Reparing HTTP authentication for Web security

Yutaka OIWA

1 Overview

This position paper proposes improvement efforts for HTTP authentication/authorization mechanisms, to solve various current problems on the Web systems.

Current HTTP specification provides two password-based authentication schemes: Basic and Digest. The former uses simple plain-text password on the wire for authentication, and the latter introduces a simple hash operation to avoid plain-text passwords sent on the wire. Unfortunately, both of these two has insufficient security strength to protect user identity in the recent computing environments. If the authentication communication is either performed with or eavesdropped by malicious people, they will easily guess the user's passwords, or will have access to the protected resources. There are also several alternatives for user authentication in the protocol layer, but most of these are not used because of problems such as security, deployment or usability.

In addition, although HTTP provides such an authentication framework, current Web applications tend to use form-based, application-layer, custom-implemented authentication functionalities instead. For the security purpose this situation is very unfortunate, because it effectively limits the security level equivalent to the plain-text authentications.

We are currently proposing a new secure authentication mechanism, called "HTTP Mutual access authentication protocol", to IETF. Our proposal uses a cryptographic primitive to ensure authentication security in several aspects, and provides *mutual authentication* between Web servers and clients to solve several current Web problems. We are also proposing a generic framework extension to the HTTP authentication, which will solve (or, at least improve) the current situation of the unused HTTP authentication. We believe that this proposal will provide a solution to the current problem, at least on the protocol side.

In the last IETF in Prague, we had a side meeting on HTTP authentication for possible standardization efforts in near future. This time, we want to have more discussions on the things which will be needed in near future for improving the security situation of user authentication on the Web.

2 The problem

User authentication is, needless to say, one of the most important building block for the Web application systems. At the same time, it is currently one of the weakest block in terms of security. The current problem on the Web authentication can be summarized in twofold: (1) we do not have a good technology for authentication which provides both security and good usability; (2) even if we had such a thing, we don't have enough technology to make it useful on Web applications.

For the first part we consider four technologies which are currently useful: HTTP Basic, HTTP Digest, authentication using HTML forms, and the X.509 client certificates with TLS. Among that, Basic and Forms are very similar: it simply uses a plain-text passwords. If any communication traffic falls into the malicious people's hands, the password will simply stolen. Form authentication are, in the real world, prone to implementation mistakes which causes severe security breaches. Digest is slightly better than those, but it does not have a enough strength to the dictionary-based off-line attacks. TLS client certificate is much stronger than that, but it has a problem mainly on usability, e.g. client-side key management and key distributions.

For the latter part, Web developers are always in favor of Form authentication, rather than HTTP authentication mechanisms. Even if we excludes claims on visual design issues at this moment, the current HTTP authentication has several disadvantages or lack of features for Web applications, for example:

- it is hard to manage authentication in the time dimension. In more detail,
  - logging-out cannot be implemented without a non-portable Javascript tricks.
  - it is hard to maintain time limits on operations.
- we cannot provide pages for both authenticated and guest users on the same URL, which is common in most web systems.

TLS authentications are even much worse in this aspect, because we cannot manage authentication depending even on the web-page space. Without solving such problems, our efforts on improving HTTP authentication have risks of becoming in vain.

In addition, we must also be careful about the *server's identity* at the same time. In most protocols other than Web (e.g. POP3 or IMAP), simple server authentication with predefined server identifier and security credential (e.g. DNS host-name and a corresponding server certificates) will be sufficient to ensure the whole communication security. The server authentication embedded on the TLS protocol will work quite good in this situation. However, Web system is not as simple as those: we cannot predefine a single host to be visited. In this case *Phishing* become a real risk to the server authentication. Unfortunately, if the clients are tricked into a wrong web sites by malicious URL links, TLS server authentication *will not* detect such forgery, and all of above-mentioned client authentication mechanisms will allow some extent of forged authentications (MITM, forged successeful authentication, password stealing/guessing, depending on the mechanism.) From the novice users' and service providers' perspectives, the security impact of server impersonations are as equally severe as the user impersonations. We need to have a mechanism which mitigates such a *forged server identification* problem.

3 Our protocol proposal

As mentioned before, we are currently proposing an improvement on the HTTP authentication mechanism [1]. For the details of the cryptographic protocol, please refer the Internet Draft submitted to the IETF. Instead, this section will provide a brief overview on the proposal, especially on the design assumptions, design decisions and the provided functionalities.

3.1 Design goals and assumptions

While designing the protocol, we have settled several goals and criteria for it.

Password-based authentication without local storage The password authentication is the most easily-understood way of authentications for novice users. They have difficulty on managing local secrets (such as private keys) stored in the local hard-disk in the correct way. In addition, the existence of local secret also makes difficult for users to use more than one computer simultaneously. The protocol avoid requirements for any long-term local secret storages and/or local whitelists.

Secure mutual authentication The protocol is designed in mind that Phishing cannot be prevented before the authentication, and it must provide a protection even with such attacks. The protocol prevents several kinds of possible attacks which can be done with Phishing: the password are kept completely secret to the server, the received credentials cannot be forwarded to the genuine site to make successful authentication in a "MITM" way.

Furthermore, it provides a *mutual authentication*; *successful result of authentication cannot be imitated* in any way. This is one of the important property which the protocol provides. It complements the TLS server authentication and fills a pit-fall of that. This kind of attack is one of the common story of Phishing attacks. To this purpose, the client browser must have a solid control on the whole authentication process, unaffected by the Web-page contents.

Generic (equal opportunity) The protocol is available to use for *any* web services who want to provide user authentication, including personal websites. Current EV-SSL scheme does not provide this criteria: EV-SSL certificates can only be acquired by parties having long-enough transaction history.

Per-URL authentication configuration In real web systems, authentication is defined for sets of web resources specified by URIs. For example, most Web applications have both public contents (e.g. blog contents) and private, authenticated contents (e.g. blog editing interfaces). In addition to the above, there are web applications which have multiple set of authentication "realms" (groups of resources which share same setting of the authentication) on one server. The protocol will support set up authentication only for the part of the resources on one Web server, unlike TLS client authentication.

Support for application-controled authentication behavior When authentication is implemented using forms and HTTP cookies, the Web application can control various aspect of authentication/authorization. To replace form-based authentication with our new protocol, we have to provide alternative methods for implementing such mechanisms inside the protocol.

Support for both TLS and non-TLS communications As HTTPS is widely accepted and understood by the users, it seems wise to simply use it for integrity and secrecy protections. However, layering two or more security protocols on top of each other is not always a simple problem. Our protocol is designed in a way using a channel-binding technology so that the whole system provides the same level of identity protection as usual when used with the TLS.

At the same time, the protocol can be used with plain HTTP, when the applications' performance requirements does not allow use of HTTPS. In this case our protocol still provides a strong protection of user's credential and the result of the authentication, although the content integrity is not well assured as usual.

Mid–term solution To satisfy all the above criteria, we set our goal time-frame to be mid-term (a few years). More precisely, we decided to accept requirements to change both server and client implementation to provide the required level of security. Of course, we are carefully designing the protocol so that the required modification to the implementation will be kept as small as possible.

3.2 Design overview

Our protocol is designed on the current HTTP authentication framework (RFC 2617). As usual in HTTP, the authentication will be activated when the server sends a 401 status code to the client requesting an authentication. When users have decided to authenticate themselves to the server, the client browser and the server will exchange an authentication credential using one of the *password-authenticated key exchange (PAKE)* family of protocols. This key exchange primitive provides a strong mutual authentication between clients and servers, and gives a strong protection against Phishing attacks. There are several variants of PAKE, and we're using the one which do not require the plain-text password on the server side, to minimize the risk of server-side password database leakage.

In addition, our protocol adds several adaptation to the underlying primitive, to achieve the required security property on this specific application. For example, the server-side password database entries are made both site- and user-specific so that the leaked database cannot be easily used with other sites. Session managements and key-caching mechanism are introduced so that the overhead introduced by the cryptographic exchange made minimum. Server authentication is made mandatory, and channel-binding technique ensures that any man-in-the-middle attacks cannot be performed.

Our design also supports "single sign-on" for several servers within the same domain. If specified by a firstly-authenticating server, the client browser will automatically tries an authentication with the same credential to the servers within the same domain. This feature is made possibly only with the strong mutual authentication: we are now safe to try authentication with the same password to another server using our protocol, because the protocol prevents all kind of password-guessing attacks from the server side.

The computational overhead of the key-exchange for the authentication is roughly comparable to the TLS key exchange of the same security strength. Once the key exchange and the authentication is done for the first time, the server has a choice on space-time trade-off: the exchanged key can be reused for several minutes to avoid performance overhead, or it can be disposed for reducing memory footprints. The design also considers use of separate authentication servers and TLS accelerators for further reduction of the overhead to the Web server.

(a) Mutual authentication is requested. Input fields are in the chrome area.

(b) Mutual authentication has been succeeded. Username "mutualtest" is displayed in the chrome area.

Figure 1: Chrome-area input fields for user-names and passwords

We also added several generic extensions to the HTTP authentication framework to enable use of this protocol with the recent Web applications with a minimum efforts. Our proposal includes two new headers to the HTTP specification: `Optional-WWW-Authenticate` and `Authentication-Control`. The former header allows the parallel distribution of both unauthenticated and authenticated Web-pages on the same URL, depending on the authentication status. The latter gives Web applications a precise control of the HTTP-level authentication behavior. The currently-supported features using that header includes time-out control of authentication, forced logging-out from the authenticated realm, or conditional Web-page redirection depending on the authentication status. We are currently considering to separate and extend this part to a new Internet-draft, so that this facility become more powerful and can be used even with other authentications schemes and Web applications.

3.3 User Interface Consideration

Existing Basic and Digest authentication usually uses a modal dialog box for asking usernames and passwords. However, it can be easily imitated by using either a stylized HTML, an HTML pop-up window or even a picture of a pop-up dialog box. In general, any region inside the content area is always vulnerable for forging by phishers. In this context, the weakness of the forms inside the document is needless to say. If plaintext passwords are sent to the phishers outside of this proposed protocol, we can not prevent any patterns of phishing attacks.

To prevent such kind of attacks, we propose to use the "chrome area" (the area where an address bar and other browser UI components are exist) for the new authentication protocol. In our implementation, the input boxes are initially not displayed. When users visit a website which requires mutual authentication, two input boxes for a username and a password automatically appear on the right of the address bar (Figure 1 (a)). When the user enters the user-name and the password the authentication begins, and after the authentication success has been recognized by the client (by checking the authentication credential sent from the server to the client), it displays a mutually-authenticated user-name with a green background on the right of the address bar (Figure 1 (b)), showing that the identity of the host is verified by the password. As long as the users are educated minimally to enter passwords in this area (like the current education of the pad-lock icon), the passwords are kept always safe.

Until recently, there seem some "borders" between protocol and application designs. Existing protocol-level specifications are not mentioning much about user-interface designs, and even such mentions are often ignored by the application developers. At the same time, the protocols are sometimes designed without good considerations on the real application usages. In that aspect, our proposal is an interesting instance to such a problem: we need to put minimum requirements on the user-interface on the protocol specification, yet allowing enough freedom to the application developers. We want to get a good advises on that.

4 Action plans

For future secure Web application authentication, we believe that our proposal is one of important building blocks, but, at the same time, not the whole part of that. For example,

- User-interface experience can be further improved, with possible integration with the identity management facilities.
- To use our protocol more securely and effectively, some secure API to the authentication facility from the browser contents may be needed.
- The relation with federated/delegated authentication should be more clarified and researched.
- Using existing secure authentication framework (e.g. Kerberos) on the Web site should be more investigated.
- There may be still the needs for client certificates. UI experience of that must be greatly improved.
- Support for two-factor or hardware-assisted authentication may be needed.
- HTTP authentication can be also useful with non-Web applications (including SOAP-style accesses Web application).

We are now proposing IETF for starting an standardization efforts for the HTTP authentication extensions. We will be happy if we can have some discussion and advises for the future possibilities.

References

[1] Yutaka Oiwa, Hajime Watanabe, Hiromitsu Takagi, Yuichi Ioku, and Tatsuya Hayashi. Mutual Authentication Protocol for HTTP. Internet Draft draft-oiwa-http-mutualauth-08, October 2010.

More resources are available on:
`https://www.rcis.aist.go.jp/special/MutualAuth/index-en.html`