PKAuth: A Social Login Protocol for Unregistered Applications

Francisco Corella
Pomcor
fcorella@pomcor.com

Karen P. Lewison
Pomcor
kplewison@pomcor.com

***Abstract*—Social login is a double-redirection mechanism whereby a Web application delegates user authentication to a social site and obtains access to the user's social context. Today social login is implemented using OAuth, which requires registration of the application with the site for authentication of the application to the site and identification of the application to the user by the site. As social login gains in popularity, this may lead to a situation where every application must register with the dominant social site (currently Facebook) just to be able to authenticate its users, and the dominant social site has the power to disable any application on the Web by revoking its registration.**

PKAuth is a protocol for social login that does not require registration and yet provides strong application authentication and identification. It relies for that purpose on the public key infrastructure of the Web. The application submits its TLS certificate as client certificate in a TLS handshake, and the site identifies the application to the user by displaying the information contained in the certificate. Additional information about the application may be provided to the site by a holder-of-key assertion or by optional prior registration.

ACKNOWLEDGEMENTS

This work was supported in part by NSF grant 1013594. The authors would like to thank Marius Scurtescu for a detailed reading of an earlier paper on PKAuth and many useful comments, and him and other subscribers to the OAuth mailing list and to the OpenID mailing list for very helpful discussions, which can be found in the archives of those lists.

I. INTRODUCTION

Social login [1] is a user authentication method used by Web applications that uses a double redirection mechanism. The application redirects the user's browser to a social site such as Facebook, Twitter or LinkedIn, which authenticates the user and then redirects the browser back to the application, conveying to the application the fact that the user has been successfully authenticated; the details vary from one protocol to the next. The application obtains from the site user identity data, including a user identifier relative to the site and other data such as the user's full name, avatar, birthday and/or verified email address. It also obtains subsequent access to the user's social context, including the ability to retrieve a list of related users and the ability to broadcast messages such as updates or tweets on behalf of the user. The application does not obtain the user's credentials for the site.

Social login via Facebook, Twitter and LinkedIn is implemented today using early versions of OAuth 2.0. OAuth is a protocol originally conceived as an authorization protocol; version 2.0 is currently being specified by an IETF working group. OAuth requires prior registration of the application with the site. The site uses the registration process to establish a shared secret used to authenticate the application, and to obtain information that it uses to identify the application to the user as it asks the user for permission to execute the social login protocol and provide the application with access to the user's account.

But compulsory registration is a dangerous thing. Since social login has compelling advantages for users and application providers, it may well become the de facto user authentication method of the Web. When that happens, if social login requires registration, every application will have to register with the dominant social site (currently Facebook) just to be able to authenticate its users. The dominant site will then have the power to disable any application on the Web by revoking its registration. This will be bad for the Web and for all parties involved, including the dominant social site, which will no doubt face regulation by governments.

In this paper we propose a protocol, called PKAuth, designed for social login, that provides strong security without requiring registration of the application with the social site, nor prior knowledge of the site by the application. The name of the protocol comes from the fact that it relies on the public key infrastructure of the Web. The social site uses the TLS certificate of the application, obtained in the course of a TLS handshake, to authenticate the application and identify it to the user. (TLS is also known as SSL.) Information in the certificate may be augmented by a holder-of-key assertion made by a party trusted by the social site, or by information verified by the site in the course of an optional registration process.

PKAuth belongs to a class of double-redirection protocols that includes Microsoft Passport (now Windows Live ID [2]), SAML Browser SSO Profile [3, section 4.1], OpenID [4], OAuth 1.0 [5], and the version of OAuth currently under development, OAuth 2.0 [6]. These protocols allow an application to delegate authentication to a site, or a user to delegate authorization to access the user's account on a site to an application, or both.[1] Besides not requiring registration, PKAuth avoids several security flaws that can be found in

[1]The site, however, is not necessarily a social site. We use the words "site" and "application" to refer to roles played in the protocol, without implying that there is an essential distinction between a Web site and a Web application.

other protocols of this class.

PKAuth is a federated authentication protocol like OpenID, and an authorization protocol like OAuth, but rather than combining the two protocol as proposed in [7] or [8], it combines the best ideas of OpenID and OAuth while avoiding the flaws. PKAuth also features new ways of supporting browser-resident applications and native applications running on desktops and mobile platforms such as smart phones and tablets.

The rest of the paper is organized as follows. Section II describes security flaws in redirection protocols that are addressed by PKAuth, including flaws described by prior work, and a new flaw that we discovered and is now being heatedly discussed on the OAuth mailing list. Section III is a high-level description of PKAuth. It includes an overview of the protocol flow in section III-E, a detailed protocol flow in section III-F, and protocol security considerations in section III-G. Section IV recapitulates and points out how PKAuth could be used for decentralized social networking.

II. Avoiding Double-Redirection Vulnerabilities

PKAuth addresses vulnerabilities that affect other double-redirection protocols.

As we were designing PKAuth, we thought of the following attack. As the user's browser is redirected back from the site to the application's callback endpoint, the attacker intercepts or observes the redirected request sent by the browser to the application. Interception is a man-in-the-middle attack where the attacker obtains the redirected request and does not relay it to the application. Observation is a passive attack. Both types of attack can be mounted, e.g., using a rogue WiFi access point [9]. Having obtained the redirected request, the attacker sends it to the callback endpoint of the application from the attacker's own browser. The attacker can thus impersonate the user vis-a-vis the application, and can use the application to access the user's account at the site.

The attack can be prevented easily by requiring TLS protection for the callback endpoint of the application, so we included this requirement in the design of PKAuth.

Then we observed that none of the above-mentioned double redirection protocols requires TLS protection for the callback endpoint. Furthermore, developer documentation provided by at least one social network uses http rather than https in examples of redirect URLs that target the callback endpoint. As a result, it seems that many applications that use OpenID or OAuth today do not protect the callback endpoint and are vulnerable to the attack.

We surveyed the literature to see if the attack had been mentioned before. It is not mentioned in Kormann et al.'s security analysis of Microsoft Passport [10], nor in the security considerations section of OpenID 2.0 [11], nor in the security considerations section of OAuth 1.0 [5], nor in Barnes et al.'s paper on the OAuth security model (which does not actually discuss the double redirection mechanism) [12]. The section on the Browser SSO Profile of the security and privacy considerations document for SAML version 2.0 [13, section 7.1.1] has a blanket warning against man-in-the-middle attacks, but does not mention this particular attack, and the SAML Browser SSO Profile itself [3, section 4.1] does not require TLS. Gross [14] briefly mentions the possibility of a man-in-the-middle attack between the browser and the callback endpoint against version 1.1 of the SAML Browser SSO Profile, but does not draw any consequences, and the subsequent analysis of version 2.0 by Gross et al. [15] does not mention the attack.

There are two attack variants besides the basic attack described above. In one variant, the attacker intercepts or observes an authentication cookie sent by the application to the browser in response to the redirected request. This variant is also prevented by requiring TLS at the callback endpoint. In the other variant, the attacker intercepts or observes the redirect response from the site to the browser which precedes the redirected request from the browser to the application. This redirect response is typically a response to a request from the browser to the site that authenticates the user by submitting a username and password or an authentication cookie. One would think that such request and its response would be protected by TLS, preventing the attack. But social networks do not always provide such protection. PKAuth explicitly requires it.[2]

PKAuth addresses several other vulnerabilities that affect double redirection protocols: phishing attacks, information leaks through referrer headers, login cross-site request forgery (CSRF) attacks, denial-of-service attacks on the callback endpoint, and denial-of-service attacks by storage exhaustion. Phishing attacks have been discussed by Kormann et al. [10] in connection with Microsoft Password, and by many critics of OpenId such as Laurie [16]. Leakage through referrer headers has been discussed by Gross et al. [15]. We explain these vulnerabilities and how PKAuth addresses them throughout section III, and more particularly in the security considerations section III-G.

III. PKAuth

This section is not a formal specification of PKAuth. It is only a high-level description of the protocol, omitting discussion of error handling and details such as the names and encodings of parameters. On the other hand it explains how the site and the application can implement the protocol, referring to storage of information in database records and cookies. Such explanations are usually omitted from protocol specifications, but they help understand the protocol and evaluate its security.

A. Application Types

PKAuth can be used with an application of any of the following types:

[2]When we discovered the attack we saw that the latest draft of OAuth 2.0 did not require TLS for the callback endpoint, so we explained the attack to the IETF OAuth working group. After many arguments, the working group is now realizing that the attack threat is real and having a heated discussion on whether TLS should be required for the callback endpoint. The working group has been transferred from the application area of the IETF to the security area due to the intensity of the discussion of this security issue.

1) A traditional Web application running on a Web server.
2) A browser-resident application, such as an AJAX application implemented in Javascript or a rich application implemented in Flash, Flex or Silverlight. A browser-resident application is downloaded from a Web server and has a server-side component running on that server.
3) A native application running on a desktop, laptop, or netbook under a desktop operating system such as Windows, MAC OS, or Linux, or on a mobile platform such as a smart phone or tablet under an operating system such as iOS, Android, or Windows Mobile. A native application may or may not rely on an ancillary Web server.

The protocol flow is the same for all three types of applications, and the social site does not need to know what type of application it is interacting with, as far as the protocol is concerned.

B. POST Redirection

While PKAuth uses a double-redirection mechanism, it does not use traditional redirections. A traditional redirection is a GET redirection, consisting of an HTTP redirect response, followed by a GET request. The parameters passed by the redirection are embedded in the redirection URI and may therefore be leaked via referrer headers, the browser history, or server logs.

PKAuth uses instead POST redirections. A POST redirection is implemented by downloading or otherwise creating a form, with POST as submission method. The parameters of the redirection are encoded as hidden inputs, and Javascript code is used to automatically submit the form. If Javascript is disabled in the browser, the user can click a button to submit the form; but in today's Web, Javascript is unlikely to be disabled.

POST redirections are also used in the SAML Browser SSO profile [3, section 4.1] in OpenID [11].

C. Certificates and certificate chains

Every application must have an application certificate signed by a CA and backed by a certificate chain ending in a generally trusted root certificate.

The application certificate of a traditional Web application is a TLS certificate, which the application uses as a client certificate when authenticating to the site, and as a server certificate when authenticating to the user's browser.

The application certificate of a browser-resident application is also a TLS certificate. The certificate and the corresponding private key are kept in the server-side component of the application. The certificate is used as a client certificate to authenticate connections to the site proxied through the Web server, and as a server certificate for authentication to the browser.

The application certificate of a native application is used to sign application instance certificates. An instance of a native application running on the user's machine has a TLS certificate and corresponding private key. The TLS certificate is signed by the application, and backed by the application certificate and certificate chain. The instance uses the instance certificate to authenticate itself to the site. In the case where the application uses an ancillary Web server, the application certificate is also used as a TLS server certificate by the ancillary Web server.

A native application certificate will be recognizable by a special key purpose in the extended key usage field of the X.509 certificate.

D. Optional Registration

Although PKAuth does not require the application to register with the site, it does not preclude it. Registration is optional. During the optional registration process the site obtains information about the application and stores it in a registration record, which may be updated as needed from time to time. The information is later used to identify the application to the user as the protocol is executed.

If an application registers with the site, it must register an application certificate signed by a CA and backed by a certificate chain ending in a generally trusted root certificate. If the application's certificate, or a CA certificate in the certificate chain, expires or is revoked, the application must update the registration.

In addition to the information contained in the application's certificate and certificate chain, the site may collect other information about the application during the optional registration process, store it in the registration record, and use it as additional information to identify the application to the user during an execution of the protocol.

E. Protocol Flow Overview

The following is a summary of the protocol flow. Each step us explained in more detail in the next section.

1) The user specifies a social site.
2) The application discovers a direct request endpoint for the site, a user interaction endpoint, and one or more access endpoints including an identity-data access endpoint.
3) The application generates an application presession key and an application presession token containing the presession key and signed by the application for later verification by the application itself. The application sends a request to the direct request endpoint, including the presession token, a callback URI, and a specification of the desired identity data and the desired scope and duration of subsequent access to the user's site account. The application authenticates itself with a TLS client certificate and an optional holder-of-key assertion. The site creates a site presession record and returns a site presession key that refers to the site presession record.
4) After receiving an acknowledgement of the direct request, the application redirects the browser to the site, passing the site presession key, and setting a cookie with the application presession key in the browser.
5) The site verifies that the user is logged in to the site, identifies the application to the user, and asks the user for permission to proceed.

6) The site deletes the site presession record and creates an access session record referenced by an access token. It redirects the browser to the application, passing the access token and the application presession token, to which the browser adds the cookie containing the application presession key.
7) The application uses the access token to obtain user identity data from the site's identity-data access endpoint.
8) The application logs the user in.
9) The application further uses the access token to access the user's social context at the site by sending requests to site access endpoints.

F. Protocol Flow Details

Step 1 – User specifies social site: The application invites the user to log in and gives the user the option of being authenticated by a social site. The application may offer the user a choice of sites, and may use XAuth [17] to suggest a site to which the user is currently logged in. The application must allow the user to specify a site that is yet unknown to the application by typing in the domain name of the site, or by using an ad-hoc search facility. As the user types in a domain name, the application may suggest completions.

Note. The application should allow the user to refer to a site using an alternative domain name that differs from the official domain name of the site by the presence or absence of a "www" prefix. The site should redirect URIs that use the alternative domain name to URIs that use the official domain name.

Step 2 – Application discovers site endpoints: Once the user has specified a site, the application downloads a file, the "site info file," from a well known location to be determined (e.g. from a URI of the form https://SITE-NAME/.well-known/RELATIVE-PATH where SITE-NAME is the official domain name of the site, and RELATIVE-PATH is a relative path to be registered with the Well-Known URI registry once the Well-Known URI standard [18] has been adopted). The site info file specifies the URIs of a direct request endpoint, a user interaction endpoint, and one or more access endpoints that provide access to user accounts. One of the access endpoints is an identity-data access endpoint that can be used to obtain user identity data including a user identifier that is unique relative to the site, and other user data such as the user's full name, the user's birthday, the user's photo or avatar, and/or a verified email address.

All site endpoints are TLS endpoints, i.e. endpoints whose URIs use the https scheme. The host portion of those URIs must coincide with, or be a subdomain of, the official domain name of the site.

Step 3 – Application sends direct request: The application sends an HTTP POST request to the direct request endpoint of the site. This direct request is sent over a TLS connection with authentication provided by a client certificate.

If the application is a traditional application or a browser-resident application, it uses as client certificate an application certificate signed by a CA and backed by a certificate chain ending in a generally trusted root certificate.

If the application is a browser-resident application, it proxies the direct request through its server-side component, and in the proxied connection, the server-side component authenticates itself to the site using the application certificate. The client-side component running in the browser must authenticate itself to the server-side component to avoid an open proxy, but this is a matter internal to the application.

If the application is a native application, the instance of the application running on the user's machine uses as client certificate an instance certificate signed by the application and backed by the application certificate.

In all cases, if the application has registered with the site, the application certificate and certificate chain must be the ones registered with the site.

The direct request includes the following parameters:

- Optionally, if the application has registered with the site, a registration key assigned during registration.
- The URI of a callback endpoint (the callback URI) exposed by the application.
  If the application is a traditional Web application or a browser-resident application, the callback endpoint is a TLS endpoint, and the application authenticates itself at that endpoint using as server certificate the same application certificate that it uses as a client certificate to send a direct request.
  If the application is a native application that relies on an ancillary Web server, the callback endpoint is a TLS endpoint, and the application authenticates itself at that endpoint using as server certificate the same application certificate that it uses to back instance certificates used as client certificates.
  If the application is a native application that does not rely on an ancillary Web server, the callback endpoint is local to the user's machine. If the user's machine has a local Web server, the callback URI may be a local URI (i.e. a URI whose host portion is localhost or a local IP address, a local IP address being one whose first octet is 127). The local Web server is then configured to dispatch requests that target the local URI to the application. If the user's machine is a mobile platform, the callback URI may use a custom scheme that the application is registered to handle.
- Optionally, a holder-of-key assertion, such as a SAML holder-of-key assertion [3, section 3.1], that binds the application certificate to additional information that may be used to identify the application to the user.
- A specification of the identity data that the application wishes to obtain from the site.
- A specification of the scope and duration of subsequent access to the user's account that the application wishes to obtain from the site.
- An application presession token, used as a countermeasure against login CSRF as explained below in section III-G7, and against a denial-of-service attack on the callback endpoint, as explained below in section III-G6.

The application presession token is a virtual presession record (i.e. a record that is signed rather than stored), which is the concatenation of the following fields:

- A high entropy random presession key,
- A timestamp,
- The official domain name of the site, and
- A signature of the above fields computed by the application for later verification by the application itself.

The application does not store the presession token nor the information it contains, except that it keeps in memory the presession key until it receives a response to the direct request.

When it receives the direct request, the site verifies the TLS client certificate and its certificate chain as the connection is established, checking in particular that the certificates have not expired and are not revoked, and that the chain ends in a root certificate trusted by the site.

The site derives the application certificate and certificate chain from the client certificate and chain. If the certificate that backs the client certificate (the first certificate in the client certificate chain) is a native application certificate, as indicated by an extended key usage field, the application certificate is that native application certificate, and the application certificate chain is the rest of the client certificate chain. Otherwise the application certificate and chain coincide with the client certificate and chain. If the direct request includes a registration key parameter, the site verifies that the application certificate and chain derived are the ones that were registered.

If the callback URI is not a local URI, the site verifies that the host portion of the callback URI coincides with the domain name specified in the subject field of the application certificate.

If the direct request includes a holder-of-key assertion, the site verifies the assertion against a collection of assertion authorities that it trusts.

The site creates a site presession record, containing:

- A random, high-entropy site presession key, which can be used to retrieve the record.
- All the parameters in the direct request.
- The client certificate and certificate chain presented by the application, which include the application certificate and its certificate chain as explained above.

Then it returns the site presession key to the application in the HTTP response. (Notice that there is a site presession key and an application presession key.)

Step 4 – Application redirects browser to site: When the application receives the site presession key, it redirects the browser to the site using a POST redirection as described above in section III-B, passing the site presession key as only parameter, and setting a cookie in the user's browser whose value is the application presession key.

If the application is a traditional application, the POST redirection is in response to the user's request for a social login in step 1. The HTTP response to that request sets the cookie, downloads the form, and downloads Javascript code that submits the form.

If the application is a browser-resident application, the POST redirection is accomplished by creating a form in a window, tab or frame and submitting it. The cookie is set directly by the browser-resident code.

If the application is a native application, the POST redirection is accomplished in two steps. First, the application launches an external browser and points it to the URI of an application endpoint, called the pre-redirection URI. The application responds to the request from the external browser targeting the pre-redirection URI by setting the cookie, downloading the form, and downloading Javascript code that submits the form. The pre-redirection URI, like the callback URI, may be a TLS URI targeting an ancillary Web server, or a local URI, or a URI with a custom scheme. It must be structured so that it can set a cookie that will later be sent by the browser to the callback URI.

Step 5 – Site verifies user is logged in, identifies application to user, asks permission: When the site receives the redirected request, it verifies that the user is already logged in at the site, e.g. as determined by the presence of a site authentication cookie in the browser, rejecting the request if this is not the case.

The site uses the site presession key contained in the request to retrieve the site presession record.

The site identifies the application to the user by displaying:

1) The information contained in the application certificate, found in the site presession record. The site displays all the fields in the application certificate that have relevant information about the application, indicating which of them are known to have been verified by the CA that signed the certificate.
2) The names of the CAs that are the subjects of the certificates in the certificate chain that backs the application certificate, found in the site presession record, with the option to display additional information about those CAs.
3) If the application has registered with the site, any additional identifying information present in the registration record.
4) If an assertion is present in the site session record, the information about the application stated by the assertion.
5) The callback URI.
6) The following information extracted from the callback URI:
   - If the scheme of the URI is https, the registered-domain portion of the host portion of the URI.
   - If the URI is local, an indication that the browser will be redirected to a local application running in the user's machine.
   - If the URI has a custom scheme, any information that the site may have on how the custom scheme is usually handled in various platforms, particularly in the user's platform if the site can guess what it is.

The site asks the user permission to identify the user to the

application, and provide the application with identity data and subsequent access to the user's account. The site may restrict the identity data and the scope and duration of access to be provided, and may allow the user to restrict them further.

To grant permission, the user is asked to click a button that is a part of a form with a POST submission method. The form must target a TLS endpoint of the site, and must be protected against CSRF as described below in section III-G5.

Step 6 – Site redirects browser to application: When the user grants permission, the site sets up a session that will allow the application to access the user's account. The session is implemented by an access record, where the site stores:

- A random high-entropy string, called an access token, that can be used to retrieve the record.
- An identifier that uniquely identifies the user among all registered users of the site
- A specification of additional identity data to be provided in addition to the identifier, based on the request by the application found in the presession record, and any restrictions applied by the site or the user.
- The scope and duration of subsequent access to be provided to the user's account, based on the request by the application found in the presession record, and any restrictions applied by the site or the user.
- The client certificate and certificate chain used by the application in step 3 and found in the site presession record, or, equivalently, a cryptographic hash of the concatenation of the certificate and chain.

Then the site deletes the site presession record and redirects the user's browser to the callback URI (using a POST redirection), passing the following information:

1) A status parameter indicating that the request succeeded.
2) The application presession token that the application sent in step 3 and was stored in the site presesssion record.
3) The access token.
4) A specification of the identity data obtainable with the access token.
5) A specification of the scope and duration of subsequent access obtainable with the access token.

If the application is a traditional Web application, it receives this information directly through the callback endpoint.

If the application is a browser-resident application, it receives the information via its server-side component, which sets cookies to convey the redirection parameters to the client-side component running in the user's browser. The cookies can be set without application-specific programming using a generic middleware module such as an Apache module that encodes into cookies the values of any POST parameters.

If the application is a native Web application and the callback URI targets an ancillary Web server, the application instance running on the user's machine receives the information via the ancillary server, which copies the body of the redirected HTTP request to the body of the HTTP response and sets the content-type header of the response to a media type handled by the application. Again this can be achieved without application-specific programming using a generic middleware module such as an Apache module that copies the body of any POST request to the body of the response.

If the application is a native Web application and the callback URI is local or has a custom scheme, the instance of the application running on the user's machine receives the information directly through the callback endpoint.

In all cases the application verifies its own signature in the presession token and verifies that the token is reasonably recent and is accompanied by a cookie containing the presession key found in the token. If so, the application uses the official domain name of the site found in the presession token to retrieve the site info file as it did in step 2. (The application may cache the site info file, as discussed below in section III-G8.)

Step 7 – Application uses access token to obtain user identity data: The application sends a request to the identity-data access endpoint found in the site info file, including the access token as a parameter, and authenticating itself using the same TLS client certificate and certificate chain used in step 3. The identity-data access endpoint uses the access token to locate the access record, and verifies that the client certificate and certificate chain are those stored in the access record, or that the cryptographic hash of the client certificate and chain coincides with the one stored in the record.

In response to the request the site provides an identifier that uniquely identifies the user among the users of the site, and the identity data specified in the access record.

Step 8 Application logs user in: The application uses the combination of the user identifier provided by the site and the official domain name of the site found in the presession token as an identifier for the user within the application. We refer to such an identifier as a remote identifier. The user may have multiple remote identifiers, as well as a local identifier associated with local credentials such as an ordinary user ID and password.

When the site provides the user identifier, the application looks for a user account that is associated with that identifier. If it finds one, it creates a login session for the user, and sets an authentication cookie in the browser. The application may use the identity data provided by the site to update the user's account or to register the user if no account is found.

Step 9 Application accesses user's social context: The application further uses the access token from time to time to access the user's social context by sending access requests to site access endpoints found in the site info file. The application includes the access token with each request, and authenticates itself using the same TLS client certificate and certificate chain used in step 3. The site verifies the access token and the client certificate and certificate chain as in step 7.

G. Security Considerations

1) Reliance on the Browser: The site relies on the browser to deliver the access token to the application that was identified to the user. Security hinges on the fact that the application was identified to the user based on information contained in the certificate with which the application authenticated itself

to the site when establishing the very same connection through which it communicated the callback URI to the site, viz. the direct request connection.

The fact that the site has to rely on the browser is a security weakness. The browser has its own root certificate store, which may contain CA certificates that the site does not trust. Furthermore, if an attacker attempts the man-in-the-middle attack between the browser and the application discussed in section II, the browser will detect an invalid certificate, but will only issue a warning to the user. The user may ignore the warning based on the fact that the site has previously identified the correct application to the user in step 5.

With the current state of Web technology, this weakness seems unavoidable. However, we plan to propose new standards that will allow the site to tell the browser what certificate it should expect at the callback endpoint (viz. the same certificate that the application used in step 3). If a different certificate is presented at the callback endpoint, the browser will then refuse to connect instead of just warning the user.

2) Host Portion of Callback URI: The site checks that the host portion of a non-local callback URI coincides with the domain name specified in the subject field of the application certificate. This is not essential: the site could allow the application to use a differenty domain name and a different certificate to receive connections at the callback URI. The protocol requires it for the sake of defense in depth.

3) Role of Authentication in Steps 7 and 9: Since the site relies on the browser to deliver the access token to the correct application, the fact that the application authenticates itself with a client certificate when using the access token is not essential. It is useful, however, for two reasons.

First, it plays a role of defense in depth. If the access token is delivered to an application controlled by an attacker, e.g. because the attacker has been able to plant a bogus CA certificate in the root certificate store of the user's browser, the attacker will be able to submit the access token to the callback endpoint of the legitimate application, impersonating the user vis-a-vis the application and gaining access to the site through the application. But the attacker will be constrained by having to go through the application. Without application authentication, the attacker would be able to access the site directly and the attack might be able to do more damage; furthermore, the application may be blamed for the attack.

Second, it prevents an attack where a rogue application controlled by the attacker legitimately obtains an access token, intended for the rogue application, and uses it to impersonate the user vis-a-vis a legitimate application. With application authentication in step 7, the site will reject the access token intended for the rogue application when presented by the legitimate application and the attack will fail. However, this attack can be prevented more simply by the legitimate application declaring its identity to the site in step 7 without proving it.

4) Phishing Attacks: Double redirection protocols such as OpenID and OAuth are highly vulnerable to phishing attacks [10], [16]. A malicious application may redirect a user to a site that impersonates a legitimate site and obtains the user's credentials for the legitimate site. If the site endpoint is not protected by TLS the user is defenseless against the attack. Even if the endpoint uses TLS, the user may be easily tricked, as explained by Kormann et al. [10]. Furthermore making users accustomed to providing credentials after a redirection is bad for the Web, because it facilitates phishing attacks in general.

PKAuth avoids this vulnerability by requiring the user to be already logged in at the site when the application delegates authentication to the site. This countermeasure is currently used by OpenID providers such as WordPress. But the countermeasure is more effective when its use is required by the protocol rather than being left to the discretion of particular implementations.

5) CSRF Attacks against the Site: The fact that the user is already logged in to the site makes it possible to use a traditional technique to provide protection against CSRF for the form that contains the button used by the user to grant permission. The form can have a hidden field whose value is a high-entropy random string contained in the login session record.

6) Denial of Service Attacks on the Callback Endpoint: One purpose of the application presession token is to mitigate denial-of-service attacks against the callback endpoint of the application. Without it, an attacker could submit many bogus access tokens to the callback endpoint which the application would have to forward unconditionally to the site's identity-data access point, possibly causing the site to lock out the application or revoke its registration. The presession token serves as a countermeasure because a presession token with a valid signature can only be obtained by an authenticated user of the site.

In case of attack by an authenticated user, the application can send the presession token to the site, and the site can use the token to identify the attacker. The site will have the burden to decide whether it is the user or the application that is responsible for the sending bogus tokens.

7) Login CSRF Attacks against the Application: The purpose of the setting the cookie with the application presession key in step 4 is to prevent a modified version of the login CSRF attack described in [19]. Without it, an attacker who is a legitimate user of the site and the application could legitimately obtain an access token and a presession token, and trick the user into submitting them to the callback endpoint, causing the user to log in to the application as the attacker. Consequences of a user logging in as an attacker are explained in [19].

8) Denial of Service by Storage Exhaustion: To avoid a denial of service by storage exhaustion, the application avoids keeping storage allocated while waiting for the user to be authenticated by the site. That is the reason for not creating a physical application presession record, using instead a signed presession token, which plays the role of a virtual presession record. While the application does not keep storage allocated before authentication, it may use a cache of bounded size to remember information that can be retrieved again if necessary, such as the contents of the site info file.

The site allocates a presession record when it receives the direct request in step 3 before user authentication. But the user is required to be already logged in, so the site does not keep storage allocated while waiting for input from an unauthenticated user.

The access record is a physical record because it is created for an authenticated user. The site could use a virtual access record instead by including all the access information in the access token and signing the token. But that would require a mechanism for revoking access tokens.

IV. Conclusion

We have proposed a social login protocol that allows a Web application to delegate user authentication to a social site and thereby gain access to the user's social context. We envision PKAuth as a successor to OAuth, the protocol used today to implement the "Login With Facebook" button and similar social login facilities. Whereas OAuth requires prior registration of the application with the social site, in PKAuth registration is optional. A social site using PKAuth could choose to require registration, for example, only for applications that perform sensitive tasks such as account administration on behalf of users.

PKAuth is primarily intended as a solution for a looming danger on the Web: the danger that, as social login becomes more and more popular, all applications will have to register with the dominant social site, and the social site will gain the power to disable any application by revoking its registration, an undesirable situation for all parties involved, including the dominant social site.

But PKAuth also provides other technical and societal benefits. As technical benefits, we have seen that PKAuth solves several security problems that affect similar double-redirection protocols, and provides new ways of supporting modern browser-resident and native applications. As a societal benefit, PKAuth should enable the emergence of decentralized social networks, by allowing social sites within a decentralized network to access each other's data without prior registration.

Much work remains to be done. What we have proposed is only a high-level description of the protocol. The protocol must be fleshed out with precise parameter encodings, error handling, and interoperable means of specifying and obtaining identity data and access to a user's social context. Reference implementations must then be developed, independent implementations must demonstrate interoperability, and a standard must be published by a standards body.

References

[1] Janrain (we believe that the term social login was coined by Janrain). [Online]. Available: http://www.janrain.com/products/engage/social-login

[2] Microsoft, "Windows Live ID." [Online]. Available: http://passport.net

[3] J. Hughes *et al.*, "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard," March 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf

[4] Wikipedia, "OpenID." [Online]. Available: http://en.wikipedia.org/wiki/OpenID

[5] E. Hammer-Lahav, "The OAuth 1.0 Protocol," April 2010, IETF RFC 5849. [Online]. Available: http://tools.ietf.org/html/rfc5849

[6] ——, "The OAuth 2.0 Protocol Framework, Version 13," February 16, 2011. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-v2-13

[7] OpenID OAuth Hybrid Working Group, "OpenID and OAuth Hybrid Extension." [Online]. Available: http://wiki.openid.net/w/page/12995194/OpenID-and-OAuth-Hybrid-Extension

[8] D. Recordon, "OpenID Connect." [Online]. Available: http://openidconnect.com/

[9] P. Moceri and T. Ruths, "Cafe Cracks: Attacks on Unsecured Wireless Networks." [Online]. Available: http://www1.cse.wustl.edu/~jain/cse571-07/cafecrack.htm

[10] D. P. Kormann and A. D. Rubin, "Risks of the passport single signon protocol," *Computer Networks*, vol. 33, pp. 51–58, 2000. [Online]. Available: http://avirubin.com/passport.html

[11] OpenID Foundation, "OpenID Authentication 2.0 Final," December 5, 2007. [Online]. Available: http://openid.net/specs/openid-authentication-2_0.html

[12] R. Barnes and M. Lepinski, "The OAuth Security Model for Delegated Authorization," July 8, 2009. [Online]. Available: http://tools.ietf.org/html/draft-barnes-oauth-model-01

[13] F. Hirsch, R. Philpott, and E. Maler, "Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0," March 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf

[14] T. Gross, "Security analysis of the SAML single sign-on browser/artifact profile," in *Proceedings of the Computer Security Applications Conference*, 2003, pp. 298–307. [Online]. Available: http://www.acsac.org/2003/papers/73.pdf

[15] T. Gross and B. Pfitzmann, "SAML artifact information flow revisited," IBM Zurich Research Laboratory, Tech. Rep., 2006. [Online]. Available: http://www.zurich.ibm.com/security/publications/2006/GrPf06.SAML-Artifacts.rz3643.pdf

[16] B. Laurie, "OpenID: Phishing Heaven," January 19, 2007. [Online]. Available: http://www.links.org/?p=187

[17] XAuth Community, "Xauth spec." [Online]. Available: http://xauth.org/spec/

[18] M. Nottingham and E. Hammer-Lahav, "Defining well-known URIs," December 30, 2009. [Online]. Available: http://tools.ietf.org/html/draft-nottingham-site-meta-05

[19] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2007. [Online]. Available: http://www.adambarth.com/papers/2008/barth-jackson-mitchell-b.pdf