Analysis of a Privacy Vulnerability in the OpenID Authentication Protocol

Manuel Urueña
Universidad Carlos III de Madrid
Email: muruenya@it.uc3m.es

Christian Busquiel
Universidad Carlos III de Madrid
Email: 100032895@alumnos.uc3m.es

***Abstract*— This paper studies the privacy risks for the users of the OpenID Single Sign-On (SSO) mechanism. A privacy vulnerability in the OpenID Authentication Protocol that leads to the exposure of the OpenID user identifier to third parties is described in detail. It has been verified that many existing OpenID agents are currently leaking the (potentially unique) OpenID identifiers of their users to third parties, like advertisement and traffic analysis companies. Therefore we consider this vulnerability as a real and widespread privacy risk for OpenID users. Thus, this paper also studies the solution space of this problem and defines a number of possible countermeasures. After analyzing their advantages and drawbacks, we finally propose two solutions to this problem, one for the long term to avoid the root cause of the vulnerability, and another short-term mitigation.**

I. Introduction

OpenID is a Single Sign-On (SSO) mechanism for the Web that is becoming increasingly popular. Big Internet players like Google, Yahoo, Flicker, WordPress or AOL support and even provide OpenID identifiers, and tens of thousands of web sites support OpenID. OpenID has been developed by the OpenID Foundation [1], which is being sponsored by big corporations such as Microsoft, Google, IBM, PayPal, Verisign or Yahoo.

The main feature of OpenID, and in general of any SSO mechanism, is to provide a single user identifier to log in all the web sites that support it. Furthermore, the user authentication process is centralized in the OpenID provider, thus usually it is only necessary to authenticate once per web browsing session, hence the Single Sign-On (SSO) name. This is important, not only from its convenience, but also from a security point of view, because with OpenID the user credentials are only stored in a single, trusted place.

This clearly contrasts with the current situation where each user has multiple usernames and passwords in order to log in many different websites, with the obvious trouble for users. Even worse, because of this mess, most users use the same login and password for all the websites, and never change it. This means that if any of the websites where the user was registered is malicious or its security is breached, the attacker could employ the obtained user credentials to gain access to other similar websites. Therefore, with shared usernames and passwords, the security of the whole user's website chain is the one of the weakest link, the website with the lowest security.

With OpenID, on the one hand, user credentials and personal information are stored just in the OpenID provider chosen (trusted) by the user, thus it is a lot easier to protect it. Also, unlike previous SSO proposals [2] OpenID is not tied to a specific identity provider but anyone can setup an interoperable OpenID provider, even the end user herself. Finally, and also because of centralization, it is possible for the OpenID provider to implement advanced authentication mechanisms such as One-Time Password Tokens or Smart Cards.

On the other hand, if OpenID or any other SSO identity mechanism becomes commonplace and each user has a single and unique identifier for all her public information published in the web, like posts, photos, videos, etc. the associated privacy risks are evident. This is the center of a very interesting debate in the SSO and the social networks communities, and many alternatives, such as temporal, anonymous ids, or using multiple identities per user, are being proposed as potential solutions to this problem.

This paper does not study the problem of unique identifiers for public postings, but the potential ability of third parties (e.g. advertisement and audience metering agencies) to track all the web visited by a given OpenID user, because of how the OpenID Authentication Protocol works and exchanges information on top of the HTTP protocol.

The structure of this paper is as follows: Section II provides a summary of how the OpenID Authentication Protocol works, in order to fully understand the privacy vulnerability that has been found, which is analyzed in detail in section III. Then, different countermeasures to this problem are presented in section IV. Finally, the conclusions of this work are presented in section V.

II. OpenID

First of all, it is necessary to define the terminology employed by OpenID. The OpenID architecture is composed by three types of agents:

- **User Agent**: The browser of the User that wants to log in in a website (*Relying Party*) with an *OpenID Identifier*, obtained from the *OpenID Provider* of her choice.
- **OpenID Provider (OP)**: An authentication server that provides one or more unique identifiers for the end User, and validates the User's credentials on behalf of the *Relying Parties*.
- **Relying Party (RP)**: The website where the end user tries to log in employing her *OpenID Identifier*. This identifier will be validated by the *OpenID Provider* of the User.

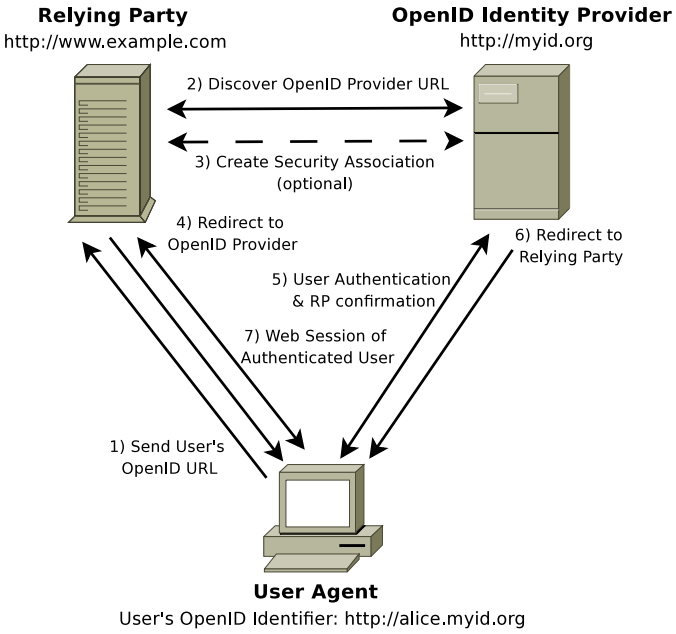

Fig. 1. The different phases of the OpenID Authentication Protocol [4].

In OpenID, Users are uniquely identified by means of a particular `http:` URL, which is called **OpenID Identifier**[1] (e.g. `http://alice.myid.org`). Initially, this is the only information provided by the User to the Relying Party when trying to log in. There are two benefits of using an URL as an identifier: First of all, URLs are hierarchical in nature by means of its domain part, thus each OpenID Provider is able to define its own set of globally unique identifiers without colliding with other OpenID Providers. Also, by providing a URL from the OpenID Provider DNS domain, Relying Parties could easily found the IP address of the OpenID authentication server and begin with the discovery and authentication phases of the OpenID Authentication Protocol.

The OpenID Authentication Protocol is the cornerstone of the OpenID Single Sign-On (SSO) mechanism. Nowadays there are two versions of this protocol in use, version 1.1 [3] and version 2.0 [4], which is the latest one. The OpenID Authentication Protocol allows Relying Parties and OpenID Providers to exchange information on top of the Hypertext Transfer Protocol (HTTP) [5] both, directly or transparently through the User Agent. An very important feature of the OpenID Authentication Protocol is that it does not require any kind of special support from User Agents. Any HTTP 1.1 standard browser can be employed as a OpenID User Agent, even if it does not support javascript or cookies have been disabled by the user.

Following, and with the aid of figure 1, we will explain in detail the different phases of the OpenID Authentication Protocol and the format of its messages, as well as its interaction with the underlying HTTP protocol:

1) The OpenID authentication process starts when a User, employing an User Agent visits a website (*Relying Party*) and tries to log in. To that end she provides her OpenID URL identifier (e.g. `http://alice.myid.org`) to the Relying Party, usually in a HTML form with a HTTP POST operation (the recommended name for the HTML form field is "`openid_identifier`").
2) The Relying Party employs the URL supplied by the User in order to discover the endpoint of the OpenID Provider to be employed for the authentication of the User. Although there are different OpenID discovery mechanisms, this is usually performed just by requesting the web page pointed by the User's OpenID Identifier URL and looking for a particular `<link>` tag within the `<head>` part of this HTML page. This `<link>` tag must include the `rel="openid2.provider"` or `rel="openid.server"` attribute values (depending on the OpenID protocol version), plus a `href` attribute that defines the OpenID Provider endpoint URL where the user authentication takes place (e.g. `href="http://myid.org/openid-auth/"`).
3) After discovering the OpenID Provider endpoint, the Relying Party may (optionally) create a security association with the OpenID Provider, in order to negotiate a shared secret using a Diffie-Hellman [6] secure key exchange. This key is later employed to sign and verify the messages exchanged between them. This security association is recommended because otherwise it is necessary to address direct requests between them in order to verify each authentication request/response message.
4) Then, the Relying Party redirects the User Agent to the discovered OpenID Provider endpoint by means of a 302 HTTP Temporary Redirect message, where the HTTP `Location` header specifies the target OpenID Provider endpoint. This URL must also contain all the parameters of the OpenID Authentication Request (`openid.mode=checkid_setup`). The most important ones are the User's OpenID Identifier (`openid.identity`), the identifier of the optional security association established between the RP and the OP (`openid.assoc_handle`), and the URL of the Relying Party where the result of the Authentication operation should be sent back (`openid.return_to`). Additionally, the Relying Party can request, by means of protocol extensions, some additional information about the User such as her full name, gender, or e-mail.
   Figure 2 shows an (quite simplified[2]) example of an OpenID Authentication Request message, sent by the User Agent to the OP because of the RP redirect reply.

[1]Instead of a `http:` URL, OpenID users can employ a XRI URL pointing to an XML document as an alternative to identify themselves. For simplicity in this paper we will only consider plain OpenID `http:` URL identifiers. Nevertheless all the results of this work are still valid, irrespectively of which kind of OpenID identifier is employed.

[2]OpenID Authentication parameters must be percent-encoded before being added as URL parameters. This encoding has been ignored in all the examples of the paper in order to ease it understanding by the reader.

```
GET /openid-auth?openid.mode=checkid_setup
  &openid.identity=http://alice.myid.org
  &openid.return_to=http://www.example.org
  /openid-login
  &openid.assoc_handle=xxxxxxxxxxxxxxx
  HTTP/1.1
Host: myid.org
```

Fig. 2. An example of an OpenID Authentication Request HTTP message sent by the User Agent to its OpenID Provider (Step 5 of the OpenID Authentication process).

5) When the User Agent receives the HTTP Redirect message from the Relying Party, it starts a new connection with her OpenID Provider in order to be authenticated. How the end user is authenticated by the OpenID provider is out of the scope of the OpenID specification [4], and thus varies among the different OpenID Providers that have been tested. This enables innovation since each OpenID Provider may deploy its own (hopefully secure) authentication mechanisms, like SSL certificates. In most cases the User needs to log in, employing a username and a password, and then she is requested to confirm the authentication request from the Relying Party, including which of the requested personal information should be returned. For User convenience, all these choices for each particular Relying Party can be remembered by the OpenID Provider, thus they will not be asked again the next time. In fact, if this is the case and the OpenID Provider also employs a HTTP Cookie in order to authenticate the user subsequently, it is quite possible that the User would not even see the OpenID Provider web page because the whole process occurs automatically and transparently, which of course it is the main selling point of Single Sign-On (SSO).
6) When the OpenID Provider authenticates the User and validates the Authentication Request from the RP, it redirects the User Agent again to the endpoint specified by the the Relying Party (i.e. `openid.return_to`). The target URL must contain all the parameters of the Authentication Response (`openid.mode=id_res`), including the OpenID User Identity (`openid.identity`), a copy of the base RP endpoint (`openid.return_to`), a nonce (`openid.response_nonce`), the optional security association handle (`openid.assoc_handle`), and a cryptographic signature (`openid.sig`), along with the list of parameters that have been signed by it (`openid.signed`). Figure 3 shows a (quite simplified) example of an OpenID Authentication Request HTTP message, sent by the User Agent to the RP because of the OP redirect message.
7) Finally, after the reply from User's OpenID Provider is validated by the Relying Party, the User will log in successfully and the web session will continue as usual.

```
GET /openid-login?openid.mode=id_res
  &openid.identity=http://alice.myid.org
  &openid.return_to=http://www.example.org
  /openid-login
  &openid.response_nonce=2010-03-22T12:00
  &openid.assoc_handle=xxxxxxxxxxxxxxx
  &openid.signed=mode,identity,return_to
  &openid.sig=yyyyyyyyyyyyyyyy
  HTTP/1.1
Host: www.example.com
```

Fig. 3. An example of an OpenID Authentication Reply HTTP message sent by the User Agent to the Relying Party (Step 7 of the OpenID Authentication process).

III. A Privacy Vulnerability of OpenID

From the previous explanation it can be seen that the OpenID Authentication Protocol heavily relies on URL-parameter encoding in order to exchange information between the Relying Party and the OpenID Provider. The parameters of the Authentication Reply are signed in order to secure the authentication process and avoid tampering. However these parameters are not encrypted, which means that all these parameters, and in particular the (probably unique) User OpenID Identifier (`openid.identity`), can be seen by anyone that has access to the full URLs of the Authentication Request or the Authentication Reply messages. Here resides the Privacy Vulnerability of the OpenID Authentication Protocol that has been found by this work.

The question then is how these URLs, that in principle should be only processed by the Relying Party or the OpenID Provider, can be leaked to a third party. The answer does not lie in OpenID itself but in the underlying Hypertext Transfer Protocol (HTTP) that is employed to access web pages, and thus also to transport the URL parameter-encoded OpenID authentication messages. In particular when accessing to a resource (e.g. image, link) the HTTP message header should include a `Referer` [3]field that specifies the full URL of the web page where this resource was accessed from. The objective of this `Referer` field is to help webmasters to locate invalid links, and can be very useful for web traffic analysis in order to know how users actually browse the website or what web pages are linking to a particular page or resource.

However this means that the OpenID Authentication URLs, including the User OpenID Identity parameter, can appear inside the HTTP `Referer` field when accessing any resource linked by the web page that is generated when processing such URLs. If the OpenID Authentication URLs were always processed by a CGI or Servlet that does not return any web content but just redirects the user to other web page, or if the target web page does only include local resources, this should not be a problem. Sadly, and because nowadays most

[3]This field is misspelled in English. It should be "Referrer" but as it appears as `Referer` in the HTTP specification, implementations use it as it is.

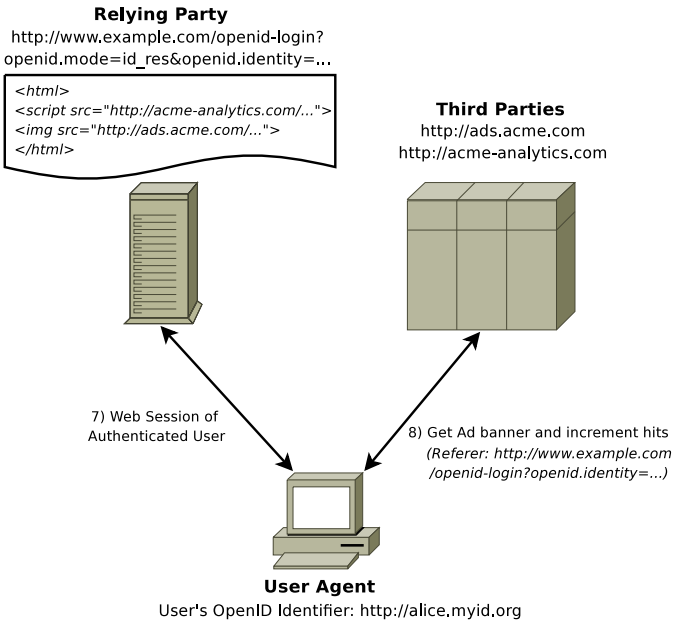

Fig. 4. User OpenID Identity leaking to third parties.

websites are funded by advertisements, it is quite common that all web pages of a site contain some kind of banner, text advertisement, or traffic analysis script from some third party.

Therefore, because of this combination of OpenID URL-parameter encoding and HTTP `Referer` field, there is a potential privacy vulnerability in all OpenID Providers and Relying Parties where the web page generated from the Authentication Request/Reply URL contains any reference to a third party resource, as depicted[4] in figure 4.

In order to study the real impact of this potential vulnerability we have analysed multiple web sites that support OpenID Identifiers (Relying Parties), as well as different OpenID Providers. We have found that this vulnerability affects all the studied Relying Parties and even one of OpenID Providers, which employed a traffic analysis script. Even worse, in many cases, the third parties where the User OpenID Identity was leaked were the same ones: a well-known Internet advertisement company and a free traffic analysis website (that even belong to the same big Internet corporation). In fact, since the traffic analysis service was based in javascript, the OpenID Authentication URL appeared twice in the HTTP requests to this third party: Once as expected in the `Referer` field, but also encoded as a parameter of the requested traffic analysis URL itself.

Therefore we consider that this privacy vulnerability is very real and even widespread, thus it could be exploited by unscrupulous Advertisement, Audience Metering or Traffic Analysis companies to track all the websites accessed by a particular OpenID User, for instance in order to create user behaviour profiles for targeted advertising.

[4]Disclaimer: All user names, DNS domains and URLs that appear in this paper are fictitious, and they do not belong to any of the RPs and OPs studied by this work.

IV. Possible Countermeasures

Unfortunately, this vulnerability is not a bug or some kind of implementation issue. All the studied User Agents, Relying Parties and OpenID Providers do implement the OpenID Authentication and HTTP Protocols correctly. In fact we have tested major browsers (Internet Explorer, Firefox, Opera and Konqueror) with the default settings, over different Operating Systems (Windows XP and Linux) and we always have been able to reproduce this vulnerability. Therefore, we consider that this vulnerability is a design problem of the OpenID Authentication Protocol because of using URL parameters to exchange private information.

Moreover, as stated in the Security Considerations section of the HTTP 1.1 specification [5] (Subsection 15.1.3 "Encoding Sensitive Information in URI's"): "*Authors of services which use the HTTP protocol SHOULD NOT use GET based forms for the submission of sensitive data, because this will cause this data to be encoded in the Request-URI. Many existing servers, proxies, and user agents will log the request URI in some place where it might be visible to third parties. Servers can use POST-based form submission instead.*"

Therefore a solution to this vulnerability must be found by:

1) Redesigning the OpenID Authentication Protocol.
2) Disabling the HTTP Referer field in User Agents.
3) Recommending Relying Parties and OpenID Providers to do not include links to third parties in the web pages processing OpenID Authentication URLs.

Clearly, the first alternative is the optimal solution of the three, since the encoding of information as URL parameters done by the OpenID Authentication Protocol is the root cause of this privacy vulnerability. There are two alternative solutions for a new OpenID Authentication Protocol:

- To avoid exchanging information between the RP and OP by means of URL parameters, but using instead POST forms where data is carried in the body of the HTTP message[5].
- To encrypt the whole OpenID Authentication URLs by leveraging the existing signing key or by generating a new encryption key during the security association phase. This way the User OpenID Identifier parameter will be meaningless, even if leaked to a third party.

Although clearly desirable, these solutions can only be applied in the long term. First, it is necessary to define a new OpenID version that deprecates URL parameters, and then wait for all OpenID Providers, and specially all existing Relying Parties, to migrate to the new protocol version. Therefore a short term solution is needed meanwhile.

The third alternative is probably the worst one because requires many of the existing websites that employ OpenID to limit where third party resources can appear, to change its implementation and even its internal structure. Although this

[5]This HTTP POST mechanism has been already defined in the version 2.0 [4] of the OpenID Authentication Protocol, but none of the analyzed RPs or OPs employ it. Probably this is because, in order to send a HTML automatically, the User Agent must support and enable javascript.

may be a plausible solution for the OpenID Providers currently affected by this vulnerability, it is clearly an unreasonable solution for all existing and future websites that just want a simple Single Sign-On (SSO) solution for their users.

Therefore the only short-term alternative seems to be the second one. Many privacy advocates have suggested in the past to get rid of the HTTP `Referer` field since it could expose the behaviour of the user to the web server. However the Referer field was finally included in the HTTP specification because it is an legitimate and useful tool for webmasters. Nevertheless some browsers are able to disable this field, by changing its default configuration (e.g. by setting `network.http.sendRefererHeader=0` in Firefox) or by enabling some kind of "Private Browsing" mode where visited web pages are not cached nor stored in the history list. However, as in the previous case, this is not a global solution as it requires all OpenID Users to change the default behaviour of their browsers.

Fortunately, there is an alternative, standard mechanism to clear the HTTP Referer field. The HTTP 1.1 specification [5] states that: "*Clients SHOULD NOT include a Referer header field in a (non-secure) HTTP request if the referring page was transferred with a secure protocol*". This statement was added to avoid any information -including URLs- that is exchanged with a secure web site, to be leaked to a different web site. The good news is that, although it is not explicitly stated, this also includes the target URL carried in a redirect message from a secure website. Therefore, if the User Agent employs HTTPS to connect to the OpenID Provider, when it is then redirected to the Relying Party, a standard-compliant User Agent should not include the OpenID Authentication Reply URL in the `Referer` field of any resource requested from that page. We have tested this behaviour in Firefox, and we have confirmed that when the OpenID Provider employs HTTPS, the User OpenID Identifier is never leaked to third parties in the `Referer` field in none of the tested Relying Parties.

It should be noted that enabling HTTPS and TLS/SSL security protocols could add some overhead to the OpenID Providers servers, and also increase the total delay of the OpenID authentication process. However we consider that adding this additional security layer is a good idea by itself, in order to protect the Users credentials and personal information stored by the OpenID Providers, even without considering the found privacy vulnerability.

Therefore we recommend all OpenID Providers to switch from HTTP to HTTPS to authenticate their users, in order to mitigate this privacy vulnerability in the short term. Nevertheless we still consider that the usage of URL-encoded parameters by the current OpenID Authentication Protocol is flawed and should be redesigned in the long term. Notice that the HTTPS solution may solve the `Referer` leakage but does not guarantee that the OpenID Authentication URLs could not appear in other places like proxies, caches, history lists, or server logs, and thus being exposed to attackers or other third parties.

V. Conclusions

OpenID is an increasingly popular Single Sing-On (SSO) mechanism that enables users to have a single identity across the whole web, thus they only need to log in once during a web browsing session, hence its name. From a security point of view, OpenID could also become a powerful tool to enhance the security of public websites, since user credentials and personal information are only stored by the OpenID Provider, which is chosen and trusted by the end user. Therefore user data is much easier to protect because OpenID Providers are able to implement advanced authentication mechanisms, such as One-Time Password Tokens, SSL certificates or Smart Cards.

However, among other privacy problems derived of using a unique user identifier, the OpenID Authentication Protocol is prone to a privacy vulnerability, where a third party like a web advertisement or audience metering agency is able to obtain the OpenID identity of a registered user, for instance in order to known all the OpenID web sites visited by each individual user. This vulnerability is caused by the usage of URL parameters in order to exchange information between the OpenID Providers and the Relying Parties. Then the (potentially unique) OpenID identifier of an user can be leaked to a third party by means of the HTTP `Referer` header. This is not a bug or an implementation problem, but a design one. Actually, after studying real Relying Parties and OpenID Providers, we have found that most of them (unless protected by HTTPS) are affected by this widespread vulnerability.

In order to mitigate this real privacy risk, this paper also studies the possible solution space and presents a number of countermeasures that could be applied. From all the proposed solutions, we consider that the OpenID Authentication Protocol should be redesigned in the long term in order to avoid exchanging URL-encoded parameters. Meanwhile, we suggest the OpenID Foundation to declare the use of HTTPS as mandatory for all OpenID Providers, as it is the simplest, easiest and proved way to avoid leaking the OpenID Authentication URLs to third parties.

As future work, we will study if additional personal information of the end user, other than the OpenID identifier, could be leaked or otherwise obtained by a third party, by means of some attack that further exploits the vulnerability presented in this paper.

Acknowledgment

The work presented in this paper has been funded by the INDECT project (Ref 218086) of the 7th EU Framework Programme.

References

[1] OpenID Foundation website. `http://openid.net/`
[2] Microsoft Passport Network. `http://www.passport.net`
[3] D. Recordon, B. Fitzpatrick. *OpenID Authentication 1.1*. May 2006.
[4] OpenID Foundation. *OpenID Authetication 2.0 - Final*. December 2007.
[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. June 1999.
[6] R. Escola. *Diffie-Hellman Key Agreement Method*. IETF RFC 2631. June 1999.