通信情報保護學會誌
第7卷 第1號. 1997. 3

WWW 보안

박 현 동*, 류 재 철*

요 약

WWW(World Wide Web)은 인터넷에서 가장 널리 사용되고 있는 서비스 중의 하나로써 인터넷 초기에는 일부의 계층에서 주로 교육, 연구용으로 사용하여 왔으나, 현재는 일반 사용자 계층에까지 파고 들고 있다. 일반 사용자들의 참여로 인해 이제는 WWW가 상업용으로까지 이용되고 있는 실정이므로 안전한 WWW 통신의 구현이 필수적이다. 본 고에서는 안전한 WWW구축에 필요한 보안 요구사항들을 살펴 보고, WWW에서 기본적으로 제공되는 보안 메카니즘과 현재 연구 개발중인 보안 기법들을 소개한다.

1. 서 론

WWW은 1990년 Tim Berners-Lee에 의해 처음으로 제안되었다. WWW은 처음에 많은 관심을 받지 못하다가 사용자 인터페이스를 편리하게 설계하고 여러가지 형태의 자료를 손쉽게 보여 주는 브라우저(Browser)의 출현으로 인하여 널리 퍼져 나가기 시작하였다. Mosaic, Netscape와 같은 WWW의 편리한 사용자 인터페이스는 전문가들뿐만 아니라 일반인들의 인터넷 사용을 증가시켰고, 결과적으로 인터넷 전체의 크기를 증가시키고 있다[1]. 인터넷은 초기에는 전문가들이 주로 교육, 연구용으로 사용하였고, 그 결과로 인터넷에는 수많은 정보들이 저장되게 되었다. 정보를 찾아 사람들이 인터넷에 모이게 되었고, 결국은 WWW을 도구로 하여 인터넷을 상업적으로 사용하려는 시도가 이미 시작되고 있다.

* 충남대학교

인터넷에는 많은 연구 자료들이 저장되어 있는데, 이들 중에는 대외비의 성격을 지닌 것들도 상당수에 달한다. 어느 특정한 집단 안에서만 정보를 공유하고 그 외의 사용자들에게는 열람을 허락치 않는 정보의 접근제어가 필요하게 된 것이다. 이러한 선별적인 정보의 공유뿐만 아니라 현재 외국의 많은 회사들이 WWW을 사용한 가상 상점을 개장하여 영업중에 있다. 상점을 운영하는 입장에서 보면 WWW은 매우 매력적인 상점 운영 방식이다. 실세계에서 상점을 운영할 때에 필수적인 상점 건물 등이 필요하지 않고, 광고비도 절감할 수 있다. 또한 구매자의 입장에서도 쇼핑의 시간적 제약이나, 공간적 제약이 없다는 점에서 사용이 늘어날 것으로 보이고 있다. 이러한 이유들로 인해 앞으로 인터넷의 가상 상점의 수는 그 수가 기하급수적으로 늘어날 것이라는 예측을 하고 있다. 현재 운영되고 있는 가상 상점들 중에는 시범적으로 운영되는 것도 있지만, 대다수의 상점들이 구매자의 신용카드를

지불수단으로 하여 네트워크를 통해 지불 서비스까지 제공해 주고 있다[2].

하지만, WWW의 근간인 인터넷이 사용하고 있는 TCP/IP는 본래 개방형 시스템을 기반으로 설계되었다. 그래서 WWW 또한 보안의 측면에서는 취약한 것으로 인식되고 있다. 일반 사용자들의 상거래 수단으로까지 이용되고 있는 WWW의 보안 문제는 더이상 간과할 수 없는 사항임에 따라, 본 고에서는 WWW 보안 관련 요구 사항들을 살펴 보고자 한다.

2장에서는 시스템 보안과 안전한 WWW의 사용을 위해 요구되는 보안 요구 사항들을 살펴 보고, 기존의 HTTP에서 지원해 주고 있는 사용자 인증 방법 등을 소개한다. 3장에서는 암호 알고리즘을 도입하여 강력한 사용자 인증과 암호 통신을 하는 기법들을 살펴 보고, 마지막으로 4장에서 결론을 맺고자 한다.

2. WWW 보안 요구사항과 현재 사용되고 있는 방법

2.1. 시스템 보안

WWW에서 사용되는 브라우저와 서버에는 다음과 같은 보안상의 취약점이 있으므로, 사용자와 서버 관리자는 사용을 주의해야 한다.

- 브라우저

브라우저를 사용하는 시스템은 불법 침입을 당할 수 있다. 이는 브라우저 자체의 문제라기보다는 브라우저가 처리해 주지 못하는 형태의 문서를 처리해 주는 ghostscript이나 MPEG player와 같은 external viewer의 security hole에서 발생한다고 본다. 예를 들어 ghostscript이 잘못 구성되어 있으면 PostScript 화일이 브라우저가 있는 컴퓨터에서 임의의 명령을 실행시킬 수 있다. 이 밖에 생각할 수 있는 것은 브라우저 프로그램의 크기이다. 모든 프로그램은 크기가 클 수록 버그를 가지고 있을 확률이 크다. 브라우저는 매우 큰 프로그램이기 때문에, 사용자가 발견하지 못하는 버그를 가지고 있을 것으로 예상되고, 이 버그를 통하여 매우 심각한 보안 취약점이 발생할 수 있다. 이미 발견된 버그는 URL의 길이에 관한 것이다. 만약, 사용자가 356자가 넘는 스트링을 URL로 입력하면 버퍼가 오버플로우를 일으켜 초과부분이 컴퓨터의 stack에 옮겨져 program code처럼 실행될 수 있어서 컴퓨터에 문제를 발생시킬 수 있다.

- 서버

Web의 서버와 클라이언트를 비교해 볼 때 클라이언트보다는 많은 정보를 소유하고 있는 서버가 공격의 대상이 될 확률이 더욱 높다. 하지만, 서버는 클라이언트에 비해 크기가 훨씬 작기 때문에 버그가 발생할 경우도 작다. 하지만 브라우저에서 발견된 "string overflow"는 서버에서도 발생하므로 침입자는 긴 URL의 끝에 program code를 붙여 서버가 있는 컴퓨터에서 실행시킬 수 있다.

서버에서의 가장 큰 보안 취약점은 CGI의 수행에서 발견된다. 만약, 전자우편을 서버에 보내는 CGI 프로그램일 경우, 프로그램 코드안에 system("/usr/lib/sendmail -t $USER < input_file") 과 같은 라인이 존재하게 된다. 클라이언트는 USER변수에 USER = "what @wherever.com < /dev/null ; rm -rf /" 와 같은 값을 주면, system명령이 수행되면서 먼저 지정된 주소로 빈 메일을 보내고, 두번째 명령인 rm을 수행하게 된다. 이와 같은 문제는 WWW 데몬의 UserID를 권한이 극히 제한된 user로 지정함으로써 피해를 최대한 줄일 수 있다.

그러나 이와 같이 브라우저와 서버에서 발생할 수 있는 보안의 문제점들은 보고될 때마다 그 문제점을 해결한 새로운 버젼들이 나오

고 있다. 서버와 브라우저에서 발생하는 버그는 이 프로그램을 제작한 제작팀 이외의 사람들은 수정하기가 어렵고, Netscape등 일부 프로그램은 소스가 공개되지 않고 있기 때문에 최신정보를 입수하여 새로운 버젼의 프로그램을 사용하는 방법으로 위험의 부담을 줄인다.

2.2. WWW 보안 요구사항

앞 절에서 기술한 시스템 보안과 함께 WWW의 안전한 사용을 위해서는 여러가지 보장되어야 할 요소가 있다. 이 절에서는 WWW의 안전한 통신을 보장해 주기 위해서 필수적인 요소로 다음과 같은 것을 정의한다[3].

- HTTP 트랜잭션의 기밀성 (Confidentiality)

WWW의 통신에서 클라이언트 사용자와 서버만이 통신되는 내용을 파악할 수 있도록 하는 기능을 말한다. 클라이언트와 서버 사이의 통신 내용을 제3자가 중간에서 가로챈다 하더라도 그 내용을 알아볼 수 없게 해야 한다. 이를 위해서 HTTP 메세지를 암호화하는 방법이 필요하다. 클라이언트의 request와 서버의 response에 포함되는 HTTP헤더들과 본문의 내용을 암호화한 후에 수신자를 나타내는 기본적인 헤더만을 암호문에 붙여 주는 방법이다.

- 서버와 클라이언트에 대한 인증 (Authentication)

HTTP 트랜잭션이 발생하기 이전에 클라이언트는 서버를, 서버는 클라이언트의 신원을 확인해 줄 수 있게 하는 기능이다. 상대방에 대한 인증을 상호 가능케 함으로써 서버는 적합한 사용자에게만 문서를 볼 수 있게 하고 클라이언트는 위장된 서버로부터 문서를 전달받는 상황을 방지할 수 있다. 이는 어느 특정한 메세지에 각자의 비밀키를 이용한 전자서명(Digital Signature)을 붙여 줌으로써 구현이 가능하다.

- HTTP 트랜잭션의 무결성(Integrity)

서버는 클라이언트의 request가 전송 도중에 불법적인 변경이 없이 제대로 도착했는 가를 확인할 수 있어야 하고, 반대로 클라이언트는 서버가 보내준 response가 전송 도중에 불법적인 변경이 없이 제대로 도착했는 지의 여부를 확인할 수 있어야 한다. 이는 request나 response를 전송할 때에 이들 메세지의 해쉬함수(Hash Function) 값을 함께 보냄으로써 수신자가 메세지를 받아 다시 계산한 해쉬값과 송신자가 계산해 보내 준 해쉬값을 서로 비교함으로써 구현할 수 있다. 두 값이 서로 일치하면 전송 도중에 아무런 문제가 발생하지 않았다는 의미가 되고, 서로 일치하지 않는다는 것은 메세지가 전송 도중에 어떤 형태로든지 변경되었다는 것을 의미한다.

위에서 살펴 본 바와 같이 WWW 보안에서는 서버와 클라이언트에 대한 인증, 전송된 메세지에 대한 기밀성, 무결성 등을 요구하고 있다. 다음 절에서는 현재 사용되고 있는 WWW 보안 방법들을 살펴 봄으로써 위의 요소들 중 현재의 HTTP가 지원해 주고 있는 것은 어떤 것들이며, 지원해 주지 못하고 있는 것은 어떤 것들인지를 알아 보도록 한다.

2.3. 현재 사용되는 WWW 보안

앞의 절에서 WWW의 안전한 사용에 필수적인 요구사항들을 보았지만 현재의 WWW은 이러한 요구사항들 중 메세지의 기밀성과 무결성은 지원해 주지 못하고 있다[4]. 클라이언트에 대한 인증은 어느 정도는 지원해 주고 있으나, 이 방법들도 각각의 약점이 있어 좋은 방법이라 할 수는 없다. 대개의 브라우저 사용자들은 신원을 밝히지 않은 상태로 서버에 접근하여 문서들을 열람할 수 있고, 서버의 신원

을 확인해 줄 수 있는 확실한 방법이 없어서 클라이언트들은 의도하지 않은 다른 서버에서 페이지를 전송해 주어도 이를 확인할 방법이 없다. 결과적으로 서버쪽에서는 적절한 접근 제어를 구현하기가 쉽지 않고, 메세지는 평문의 형태로 전송되기 때문에 기밀성이나 무결성도 기대할 수 없다. 보안 요구사항들 중에서 접근제어를 위해 제공되고 있는 방법들을 보면 다음과 같다.

(1) 사용자ID와 패스워드를 이용한 Basic Authentication

HTTP 1.0 버젼부터 제공되어진 클라이언트 인증방법이다. HTTP에서 지원하기 때문에 HTTP를 사용하는 모든 브라우저와 서버 사이에서 이용할 수 있다. 이 방법을 사용하기 위해서는 몇 개의 화일을 작성해 주어야 한다. 필요한 화일들은 ".htaccess", ".htpasswd", ".htgroup"이다. 각 화일들의 내용을 보면, ".htaccess"는 접근제어를 요구하는 디렉토리마다 하나씩 존재한다. 이 화일에는 해당 디렉토리에 접근을 허용하는 사용자ID의 리스트, .htpasswd화일의 위치 등을 명시해 준다. 내용을 보면 다음과 같다.

```
AuthUserFile /otherdir/.htpasswd
AuthGroupFile /dev/null
AuthName ByPassword
AuthType Basic

〈Limit GET〉
require user hdpark tgkim
〈/Limit〉
```

의미를 보면, 제일 윗줄의 "AuthUserFile"에는 사용자들의 패스워드를 저장하고 있는 화일의 패스를 지정해 준다. ".htpasswd"에는 각 사용자들의 등록된 로긴네임과 암호화된 패스워드가 저장되어 있다. 접근이 허용된 사용자들의 수가 클 경우, 관리의 효율성을 위해 ".htgroup"화일을 만들어 이곳에 모든 사용자 ID를 저장하고, .htaccess화일에는 .htgroup화일이 있는 곳의 패스를 지정해 준다. 사용자ID를 모두 저장하고 있는 화일을 지정해 준 곳이 두번째 줄의 "AuthGroupFile"이다. "AuthName"의 "ByPassword"는 접근제어 방식으로 패스워드를 사용하고 있다는 것을 나타낸다. "AuthType"은 이 방법이 "Basic Authentication"이라는 것을 나타낸다. "〈Limit GET〉"은 이 접근제어 방식은 GET method를 가진 request에 해당한다는 것을 나타낸다. "Require user"는 패스워드 검사를 통해 접근이 허용되는 사용자들의 ID리스트를 지정해 놓는다. 위의 예에서는 hdpark과 tgkim이 접근이 허용되고 있다. Hdpark과 tgkim은 올바른 패스워드를 입력함으로써 해당 디렉토리에 있는 자료들에 대한 열람이 허용된다.

Basic authentication의 인증 절차를 보면 다음과 같다.

① 클라이언트는 접근이 제어되는 화일에 인증 자료 없이 request를 보낸다.

```
GET /protected/secrets.html HTTP/1.0
Accept: */*
User-Agent: NCSA__Mosaic/2.7
```

② 서버는 사용자에게 Basic authentication이 필요하다는 사실과 401 에러 메세지를 보낸다.

```
HTTP/1.0 401 Unauthorized
Date: Mon, 16 Oct 1995 04:12:49 GMT
Server: NCSA/1.4.2
Content-type: text/html
WWW-Authenticate: Basic realm=
"SuperSecret"
```

〈HEAD〉〈TITLE〉Authorization Required〈/TITLE〉〈/HEAD〉
〈BODY〉〈H1〉Authorization Required〈/H1〉
Browser not authentication-capable or authentication failed.〈/BODY〉

③ 브라우저 사용자는 자신의 사용자ID와 패스워드를 추가해서 다시 request를 보낸다.

GET /protected/secrets.html HTTP/1.0
Accept: */*
User-Agent: NCSA__Mosaic/2.7
Authorization: Basic zm51cmc6eWFnz2EteWfnZ2E=

④ 서버는 사용자ID와 패스워드를 검사하여 적합한 사용자로 판명되면 요청한 문서를 전송해 준다.

HTTP/1.0 200 Document follows
Date: Mon, 16 Oct 1995 04:13:05 GMT
Server: NCSA/1.4.2
Content-type: text/html
Content-length: 227
--- Document starts here ---

하지만, 이 방법이 완벽한 접근제어를 구현해 준 것은 아니다. 클라이언트가 입력한 패스워드는 암호화되지 않고 단지 uuencode된 형태로 서버에까지 전송되기 때문에 도중에 네트워크 상에서 패스워드가 노출될 수 있다. 패스워드가 노출되면 접근권한이 없는 사람이 그 패스워드를 사용하여 적합한 사용자인 것처럼 가장하여 서버에 접근할 수 있다. 또한 등록되어 있는 사용자의 수가 너무 크거나, 이들의 패스워드를 자주 변경해 주어야 할 경우에는 서버 관리자에게 너무 큰 오버헤드가 주어지는 문제점이 발생한다.

(2) 네트워크 주소를 이용한 접근제어

사용자 인증을 위해서 HTTP에서 제공하는 또 하나의 방법으로 클라이언트의 네트워크 주소를 이용한 "IP filtering" 방법이 있다. 이 방법을 사용하기 위해서는 ".htaccess" 화일에 사용자ID가 아닌 클라이언트의 주소를 지정해 줌으로써 접근을 허용하거나 거부할 수 있다. 만약, ".chungnam.ac.kr"이라는 도메인에서 오는 request를 허용하려면 ".htaccess"화일에 "allow from chungnam.ac.kr"이라고 지정해 주고, 거부할 경우에는 "allow" 대신에 "deny"를 사용한다.

```
AuthUserFile /dev/null
AuthGroupFile /dev/null
AuthName ExampleAllowFromChungNam
AuthType Basic

〈Limit GET〉
order deny allow
deny from all
allow from .chungnam.ac.kr
〈/Limit〉
```

사용자의 ID나 패스워드를 필요로 하지 않기 때문에 "AuthUserFile"과 "AuthGroupFile"에는 아무런 패스도 지정되지 않았다. "Deny from all"과 "Allow from .chungnam.ac.kr"에서 chungnam.ac.kr에서 오는 request만을 허용하고 그 이외의 다른 곳에서 오는 request는 거부한다는 것을 알 수 있다.

이 방법은 사용자ID와 패스워드가 네트워크를 통해 전송되지 않아도 된다는 점에서 더욱 간단하고 안전하다고 볼 수도 있다. 하지

만, 대개의 해커가 자신의 IP주소를 변조할 수 있다는 점을 감안할 때에는 안전하다고는 할 수 없는 방법이다. 또한 이 방법은 접근제어가 브라우저 프로그램이 수행되는 컴퓨터 단위로 구현되었기 때문에 진정한 사용자별 접근제어는 기대할 수 없다.

(3) 패스워드 검사와 네트워크 주소를 병합한 접근 제어

접근제어를 하는 대부분의 서버는 패스워드를 검사하는 방법과 네트워크 주소로 접근을 제어하는 두가지 방법을 혼합하여 사용하고 있다. "chungnam.ac.kr"에서 오는 request들 중에서 hdpark과 tgkim에 대해서만 패스워드를 검사하여 접근을 허락하고, 그 이외의 도메인이나 "chungnam.ac.kr"에 있는 사람일지라도 hdpark과 tgkim 이외의 사람들에게는 접근을 허용하지 않는다" 이를 표현하는 .htaccess화일의 구성은 다음과 같다.

```
AuthUserFile /otherdir/.htpasswd
AuthGroupFile /dev/null
AuthName ExampleAllowFromChungNam
AuthType Basic

    〈Limit GET〉
    order deny allow
    deny from all
    allow from .chungnam.ac.kr
    require user hdpark tgkim
    〈/Limit〉
```

두 가지 방법을 혼합하여 좀 더 안전한 인증을 꾀하고 있지만, 패스워드가 평문으로 전송된다는 약점과 IP주소가 변경될 수 있다는 약점을 근본적으로 고치지 못한 방법이기 때문에 좋은 해결책은 될 수가 없다. 확실한 사용자 인증과 접근제어를 구현하기 위해서는 공개키 암호 알고리즘의 전자서명에 의한 사용자 인증 등이 사용되는 새로운 방법이 필요하다.

(4) Log 화일

서버 프로그램에는 logs라는 디렉토리가 존재하는데, 그 밑에는 access.log와 error.log라는 화일이 존재한다. access.log에는 서버에 access하여 성공한 request가 기록되고 error.log에는 access에 실패한 request가 기록된다. access.log에 기록된 내용을 보면 다음과 같다.

```
〈ngrims.kepcorc.re.kr--[08/Nov
/1995:14:41:53 +0900] "GET /security/
title.html HTTP/1.0" 200 325〉
```

이 기록에 의하면, ngrims.kepcorc.re.kr이라는 호스트에서 1995년 11월 8일 오후 2시 41분에 서버에 접근하여, security/title.html화일을 열람했다는 것을 알 수 있다.

Error_log에 남는 기록의 내용은 다음과 같다.

```
〈[Wed Nov 8 14:37:09 1995] httpd: access
to /home/user/www/httpd/htdocs/ auth3/
tip.html failed for 168.188.66.84, reason:
client denied by server configuration〉
```

이 기록은 1995년 11월 8일 오후 2시 37분에 누군가가 /home/user/www/httpd/htdocs/auth3/tip.html문서에 접근하려다가 서버가 요구하는 configuration에 맞지 않아서 접근이 거부되었던 것을 알수 있다.

Access.log와 error.log를 사용하여 서버의 이용상태 정보를 알 수 있고, 만약 서버에 문제가 발생했을 경우에, 이 log 화일들을 통하여 침입에 대한 증거를 확보할 수도 있다. 하지만, 이 화일들이 불법침입과 같은 사고를 미리

방지해 줄 수는 없고, 또한 위의 형태와 같은 자료들이 화일에 계속 누적되어 쌓이므로 나중에 문제가 발생하여 화일들을 조사할 때에 조사자에게 편리한 인터페이스를 제공해 주지 못한다. 또한, 접근이 빈번하게 일어나는 서버의 경우에는 이 화일들의 크기가 매우 빠르게 커지기 때문에 이 화일들의 관리 자체가 큰 문제점으로 남게 된다.

3. 현재 연구중인 WWW 보안

2장에서는 WWW의 안전한 통신을 위해 보장되어야 할 요소들과 현재 사용되고 있는 WWW의 보안 방법들을 알아 보았다. 하지만 2장에서 알아 본 사용자 인증 방법들은 본격적인 보안 요구사항들을 만족시켜 주지 못하고 있다. 보안 요구사항들을 만족시켜 주기 위해선 현재의 HTTP에 암호 알고리즘을 추가시키는 방법이 필요하다. 암호 알고리즘을 사용하는 것만이 문서의 기밀성과 사용자 및 서버의 인증을 확실히 보장해 줄 수 있는 방법이다. 하지만 이와 같이 WWW에 암호 알고리즘을 추가시키는 것이 매우 어려운 작업으로 간주되고 있는 데 그 이유는 다음과 같다[4].

- 새로운 표준이 제정되어야 할 정도로 HTTP와 HTML의 확장이 필요하다.
- 브라우저와 서버 프로그램 자체에 암호 알고리즘을 추가시켜야 한다.
- 현존하는 암호 알고리즘들은 심각한 취약점이 발견될 수 있으므로 앞으로 개발될 암호 알고리즘들까지 수용할 수 있는 서버와 브라우저의 개발이 필요하다.
- 몇몇 나라에서는 암호 기술의 수출입을 제한하고 있어서 암호 알고리즘의 사용이 자유롭지 못하다.
- WWW에 암호 기술을 추가하기 전에 모든 서버와 모든 사용자들의 키 인증을 전담하고 관리할 기구의 설립 등 WWW이외의 다른 요소가 미리 설계되어야 한다.

위와 같은 이유에도 불구하고, WWW의 인기로 인하여 안전한 사용을 위해 암호 기술을 WWW에 접목하는 여러 기술들이 소개되고 있다. 다음 절에서는 현재 연구 중에 있는 HTTP에 암호 알고리즘을 추가하는 방법들을 알아 본다.

3.1. Secure HTTP

1994년 EIT는 HTTP에 보안요소를 첨가한 Secure HTTP를 발표하였다. Secure HTTP는 범용으로 사용될 수 있도록 설계되었으며 트랜잭션의 기밀성, 인증, 무결성 등을 지원해 준다. 응용 레벨에서의 메세지 암호화를 통해 안전한 트랜잭션을 보장해 주고, RSA Data Security사의 공개키 암호 알고리즘을 이용하여 서버와 클라이언트가 공유하여야 하는 정보(세션키) 등을 암호화하여 전송한다. Secure HTTP에 대한 간단한 설명은 다음과 같다[5].

□ 목 적 : WWW의 상업적 이용에 대비하여 HTTP 서버와 클라이언트 사이의 안전한 통신 메카니즘을 제공하기 위하여 제작되었다.

□ 특 징

- 클라이언트와 서버에서 행해지는 암호화 동작이 동일하다. 즉, request와 response에 같은 암호화 처리를 한다.
- HTTP가 가지고 있는 트랜잭션의 형태를 변화시키지 않고 HTTP의 특성은 모두 보존하고 있다.
- PKC-7, PEM, PGP등 여러 암호 메카니즘에 사용되는 다양한 암호문 형태를 지원한다.

- 선택 협상(Option Negotiation) 과정이 있기 때문에 획일적인 암호화를 지양하고 다양한 암호 알고리즘, 모드, 파라메터의 사용에 융통성을 제공해 주고 있다.

□ 암호화 동작의 종류

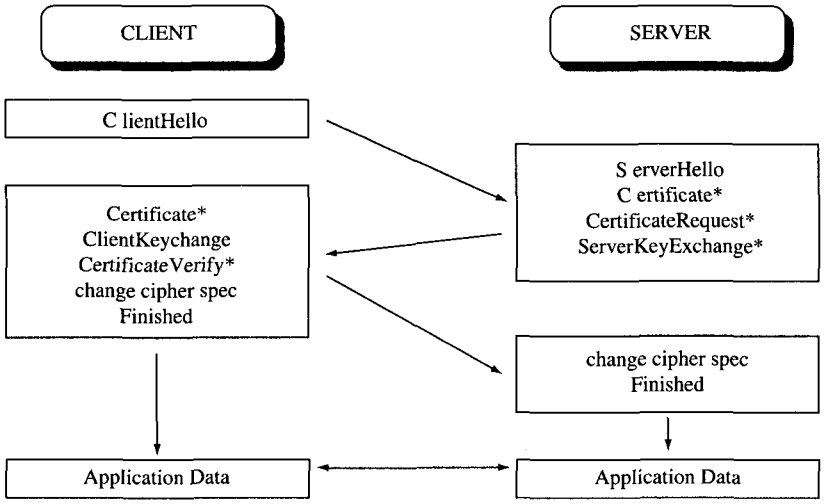
- 기본적인 3개의 동작 : 서명(Signature), 인증(Authentication), 암호화(Encryption) 등의 동작이 서로 독립적으로 수행된다.
- 서 명 : 서명은 메세지에 붙어서 전송되는 방법이 있고, 메세지와는 독립적으로 따로 전송되는 방법이 있다.
- 키 전송 방법 : 암호화에 사용된 키의 전송 방법에는 암호문과 함께 보내는 방법과 암호문과는 다른 채널로 보내는 두 가지 방법이 있다.
- 인 증 : 메세지의 무결성을 확인하기 위해서 MAC(Message Authentication Code)을 사용한다.

□ 프로토콜

- Secure HTTP 메세지는 request나 status line의 뒤에 RFC-822 스타일의 헤더들과 encapsulated된 내용들을 포함한다.
- "HTTP/1.0" 대신에 "Secure-HTTP /1.1"이라는 프로토콜 지정어를 사용한다.
- "Secure * Secure-HTTP/1.1"이라는 새로운 request method를 사용한다.
- Request의 성공/실패 여부에 관계없이 response의 첫 라인은 "Secure-HTTP /1.1 200 OK"가 사용된다.
- Secure HTTP에는 다음과 같은 헤더들이 있다.
  - Content-Privacy-Domain : PEM, PKC-7,...
  - Content-Transfer-Encoding : BAS-E64, 8BIT, 7BIT,...
  - Content-Type : application/http
  - Prearranged-Key-Info : Inband, krb-{4,5}, outband
  - MAC-Info
- Negotiation : 서버와 클라이언트의 암호 알고리즘 구현 능력이나 어플리케이션에서 필요로 하는 강도에 따라서 암호 알고리즘의 종류를 선택할 수 있다. Negotiation의 결과는 암호 알고리즘의 종류와 모드, 전송방향 등으로 표현되는데, 예를 들면 "You may use DES-CBC when encrypting", "I insist on using DES-ECB when encrypting" 처럼 표현될 수 있다.
- 기존의 URL 형태인 "http://" 대신에 "shttp://"라는 새로운 URL형태가 사용된다.
- 추가적으로 DN, NONCE, CRYPTOPTS와 같은 새로운 anchor들이 필요하다.

3.2. Secure Socket Layer(SSL)

SSL은 Netscape Communications사에서 개발한 것으로 어플리케이션 프로토콜과 TCP/IP 사이에 위치하며 데이타의 암호화, 서버의 인증, 메세지의 무결성을 제공해 주고 있다. 서버에 대한 인증은 반드시 수행되지만 클라이언트에 대한 인증은 선택적으로 수행할 수 있는 방법을 제공해 준다. SSL은 서버와 클라이언트 양쪽의 TCP/IP연결을 위해서 "handshake" 프로토콜을 수행한다. 이 결과로 양쪽은 암호화 통신에 합의하고, 암호화 통신과 인증에 필요한 값들을 준비한다. 이 단계가 지나면 SSL은 어플리케이션 프로토콜에서 생성해 낸 bytestream의 암호화와 복호화만 수행하게 된다. 이는 HTTP request와 HTTP response에 포함되는 모든 정보들(URL, form의 데이타, access 인증자료,...)이 암호화되어 전송된다는 것을 의미한다. Handshake 프로토콜을 그림으로 나타내면 〈그림 3-1〉과 같다[6].



* 표는 항상 필요한 내용이 아니라 상황에 따라 필요한 때에만 보내지는 내용을 뜻한다.

〈그림 3-1〉 Handshake 프로토콜

〈그림 3-1〉에서 클라이언트가 서버와 암호화된 통신을 위해 먼저 "ClientHello" 메세지를 전송한다. 서버는 이에 대한 대답으로 "ServerHello"메세지를 전송한다. 만약, 서버가 자신을 인증할 수 있는 정보인 "Certificate"을 소유하고 있다면 이 메세지도 함께 보낸다. 또한 서버는 클라이언트를 인증하고 싶으면 "CertificateRequest"를 전송함으로써 클라이언트에게 인증자료를 요청한다.

클라이언트는 서버가 전송해 준 메세지를 수신한 후, 서버가 클라이언트의 인증을 요구하면 자신을 인증할 수 있는 "Certificate"를 전송한다. 통신에 사용할 공개키 알고리즘을 지정하는 정보는 "ClientKeyExchange"에 포함된다. "change cipher spec"에는 application data의 암호화 통신에 이용될 암호화 정보들이 포함되어 있다. 이 "change cipher spec"은 현재 지정되어 있는 cipher spec으로 암호화되어 전송된다. "change cipher spec"을 수신한 서버는 현재의 cipher spec을 갱신하여 application data의 암호화를 수행한다.

SSL은 port번호 443번을 사용하고 같은 서버에서 안전한 서버와 안전하지 않은 서버를 모두 운영할 수 있다. 예를 들면, 상점의 경우에 상품을 소개하는 데이타는 반드시 안전해야 할 필요가 없으므로 안전하지 않은 서버에서 운영하여 모두에게 공개하고, 주문과 지불에 관련한 데이타는 안전한 서버에서 관리할 수 있다.

▶ 별첨 ◀

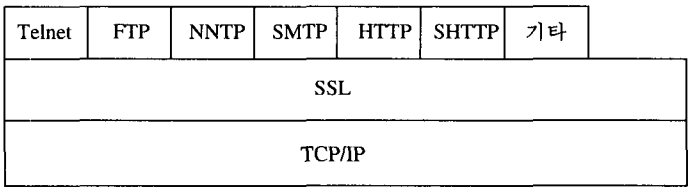
미국이외의 나라에서 SSL을 사용할 때에는 40비트 크기의 키를 사용하도록 되어 있다. 이는 미국의 법적 문제로 암호 알고리즘을 외국으로 수출할 때 알고리즘의 보안도를 낮추기 위한 방법이다. 물론, 미국 내에서는 128비트의 키를 사용한다. 그런데, 미국 이외의 나라에서 사용하는 40비트의 SSL이 이미 해독되었다는 보고가 있었다. 그 상황은 다음과 같다. SSL에서는 관용 암호 알고리즘으로 RC4를 사용한다. RC4는 128비트의 키를 사용하고, 소스는 공개되어 있지 않다. 미국 이외의 나라에서 사용하

도록 되어 있는 SSL에서도 RC4의 키로 128비트를 모두 사용한다. 하지만, 이 128비트 중에서 40비트만 암호화하여 전송하고, 나머지 88비트는 평문의 형태로 전송한다. 그래서 40 비트의 키를 사용하는 셈이 된다. RC4의 소스가 비공개로 되어 있지만 리버스 엔지니어링(Reverse engineering)기법을 사용하여 RC4의 소스를 도출해 내었다. 또한 암호화되어 전송되는 40비트의 키를 brute force 공격으로 해독하였다. 결국, 129비트의 키가 모두 노출되고, 이 키를 사용한 암호 알고리즘의 소스도 노출됐었으므로 SSL이 해독된 것이다.

□ 특 징

- 데이타의 암호화를 위한 비밀키를 handshake 프로토콜 단계에서 정한 후 그 키를 사용하여 암호화된 메세지로 통신하기 때문에 기밀성이 보장된다.
- 공개키 알고리즘을 사용하여 상대방의 신원을 인증할 수 있다.
- Secure hash function(MD5, SHA....)을 사용하여 Message Authentication Code를 만들어 메세지에 포함시키기 때문에 통신이 reliable하다.
- Secure HTTP와 SSL과의 관계

Secure HTTP와 SSL은 서로 다른 접근 방법을 가지고 있다. SSL은 HTTP, NNTP와 같은 어플리케이션 프로토콜의 하위에서 동작하는 것에 반해, Secure HTTP는 PEM처럼 message-based인 접근 방법을 사용한다. 이를 그림으로 보면 〈그림 3-2〉와 같다. 하지만 이 둘은 서로 배타적인 구조를 가지고 있는 것이 아니기 때문에 SSL의 상위에 Secure HTTP가 위치하는 구조로 각자가 가지고 있지 않은 부분들을 서로 보완하여 새로운 WWW보안 메카니즘을 개발할 수 있다. Secure HTTP와 SSL은 모두가 공식적인 표준으로 지정되어 있는 상태가 아니므로 현재 EIT사와 Netscape Communications사는 공동연구를 통해 Secure HTTP와 SSL을 하나의 방법으로 통합하는 방법을 개발하여 공식 표준으로 만들기 위한 노력을 하고 있다.



〈그림 3-2〉 SHTTP와 SSL의 관계

3.3. Simple message Digest authentication

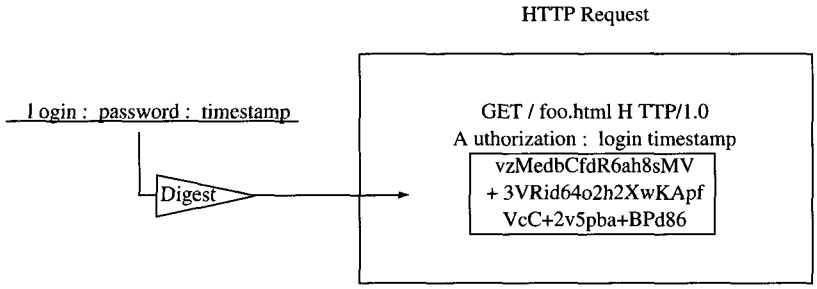
사용이 자유로운 알고리즘들을 이용한 방법 중에서는 가장 효과적인 사용자 인증 기술로 인정받고 있다. 클라이언트와 서버의 간단한 수정만 거치면 HTTP/1.0에서 지원해 주는 Basic Authentication방법을 대치할 수 있다[4].

□ 목 적

- HTTP가 가지고 있는 특징들을 모두 수용하면서 좀 더 안전한 사용자 인증을 추구한다.
- 특허나 수출입법에 저촉되지 않는 사용이 자유로운 알고리즘을 사용한다.

Basic authentication 방법에서 패스워드가

평문으로 전송되는 것을 개선하는 방법으로 사용자의 인증이 필요한 경우, 브라우저 사용자는 사용자ID와 패스워드, 그리고 replay 공격을 방지할 수 있는 timestamp를 작성한 후, 이를message digest 함수를 이용하여 특정한 길이의 bit으로 표현한다. 여기에 사용자ID와 timestamp를 다시 붙여서 서버에게 보낸다. message digest 함수는 일방향 함수이므로, 누군가가 도청한다 하더라도 본래의 패스워드를 추출할 수 없다. 서버는 request를 받은 후, 사용자ID를 보고 패스워드 화일에서 패스워드를 찾아내 브라우저에서 했던 것처럼 사용자ID, 패스워드, timestamp를 message digest 함수에 적용시킨다. 이 결과값과 request에 있던 값을 비교하여 사용자 인증을 수행한다. 그림으로 나타내면 〈그림 3-3〉와 같다.



〈그림 3-3〉 Simple message Digest authentication

하지만 이 방법은 서버가 패스워드 화일을 관리하는 측면에서 Basic authentication 방법보다 복잡한 면을 가지고 있다. Basic authentication에서는 사용자가 전송한 평문의 패스워드를 암호화하여 저장되어 있는 암호화된 패스워드와 비교한다. 이 때 사용하는 암호 알고리즘은 복호화가 불가능하여 평문의 패스워드로 변환시키지 못한다. 하지만 Simple message digest scheme에서는 패스워드 자체가 로긴네임과 timestamp와 함께 해쉬값으로 계산되어 전송되기 때문에 평문의 패스워드 만을 추출해 내어 암호화를 할 수가 없다. 이를 해결하기 위해서는 WWW서버가 패스워드를 평문의 상태로 저장해야 하는 데 패스워드를 모든 사람이 볼 수 있게 관리할 수는 없으므로 WWW 데몬만 읽을 수 있는 방법을 사용해야 한다.

3.4. Kerberos와 HTTPD

Kerberos는 안전성이 보장되지 않은 통신로상에서 클라이언트와 서버 사이에서 서로의 신원을 확인해 주는 상호 인증 시스템이다. 모든 사용자들은 자신들을 인증하기 위해 필요한 정보들을 "Kerberos"라고 불리는 서버에 등록시켜 놓아야 한다. 사용자는 이용하려는 응용서버에게 자신의 신원을 확인시켜 주기 위해 Kerberos서버에게 자신의 패스워드를 전송해 ticket을 얻고 이 ticket을 이용하려는 응용 서버로 보낸다. 사용자가 이용하려는 응용 서버는 ticket을 검사하여 사용자의 신원을 인증한다. Kerberos의 본래의 목적은 암호화되지 않은 패스워드가 전송되는 횟수를 줄이자는 것이다. 현재 V4와 V5의 두 가지 버젼이 나와 있지만 호환성은 제공되지 않고 있다. NCSA(National Center for Supercomputing Applications) httpd 1.5에서는 Kerberos의 세션키와 DES를 사용하여 메세지를 암호화시키는 방법은 현재 개발중에 있다[7].

□ 목 적

- WWW 클라이언트와 서버 사이의 상호 인증을 지원하여 안전한 접근제어를 구현한다.
- Request와 response를 암호화하여 통신한다.

Web에 Kerberos를 추가했을 때의 메세지들의 통신 예는 다음과 같다.

① 클라이언트가 인증자료 없이 본래의 request를 보낸다.

```
GET  /krb__protected/poems.html
HTTP /1.0
Accept: */*
User-Agent: NCSA__Mosaic/2.7
```

② 서버는 Kerberos ticket이 필요하다는 메세지와 401 에러 메세지를 보낸다.

```
HTTP/1.0 401 Unauthorized
Date: Mon, 16 Oct 1995 04:12:49 GMT
Server: NCSA/1.4.2
Content-type: text/html
WWW-Authenticate: KetberosV4
--HTML for 401 error message--
```

③ 클라이언트는 Kerberos로부터 ticket을 얻은 후 이를 16진수 형태로 바꾸어 다시 request를 보낸다.

```
GET /krb__protected/poems.html HTTP/1.0
Accept: */*
User-Agent: NCSA__Mosaic/2.7
Authorization : KerberosV4 acain
                e4fc3e0a356352d  4e42b19a
                33b18d
                47ce f135eb351352df35 0f
                cba fc3e eb8 f9352
                c80fce026ff9352dba3502466
                00e4d9571
```

④ Kerberos ticket이 적합한 것으로 인증이 되면 서버는 자신을 인증해 줄 자료와 함께 문서를 보낸다.

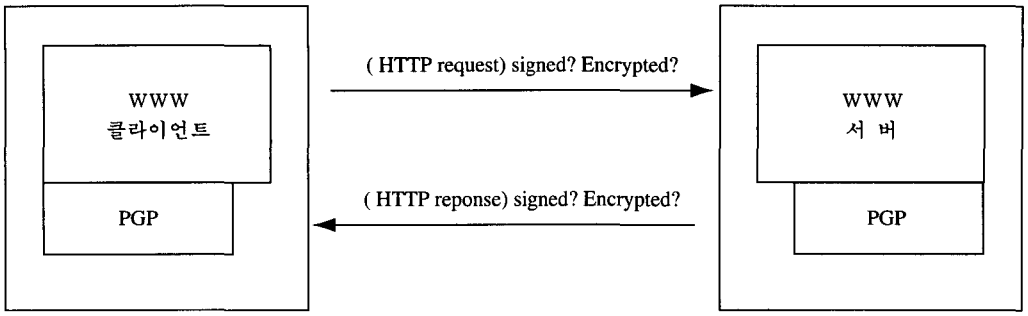
```
HTTP/1.0 200 Document follows
Date: Mon, 16 Oct 1995 04:13:05 GMT
Server: NCSA/1.4.2
Content-type: text/html
Content-length: 624
WWW-Authenticate : KerberosV4
[c3602 905a92b683f] User authentcated

--Document starts here--
```

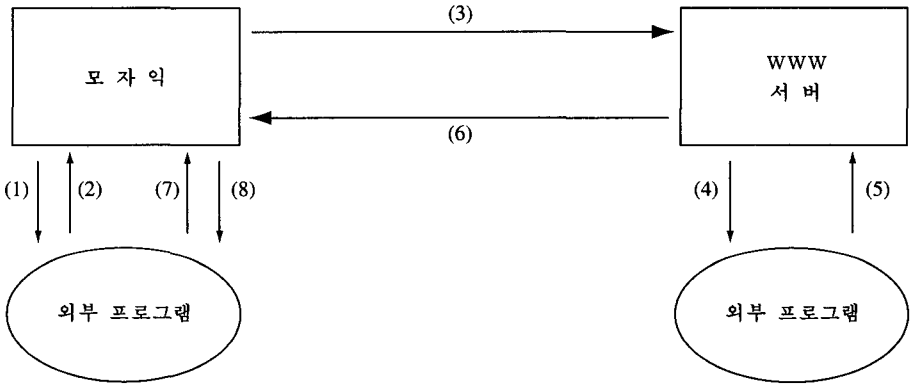
3.5. WWW과 PGP의 정합

이 방법은 NCSA에서 고안해 낸 방법으로 전자우편 보안도구로 널리 이용되고 있는 PGP를 WWW의 보안 모듈로 사용하는 방식이다. PGP는 다른 응용 프로그램에 독립적으로 구성되어 있기 때문에 이를 WWW에 접목시켜 HTTP를 암호화 시킴으로써 안전한 WWW 통신을 구현할 수 있다. 또한, 이 방법은 기존의 HTTP와 HTML에 수정을 요구하지 않는 방법이다. 많은 WWW 보안 방법들이 HTTP나 HTML에 수정을 요구하고 있는 것에 비하면 이 방법은 호환성 면에서 장점을 가지고 있다. 전체적인 구성을 보면 〈그림 3-4〉와 같다[4].

이 방법에서는 브라우저가 암호화를 담당하는 외부 프로그램과 통신을 할 수 있어야 한다. WWW서버와 외부 프로그램과의 통신은 CGI(Common Gateway Interface) 프로그래밍을 이용하여 구현할 수 있다. 브라우저와 외부 프로그램과의 통신은 NCSA에서 개발한 CCI(Common Client Interface) 라이브러리를 이용하여 구현한다. CCI 라이브러리는 NCSA에서 개발한 관계로 현재는 유닉스용 Mosaic만을 지원해 주고 있다. 동작하는 원리는 〈그림 3-5〉와 같다[8].



〈그림 3-4〉 WWW에 PGP를 접목한 구조



〈그림 3-5〉 PGP를 이용한 WWW의 암호화 통신

클라이언트는 문서를 요청하는 request를 처음에 서버로 보내지 않고, (1)번과 같이 외부 프로그램으로 보내 준다. 외부 프로그램은 request 메세지를 받아 전자서명을 붙이고, 서버만이 알아볼 수 있도록 암호화하여 (2)번을 통해 브라우저로 되돌려 보낸다. 클라이언트는 암호화된 request 메세지를 (3)번 단계에서 서버로 post한다. (4)번을 통해 서버의 외부 프로그램으로 전달된 request 메세지는 복호화가 되어 클라이언트의 인증을 확인하고, 요청된 문서를 찾아 서버의 전자서명을 붙이고, 이 문서를 요청한 클라이언트만이 알아볼 수 있도록 암호화한다. (5)번 단계에서 암호화된 response는 다시 서버로 보내진다. (6)번에서 암호화된 response는 클라이언트에게 전송되고, 복호화를 위하여 (7)번에서 다시 외부 프로그램으로 보내져 복호화된 후 (8)번에서 복호화된 response를 클라이언트에게 보여 준다.

이 방법은 전체적인 WWW 시스템에 수정을 요구하지 않는다는 장점 이외에도 PGP가 WWW 보안에만 사용되는 것이 아니고 전자우편 등의 다른 어플리케이션에도 사용될 수 있다는 장점을 지니고 있다. 하지만, 암호화 모듈이 브라우저와 서버 프로그램의 외부에 위치하는 관계로 브라우저와 암호화 모듈 사이의 통신, 서버 프로그램과 암호화 모듈 사이의 통신이 많아져 속도 면에서는 다른 방법들에 비해 느리다는 평가를 받고 있다.

4. 결 론

인터넷의 급속한 성장과 함께 초고속 정보

통신망이 구축되면 WWW의 이용 중에서도 안전한 통신을 요구하는 분야가 크게 늘어날 것으로 보인다. 특히 WWW을 이용한 전자 쇼핑과 전자 상거래 분야에서는 HTTP 메세지의 인증과 기밀성을 반드시 보장해 주어야 한다. 이러한 요구사항들에 대처하기 위해 본 고에서는 현재 HTTP에서 지원해 주고 있는 WWW 보안 메카니즘들과 암호 알고리즘을 이용한 방법들을 살펴 보았다.

참 고 문 헌

[1] 박현제, "인터넷 입문", KRNET '96, 1996

[2] 권도균, "전자 거래 보안", NETSEC-KR '96, 1996

[3] Greg Bossert, "Requirements for HTTP Security", Internet-Draft, 1995

[4] Adam Cain, "Introduction to Web security", 2nd WWW-Forum KOREA, 1995

[5] E. Rescorla, "The Secure HTTP", Internet-Draft, 1995

[6] Alan O. Freier, "The SSL Protocol", Internet-Draft, 1996

[7] Adam Cain, "Kerberizing the Web", http://snapple.ncsa.uiuc.edu/adam/khttp/intro.html, 1996

[8] Judson D. Weeks, "CCI-Based Web Security: A Design Using PGP", 4th International WWW Conference, 1995

□ 著者紹介

박 현 동

1995년 충남대학교 전산학과 (학사)
1995년 충남대학교 대학원 컴퓨터과학과 (석사)
1997년 ~ 현재 충남대학교 대학원 컴퓨터과학과 박사과정

※ 관심분야 : 네트워크 보안 시스템, 암호학

류 재 철

1985년 2월 한양대학교 산업공학 학사
1988년 5월 Iowa State Univ. 전산학 석사
1990년 12월 Northwestern Univ. 전산학 박사
1991년 2월 - 현재 충남대학교 컴퓨터과학과 조교수

※ 관심분야 : 컴퓨터 및 통신 보안체제, 네트워크 관리, 분산처리