# Ajax를 기반으로 한 인증 및 세션 관리

## Authentication and Session Management based on Ajax

남 상 온* Rolyn C Daguil** 김 기 원*** 송 정 길****
Sang-On Nam Rolyn C Daguil Gi-Weon Kim Jung-Gil Song

### 요 약

Ajax의 상호작용 모델은 웹 어플리케이션 상황을 HTTP에서 상태 기반형이 되도록 바꾸어 주며, Ajax 응용프로그램은 브라우저상에서 오래 지속된다. XHR(XML HTTP Request)는 데이터 교환을 촉진하는데 사용된다. 이러한 상호작용 조건에서 HTTPS를 사용하는 것은 데이터 교환의 빈도 때문에 실용적이지 못하다. 더욱이 민감한 정보의 경우 HTTP에서 HTTPS 로 프로토콜을 전환하는 것은 원본서버 유지정책으로 인해 허용되지 않는다. Ajax 어플리케이션의 장기 지속성, 구속성, 비동기성과 같은 특징들은 재 인증을 촉발하는 각기 다른 인증 및 세션 처리 메카니즘을 필요하게 되는데, 이 논문은 Ajax를 사용하는 인증 및 세션 관리의 설계를 제시한다. 본 설계는 OTP(One Time Password)와 유사한 자동 발생 패스워드가 있는 요약 인증을 사용하는 환경에서 기간 단위 및 이벤트 기반의 재인증을 촉발하도록 고안되었다. 이 인증 및 세션 관리는 인증 및 세션 관리가 손상되지 않도록 커플링의 AWAsec(Ajax Web Application Security) 이라고 불리는 체제 안에 포함된다.

### Abstract

Ajax interaction model changes the posture of web application to become a stateful over HTTP. Ajax applications are long-lived inthe browser. XMLHTTPRequest (XHR) is used to facilitate the data exchange. Using HTTPS over this interaction is not viable because of the frequency of data exchange. Moreover, switching of protocols from HTTP to HTTPS for sensitive information is prohibited because of server-of-origin policy. The longevity, constraint, and asynchronous features of Ajax application need to have a different authentication and session handling mechanism that invoke re-authentication. This paper presents an authentication and session management scheme using Ajax. The scheme is design to invoke periodic and event based re-authentication in the background using digest authentication with auto-generated password similar to OTP (One Time Password). The authentication and session management are wrapped into a framework called AWASec (Ajax Web Application Security) for coupling to avoid broken authentication and session management.

☞ Keyword : AJAX, Authentication, Session Management

## 1. Introduction

Web application facilitates wide area access that can stretch across the globe. It delivers services to the client effectively of less cost using the World Wide Web (WWW). WWW has two major limitations: un-trusted domain (Internet) and stateless protocol (HTTP). However, various technology and mechanism such as HTTPS and session handling has been developed to answer the problem. Yet web applications are still facing various attacks because vulnerabilities due to poor development practices. Broken authentication and session management is one of the vulnerabilities listed by OWASP [2].

* 정 회 원: 대원과학대학 컴퓨터정보계열 교수
nso@daewon.ac.kr
** 정 회 원: Graduate School, Computer Engineering, Hannam University,Korea
rcd@hannam.ac.kr
*** 정 회 원: 초당대학교 컴퓨터과학과 교수
kwkim@chodang.ac.kr
**** 종신회원: 한남대학교 컴퓨터공학과 교수
jksong@hananm.ac.kr
[2006/10/09 투고 - 2006/10/10 심사 - 2006/11/21 심사완료]

With the desire to raise the web application into higher level of interaction as describe in Web 2.0 [3], various plugins and browser extension such as Flash and applets were designed and deployed. Finally, Ajax is coming into the picture, which provides synergy of the existing technology that gives shine to web application. Ajax provides a mechanism and functionality for asynchronous interaction to the server. It request data in the background that the browser can perform some navigation on the client while waiting for the response from the server. Ajax's functionality comes along with strong and weak points in the web application. The latter have more concern in security issue in web application as a whole.

Web applications are vulnerable to various attacks. Incorporating Ajax into it increases the surface area of attack. It provides additional entry points into our application. However, it also provides a way to perform various security operations in the web such as authentication and session management.

With Ajax, the web application becomes stateful and long-lived in the browser. To avoid compromise within the interaction, a new authentication mechanism must be in-placed to augment the user/password authentication. The authentication process must be asserted in a repetitive manner within the active session. Session on the other hand, provides state between pages in HTTP transaction. It is used as a ticket to avoid typing of user/password over and over again. The handling of this session must be tightly coupled to authentication to avoid breaks. This coupling can be done in the background using Ajax.

JavaScript controls the flow of the client in Ajax application. It can do operations such as MD5 computation in the background without diverting the flow of the application. Re-authentication process is done using script and XHR. The process is asserted periodically or on event basis where authentication and session management parameters are exchanged between the server and the client. HTTPS is a secured protocol. However, the frequency of data exchange in Ajax application degrades the performance and switching of protocols from HTTP to HTTPS is not allowed by default because of Javascript security model.

This paper presents an authentication and session management mechanism using Ajax. It describes the framework and its components that are tightly coupled to avoid broken authentication and session management.

## 2. Background and Related Works

### 2.1 AJAX

AJAX [1] is not a technology. It is an approach to web applications that includes a couple of technologies listed below:

- For binding all the technologies below and user interactions JavaScript
- For styling and presentation HTML or XHTML and Cascading Style Sheets (CSS)
- For returned structured document handling Document Object Model (DOM)
- For data manipulation and conversion XML and XSLT
- For asynchronous request sending and retrieval XMLHttpRequest as a messaging protocol

These core AJAX technologies are mature,

well-known and used in web application widely. AJAX is a new model or an approach to web applications that brings some benefits to web users. It eliminates the stop-start nature of web interactions, which will happen asynchronously. The data can be manipulated without having to render the entire page again and again in the web browser. This prevents unchanged information to be send repeatedly back and forth across the network. It uses text and XML for data exchange over XHR. It leverages the server from processing demands and thus reduces the bandwidth because JavaScript is running in the client

The increase of JavaScript code in AJAX application, posed for development of Ajax framework to simplify task. After Google started to develop some new applications with the AJAX, AJAX has drawn some attention in the public after Google introduced application such as Google Groups, Google Suggests, and Google Maps. Besides the Google products Amazon also have used AJAX approach in its search engine application. These projects demonstrate that AJAX is not only sound technically, but also practical for real world applications.

You can use AJAX in your web applications just by writing your own custom JavaScript codes that directly use the XMLHttpRequest protocol's API. The implementation of XHR varies between browsers. However, there are a lot of frameworks that has been developed to provide higher level AJAX services and hide the differences of different browser. Among these are DWR, Prototype, Sajax, AJAX.NET, OpenRico, and ScriptAculous. The list of frameworks and its classification can be found in this site [4].

## 2.2 Authentication

The HTTP specifications provide two mechanisms for authentication: Basic authentication and Digest authentication [5]. Basic authentication requires the client to send a username and password in the clear as part of the HTTP request. This pair is typically resent preemptively in all HTTP requests for content in subdirectories of the original request. Basic authentication is vulnerable to an eavesdropping adversary. It also does not provide guaranteed expiration (or logout), and repeatedly exposes a user's long-term authenticator. Digest authentication, a newer form of HTTP authentication, is based on the same concept but does not transmit cleartext passwords. In Digest authentication, the client sends a cryptographic hash of the username, password, a server-provided nonce, the HTTP method, and the URL. The security of this protocol is extensively discussed in RFC 2617 [5]. Digest authentication enjoys very little client support, even though it is supported by the popular ApacheWeb server.

With an interface limitation of HTTP authentication, a form-based authentication is introduced. It provides the web application designer the most control over the user interface. However, it requires the application to do a fair amount of work to implement authentication.

The main risk of these schemes is that a successful attack reveals the user's password, thus giving the adversary unlimited access. Further, breaks are facilitated by the existence

of freely available tools capable of sniffing for authentication exchanges.

The Secure Sockets Layer (SSL) protocol is a stronger authentication system that provides confidentiality, integrity, and optionally authentication at the transport level. It is standardized as the Transport Layer Security protocol [6]. HTTP runs on top of SSL, which provides all the cryptographic strength. Integration at the server allows the server to retrieve the authentication parameters negotiated by SSL. SSL achieves authentication via public-key cryptography in X.509 certificates [7] and requires a public-key infrastructure (PKI). This requirement makes SSL difficult for authentication because there is no global PKI. Several major certificate authorities exist (e.g., Verisign), but the space is fractured and disjoint. To some degree, users avoid client certificates because certificates are practically incomprehensible to non-technical users. Other arguments suggest that the merits of PKI as the answer to many network security problems have been somewhat exaggerated [8]. Client support for SSL is non-standard and thus can have interoperability problems (e.g., Microsoft Internet Explorer and Netscape Navigator client certificates do not interoperate), and performance concerns. SSL decreases Web server performance and often provides more functionality than most applications need. In an effort to avoid using SSL, Bergadano, Crispo, and Eccettuato use Java applets to secure HTTP transactions [9].

Park and Sandu identify security problems of regular cookies, network threats, end-system threats, and cookie harvesting threats [10]. Samar describes a cookie-based distributed architecture for single-signon [11].

There are a lot of authentication protocols have been developed, including AuthA [12], EKE [13], provably secure password authenticated key exchange [14], and the Secure Remote Password protocol [15]. These protocols are not well-suited for the Web because they are designed for session initialization of long running connections, as opposed to the many short-lived connections made by Web browsers. Long-running connections can easily afford a protocol involving the exchange of multiple messages, whereas short-lived ones cannot absorb the overhead of several extra round-trips per connection. Additionally, these protocols often require significant computation, making them undesirable for loaded Web servers.

One-time passwords can prevent replay attacks. Lamport's user password authentication scheme defends against an adversary who can eavesdrop on the network and obtain copies of server state (i.e. the hashed password file) [16]. This scheme is based on a one-way function. Haller later implemented the S/Key one-time password system [17] using techniques from Lamport. De Waleffe and Quisquater extended Lamport's scheme with zero-knowledge techniques to provide more general access control mechanisms. With their one exchange protocol, a user can authenticate and prove possession of a ticket. This scheme requires the client to perform computation such as modular exponentiation. However, with Ajax, JavaScript can do the computation in the client.

Kerberos uses tickets to authenticate users to services [18]. The Kerberos ticket is encrypted

with a key known only to the service and the Kerberos infrastructure itself. A temporary session key is protected by encryption. The ticket approach differs greatly from schemes such as ours because tickets are message preserving, meaning that an adversary who compromises a service key can recover the session key. If an adversary compromises the key in our scheme, it can mint and verify tokens, but it cannot recover the contents that were originally authenticated. Authentication and encryption should be separated, but Kerberos does both in one step.

The Amoeba distributed operating system cryptographically authenticated capabilities (or rights) given to a user. One of the proposed schemes authenticated capabilities by XORing them with a secret server key and hashing the result. Client authentication on the Web falls into the same design space. A Web server wishes to send a user a signed capability.

Microsoft Passport offers a managed cookie authentication scheme [19]. Microsoft mints a cookie authenticator after a user logs in. Vendors participating in the passport service can verify the authenticator to determine authenticity and authorization. The details of the authentication scheme have not been published, but the white paper indicates that Microsoft shares a unique symmetric key with each vendor. These keys can both mint and verify authenticators.

With the host and web authentication scheme presented above, the strong and weak points can be compensated with each other depending on the implementation. Ajax on the other hand, can perform computation on the client with JavaScript. Ajax applications are long-lived in the browser that a traditional web application; combining host based authentication into with web authentication scheme provides secure authentication.

## 2.3 Session Management

Session management allows the web-based system to create a session so that the user will not have to re-authenticate every time they wish to perform an action. Session management ensures that the client who is connected to a server is the same person who logged in originally. Sessions are thus a target for malicious users because they can be used to gain access to the system without having to authenticate. Sessions allow a web-based system to avoid repeatedly performing authentication. Web applications perform an authentication process to determine if a user should be granted access to the web based resource. The user is granted a session ID when they have successfully authenticated (i.e. presented a valid login name and password). This session ID can be presented wherever authentication is necessary to avoid repeating the login/password process.

The session ID itself is simply a string of characters or numbers and functions much like a social security number. The server remembers that the session ID (SID) was given to the user and allows access when it is presented. Session IDs may be valuable to hackers, but they oftentimes are taken for granted by developers. Good session management is the barrier between the public and the data stored in the system. It is as important as the data the system maintains. Protecting the

session ID is critical to the security of an online system.

Sessions allow web applications to maintain state. After the user logs in, the server and client will operate together to maintain an authenticated state until the user logs out. Hyper-Text Transfer Protocol (HTTP) is a stateless protocol [20], and thus, special techniques are required to create stateful web applications. Three methods are used to perform this function: cookies, GET form data, and POST form data. Developers use one or more of these methods to propagate the SID from one page to the next so that a user's session can be maintained.

### 2.3.1 GET Method

The GET method encodes data as part of the URL. Consider the following link (the field names are highlighted):

*http://dummysite.com/somepage.php?VarOne=one&VarTwo=two*

This URL opens the page "somepage.php" and sets two variables, VarOne and VarTwo. This method allows form data and arguments to be passed to web pages. Oftentimes, this method is used to maintain the session identifier. Consider the following URL:

*http://dummysite.com/Admin.php?SessionID=12345678*

This URL is encoded with the session ID (underlined above). The "Admin.php" will receive the session variable and check to determine if the session ID is valid. If the session is valid, then the system can assume the user has authenticated successfully and will allow appropriate access to the page. Oftentimes, the URLs are crafted by the browser based on forms present in the web page.

### 2.3.2 POST Method

Another method for passing data to pages is through the use of POST data. Form elements have a method attribute that allows the developer to select the way in which the form data is presented to the server. POST data is retained in input, select and textarea form tags in HTML pages. This method is not reflected in the URL so users do not see the variables and values in the address bar of their browser. POST can be used to maintain the session ID like GET does. This is accomplished with a form tag on each secured page that contains the SID. The type attribute of this tag is set to hidden which prevents the form element from being displayed yet provides a placeholder for the SID.

### 2.3.3 Cookies

The definition and purpose of a cookie is:

*An HTTP cookie (usually called simply a cookie) is a packet of information sent by a server to a World Wide Web browser and then sent back by the browser each time it accesses that server. . . . Cookies can contain any arbitrary information the server chooses and are used to maintain state between otherwise stateless HTTP transactions.[22]*

Cookies were originally designed to circumvent the stateless nature of HTTP. Cookies can exist on disk indefinitely depending on the parameters used to configure the cookie during its initial creation. Session cookies are a spe-

cific type of cookie that do not have an expiration date and cease to exist when the browser is closed. Session cookies are not written to disk and only exist in memory. However, persistent cookies have an expiration date and are written to disk and stored until the expiration date arrives. Cookies were designed with privacy in mind. Browsers enforce rules that limit cookie access to sites within the domain specified when a cookie was set. Consider the following cookie:

*Set-Cookie:Last=12;Domain=.hannam.ac.kr*
*Expires=Fri,18-Oct-2006 21:27:20 GMT;*

The above HTTP response would set a persistent cookie (note the existence of an expiration date) for the domain hannam.ac.kr (and sub-domains of). Only the hannam.ac.kr and sub-domains wouldhave permission to access and modify this cookie. A website outside of this domain such as othersite.net would be denied access by the web browser.

### 2.3.4 Comparing GET, POST and cookies

All three methods of session ID propagation (GET, POST and cookies)can be considered to be roughly equivalent methods of maintaining state in regards to security. None are inherently secure and they all must be combined with other security countermeasures. Nevertheless, many developers will claim that one method is secure and another is insecure. However, all three methods can be made reasonably secure through intelligent design. Conversely, negligence and incompetence can render any of the methods insecure. The attack vectors against all three methods are similar although some methods may have unique characteristics that allow attacks not possible with other methods. Moreover, attacks on session management focus on retrieving a valid session ID. The list of attacks and countermeasures on session management were described by Luke Murpey on his article at SANS Institute [21].

## 3. AJAX Security Model and Vulnerabilities

In Ajax application, the browser performs logic apart from the server. Ajax application completely relies in the browser security model. In IE, it is based on the concept of zones while Mozilla is based on privileges. Activities in the client that violates the model such talking to the third party servers will be prompt with a message in the case of Internet Explorer and disable by default in Mozilla. However, Mozilla security policy can be programmatically enabled using *netscape.security.privilegeManager* object. Thus, different browser provides different client side security to Ajax application.

Javascript on the other hand, perform the logic on the client side in Ajax application. The current JavaScript security solution is based on executing JavaScript code within a sand-box [23]. The JavaScript sandbox is similar to the Java sand-box, but it is more restrictive since JavaScript does not provide any built-in support for file access. This means that by default, a JavaScript program cannot read files on the local drive. Other examples of operations that are not allowed are opening a window smaller than 100x100 pixels, using the History object to find out recently visited pages, and unconditionally close a browser win-

dow. In addition to restricting many operations, browsers use two main JavaScript security policies. The first policy, called the *same-origin policy*, is used to isolate one document from another, while the second one, called the *signed-script* policy, provides means to enforce finer-grained access control.

The same-origin policy prevents documents or scripts loaded from one origin (i.e., a web server), from getting or setting properties of a document from a different origin. In this context, "same origin" means same protocol, host, and port. This policy provides the foundation for isolating one script from another, and ensures that a document downloaded from one source cannot be changed by JavaScript code downloaded from another origin. There is one exception to this rule: a document can set its domain to a suffix of its current domain. This means that a document from http://syssw.hannam.ac.kr can access a document from http://hannam.ac.kr after setting its domain to factory.com. This policy applies to both windows and frames. However, it does not apply to different protocols such as https://syssw.hannam.ac.kr and http://hannam.ac.kr

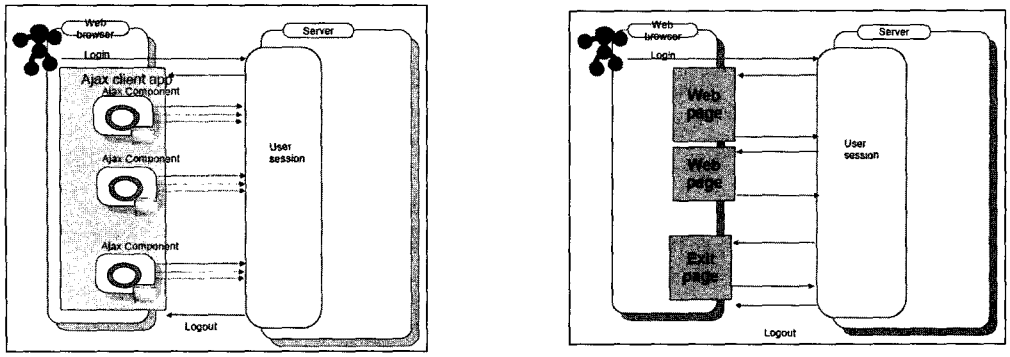
Signed-script policy give scripts more functionality and give a user the option to define a finer-grained security policy. Script signing allows a script to get out of the sand-box and is similar to the mechanisms used for signed Java applets. When the browser downloads a script that is digitally signed, the browser first verifies the signature and then extracts the principals of the script. The principals can either be derived from validating the signature of a script or they can be derived from the origin of the script. If a principal is derived from the origin of the script, the principal is called a *codebase principal.*

Signed scripts are allowed to ask for extended privileges/capabilities at runtime, using the command *netscape.security.Privilegemanager. enableprivilege*(). The privilege represents permissions to access specific targets and a prompt will appear in the browser window whenever a script asks for a privilege. The privileges range from automatically bypassing the same-origin checks to reading random files on the local drive.

Ajax application does not have new vulnerabilities [27, 28]. It has same security issues with traditional web application. But Ajax increases the attack surface area. The new security flaws openedby Ajax are potential for information leakage and non-repudiation and cross-site scripting. However, OWASP Ajax security project posted valuable information on Ajax vulnerabilities and countermeasures [29].

## 4. Design of AWASec Framework

AWASec is focus on authentication and session management to provide security over Ajax-enabled web application, as shown in Figure 2. Authentication is first security component to be considered in client/server environment. It provides assurance in both parties that they are as they claim to be. Most web application used username/password authentication and certificate based. The former is widely used among small and medium web driven business enterprises because it requires least amount of infrastructure and it is easy to implement. However, as shown Figure 1, au-



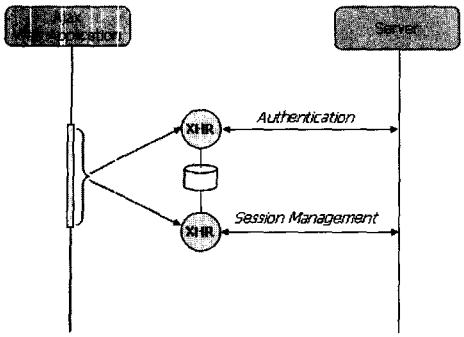

〈Figure 1〉 Traditional web application (a) and Ajax application (b)

thentication in traditional web and Ajax application is having same pattern. Re-authentication is not asserted within the active session. Moreover, Ajax component may access sensitive information that needs authentication.

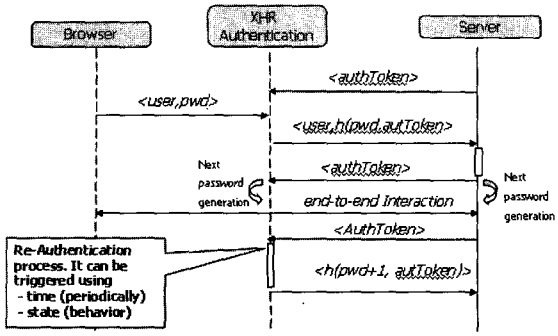
Username/password authentication (Basic, Digest, Form-based) is vulnerable to eavesdropping and other attacks. To mitigate this compromise, pseudo random password similar to OTP scheme must be in-placed. The password is used as the passphrase in both ends. Re-authentication is asserted based on time or behavior of the application. It is invoked using JavaScript and XHR in the background. The invocation provides no hassle to the client and the flow of the application will is not diverted during the authentication process.

State management usually coined to session management. In our context, we extend the concept because client has logic. HTTP by itself is a stateless protocol [20], where pages are independent with each other if it rendered into the browser. However, to facilitate the stateful nature of application over the web, a session is provided by the server to store information needed by the subsequent pages. Session management comes after authentication was successfully completed. The security of subsequent interaction in web application is dependent to the security of the session parameter. AWASec framework provides a secure mechanism in session management. It asserts authentication driven by events and time even the session information is still valid. This authentication assertion is one of the best practices to avoid compromise in session management.

The handling of session information is done on the background using AJAX. It facilitates exchange of session information not dependent to the host page requesting session information but with the time set in the server for session's duration. When session expires, the assertion of authentication is invoked seamlessly without diverting the flow of the application. The state of the client will also be preserved with new session information. The authentication and state management will be described in details in the following sections.




〈Figure 2〉 AWASec Framework




〈Figure 3〉 AWASec Authentication System

## 4.3 Authentication Method

Username/password authentication is employed in our framework as shown in Figure 3. The authentication interfaceis displayed to take the username and password. Initially, *Token* from the server is given along with the page and stored it as a temporary cookie. The client will hash the password along with the authToken. The hashed value and username will be sent to the server for matching process. The server will lookup the username and takes the password; then hashed the password with the AuthToken that was initially sent to the client. If matched, the server generates new authToken. This token and the password is use to generated the subsequent password which is similar to OTP password generation scheme [17] using MD5 [26] hashing algorithm. To accomplish two-way authentications, both server and client can compare the password generated using common token to authenticate each other. After the authentication is successful, the session connection will be established.

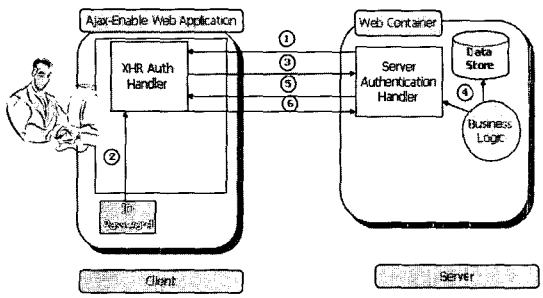
Listing 1 shows the JavaScript Object for AWASec Authentication. The login, requestAuth and logout and authCallbackHandler objects are describe as follows:

***Login*** – functionsfor logging in the username and the hash value of the password and authToken to the server,

***requestAuth*** – This function is used to request authentication to the server.

***authCallbackHandler*** – a callback function that is triggeredin response to the authentication response from the server.

***Logout*** – this method is used to logout the application. It will inform the server to revoke all the authenticators assigned to the host.




〈Figure 4〉 Authentication Sequence

Authentication steps in Figure 4:

1. Server sends AuthToken along with the

page.

2. Username and password is taken by client handler to perform hashing of password and AuthToken
3. The computed hashed value and username is sent to the server.
4. The server searchesthe database for valid username and password. It computes the hashed value using the authToken sent to the client previously and the password. If equal, next authToken will be generated randomly. The server will also assert other access control pertaining to user. Otherwise, a failure response is sent to the client.
5. The newly generated authToken is sentto the client and be the token use to generate the succeeding password for re-authentication. Both ends will generate the next password for the next authentication process. The server will generate OTP once while the client is on need basis.
6. The client will perform MD5 hashing with password generated and authToken from the server and send it to server for comparison. If matched, the server established a session connection for a successful authentication.

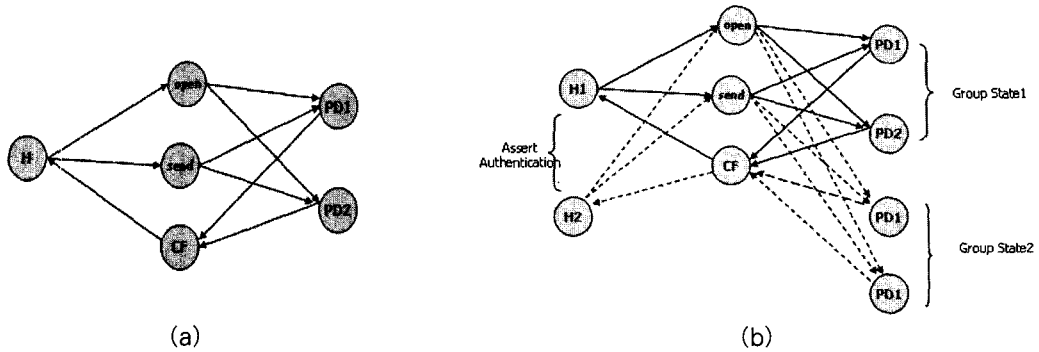
The re-authentication process will start at step 2, where the client request token from the server and compute the hashed value of next password plus the token and send the result to the server for authentication. Moreover, for subsequent interaction, the authToken is stored in ephemeral or temporary cookie in the browser.

```
var Auth = {
          AuthToken:0,
          login: function(uname,hpass) {},
          assertAuth: function(type)
                        {
                         .....
                         RequestAuth(HashToken(this.AuthToken));
                         ......
                         } ,
          requestAuth:function(token)
                        {
                         ....
                         loadXHRequest("serverAuthHandler.php", token, POST);
                        },


          authCallbackHandler: function()
                        {
                         .....
                        this.AuthToken = AuthToken;
                        },

          logout: function()
                   {
                    ....
                   This.AuthToken = 0;
                   }
```

〈Listing 1〉 AWASec Authentication JS Object

## 4.4 Session Management

Ajax-enabledweb application has a host page, XHR components and server documents as shown in Figure 5 (a). XHR objects have a host page (H) and interact with the participating documents (PDs) such as XML, text, PHP, ASP and JSP scripts in the server. The host page invokes document request to the server with XHR open and send function along with some application headers. The PD will send back the requested documents or response to the host page; and it will be managed by the callback function (CF). In real web-based systems, application has a lot of host pages and participating documents. Figure 5 (b) shows the interaction of XHR components and server documents with multiple host pages. The broken line represents the interactions of the other host page to the server. The illustration of flow between host page, XHR, and other server documents implies that XHR is a potential tier in maintaining the state of a web application. This concept is the basis of AWASec framework in session management design.

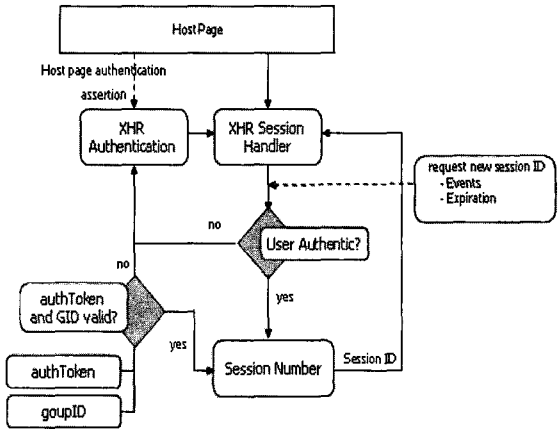



〈Figure 5〉 Host - Server document mapping with XHR

The AWASec session management group the sessions by host page and server documents mapping. Each group will have its group ID that is randomly generated every time the host page is loaded into the browser. A session number will also be generated randomly and hashed for session identifier (session ID). But before generating valid session ID, the server should always assert the validity of the authToken and group ID. Each session in the group has its time duration, which will be revoked if it is elapsed. The revocation and renewal of sessions will be triggered in the client based on events and time. A time-driven revocation can be done by calculating the approximated expiration time of the session in the server; and prompt the user of his appropriate actions. The event-driven revocation happens if the security policy is violated both in the client and server. However, re-authentication must be asserted for a revoked and expired session in the group. Authentication is also invoked if the flow of the application moves to the other host page.

Subsequent access within the host page that requires new session ID, a new session number is generated but the authToken and groupID is not changed. In the case of re-authentication process, which will fired periodically within the host page a new authToken is generated and so with the session number but the groupID is the same. However, authToken, groupID and session number will be changed if the flow moves from one host page to another.

Figure 6 shows the AWASec session management scheme. Host page will invoke authentication every time the page is loaded. XHR authentication notifies the session handler for successful authentication. The session will always be revoked every time the XHR authentication notifies successful authentication for the handler to invoke session request. Each session request should always start with validation of the user authenticity before generating new session identifier for the subsequent interaction. AWASec session management is design to work XHR authentication and XHR session handler coupled together to avoid broken authentication and session management.




〈Figure 6〉 AWASec Session Management Diagram

## 4.5 Event Logging System

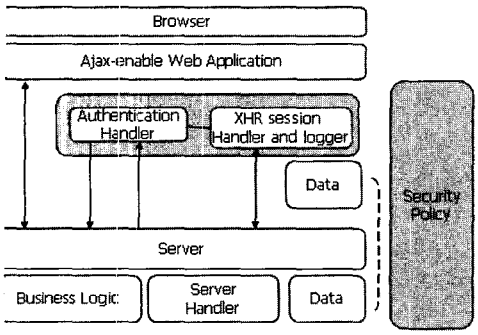
The framework provides a mechanism to keep track the background activity of XHR. This feature provides the user information of background activities. Object ID is automatically generated at aninstance of an XHR object. The XHR object is overridden with a new object that has additional methods and attributes as shown in Listing 2. The XHR object is extended by saving the old open and send method; add new attribute and functions, then invoked the old open and send functions to request the server. To keep track the response of modified XHR object, an event handler is place to trigger an event logging function. This feature can be incorporated to your application by defining a DIV area to contain the XHR messages in the background. We have tested the concept using mozilla firefox browser and other browser that supports XMLHTTPRequest object.

```
XMLHttpRequest.prototype.uniqueID = function( ) {
    if (!this.uniqueIDMemo) {
        this.uniqueIDMemo = Math.floor(Math.random( ) * 1000);
    }
    return this.uniqueIDMemo;
}

XMLHttpRequest.prototype.SetSecure = function(value) {
  if(value != 0)
    this.secureFlag = 1;
  else
    this.secureFlag = 0;
 }


XMLHttpRequest.prototype.oldOpen = XMLHttpRequest.prototype.open;

    var newOpen = function(method, url, async, user, password) {
    display("[" + this.uniqueID( ) + "] intercepted open (" +
            method + " , " +
            url + " , " +
            async + " , " +
            user + " , " +
            password + ")");
    this.oldOpen(method, url, async, user, password);
}

XMLHttpRequest.prototype.open = newOpen;
```

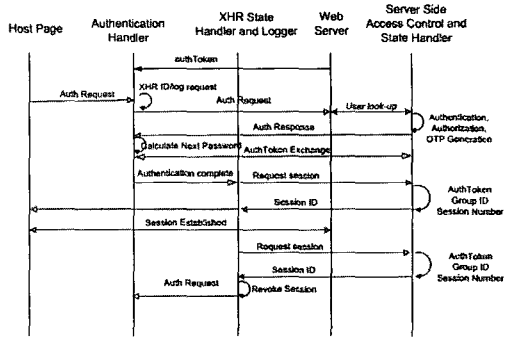
〈Listing 2〉 XHR Logging System

# 5. Framework Components

In this section, we describe the client side of the framework. The overall architecture of web application with our framework is shown in Figure 7. The framework mediates the web application and the server. It will serve as security portal of the client for XHR request. There are only two major components of the framework; the Authentication Handler and State Handler. Each component has client and server side implementation to facilitate persistent interaction. These two handlers are coupled with each other to provide consistency of the application for tight security. The framework maintains data storage use to match the information stored in the server for interaction. A security policy is also formulated especially in client side to ensure that the script and the flow of execution will be in restricted environment. It has also common data storage for authentication and session management information.




〈Figure 7〉 Architecture of Web Application with AWASec Framework




〈Figure 8〉 AWASec end-to-end interaction of Authentication and Session Management

## 5.1 Authentication Handler

The Authentication Handler handles the authentication process in the client. Initially, the handler will be fired up using onload event of the body ***<body onload = "Auth.assertAuth ()">*** in the host page. The ***Auth.Login*** will be triggered from the authentication assertion in the case first login and revoked token. The interface for login will just float over the host page for seamless experience. The subsequent authentication can be invoked periodically or behavior-based. Periodic authentication can be fired up with ***setTimeout(Auth .assertAuth(),period)*** while behavior-based can be triggered when an events occur.The ***addlistener ("events", "Auth.assertAuth")*** object will be used to invoked the behavior-based or event-based authentication. The logout can also be invoked using unload events in the body tag ***<body onUnload="Auth.Logout()">***

The handler sends a token to session management handler. This token will be used to setup the session with ***Session.requestSession()***.

## 5.2 Session Handler and Logger

The Session Handler manages the state of the application. It is periodically changed to avoid replay attacks. The session ID has durationif the time elapsed, the session will be revoked and authentication process will takes place. The handler will approximately calculate the elapsed time of the session in the client and inform the user about the expiration of the session. However, If the user want to renew the session, the ***Session.requestSession()*** will be triggered for new session ID otherwise ***Session.revokeSession()*** will be triggered and the authentication process will be invoked if the user wanted to continue the access on the revoked session.

# 6. End-to-end Interaction with AWASec Framework

Figure 8 illustrates the end-to-end interaction of web application with AWASec authentication and session management. Both ends trust is establish using two authentications as shown

〈Table 1〉 Framework functions and Attributes for security

| Functions and Attribute | Descriptions |
|---|---|
| **Authentication** | |
| *AuthToken* | 128 bit hash value |
| | |
| *Login(username,pwd)* | Request Login to access services in the application |
| *AssertAuth(type)* | Assert setup of authentication. |
| *RequestAuth(token)* | Request Authentication from the server. |
| *Logout()* | Logout |
| **Session Management** | |
| *Session ID* | Asession ID generated from the server using the AuthToken, groupID and a random generated number. |
| *requestSession()* | Request Session ID from the server |
| *revokeSession()* | Revoke the current session |
| **Other** | |
| *HashToken(token)* | This function returns 32 character hash value of *token*. Initially, token is equal the concatenated password and AuthToken from the server. For subsequent authentication, the previous AuthToken will be use to generate the next token. |

in the figure. The server side access control and state handler play an important role to an overall security posture of our framework. It asserts access control and security policy of request base on the business logic in the server. The generation of authToken and Session ID are also done in the server in accordance to the best practices described in web application security project [2].

Table 1 shows the attributes and methods of our framework for security purposes. They are classified according to our security component.

## 7. Analysis of Our Framework

In this section we present informal analysis of our scheme. The overhead can be noted in the initial authentication setup especially in generating the password for authentication. The time required for re-authentication contributesto the overall performance of the framework because the coupling process asserts session revocation, authentication and session initiation.

The dos and don'ts of web authentication describe in [30] is considered. MD5 is used for hashing that is widely known hashing function. The password never travels into the wire. In this way replay attack can be mitigated. Forging a token is intractable because MD5 is irreversible function. In the case of captured token, it is useless for eavesdropper because token has expiration in the

server.

Deployability and user acceptability can be noted with the support of most browsers in Ajax.The re-authentication in the background gives no hassle for the user. The framework can be integrated seamlessly to existing application. However, the prompting of session renewal may cumbersome to some client.

## 8. Conclusion

Re-authentication process is crucial to a long-lived application. It establishes the authenticity of the communicating parties over time. The authentication and session management scheme described in this paper shows that host-based authentication can be applied to web application using Ajax. Periodic and event based assertion of authentication can be achieved. The repetitive invocation of authentication on the background that is tightly coupled to session management is the main contribution of this framework. Ajax provides a mechanism to avoid broken authentication and session management, as described in this paper.

Our scheme is trying to overcome the practical limitations of web authentication such as deployability and user acceptability. However, the performance of this scheme applied to different web application environment will be our future work. This work is also gearing towards state synchronization of Ajax application in distributed environment.

## References

[1] Jesse James Garret, Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/publications/essays/archives/000385.php

[2] Open Web Application Security Project, http://www.owasp.org/

[3] Web 2.0, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[4] Max Kiesler, Round-up of 50 AJAX Toolkits and Frameworks, http://www.maxkiesler.com/index.php/weblog/comments/round_up_of_50_ajax_toolkits_and_frameworks/

[5] John Franks, Phillip Hallam-Baker, Jeffrey Hostetler,Scott Lawrence, Paul Leach, Ari Luotonen, and Lawrence Stewart. HTTP authentication: Basic and digest access authentication. RFC 2617, NetworkWorking Group, June 1999.

[6] Tim Dierks and Christopher Allen. The TLS protocol version 1.0. RFC 2246, Network Working Group, January 1999.

[7] CCITT. Recommendation X.509: The directory authentication framework, 1998.

[8] Carl Ellison and Bruce Schneier. Ten risks of PKI: What you're not being told about public key infrastructure. Computer Security Journal, 16(1):1–7, 2000.

[9] F. Bergadano, B. Crispo, and M. Ecce-ttuato. Secure WWW transactions using standard HTTP and Java applets. In Proceedings of the 3rd USENIX Workshop on Electronic Commerce, pages 109–119, Boston, MA, September 1998.

[10] Joon S. Park and Ravi Sandhu. Secure cookies on the Web. IEEE Internet Computing, 4(4):36–44, July/August 2000.

[11] Vipin Samar.Single sign-on using cookies for Web applications.In Proceedings of

the 8th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pages 158–163, Palo Alto, CA, 1999.

[12] Mihir Bellare and Phillip Rogaway. The AuthA protocol for password-based authenticated key exchange. Technical Report, IEEE P1363, March 2000. http://grouper.ieee.org/groups/1363/StudyGroup/Passwd.html#autha.

[13] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 72–84, Oakland,CA, May 1992.

[14] Victor Boyko, Philip MacKenzie, and Sarvar Patel. Provably secure password authenticated key exchange using Diffie-Hellman. In B. Preneel, editor, Proceedings of Advances in Cryptology—EUROCRYPT 2000, volume 1807 of Lecture Notes in Computer Science, Bruges, Belgium, May 2000. Springer-Verlag.

[15] Thomas Wu. The secure remote password protocol. In Proceedings of the1998 Internet Society Network and Distributed System Security Symposium, pages 97–111, San Diego, CA, March 1998.

[16] Leslie Lamport. Password authentication with insecure communication. Communications of the ACM, 24(11):770–771, November 1981.

[17] Neil Haller. The S/KEY one-time password system. RFC 1760, Network Working Group, February 1995.

[18] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. IEEE Communications Magazine, 32(9):33–38, September 1994.

[19] Microsoft passport. http://www.passport.com/

[20] "RFC2616, Hypertext Transfer Protocol – HTTP 1.1." Internet Engineering Task Force (IETF). June 1999: p. 1. Internet. Available: http://www.ietf.org/rfc/rfc2616.txt

[21] Luke Murpey, Secure Session Management, SANS Institute, January 2005

[22] "HTTP Cookies." Wikipedia. December 30, 2004: http://en.wikipedia.org/wiki/HTTP_cookie

[23] V. Anupam and A. J. Mayer. Secure Web Scripting. IEEE Internet Computing, 2(6):46–55, 1998. citeseer.ist.psu.edu/anupam98secure.htm

[24] David Crane, et. al., Ajax In Action, Manning Publications, 2006

[25] Paul Johnson, JavaScript MD5 libraries: http://pajhome.org.uk/crypt/md5src.html

[26] R. Rivest, RFC 1321, MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992

[27] Stewart Twynham, "AJAX Security", Feb. 16th, 2006. http://www.it-observer.com/articles/1062/ajax_security

[28] Jaswinder S. Hayre,Ajax Security Basics June 19, 2006. http://www.securityfocus.com/infocus/1868/

[29] OWASP AJAX Security Project, http://www.owasp.org/index.php/Category:OWASP_AJAX_Security_Project

[30] K. Fu, E. Sit, and N. Faemster. Dos and Don'ts of Client Authentication on the Web. In Proceedings of the USENIX Security Symposium, pages 251--269, August 2001.

## ◐ 저 자 소 개 ◑


### 남 상 온 (Sang-On Nam)

1983년 계명대학교 전자계산학과 졸업(학사)
1995년 한남대학교 대학원 컴퓨터공학과 졸업(석사)
1997년 한남대학교 대학원 컴퓨터공학과(박사)
1996~현재 대원과학대학 컴퓨터정보계열 교수
관심분야 : 분산 시스템 및 실시간 시스템, XML, 웹 보안, 데이터베이스
E-mail : nso@daewon.ac.kr

### Rolyn C Daguil

1998 BS Computer Science
Mindanao State University
Marawi City, Philippines
2003 MS Information Technology
Hannam University- Extension Program
Daejeon, Korea
Research Interest : Web Security, XML, Ubiquitous Web
Email: rcd@hannam.ac.kr

### 김 기 원 (Gi-Weon Kim)

1987년 한남대학교 전자계산학과 졸업(학사)
1989년 숭실대학교 대학원 컴퓨터공학과 졸업(석사)
2001년 한남대학교 대학원 컴퓨터공학과(박사)
1996~현재 초당대학교 컴퓨터과학과 교수
관심분야 : 멀티미디어, 실시간 영상처리, 음성인식.
E-mail : kwkim@chodang.ac.kr

### 송 정 길 (Jung-Gil Song)

1966년 한남대학교 수학과 졸업(학사)
1982년 홍익대학교 대학원 전자계산학과 졸업(석사)
1988년 중앙대학교 대학원 전자계산학과(박사)
1900~1991 University of illinois 객원교수
1979~현재 한남대학교 컴퓨터공학과 교수
관심분야 : XML, UML, 분산 시스템 및 실시간 시스템
E-mail : jksong@hananm.ac.kr