선배, 성실 = 정화선배, 저를 많이 도와 주고 많이 괴롭혔던(?) 종필선배, 말없이 도움

을 줬던 영호선배, 이제 교수님(^^) 현호교수님(?), 동기 ? 선배 = 서철선배, 2년동안 동고 동락한 동기 명진, 희연, 1년동안 많은 도움 주지 못한 것 같아 아쉬운 성렬, 영경 이제 연구실의 가장 중요한 구성원으로써 잘 하기를..., 연구실 새내기들 지원, 미선, 인경 모두 열심히 해서 좋은 결과 있기를, 올해까지 연구실 막내로 지내다 좋은 직장을 얻은 종금, 지금의 막내 화경, 경섭선배를 비롯한 센추리 식구들, 영원히 친구이면서 끗빨(?)이 딸리는 춘, 연구실 공부하랴! 일하랴! 수고가 많으신 산·교대 분들 모두 모두 감사의 말을 전합니다.

그리고 지금까지 다짐에 다짐을 새로이 하며 열심히 해온 저 자신에게도 감사하다는 말과 저의 우둔함을 옆에서 미소로 지켜주었던 은경에게 사랑한다는 말과 함께 항상 성실히 노력하겠다는 저와의 약속을 남기며 마무리를 하려고 합니다.

ps : 모든 분들의 이름 다 적지 못하는 점 이해해 주세요.