웹 환경에서 자바 기술을 이용한 안전한 사용자 식별 및 인증 모델 설계

송기평* · 손홍* · 김선주** · 조인준**

* 한국전자통신연구원, ** 배재대학교

Design and Implementation of Secure Identification and Authentication Model using Java Technology on the Web Environment

Gi-Pyeung SONG*, Hong Son*, Sun-Joo Kim** , In-June Jo**

*ETRI , **Paichai University

E-mail : gpsong@pec.etri.re.kr , hshon@pec.etri.re.kr, uneedme@network.paichai.ac.kr , injune@mail.paichai.ac.kr

요 약

현재 널리 사용되고 있는 웹 환경에서 사용자 식별 및 인증관련 보안서비스가 취약할 뿐만 아니라 이의 활용에 문제를 안고 있다. 즉, HTTP 1.0에서는 BAA(Basic Access Authentication)을 채택하여 사용자 ID와 패스워드를 평문 상태로 전달한다[1]. HTTP1.1에서는 이를 개선하여 DAA(Digest Access Authentication) 방법을 권고하고 있지만, 웹 브라우저에서 이를 채택하지 않아 범용적으로 사용되지 못하고 있는 실정이다.

본 논문에서는 이러한 문제해결을 위해 자바기반 사용자 식별 및 인증 보안 서비스(JAA : Java-based Access Authentication)를 제안하고 이를 설계하였다. 제안시스템은 기존의 웹 환경과 독립성을 유지하기 때문에 모든 웹 환경에서 자연스러운 채택이 가능하다.

ABSTRACT

Now, It is vulnerabilities and problems of adaption in user identification and authentication on the Web environments; the BAA(Basic Access Authentication) of HTTP1.0 is that user ID and password is passed with clear-text between client and server. For this enhancement, the DAA(Digest Access Authentication) of HTTP1.1 is that user password is digested by MAC(Message Authentication Code) mechanism. but, this mechanism is not adapted by venders of Web browsers.

This paper propose the Java based user identification and authentication model to resolve the above problems. Proposed systems are applied to the Web environment, since it has independence to web server and client.

Ⅰ. 서 론

현재 범용적으로 사용되고 있는 HTTP 1.0[1] 프로토콜에서는 서버가 클라이언트의 UA(User Agent)를 인증하는 방안으로 BAA를 채택하고 있다. 이 방안은 사용자의 인증정보(즉, 아이디, 패스워드)가 평문상태로 네트워크 상에 전송되기 때문에 안전한 사용자 인증을 제공하지 못하고 있다. 이의 대체 방안으로 IETF(Internet Engineering Task Force)의 RFC 2069[2]에서는 HTTP1.1에 DAA방안 채택을 권고하고 있다. 이 방안은 BAA와 같이 단순한 챌린지/리스판스(Challenge/Response)[3] 방식을 사용하지만, MAC(Message Authentication Code)[5] 기법을 추가하여 사용자 기밀 데이터를 네트워크 상에 노출시키지 않고 통신하는 방안을 제시하고 있다. 이 방안은 HTTP헤더 일부와 HTTP의 개체 몸체

(Entity Body)의 무결성을 보장하는 방안도 추가로 제시되어 있다.

하지만, 상기의 방안 모두는 HTTP 프로토콜 수준에서 사용자 식별 및 인증을 행함에 따라 다음과 같은 문제가 내재되어 있다. 즉, HTTP 1.0에서는 웹 클라이언트와 서버간에 사용자 식별 및 인증에 필요한 정보(즉, 사용자 아이디, 패스워드)를 평문으로 전달하는 취약성이 있고, HTTP 1.1에서는 MAC기법을 이용하여 사용자 패스워드를 노출하지 않지만, 웹 서버 및 브라우저 모두를 HTTP 1.1로 갱신할 것을 요구한다는 문제를 안고 있다. 이는 현실적으로 웹 서버와 브라우저를 제공하는 제작사의 의도에 따라 이의 채택 유무가 결정되기 때문에 현재 HTTP 1.1을 채택한 브라우저를 찾아보기가 힘들다.

본 논문에서는 상기와 같은 HTTP 수준에서 사용자 인증 및 식별의 취약점을 해결하고자 자바 기술[7,8,9]을 사용하여 응용수준에서 이의 해결방안(JAA : Java-based Access Authentication))을 제안하였다. 제안된 방안은 개발자가 웹 서버에 사용자 식별 및 인증 모듈만을 설치하고 웹 클라이언트에게는 아무런 변경을 요구하지 않기 때문에 현재 사용되고 있는 모든 웹 환경에 적용이 가능하다.

본 논문의 구성은 제 I 장에 서론, 제 II 장에 HTTP프로토콜에서 사용자 식별 및 인증 기법을 요약하고, 제 III 장에서 본 논문에서 제안한 자바 기반 사용 식별 및 인증 기법(JAA)의 설계 및 동작을 설명하고, 마지막으로 제 IV장에 결론을 맺었다.

II. HTTP프로토콜에서 사용자 식별 및 인증

2.1 HTTP 환경에서의 접근 인증 개요

HTTP환경에서 사용자 식별 및 인증은 클라이언트가 접근이 보호된 서버의 URI자원을 요구했을 때, 서버에 의해 시작된다. 즉, 서버가 클라이언트 요구에 챌린지 하고, 이를 수신한 클라이언트의 UA(User Agent)가 사용자 아이디와 패스워드로 이에 리스판스하여 사용자 식별과 인증이 이루어지는 방안이다. HTTP프로토콜은 접근 인증에 이러한 단순한 챌린지/리스판스 패러다임만을 사용할 것을 제약하지 않는다. 추가적인 방안으로 전송계층에서 암호화, 메시지 인캡슐레이션 기법, 그리고 인증정보를 명시하는데 추가적인 헤더 등의 사용을 허용하고 있다. 이러한 인증 기능을 위해 HTTP 1.0에서는 BAA방안을, HTTP 1.1에서는 DAA방안을 채택하였다.

2.2.1 BAA 방안

BAA는 UA가 보호된 URI자원을 접근하고자 할 경우 사용자 이름과 패스워드를 제시하여 자신을 식별 및 인증 하는 모델이다. 서버는 UA가 요구한 URI자원의 보호를 위해서 사용자 이름과 패스워드가 타당할 경우에만 UA의 사용 요구를 허락한다. 세부 동작을 살펴보면 다음과 같다.

서버가 보호 영역내의 한 URI에 대해서 권한이 없는 요구 메시지를 받게 되면, 서버는 다음과 같은 챌린지 코드를 생성하여 클라이언트에게 응답한다[1,2].

```
WWW-Authenticate :
Basic realm = "WallyWord"
/*WallyWord: 요구 URI의 보호 영역을 식별하기 위해서 서버가 할당한 문자열*/
```

이를 수신한 클라이언트 UA가 서버로부터 인증을 허락 받기 위해, 클라이언트는 사용자 이름과 패스워드를 Base64로 인코딩하여 서버에게 전송한다.

```
basic-credentials="Basic" SP basic-cookie
basic-cookie = <base64 encoding of user-pass,
        except not limited to 76 char/line>
user-pass=userid ":" passwd
userid=*<TEXT excluding ":">
password = *TEXT
```

예를 들어, UA의 사용자 이름이 "uneedme"이고, 패스워드가 "testtest"인 것을 보내고자 할 경우에는 다음과 같은 허가 요청 헤더가 만들어진다.

```
authorization :
  Basic QWxhZGRpbjpvcGVuIHNIc2FtZQ
```

BAA에서는 상기와 같이 인증 허락을 요청하는 헤더가 평문으로 네트워크에 전송되기 때문에 안전하지 못한 사용자 인증 방법이다. 또한 무결성 및 기밀성을 지원하는 방안이 포함되어 있지 않기 때문에 교환되는 어떤 HTTP개체도 안전하게 보호하지 못한다. 이중에서 가장 큰 문제점은 사용자의 아이디, 패스워드를 평문으로 전송한다는 점이다. 따라서, 중요한 인증정보가 보호되지 않는다. BAA의 일반적인 용도는 보안용보다는 식별용이라고 볼 수 있다. 즉, 식별의 수단으로써 사용자 아이디와 패스워드를 제공할 것을 사용자에게 요구한다.

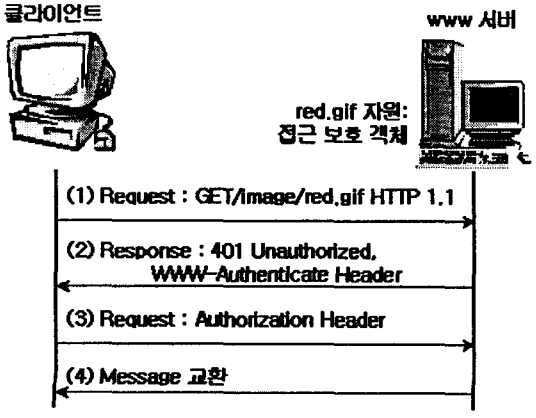
BAA는 위장된 서버에 의해 속임공격(Spoofing)에 취약하다. 즉, 사용자가 악의적인 공격자의 서

버에 연결되었을 때, 이 서버가 사용자에게 BAA로 보호된 정보를 가지고 있는 서버에 정상적으로 연결 중이라는 것을 믿게 유도하여, 공격자가 사용자의 아이디와 패스워드를 요구하고, 이를 후에 사용을 위해 저장하고, 오류가 발생한 것처럼 위장할 수 있다. 따라서, 웹 개발자들은 CGI(Common Gateway Interface)스크립트나 게이트웨이를 사용하여 이러한 종류의 속임공격에 대비하여야 한다. 이러한 형태의 공격은 DAA에서는 불가능하다.

2.2.2 DAA 방안

DAA방안도 BAA처럼 챌린지/리스판스라는 단순한 패러다임을 기초로 한다. 하지만 BAA의 취약점인 속임 공격과 아이디와 패스워드 노출을 방지하기 위한 몇 가지 고안이 추가되었다. 즉, 클라이언트 UA를 인증하기 위해서 서버는 MAC(Message Authentication Code)기법[6]을 사용하여 넌스(nonce)값을 계산하고 이 결과를 클라이언트에게 챌린지 한다. 이때의 MAC 값은 속임공격을 방어하기 위한 수단이다. 이를 수신한 클라이언트는 이에 대응하는 리스판스를 제작하여 서버에게 보내게 된다. 이 메시지에는 사용자 이름(UA가 제공), 보호영역(서버가 제공), 넌스값(서버가 제공), URI값(서버가 제공), 알고리즘(클라이언트가 선택), 패스워드를 노출시키지 않기 위한 MAC값, 그리고 헤더 및 개체 몸체의 무결성을 지원하기 위한 또 다른 MAC 값이 계산되어 포함된다.

이를 수신한 서버는 전송되어 온 메시지로부터 사용자 이름과 URI를 추출하여 이에 대응하는 인증 데이터(사용자 패스워드)를 자신이 유지하고 데이터 베이스로부터 취득한다. 서버는 이들 데이터를 이용하여 클라이언트에서와 같이 동일한 방법으로 MAC값을 계산하여 보내온 MAC값과의 일치 유무로 UA를 인증하고, 넌스값을 사용하여 속임 공격 유무를 검증한다. 따라서, 사용자 아이디와 패스워드가 전송 중에 네트워크 상에 노출되지 않고 UA인증이 가능하고, 제한된 범위 내에서 속임 공격을 어렵게 한다.



<그림 1> BAA와 DAA의 접근 인증 절차

위에서 제안된 DAA방안에는 여러 가지 제약사항들이 알려져 있다. 이는 단지 BAA를 대체한 것일 뿐이다. 이는 MAC기반의 인증 시스템이기 때문에 이러한 형태의 인증 시스템에서 발생하는 모든 문제를 그대로 내포하고 있다. 특히, 사용자의 패스워드를 설정하기 위해서 사용자와 서버사이에 이루어져야 할 안전한 초기 설정에 대해서 이 프로토콜에서는 어떤 언급도 하지 않고 있다. 따라서, 사용자와 개발자는 이 프로토콜이 기존의 Kerberos나, 클라이언트 측의 개인키 방안을 사용하는 인증방법[6]만큼 안전하지 않다는 것을 인식해야 한다. 이것은 단지, 상대적으로 BAA를 사용하는 것보다는 우수하다는 의미이다. <그림1>에서는 위에서 논의한 BAA와 DAA의 접근 인증 절차를 나타낸 것이다.

III. 자바기반의 사용자 식별 및 인증 방안 (JAA:Java-based Access Authentication)

3.1 기본 개요

이 장에서는 상기의 HTTP프로토콜 수준에서 이루어지는 사용자 식별 및 인증 방안을 개선하고자 JAA(Java-based Access Authentication)방안을 제안하고 설계 내용을 기술한다. JAA는 응용수준에서 사용자 식별 및 인증이 이루어진다. 이의 특징은 HTTP가 사용자 패스워드 노출 방지를 위해 MAC기법을 사용한 반면, JAA는 공개키 암호기술을 사용하여 보안강도를 제고시킨 점을 들 수 있다.

JAA 설계에서 중요하게 고려한 기능은 (1) 사용자 인증정보(패스워드 등) 노출 방지 기능, (2) 제한적인 재현 공격(속임 공격) 방지 기능, (3) 웹 클라이언트의 변경 없이 적용이 가능한 기능 등이다.

3.2 JAA 시스템 구성 요소

제안된 JAA는 <그림2>처럼 여러 개의 구성요소로 이루어진다. 그림에 표기된 문자를 사용하여 이들 각각의 기능을 설명하면 다음과 같다.

가. 클라이언트('C')

이는 웹 브라우저('wb')를 사용해 웹 서버('ws')에 접속을 시도하는 개체이다. 이는 자신의 신원을 웹 서버로부터 식별 및 인증 기능을 하는 인증 모듈(즉, JAA모듈)을 실행한다. 이는 애플릿

형태로 서버로부터 내려 받은 모듈이다. 클라이언트에서 JAA모듈의 주요 기능은 클라이언트 UA 인증에 필요한 일련의 보안 메시지를 생성하여 서버에게 보내는 기능을 한다. 또한 서버로부터 UA인증 결과를 수신하여 이에 대응된 처리를 한다. 마지막으로 클라이언트는 동일한 문서에 접근 인증을 편리하게 하기 위해 접근이 보호된 문서의 URI정보, 서버의 공개키와 사용자 UA정보 등을 로컬 파일에 저장한다. 이에 대한 자세한 내용은 다음절에서 다룬다.

나. 웹 서버('S')

접근통제가 필요한 URI문서를 클라이언트('C')로부터 요구받았을 때, HTTP프로토콜을 통하여 JAA인증모듈(자바 애플릿 : 'jaam')을 해당 클라이언트에 전송한다. 그리고 나서, JAA데몬('jaad')을 통해서 클라이언트로부터 식별 및 인증을 위한 정보를 수신한다. 클라이언트로부터 수신한 정보를 JAA 데몬('jaad')에서 사용자 식별 및 인증 여부를 결정한다. 식별 및 인증이 확인 된 경우에는 접근 통제가 된 URI 문서를 클라이언트에게 HTTP프로토콜을 통해서 전송한다. 또한 식별 및 인증이 실패한 경우 실패 메시지를 클라이언트에게 채널을 통해서 전송한다. 이에 대한 자세한 동작 설명은 다음절에서 자세히 설명한다.

다. JAA데몬('jaad')

JAA데몬('jaad')은 클라이언트로부터 수신된 정보를 이용해 사용자를 식별 및 인증을 하는 서버 프로그램이다. JAA데몬('jaad')은 자신이 소유하고 있는 사용자 식별 및 인증 정보와 클라이언트로부터 수신한 정보와 비교하여 인증을 한다. 마지막으로 인증 여부를 채널을 통해 클라이언트에 전송한다.

라. JAA 인증 모듈 ('jaam')

JAA 인증모듈('jaam')은 클라이언트가 웹서버에 접근 통제된 URI문서를 요구할 때 자동으로 클라이언트에게 다운로드 받도록 제작된 자바 애플릿 프로그램이다. 이 애플릿은 암호 알고리즘(RSA, MD5 등)이 포함되어 있다. JAA 인증모듈('jaam')은 JAA 데몬('jaad')과 채널 형성을 통해 클라이언트를 식별 및 인증 할 수 있도록 해준다.

마. JAA 로컬 파일 ('lf')

이 파일은 클라이언트가 웹서버에 재접속을 시도하는 경우 UA로부터 매번 입력받는 것을 줄이기 위해 사용자 정보를 저장한 파일을 말한다. 이 파일의 구성은 사용자 아이디, 패스워드, 웹서버의 URI정보, 웹서버 공개키, 파일 유효 확인필드(TTL) 등으로 구성된다. JAA데몬('jaad')과 JAA 인증모듈('jaam')간의 식별 및 인증 결과를 클라이언트측 로컬 컴퓨터에 저장되는 파일이다.

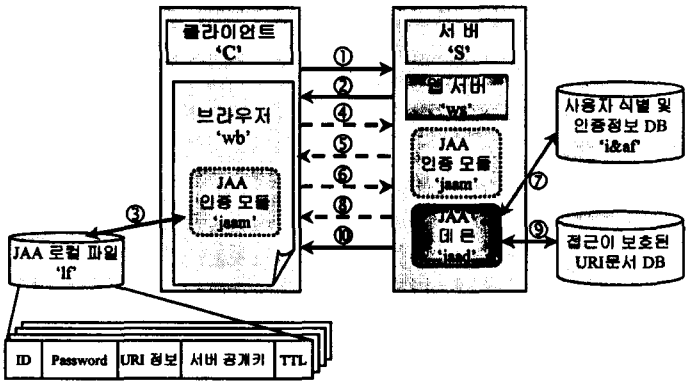
바. 사용자 식별 및 인증 정보 데이터 베이스 ('i&af')

이것은 사용자에 대한 식별 및 인증 정보가 저장된 데이터 베이스('i&af')이다. 즉, 사용자 정보가 저장된 데이터베이스로 클라이언트로부터 수신한 사용자 정보를 식별 및 인증하는데 사용하는 데이터 베이스이다.

사. 접근이 보호된 URI 문서

일반사용자의 접근이 보호된 문서로 JAA 인증 과정을 통해서 사용자 식별 및 인증된 사용자만이 접근 가능한 문서이다. 즉, 일반 문서 파일이나 HTML문서를 가리킨다.

3.3 JAA 프로토콜의 기본동작



<그림 2 > JAA 구성요소 및 동작도

JAA방안은 전술한 BAA/DAA와 같이 UA에 의한 URI 자원의 접근통제를 위해 사용자 아이디와 패스워드를 방식을 사용하여 사용자 식별 및 인증을 행한다.

본 논문에서 제안한 JAA 방안의 기본동작 절차는 다음과 같다.

[단계1] 사용자(UA)가 웹 서버의 보호된 URI자원에 접근을 요청한다.

[단계2] 이를 수신한 웹 서버는 JAA 인증 모듈이 포함된 자바 애플릿을 클라이언트에게 전송한다.

[단계3] 클라이언트는 수신된 인증 모듈을 실행한다. 이의 실행 기능은 다음과 같다.

(1) 현재 접속하려는 서버의 공개키 및 URI정보가 JAA 로컬 파일에 저장되었는지 확인한다.

(2) JAA 로컬 파일 내에 존재하지 않으면, 서버의 공개 키 수신을 위해 서버의 JAA데몬과 클라이언트간에 채널을 설정한다.

(3) 설정된 채널을 통해 서버의 공개키를 수신받는다.

(4) 사용자의 식별 및 인증 정보(즉, 사용자 아이디, 패스워드)를 사용자로부터 입력받는다.

(5) 사용자 식별 및 인증 정보를 서버의 공개키로 암호화한다.

(6) 위의 (1)의 과정에서 JAA 로컬 파일 내에 서버의 공개키 및 URI정보가 존재하는 경우에, 곧바로 (4),(5)의 과정을 수행하게 된다.

[단계4] 암호화 된 사용자 식별 및 암호화 정보를 [단계 3]의 (1)항에서 설정된 채널을 통해서 JAA 데몬으로 전송한다.

[단계5] 이를 수신한 서버의 JAA데몬은 다음과 같은 기능을 한다.

(1) 사용자 식별 및 인증 정보를 자신의 개인키를 사용하여 복호화 한다.

(2) 웹 서버에 저장된 사용자 식별 및 인증 정보와 복호화된 정보를 비교하여 인증을 행한다.

[단계6] 인증 결과에 따라 URI 문서 혹은 인증 실패 메시지를 클라이언트에게 전달한다

3.4 JAA 프로토콜 설계

상기와 같은 기본동작들을 기반으로 JAA프로토콜 설계 내용을 보면 다음과 같다. <그림2>에서 표시된 번호를 중심으로 설명하면 다음과 같다. 설명의 표기법에서 C(x)는 클라이언트에서 x라는 구성요소(혹은 프로그램 모듈)이고, S(y)는 서버에서 y라는 구성요소(혹은 프로그램 모듈)이다.

① C(wb) => S(ws) ; 사용자(UA)가 웹 브라우저(즉, wb)를 통해서 웹 서버(즉, ws)의 보호된 URI자원에 접근을 요청한다. 이때 HTTP프로토콜이 사용된다.

② S(ws) => C(wb) ; 웹 서버가 JAA 인증모듈이 포함된 자바 애플릿(즉, jaam)을 HTTP 프로토콜을 통하여 클라이언트에게 전송한다.

③ C(jaam) <=> C(lf) : C에 수신된 인증 모듈(즉, jaam)이 실행된다. 이 인증 모듈의 기능은 접근 요청한 URI 자원에 접근 시 필요한 정보가 JAA 로컬 파일(즉, lf)에 존재하는지 검사한다. 만약, 존재한다면 파일 유효항목 정보를 확인한다. 유효하다면 ⑥단계 실행으로 제어권을 넘긴다. 만약, URI정보가 존재하지 않거나 파일 유효항목 정보가 유효하지 않다면 ④단계 실행으로 제어 권을 넘긴다.

④ C(jaam) <=> S(jaad) : 클라이언트의 식별 및 인증 기능을 수행하는 JAA 데몬(즉, jaad)과의 채널을 설정한다.

⑤ S(jaad) => C(jaam) : 설정된 채널을 통해서 서버에서 실행중인 JAA 데몬(즉, jaad)이 클라이언트(즉, jaam)에게 서버의 공개키를 전송한다.

⑥ C(jaam) => S(jaad) ; 클라이언트는 실행중인 자바 인증 모듈(즉, jaam)을 통해서 사용자의 식별 및 인증정보(사용자 아이디, 패스워드)를 사용자로부터 입력을 받는다. 이를 ⑤단계에서 수신한 웹서버의 공개키로 암호화하여 서버의 데몬(즉, jaad)에게 전송한다. 또한, 동일한 URI문서 요구 시, 사용자 아이디 및 패스워드 재 입력을 방지하기 위해 로칼 파일(즉, lf)에 레코드를 추가한다.

⑦ S(jaad) => S(i&af) ; JAA 데몬(즉, jaad)은 ⑥의 과정에서 수신한 암호문을 웹서버의 개인키로 복호화 한다. 복호화된 사용자 식별 및 인증정보를 웹 서버에 저장되어(즉, i&af파일) 있는 사용자 식별 및 인증 정보와 비교하여 인증을 행한다.

⑧ S(jaad) => C(jaam) ; JAA 데몬(즉, jaad)은 인증 여부를 ④에서 설정된 채널을 통해서 클라이언트에 내려받기 된 JAA인증 모듈(즉, jaam)에게 전송한다. 이때, 인증이 성공한 경우에는 웹서버(즉, ws)가 데이터베이스로부터 접근이 통제된 URI문서를 선택하여(그림에서 ⑨) HTTP프로토콜을 통해서 클라이언트(즉, wb)에 전송(그림에서 ⑩)한다.

이 단계 이후 동일한 UA가 보호된 서버의 문서에 접근할 경우에는 JAA 인증모듈이 JAA로컬 파일에 저장된 정보를 이용한다. 즉, JAA 로컬 파일에 저장된 식별 및 인증 정보를 이용함으로써 사용자가 접속할 때마다 매번 이를 입력해야 하는 불편을 없앴다. 이와 같은 과정을 통해 접근 통제된 문서에 대해 안전한 사용자 식별 및 인증이 가능하다.

JAA를 설계함에 있어서 중요하게 고려한 보안 서비스기능을 요약하면 다음과 같다.

(1) 사용자 인증정보(패스워드 등) 노출 방지 기능이다. <그림2>에서 ⑥의 부분이 사용자의 식별 및 인증 정보가 웹서버의 공개키로 암호화됨을 보여주고 있다.

(2) 클라이언트가 서버를 인증하도록 하였다. 이는 제안한 JAA 방안에서 자바 애플릿 기술을 사용함에 따라 부수적으로 얻은 효과이다.

(3) 웹 클라이언트의 변경 없이 적용이 가능하도록 하였다. 즉, 제안된 JAA 방안은 자바 기술을 사용한다. 즉, 클라이언트가 기존의 웹 브라우저를 통해서 자바 인증 모듈(자바 애플릿)을 내려받기 하여 이를 자바 가상 머신에서 실행시킨다. 따라서, 클라이언트의 변경작업 없이 제안된 시스템을 적용할 수 있다.

3.5 제안된 JAA 시스템 특징

제안된 JAA시스템의 특징을 기술하면 다음과 같다.

(1) JAA의 효율적인 실행을 위해 기존에 구현된 JCE(Java Cryptography Extension)와는 별도로 JAA 인증 모듈을 자체 제작하였다. 이는 JCE를 참조하는 타 시스템보다는 내려받기 애플릿 크기를 줄여 효율을 높였다.

(2) 암/복호화를 위한 키 크기가 미국의 수출 규정에 따라 사용에 문제점을 안고 있다. JAA에서는 국산 암호 알고리즘 채택이 가능하도록 확장성을 고려했기 때문에 키 크기 제약 문제를 해결할 수 있다.

(3) JAA의 인증 모듈이 자바 애플릿 프로그램으로 구현되어 있어서 클라이언트 측 브라우저에서 별도의 설치 과정이 필요 없이 바로 실행시킬 수 있다.

IV. 결 론

현재 인터넷에서 서비스중인 대부분의 웹 시스템에서 사용자 식별 및 인증 방법은 아주 간단하다. 즉, 클라이언트가 인증정보로 사용자 아이디 및 패스워드를 입력하면 이를 수신한 서버가 해당사용자에 일치하는 인증정보 보유 유무를 검사하여 인증이 이루어지는 방법을 택하고 있다.

서론에서 언급했듯이 현재 대부분의 웹 브라우저에서 HTTP1.0을 채택하고 있기 때문에 사용자 인증 정보가 평문상태로 전송된다. 이러한 기존의 웹 시스템들은 인증정보 누출이라는 커다란 취약점을 가지고 있다. 또한 기존의 보안 프로토콜 즉 Kerberos, SSL 등과 같은 것은 기존의 웹서버에 추가로 보안 프로그램을 설치를 하여 상호 동작을 하는 구조로 되어있다. 이는 별도의 상용 보안 소프트웨어 설치를 요구한다.

이러한 문제점 보완을 위해 본 논문에서 제안한 JAA는 사용자 인증에 자바 기술을 사용하여 웹 서버/클라이언트 프로그램에 독립성을 유지하고 안전한 인증이 이루어지도록 하였다.

본 논문에서는 사용자 인증 부분만을 설계하여 그 가능성을 확인하였다. 이를 바탕으로 웹 환경에서 서버 및 클라이언트간에 유통되는 문서에 대한 인증, 기밀성, 무결성, 그리고 전자서명 등의 보안서비스가 제공되는 종합적인 보안 시스템 설계 및 구현이 지속적으로 연구되어야 할 것이다.

참고문헌

[1] Berners-Lee, T., Fielding, R., and H. Frystyk, Hypertext Transfer Protocol HTTP/1.0, Request for Comments : 1945 , IETF, May 1996.

[2] R. Fielding, J. Getty, J.Mogual, H.Frystyk, T.Berners, "Hypertext Transfer Protocol HTTP/1.1, Request for Comments : 2068, IETF, January 1997

[3] A. Freier, P.Karlton, and P.Kocher, The SSL Protocol Version 3.0, http://www.netscape.com/eng/ssl3/3-spec.ps, 1996.3

[4] T. Dierks. et al., The TLS Protocol 1.0, RFC 2246, 1999.1.

[5] E. Rescorla. et al., The Secure HyperText Transfer Protocol, RFC 2660 ,1999.8

[6] Rivest, R., The MD5 Message Digest Algorithm, RFC 1321, ETF, 1992.4

[7] "JavaTM 2 SDK, Standard Edition Documentation", http://java.sun.com/products/jdk/1.2/docs/, Sun Microsystems, 2000

[8] Johnathan Knudsen, "Java Cryptography", O'Reilly, 1998

[9] Macro Pistoia, et al., "Java 2 Network Security" , IBM , 1999