ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

안전한 HTML5 로컬스토리지 구현에 대한 연구☆

Study on implementation of Secure HTML5 Local Storage

명 희 원* 백 정 하** 이 동 훈***
Hee Won Myeong Jung Ha Paik Dong Hoon Lee

요 약

HTML5는 특정 브라우저에 종속되지 않으며 상호 운용성을 고려하는 동시에 기존 HTML과도 하위 호환성을 갖도록 개발되어 현재 표준화 작업이 진행 중인 새로운 웹 표준이다. 이는 최근 스마트폰 시장의 활성화와 함께 모바일 웹 환경에서 다양한 플랫폼에 공통적으로 적용될 수 있다는 특징으로 인해 관심을 모으고 있다. 특히 HTML5에서 새롭게 추가되는 기능들 중 하나인 *Local Storage*는 인터넷 접속이 끊긴 상태에서도 웹 어플리케이션의 동작을 가능하게 하는 오프라인 기능을 지원하며 이는 서버와 독립적인 웹 어플리케이션 개발이 가능하게 한다. 그러나 현재 *Local Storage*는 평문의 데이터를 *Client-side*에 아무런 보안조치 없이 저장하기 때문에 쿠키와 같은 기존의 클라이언트 측 저장소가 갖는 보안 위협에 그대로 노출되어 보안상 안전하지 않다.

본 연구에서는 *Local Storage*에 데이터를 저장할 때 성능부하를 최소화 하면서 데이터의 안전한 저장 및 사용을 가능하게 하는 방법을 제안한다. 제안하는 방법은 기존 *Local Storage* 표준 API와 암호 기능을 제공하는 모듈을 이용하여 안전한 사용자 정보 저장을 지원한다. 또한 제안하는 방법을 실제 구현코드를 바탕으로 성능을 측정하여 효율성을 입증한다.

ABSTRACT

HTML5 has developed not to have browser dependancy considering interoperability as same as maintaining compatability with lower versions of HTML. HTML5, the newest web standardization is on going of being structured. Along with the smart phone boom, HTML5 is spotlighted because it can be applied to cross platforms in mobile web environments. Specially the localStorage that has been listed in new features in HTML5 supports offline function for web application that enables web application to be run even when the mobile is not connected to 3G or wifi. With Local storage, development of server-independent web application can be possible. However Local storage stores plaintext data in it without applying any security measure and this makes the plaintext data dangerous to security threats that are already exist in other client side storages like Cookie. In the paper we propose secure Local storage methods to offer a safe way to store and retrieve data in Local storage guaranteeing its performance. Suggested functions in this paper follow localStorage standard API and use a module that provide cryptographic function. We also prove the efficiency of suggested secure Local storage based on its performance evaluation with implementation.

☞ keyword : HTML5, 클라이언트 사이드 스토리지(Client-side Storage), 웹 스토리지(Web Storage), 로컬스토리지(Local Storage), PBKDF2, Salt

1. 서 론

최근 웹 표준은 보다 많은 기능적 확대를 필요로 하는 사용자의 요구사항을 충족하기 위해 과거 10년간 유지되어왔던 기존 HTML 표준을 확장하여 HTML5[1] 라는 새로운 대안을 제시하였다. HTML5는 자바스크립트와의 유연한 연동과 내부의 강력한 API를 기반으로 기존에 제공하지 못했던 고수준의 웹 애플리케이션을 제공할 수 있어 웹 서비스 및 콘텐츠의 질적 향상을 제공하는 등 그 활용도에 대한 기대가 높다. 이전 HTML 표준과 대비되는 새로운 HTML5 기능 중 하나는 멀티미디어 지원의 강화로써 외부 플러그인에 의존하지 않고 자바 스크립트만으로 동영상을 재생하거나 기본 태그를 통해 플래시와 같은 고수준 멀티미디어 경험을 제공한다. 또 한 가지 확연히 구별되는 기능적 측면은 HTML5 *Web Storage* API를 통한 저장 공간 기능 지원이다.

기존에도 HTML을 기본으로 하는 웹 서비스에서 사용

* 정 회 원 : 고려대학교 정보보호대학원 석사과정
mis1020@korea.ac.kr
** 정 회 원 : 고려대학교 정보보호대학원 박사과정
jungha.paik@gmail.com
*** 정 회 원 : 고려대학교 정보보호대학원 교수
donghlee@korea.ac.kr(교신저자)
[2012/04/29 투고 - 2012/05/15 심사 - 2012/07/16 심사완료]
☆ 본 연구는 지식경제부 및 한국인터넷진흥원의 "고용계약형 지식정보보안 석사과정 지원사업"의 연구결과로 수행되었음
☆ A preliminary version of this paper appeared in ICONI 2011, Dec 15-19, Sepang, Malaysia. This version is improved considerably from the previous version by including new results and features.

자의 저장 공간을 활용하기 위해 많은 기법들이 제안되었다. 가장 대표적인 기술인 쿠키[2] 이외에도 어도비의 Flash[3], 구글의 Gears 등은 모두 특정 벤더들에 의해 Plug-in 형태로 사용자의 저장 공간 활용을 위한 기술로 개발되었다. 이러한 *Client-side storage* 기술은 웹 애플리케이션의 동작에 필요한 정보를 저장함으로써 오프라인 상태에서도 웹 애플리케이션이 동작할 수 있도록 하고, 웹 서버와 클라이언트간의 트랜잭션 최소화 및 서비스 제공 속도 향상을 기대할 수 있는 등 다양한 이점을 가지고 있다. 특히, 모바일의 대중화로 인해 웹 애플리케이션 분야의 모바일 적용이 활발히 이루어지는 현재의 경우 PC와는 달리 3G나 Wi-fi가 연결되어 있지 않은 상태에서도 사용자 작업의 지속성을 유지할 수 있도록 한다는 점에서 *Client-side storage*의 필요성은 더욱 부각된다.

HTML5에서는 *Web Storage*(웹 스토리지)[4]라는 새로운 표준을 제시하여 브라우저간의 호환성을 유지하면서도 사용자 저장 공간을 활용하는 기능을 제공한다. HTML5에서 *Client-side storage*에 대한 표준 접근방법을 제시함으로서 기존 기술들이 갖는 한계점을 극복할 수 있는 방안을 마련하였다. 쿠키의 경우 최신 기술이 반영되는 서비스에 접목되기에는 너무 작은 저장 공간을 제공하는 단점이 있고, Plug-in 기반의 기술들은 브라우저 별로 상이하게 적용 되어야 하는 태생적인 한계 때문에 사용자에게 불편함을 초래한다. HTML5에서 새롭게 제안된 *Web Storage* 기술은 이러한 기존 기술을 모두 대체 가능할 뿐 만 아니라 존재하는 문제점들을 모두 극복할 수 있기 때문에 수많은 서비스에 접목될 것으로 예상된다.

하지만 현재 *Web Storage* 기술은 저장되는 데이터의 보안을 주요하게 고려하고 있지 않기 때문에 데이터의 중요도에 따라 storage 안전성에 심각한 위협이 발생할 수 있다. 특히, *Web Storage* 중 *Local Storage*는 세션종료 또는 웹 브라우저 종료 후에도 명시적으로 삭제하지 않는 한 브라우저가 지정하는 내부 경로에 영속적으로 평문 형태의 데이터를 저장한다. 이 때문에 외부 공격자가 *Local Storage*에 접근만 가능하다면 손쉽게 사용자 정보의 추출이 가능하다. 게다가 기존에 이와 유사한 기능을 제공하는 쿠키를 보호하기 위해 제시된 기법들은 *Local Storage* 기술에 접목하기 어렵다는 한계점이 있다.

따라서 본 논문에서는 *Local Storage*에 사용자 정보 저장 시, 퍼포먼스를 크게 해치지 않으면서 데이터의 안전한 저장 및 사용을 지원하는 알고리즘을 제안한다. 제안하는 기법은 *Local Storage* 기술을 사용하기 위한 표준 API 구성을 준수하여 개발자들이 쉽게 보안 기술을 접목할 수 있도록 구성하였고 보안 기술이 적용되었을 경우 직접적인 파일접근을 통한 공격 이외에도 XSS[5]와 같은 실시간 공격도 모두 차단할 수 있을 것으로 기대한다.

본 연구에서는 성능분석을 위해 Secure local storage를 구현하고 성능을 측정해보았다. 10만개의 입력 레코드에 대해 암호화 처리 후 평균 시간을 측정해 본 결과 단일 저장 정보에 대해 암호화를 고려하지 않은 속도에 비해 약 0.03 ms 정도 딜레이가 생기는 것으로 측정되었다. 이는 실제 웹 페이지 로딩 속도 등을 고려해 봤을 때 사용자 인지에 영향을 미치지 않는 것으로 판단된다.

본 논문의 구성은 다음과 같다. 2장은 *Local Storage* 개요 및 문제점을 분석하고 기존의 클라이언트 스토리지 관련 연구를 살펴본다. 3장에서는 사용자의 패스워드를 기반으로 하는 키 생성 및 암·복호화를 통해 로컬 스토리지에 데이터를 안전하게 저장할 수 있는 방법을 제안한다. 4장은 안전한 로컬스토리지의 성능 및 속도를 분석하며 5장 결론을 끝으로 논문을 마무리 한다.

2. 관련 연구

2.1 Local Storage 개요

2.1.1 Local Storage 특징

HTML5 *Local Storage* 이전에 플러그 인 기술로 개발된 *Client-side storage*들은 개별적으로 구현되어 운영 방식도 상이하다. 이들은 타 브라우저에서 실행될 경우에 해당 브라우저의 다른 추가기능과 호환이 되지 않거나 브라우저 별로 동기화를 해줘야 되는 등의 사용자 불편함을 야기할 뿐만 아니라 (표 1)에서 보는 바와 같이 특정 브라우저에서는 지원이 되지 않는다.

이러한 기능적 한계를 극복하고자 브라우저에 독립적으로 사용할 수 있는 공통의 언어로 작성된 *Client-side storage*의 필요성이 제시되어 왔다. *Local Storage*는

(표 1) 브라우저 별 지원 Client-side storage

Technology	IE 8.0/9.0	chrome 5.0	Opera 10.60	Firefox 3.6.16/4.0	sarafi 5.0.1
Cookie	√	√	√	√	√
IE userData	√	×	×	×	×
Google gears	√	√	×	√	√
Web Storage	√	√	√	√	√

(표 2) Local Storage 지원 브라우저 버전 및 기본 경로(Window 7 기준)

Browser type	IE	Chrome	Opera	Firefox	Safari
Version(+)	8.0+	4.0+	10.50+	3.5+	4.0+
Data format	plain XML	SQLite	SQLite	SQLite	SQLite
default path	%userprofile%\ AppData\ Local\ Microsoft\ InternetExplorer\DOMStore	%userprofile%\ AppData\Local\Google\Chrome\ UserData\ Default\ Local Storage	%userprofile%\ Application Data\Opera\ Opera\pstorage	%userprofile%\AppData\Roaming\Mozilla\Firefox\Profiles\webappstore	%userprofile%\ LocalSettings\ Application Data\Apple Computer\ Safari\ LocalStorage

HTML5 *Web Storage*의 하나로서 기존의 HTTP 쿠키와 비슷하게 사용자 데이터를 브라우저 타입에 상관없이 *Client-side*에 저장할 수 있게 한다. 또한 *Web Storage*의 또 다른 저장 공간인 *Session Storage*(세션 스토리지)와는 다르게 세션 종료 이후에도 영속적으로 저장할 수 있으며 도메인 당 5MB~10MB의 데이터를 사용자 PC의 하드디스크에 저장할 수 있다. 현재 크롬, 파이어폭스, 오페라 등과 같이 다양한 브라우저에서 이러한 *Client-side storage*를 지원하고 있으며 저장 경로와 데이터 처리 방식은 (표 2)와 같다.

*Local Storage*는 기존의 브라우저 저장 공간인 쿠키보다 큰 용량을 제공하며 오프라인 환경에서도 열람 및 편집이 가능한 장점이 있어 모바일과 같이 온라인(Online) 상태를 계속적으로 보장할 수 없는 환경에서 특히 유용하게 쓰일 수 있다. 또한 HTTP request시 마다 파일을 서버에 전송할 필요가 없다는 점도 쿠키와 다르며 이는 네트워크 트래픽 절감 및 페이지 재탐색 속도 향상 등에 기여한다[5].

*Local Storage*를 구성하는 interfaceWindow- -LocalStorage는 *setItem*(key, value), *getItem*(key), *removeItem*(key), 그리고 *clear*() 함수로 이루어진다. *setItem()*은 key/value 쌍으로 저장되는 *Local storage*에 두 값을 저장하는 함수이다. *getItem*()은 저장된 key/ value 쌍으로 이루어진 레코드들에 대해 key값에 해당 value를 리턴하는 함수이다. *removeItem*()은 key를 인자로 취해 매핑되는 value값을 삭제하는 함수이며 *clear*()는 *Local Storage*에 저장된 모든 key/value쌍을 삭제한다.

위에서 언급한 것처럼 *Local Storage*는 단순한 함수를 바탕으로 동작하기 때문에 구현이 쉽고 브라우저 타입에 독립적으로 적용되어 호환성이 보장된다. 또한 기존 *Client-side storage*에 비해 큰 용량을 제공하여 보다 많은 데이터를 저장할 수 있으며 애플리케이션 데이터 저장 시에는 오프라인에서도 웹 애플리케이션을 사용할 수 있도록 하는 장점 등을 바탕으로 웹 애플리케이션 및 웹 서비스에 다양하게 활용될 것으로 기대한다.

2.1.2 Local Storage의 보안 취약점

(표 2)에서 보는 바와 같이 *Local Storage*는 브라우저마다 상이한 저장형태를 가짐에도 불구하고 모든 정보를 평문 그대로 저장한다는 공통점이 있다. *Local Storage*에 존재하는 데이터에 대한 근본적인 보안 위협의 원인은 이처럼 key/value 쌍이 평문으로 로컬에 저장되는 것이다. 클라이언트 PC에 평문으로 저장되는 key/value 쌍은 XSS 공격이나 클라이언트 측에서 타인에 의해 임의로 데이터가 수정 및 삭제 될 수 있는 보안 취약점에 노출될 수 있다.

평문으로 저장되는 *Local Storage* 데이터는 여러 사용자가 공유하는 PC에 저장될 경우, 다른 사용자로 인한 데이터의 수정이나 삭제 등의 문제로부터 자유로울 수 없다. 사용자가 임의로 설정을 변경하지 않는 이상 *Local Storage*의 대부분이 기본적으로 제공되는 경로를 통해 데이터를 저장하므로 여러 사용자가 공유하는 공용 PC에서는 임의의 데이터 침해가 발생할 수 있다. 또한 데이터의 저장기한 설정옵션이 부재하기 때문에 사용자가 삭제하지 않을 경우 저장된 평문정보가 영속적으로 보관되며 이는 PC에 발생할 수 있는 클라이언트 보안 취약점에 장기적으로 노출되도록 한다. 특히 사용자의 민감한 정보를 탈취하려고 하는 악성코드 등이 설치되어 있을 경우에도 로컬에 저장된 의미 있는 평문 정보가 그대로 공격자에 전송될 수 있다[5].

클라이언트 브라우징의 대표적인 취약점인 XSS 공격 또한 위협이 된다. XSS 공격은 자바스크립트가 클라이언

트 웹 애플리케이션에 접근할 수 있는 점을 이용하여 악의적인 코드를 삽입한 페이지를 사용자 웹 브라우저에서 실행시켜 세션정보 및 쿠키정보 등을 공격자에게 전송하는 공격방법이다[6]. 최근에는 기존 이메일 항목 등을 *Client-side storage*에 저장하고 오프라인 환경에서 사용할 수 있도록 하는 애플리케이션의 개발이 활발히 이루어지고 있다. 이러한 서비스의 경우 애플리케이션 실행에 대한 서버의 간섭이 불필요하기 때문에 서버 측에서 동작하는 악성코드 필터나 인코더를 우회할 수 있게 된다. 또한 클라이언트 측의 *Local Storage*는 쿠키처럼 접근권한을 session ID 등으로 보호하는 기능이 없기 때문에 저장된 데이터는 공격자의 공격 목표가 될 수 있다.

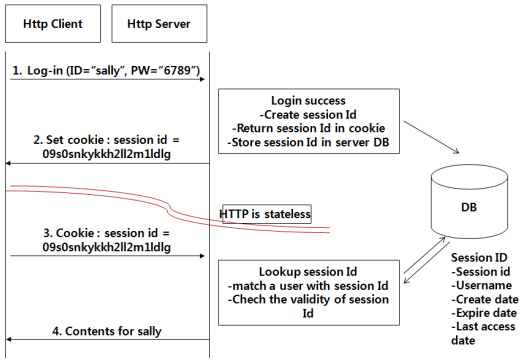
W3C에서 발표한 *Web Storage* 표준 초안에는 사용자에게 종속되는 민감한 정보는 되도록 *Local Storage*에 저장하지 않기를 권장하고 있으며[4] 이 외의 별도의 보안대책을 마련하지 않고 있다. 오프라인 환경에서 웹 애플리케이션을 사용하는 사용자의 많은 수가 인터넷을 이용할 수 없는 환경에서 언제든지 사용자의 이메일 등 개인정보에 접근하기 위해 *Client-side storage*를 이용함을 감안할 때 저장되는 정보를 보호하기 위한 구체적인 대안이 필요하다.

따라서 위와 같은 위협으로부터 사용자의 데이터를 안전하게 보호하기 위해서 다음 3장에서는 *Local Storage*를 보안 위협으로부터 안전하게 보호할 수 있는 구현방법을 제안한다.

2.2 안전한 Client-side storage 보호

HTML5 *Local storage*는 기술의 보급이 아직까지는 대중화 되어 있지 않기 때문에 보안기술에 대한 고려가 구체적으로 이루지지 않은 반면 기존의 *Client-side storage* 기능을 제공한 쿠키에 대한 보안 연구는 상당히 활발하게 이루어진 편이다. 특히 쿠키는 메모리의 지정된 경로상에 저장된다는 점에서 *Local Storage*와 유사하지만 쿠키를 보호하기 위해 제안된 기술을 *Local Storage*에 접목시키기 위해서는 다양한 고려가 반드시 선행적으로 이루어져야 할 것으로 보인다.

HTTP 쿠키의 경우 사용자의 상태 정보에 대해 서버가 트래킹을 할 수 있도록 하는 기능을 제공하기 위한 목적을 갖기 때문에 서버와 클라이언트 간에 주고받는 HTTP requests에 따라 값이 생성되고 저장된다. 또한 쿠키정보는 사용자의 개인정보 및 세션정보를 포함하는데 이는 기존에 알려진 공격들에 취약하므로 저장된 쿠키의 기밀



(그림 1) HTTP 쿠키의 동작 원리

성과 무결성을 보장하기 위한 연구가 여러 차례 발표되었다.

Jong-Phil Yang 등[7] 은 CA로부터 발행된 공개키 인증서를 이용하는 Secure cookie set을 구현하였다. Secure cookie set은 클라이언트의 인증서와 서버의 공개키로 암호화된 대칭키, 그리고 대칭키로 암호화된 사용자의 패스워드를 포함한다. 최종적으로는 서버의 개인키로 Cookie set의 다이제스트를 생성하여 쿠키를 보호한다.

Heng Wu 등[8] 은 Mac주소를 유일한 식별자로 설정하고 키 링 구조를 이용하여 클라이언트-서버 간에 공유되는 쿠키에 대해 암·복호화할 수 있는 방법을 제안하였다. 이는 클라이언트와 서버가 서로의 공개키를 교환하고 서버는 현재 시간으로부터 유도한 키로 암호화한 쿠키, 쿠키의 암호화키와 매칭 되는 key identifier, 타임스탬프와 서버의 서명 등을 포함하는 digital envelope을 사용자의 공개키로 암호화하여 사용자에 전송한다. 클라이언트는 자신의 개인키로 digital envelope을 복호화하고 서버의 서명 및 타임스탬프의 유효성을 검증한 뒤 쿠키를 저장한다. 서버와 클라이언트가 쿠키를 주고받기 위해서는 Mac 주소로 암·복호화 되는 키 링에서 key identifier와 매칭되는 키로 쿠키정보를 확인한다. 위와 같은 공개키 인증서 기반의 암호학적인 방법은 통신 시 마다 클라이언트와 서버가 각자의 인증서 정보를 교환하는 과정이 필요하다. 그러나 *Local Storage*가 네트워크에 연결되지 않은 오프라인 환경에서는 사용자가 인증서 정보를 교환할 수 없기 때문에 이를 *Local Storage*에 적용하는 것은 불가능하다.

Chan[9]은 웹 프록시를 경유하는 쿠키값을 스마트카드와 PIN을 이용하여 쿠키를 저장하고 사용자 접근 통제와 같은 쿠키 관리가 가능한 아키텍처를 제안하였다. 클

라이언트와 서버 중간에 위치한 웹 프록시는 두 개체가 주고받는 HTTP-request와 HTTP-response에 대한 쿠키를 저장하고 통신이 종료되면 이를 스마트카드 리더기를 통해 스마트카드에 저장한다. 스마트카드는 사용자의 패스워드를 통해 접근통제를 할 수 있다.

Rattipong Puttacharoen 등[10] 은 웹 서버과 클라이언트 사이에 위치하는 웹 프록시를 이용하여 랜덤하게 쿠키값을 변경하여 클라이언트에 대한 XSS 공격에 견고할 수 있도록 구현한 논문을 발표하였다. 웹 프록시는 치환 테이블을 이용하여 클라이언트에 저장할 랜덤하게 변경된 쿠키값과 서버로 전달될 실제 쿠키값을 매핑/저장한다. 따라서 웹 프록시를 통해 전달되는 클라이언트 단에 실제 쿠키값이 저장되지 않아 클라이언트를 가장하기 위해 쿠키값을 탈취하고자 하는 XSS공격을 무력화 시킬 수 있다.

웹 프록시를 이용한 보안 방법은 서버와의 통신이 비주기적으로 발생하는 *Local Storage*에 적용하기에 적절하지 않다. 쿠키에 저장되는 데이터는 서버와의 통신을 목적으로 작성되기 때문에 데이터를 주고받을 때 웹 프록시에 대한 접근이 자유롭다. 하지만 *Local Storage*는 오프라인 환경에서 웹 애플리케이션 실행 시에 필요한 정보를 로컬이 아닌 웹 프록시를 통해 저장하게 되면 *Local Storage* 본래의 기능을 제한하게 된다. 또한 온라인 환경에서 부분적으로 적용할 수 있다 할지라도 웹 프록시 서버에 저장되는 내용이 공격자에 의해 변조될 가능성이 존재한다.

이 밖에도 *심원태* 등[11] 은 쿠키파일의 안전성을 보장할 수 있는 방법으로 PC 내에 보안영역을 설정하여 쿠키파일을 저장, 관리할 수 있는 방안을 제시하였다. 이는 필요시마다 쿠키 디렉토리로 요구되는 쿠키파일을 이동시켜 사용하는 방법으로, 사용자와 서버의 통신이 종료되면 새롭게 생성된 쿠키를 보안 영역의 쿠키를 비교하고 업데이트 한 후, 보안 영역으로 해당 쿠키를 옮긴다. 이는 쿠키 디렉토리에 쿠키 정보를 남기지 않아 악성코드에 의한 정보 유출을 방지할 수 있다.

하지만 이 기법은 보안 영역이라는 별도의 영역을 확보하기 위해 하드웨어 종속적인 요소가 반드시 필요하므로 브라우저 호환성을 강조하는 HTML5의 목적과 어긋날 수 있으며 스마트폰과 같은 모바일 장치 등에 적용하기 어렵다는 단점이 있다.

*Local storage*는 HTML5의 제정이 논의된 시점부터 데이터 보안상의 이슈가 끊임없이 제기되어온 부분이다[12-14]. 현재 *Local storage* 기술문서는 권장기술 후보(Candidate Recommendation)로 앞으로 발표될 정식 표준과 기능상 거의 동일하다. 그럼에도 불구하고 *Local storage*의 보안 이슈에 대한 연구는 아직까지 활발하게 진행되지 않고 있으며 뚜렷한 연구결과도 부재한 상황이다[15].

기존 쿠키에 대한 보호기술은 쿠키와 *Local Storage*의 동작방식이 확연히 다르다는 점과 다양한 환경(온/오프라인 및 여러 플랫폼)에서 동일하게 보안을 제공해야하는 목적 때문에 *Local Storage*에 접목하기 어렵다. 따라서 *Local Storage* 기능에 특화되고 여러 환경에서 호환성을 제공하는 사용자 저장 데이터 보호 방법이 반드시 필요하다.

3. 안전한 로컬스토리지 구현 제안

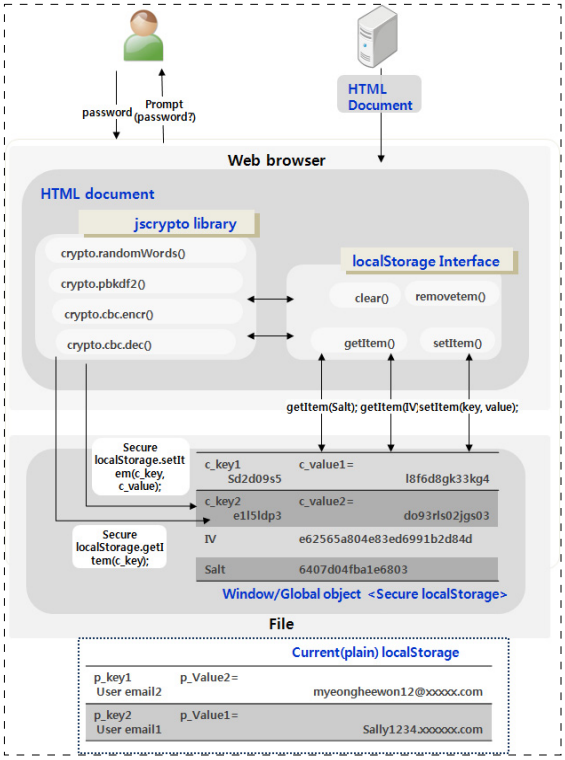
3.1 Secure localStorage Service

3.1.1 보안 요구사항

본 연구에서 제안하는 Secure localStorage는 평문형태로 저장되는 *Local Storage* 상의 데이터를 보호하기 위해 제안되는 암·복호화 할 수 있는 기능을 포함하는 *Client-side storage*이다.

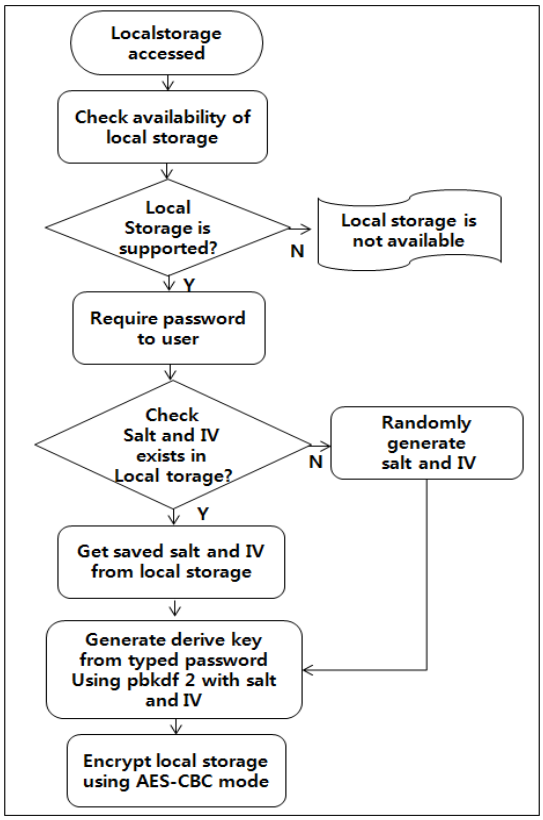
안전한 로컬스토리지 디자인 고려사항 및 보안 요구사항은 다음과 같이 정의할 수 있다.

1. 기존 사용자는 보안을 적용한 *Local Storage*에도 직접적으로 접근하여 데이터 조회, 삭제 및 수정할 수 있어야 한다. 따라서 보안에 필요한 비밀 파라미터 등은 사용자가 생성하고 관리할 수 있어야 한다.
2. 해당 사이트의 XSS 취약점 여부에 따라 L*ocal Storage* 저장정보의 안전성이 위협받기 때문에 인가받지 않은 주체의 script 실행을 통한 *Local Storage* 정보 추출에 대한 기밀성을 고려해야 한다.
3. *Local Storage*에 데이터를 저장하는 기능은 browser에서 제공하며 모두 평문 형태의 파일로 저장되기 때문에 악성 프로그램 등을 통한 외부 공격자의 절대 경로 접근 시에도 저장 정보를 안전하게 보호해야 한다.
4. 보안기법을 구현 시 웹 스크립트 언어의 특성 때문에 공격자가 구현 코드를 쉽게 조회할 수 있으므로 구현 코드 내에 키를 포함한 암호학적 비밀정보를 명시하면 안 되고, 구현 기법으로부터 어떠한 비밀정보도 노출 되서는 안 된다.



(그림 2) Secure local storage 서비스 개요

5. HTML5 로컬스토리지의 특징에 따라 인터넷에 연결되지 않은 상태를 고려하여 클라이언트에서 PKI 기반의 공개키 암호 처리는 부적절하다. 따라서 대칭키 기반의 암호 처리를 사용해야 한다.
6. 데이터는 사용자의 로컬에 파일로 저장된다. 따라서 고정된 키를 사용할 경우, 한 대의 PC를 여러 사용자가 공유하는 환경에서 모든 사용자의 합법적인 접근을 허용할 수 있으므로 각 사용자마다 다른 접근권한을 지원해야 한다.
7. 도메인마다 개별적인 *Local Storage*를 제공하며 각각 다른 패스워드를 사용해야 한다. 그러나 사용자가 모든 도메인에 대해 비밀번호를 기억할 수 없을 것을 고려하여 같은 패스워드에서도 다른 암·복호화 키가 생성되도록 한다. 이는 사용자 편의성을 보장할 수 있다. 또한 입력받는 패스워드가 너무 짧을 경우 패스워드 사전 공격이나 brute force 공격과 같은 전수 공격으로부터 안전성을 확보하기 위해 최소한 10자리 이상 입력받도록 구성해야 한다[16].



(그림 3) Secure localStorage 개념도

3.1.2 Overview

제안하는 Secure localStorage는 (그림 2)와 같은 암·복호화 과정을 거친 데이터를 제공한다. jscrypto.library는 자바 스크립트 기반의 암호 알고리즘 구현으로서 대칭키 암호 알고리즘, 패스워드 기반 키 생성함수, 랜덤 생성기를 포함한다. 이는 HTML5의 기본 Local Storage 인터페이스(getItem() / setItem() 등)와 연동되어 사용자의 저장 정보를 관리한다. 또한 (그림 2)의 Window/Global object <Secure localStorage>에서 볼 수 있듯이 제안하는 기법은 저장 value 뿐 만 아니라, key의 기밀성도 지원하기 때문에 올바른 패스워드를 모르는 주체는 정보 자체는 물론 어떤 정보를 포함하고 있는지조차 파악하기 어렵다.

Secure localStorage의 대략적인 순서는 다음과 같이 진행된다. 사용자가 웹 사이트에 접근하면 서버로부터 전송되는 html문서의 자바스크립트는 해당 브라우저가 Local Storage를 지원하는지 확인한다. Local Storage를 지

```
Algorithm1.
    Set_Parameter(password, count, derivekeylen)
<precondition>
- userPW는 interactive prompt 통해 입력
input : password, count, derivekeylen
output : 1(Success) or 0(Failure)
1   var Cipher;          // cipher instance
    var derivekey;       // secret key for block cipher
    var IV;              // Initial Vector
    var salt;            // random salt
2   if( Available_Check( ) == 0 )
        // Local Storage is not supported
        return 0;
3   if( ( IV ← localStorage.getItem("IV") ) is null )
        IV ← crypto.random.Words();
        localStorage.setItem("IV",IV);
4   if( ( salt ← localStorage.getItem("salt") ) is null )
        salt ← crypto.random.Words();
        localStorage.setItem("salt",salt);
5   derivekey ←crypto.pbkdf2
                (password, salt, count, derivekeylen);
5   Cipher ← new crypto.cipher.aes(derivekey);
6   return 1;
```

(그림 4) Set Parameters for Secure localStorage

원하는 브라우저라면 사용자는 Local Storage를 이용하여 사용자의 데이터를 저장할 수 있다.

사용자가 Local Storage에 접근 시 프롬프트를 띄워 패스워드를 입력받는다. 입력된 패스워드는 IV와 salt와 함께 PBKDF 2 함수의 인자로 사용되며, 함수처리의 결과는 *Local Storage*에 작성된 데이터 셋을 암·복호화하는 대칭키로 사용된다. 생성된 대칭키를 이용해 데이터를 암호화하여 *setItem*()함수를 이용하여 로컬 스토리지에 저장하거나 복호화를 위해 *getItem*()함수를 이용하여 데이터 셋을 브라우저 영역으로 호출할 수 있다.

3.1.3 안전한 HTML5 Local Storage 구성 기법

서비스 초기화 Local Storage 서비스 초기화는 Local Storage를 사용하는 주체가 기존 사용자인지 여부에 따라 다르게 설정된다. 만약 기존 사용자라면 Secure localStorage에 이전 접근 시에 저장된 IV 및 salt가 존재한다. 따라서 localStorage.getItem() 함수로 IV와 salt를 리턴하여 사용한다. 반면 새롭게 Local Storage를 이용하려는 사용자라면 랜덤하게 IV와 salt를 생성해야 한다. (그림 4)는 이러한

```
Algorithm2.
    Secure localStorage.setItem(key, value)
<precondition>
- Set_Parameter 함수를 통해 파라미터가 초기화 되어 있어야 함
input : key, value
output : 1(Success) or 0(Failure)
1   var Cipher;          // cipher instance
    var IV;              // Initial Vector
    var p_key;           // plain key
    var p_value;         // plain value
    var c_key;           // encrypted key bits
    var c_value          // encrypted value bits
2   if( Available_Check( ) == 0 )
        // Local Storage is not supported
        return 0;
3   if  (Any of Cipher, p_key, p_value and IV is null)
        return 0;
4   else
        c_key ← crypto.cbc.enc(Cipher, p_key, IV);
        c_value ← crypto.cbc.enc(Cipher, p_value, IV);
5   localStorage.setItem(c_key, c_value);
6   return 1;
```

(그림 5) Secure localStorage.setItem(key,value)

초기화 과정을 나타낸 것이다.

저장방법 (그림 5)에 나타난 것처럼 Secure localStorage.setItem() 암호화 알고리즘을 거쳐 생성된 (key, value) 쌍을 *localStorage.setItem*()으로 Secure localStorage에 저장한다. 암호화에 앞서 사용자가 *local storage*에 접근하면 *Local Storage*를 지원하는 브라우저인지 확인하고 지원하지 않는다면 0을 반환한다. Cipher, p_key, p_value와 IV 중 하나라도 null인 경우에도 0을 반환한다. 그러나 Cipher, p_key, p_value와 IV가 존재하는 경우에는 암호화 인스턴스 Cipher와 암호화하고자 하는 p_key, 그리고 IV를 이용하여 암호화된 key c_key를 생성하고 동일하게 c_value를 생성한다. 두 값은 localStorage.setItem(c_key, c_value)으로 Secure localStorage에 저장된다.

저장정보 획득 암호화 과정과 동일하게 사용자가 Local Storage에 접근하면 Local Storage를 지원하는 브라우저인지 확인하고 지원하지 않는다면 0을 반환한다. Cipher, p_key와 IV 중 하나라도 null인 경우에도 0을 반환한다. 그러나 Cipher, p_key, IV가 존재하는 경우 찾으려 하는 p_value에 매칭 되는 p_key를 암호화하여 검색하

```
Algorithm3.
            Secure localStorage.getItem(key)
<precondition>
- Set_Parameter 함수를 통해 파라미터가 초기화 되어 있어야 함
input : key
output : value(Success) or 0(Failure)
1   var Cipher;          // cipher instance
    var IV;              // Initial Vector
    var p_key;           // plain key
    var p_value;         // plain value
    var c_key;           // encrypted key bits
    var c_value          // encrypted value bits
2   if( Available_Check( ) == 0 )
        // Local Storage is not supported
        return 0;
3   if  ( Any of Cipher, p_key, IV is null )
        return 0;
4   else
        c_key ← crypto.cbc.enc(Cipher, p_key, IV);
5   if(( c_value ← localStorage.getItem("c_key")) is null )
        return 0;
6   else
        p_value ← crypto.cbc.dec(Cipher, c_value, IV);
7   return 1;
```

(그림 6) Secure localStorage.getItem(key)

기 위한 c_key를 생성한다.

*localStorage.getItem(c_key)*로 호출한 c_value가 null이면 0을 반환하고 그렇지 않으면 c_value를 복호화 한다.

3.2 Contribution

제안하는 알고리즘은 기존 *Local Storage*에 평문으로 저장되는 key, value쌍을 암호화하여 *Local Storage*의 표준함수인 *localStorage.setItem()*를 통해 저장하고 이를 *localStoage.getItem()*로 반환하여 다시 복호화 할 수 있도록 설계되었다.

암호화된 데이터를 저장하는 Secure localStorage는 데이터 복호화를 위해 *localStoage.getItem(c_key)*로 데이터를 브라우저 영역으로 호출한다. 받아오는 *c_value*는 오직 올바른 패스워드를 알고 있는 사용자만이 원본 값으로 복호화 할 수 있기 때문에 안전하다. 또한 브라우저가 기본 경로에 저장하고 있는 (*key,value*) 정보들은 모두 암호화 되어 있기 때문에 공격자에 의해 외부로 노출 되어도 원본 정보(value) 뿐 만 아니라, 어떤 정보로 저장하고 있는지에 대한 내용도 알 수 없다.

즉, storage 영역에 평문 정보가 머무르지 않도록 하여 *Local Storage*에 대한 강한 기밀성을 보장할 수 있다. 만약 공격자에 의해 *c_key, c_value*가 노출되고 이에 상응하는 p_key, p_value가 예측 가능하다 할지라도 PBKDF2에 사용되는 salt가 도메인마다 다르게 생성되기 때문에 생성되는 키에 대한 사전공격이 시도되기 어렵다[16]. 제안하는 알고리즘에는 PBKDF2의 안전한 사용을 위해 표준에서 권고하는 바에 따라 최소 64bits의 salt를 사용하고 최소 1000번 이상의 iteration을 사용하는지 검사하는 루틴이 포함되어 있다.

기존 *Local Storage*에는 평문 형태로 저장되기 때문에 민감한 정보를 저장할 수 없었으나, 제안하는 방법을 통해 이메일과 같은 개인정보도 *Local Storage*에 저장할 수 있도록 하였다.

4. 성능 평가

안전한 *Local Storage* 사용을 위해 제안한 Secure localStorage의 성능은 암·복호화를 수행하기 위해 Parameter setting에 걸리는 시간과 암·복호화된 레코드를 *Local Storage*에 저장 및 로드하는 *setItem*(), *getItem*() 함수의 수행 시간을 측정하여 확인하였다. 암·복호화 기능을 코드로 구현하기 위해 SJCL(Stanford Javascript Crypto Library)의 sjcl.cipher.aes을 사용하였으며 운영 모드는 CBC 모드를 채택하였다[17].

성능분석에 사용된 PC는 3.2 Ghz intel core i5 CPU, DDR3 2GBytes RAM, Window 7 운영체제를 사용하며, 파이어폭스 브라우저를 사용하여 테스트하였다.

가장 먼저 초기화 오버헤드를 테스트하기 위해 IV와 Salt가 null 값일 때(초기 사용 시)의 파라미터 세팅 과정과, *Local Storage*에 IV와 Salt가 존재할 경우의 파라미터 세팅 과정을 100회 시행하여 각각을 측정한 후 평균 시간을 구하였다. (표 3)은 각 경우에 대한 Parameter Setting 소요 시간을 나타낸 것이다. 해당 결과에 나타난 두 가지

(표 3) parameter setting time

Parameter setting	Time(ms)
(IV, Salt) == null	63
(IV, Salt) != null	59

(표 4) Storage 입/출력 시간(ms)

	Local Storage	Secure localStorage
setItem()	0.026	0.064
getItem()	0.022	0.061

경우의 속도 차이를 통해 새로운 IV와 Salt를 생성하는데 걸리는 시간도 추정해 볼 수 있다.

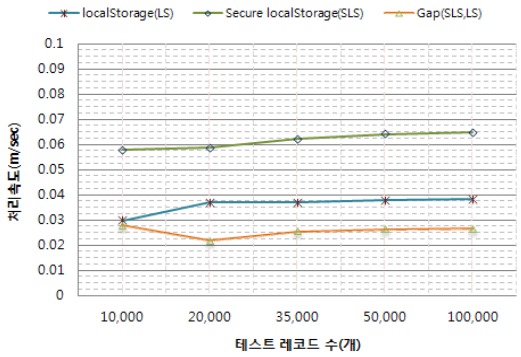
*localStorage.setItem()*함수를 이용하여 *Local Storage* 및 Secure localStorage에 10만 개의 레코드를 입력하는 작업을 각각 100회 수행하였다. (표 4)의 storage 입/출력 시간은 100회 측정된 시간을 각각 10만으로 나눠 하나의 레코드를 입력하는데 걸린 시간을 구하고 이들의 평균 시간을 구한 것이다. 또한 Storage 출력 시간은 10만 레코드가 저장되어 있는 *Local Storage* 및 Secure localStorage를 바탕으로 key 값을 랜덤 선택하여 총 10만 레코드를 불러오는 과정을 100회 시행하면서 각각의 시간을 측정 한 후, 동일한 방법으로 하나의 레코드를 불러오는 평균 시간을 구하였다.

Storage 입력 시간은 (표 4)에서 보는 것과 같이 HTML5에서 기본으로 제공하는 Local Storage의 setItem() 함수를 통해 key 및 value를 저장하는 경우 테스트 결과 약 0.026 ms의 시간이 소요되는 것으로 나타났다. Secure localStorage.setItem() 함수를 통해 key 및 value를 저장하는 Secure localStorage의 경우 약 0.064 ms의 시간이 소요되는 것으로 측정되어, 기존 Local Storage와 약 0.038 ms의 차이를 보였다.

Storage 출력 시간은 HTML5에서 기본으로 제공하는 localStorge.getItem() 함수를 통해 value를 불러오는 경우 테스트 결과 약 0.022 ms의 시간이 소요되는 것으로 나타났다. Secure localStorage.getItem() 함수를 통해 value를 불러오는 Secure localStorage의 경우 약 0.061 ms의 시간이 소요되는 것으로 측정되어 기존 Local Storage와 약 0.039 ms의 차이를 보였다.

(그림 7)은 기존 Local Storage와 본 논문에서 제안한 Secure localStorage에 저장된 데이터 레코드의 수를 각각 1만, 2만, 3.5만, 5만, 10만개로 구성한 후, 각 경우에 한 레코드의 평균 저장 속도를 측정한 그래프이다. Storage 입/출력 시간 방법과 동일하게 각 구성별로 100회씩 실시한 후 평균 시간을 측정 하였다. 측정 결과는 선형적인 증가율을 보이지만 각각의 속도 차는 0.02ms ~ 0.03ms 정도로 매우 미미하게 나타났다.

모든 테스트 결과들을 종합한 결과, 데이터 암/복호화



(그림 7) 테스트 레코드 수 변화에 따른 처리속도 측정

연산 및 현재 storage에 저장된 레코드의 수(스토리지의 크기)는 *Local Storage*의 데이터 입/출력 시간에 큰 영향을 미치지 않음을 알 수 있었으며, Secure localStorage와 기존 *Local Storage* 속도 차이는 사용자가 감지하지 못할 정도의 차이를 보인다.

5. 결　론

HTML5는 새로운 웹 표준으로서 다양한 웹 서비스 분야에 긍정적인 영향을 미칠 것으로 기대되는 기술이다. 특히 Local Storage는 표준화된 *Client-side storage*를 브라우저에 독립적으로 제공하여 오프라인 기능을 지원하는 웹 애플리케이션의 활용을 극대화 할 수 있을 것으로 보인다. 반면 평문 형태로 데이터를 저장하는 *Local Storage*는 기존 *Client-side storage*의 취약점에 그대로 노출되어 있어 이에 대한 대안이 절실히 요구된다. 기존에 선행된 *Client-side storage* 보안 관련 연구들은 *Local Storage*의 오프라인 기능 지원을 고려했을 때 적용이 어렵기 때문에 *Local Storage*만을 위한 새로운 대안이 필요하다.

본 연구에서는 *Local Storage*에 존재하는 보안 위협에 대비하여 안전하게 데이터를 저장할 수 있는 방법으로 사용자 패스워드 기반의 암호화 방법을 제안하였다. 이를 실제로 구현하여 성능을 측정한 결과, 기존의 HTML5에서 제공하는 *Local Storage*의 성능에 견줄 만큼 미미한 속도차를 보이는 것을 확인하였다.

향후에는 이와 관련하여 *Local Storage*에 대한 보안을 웹 서비스 제공자 측에서 제공하지 않더라도 사용자가 개별적으로 보안을 적용 할 수 있는 방안에 대해 연구할 계획이다.

참 고 문 헌

[1] W3C Working Draft 'A vocabulary and associated APIs for HTML and XHTML.', May 2011.

[2] W3C Working Draft 'HTTP Specifications and Drafts.', Mar 2002.

[3] Adobe Flash Player, 'Flash Professional CS6 / Tech specs.'

[4] W3C Working Draft 'Web Storage Editor's Draft', April 2002.

[5] SANS Institute InfoSec Reading Room, 'The RIsks of Client-Side Data Storage', 2011.

[6] OWASP(The Open Web Application Security Project), 'Cross-site Scripting(XSS)', Aug 2011.

[7] Jong-Phil Yang, Kyung-Hyune Rhee 'The Design and Implementation of Improved Secure Cookies Based on Certificate', INDOCRYPT 2002, LNCS 2551, pp.314-325, 2002.

[8] Heng Wu, Weiting Chen, Zhongjie Ren 'Secure Cookies with a MAC Address Encrypted Key Ring'', IEEE Computer Science, Vol 2, pp.62-65, 2010.

[9] Alvin T.S Chan 'Mobile Cookies Management on a Smart Card', Communication of the ACM, Vol 48. No.11, page 38-43, 2005.

[10] Rattinpong Putthacharoen, Pratheep Bunyatnoparat, 'Protecting Cookies from Cross Site Script Attacks Using Dynamic Cooies Rewriting Technique', ICACT 2011, pp.1090-1094, 2011.

[11] 심원태, 최요한, 서희석, 노봉남, '쿠키파일의 보안성 향상을 위한 저장 방식', 한국시뮬레이션학회 논문지, Vol 20, No 1, pp.29-37, 2011.

[12] NetworkWorld, 'HTML5 raises new security issues', [Online]. Available : http://www.networkworld.com/news/2010/082010-html5-raises-new-security.html

[13] SEC Discover, 'Abusing HTML 5 Structured Client-side Storage', July 2008

[14] Compass Security, 'HTML5 web security', Dec 2011

[15] W3C, 'Web Storage Candidate Recommendation' , Dec 2011

[16] NIST Special Publication 800-132, 'Recommendation for Password-Based Key Derivation Part 1 : Storage Applications', Dec 2010

[17] Stanford Javascript Crypto Library, [Online]. Available : http://crypto.stanford.edu/sjcl/

◐ 저 자 소 개 ◑

명 희 원 (Hee Won Myeong)
2011년 고려대학교 경영정보학과 졸업(학사)
2011년~현재 고려대학교 정보보호대학원 석사과정
관심분야 : 금융보안, 암호프로토콜, 암호이론, 인터넷보안, 개인정보보호
E-mail : mis1020@korea.ac.kr

백 정 하 (Jung Ha Paik)
2006년 고려대학교 수학과 졸업(학사)
2006년~2008년 고려대학교 정보경영공학전문대학원 졸업(석사)
2008년~현재 고려대학교 정보보호대학원 박사과정
관심분야 : 암호프로토콜, VANET, 스마트폰 보안, 클라우드 컴퓨팅, 애드 혹 네트워킹
E-mail : jungha.paik@gmail.com

이 동 훈 (Dong Hoon Lee)
1983년 고려대학교 경제학과(학사)
1987년 Oklahoma University 전산학과(석사)
1992년 Oklahoma University 전산학과(박사)
1993년~1997년 고려대학교 전산학과 조교수
1997년~2001년 고려대학교 전산학과 부교수
2001년~현재 고려대학교 정보보호대학원 교수
관심분야 : 암호프로토콜, 암호이론, USN이론, 키 교환, 익명성 연구, PET 기술
E-mail : donghlee@korea.ac.kr