User authentication for the masses

Access Granted

Outsourcing authentication services gives you access to more services – at a price. Kurt examines the pros and cons of distributed authentication. *By Kurt Seifried*

Last month, I talked about password management for users. But what can server and system administrators really do to make life easier for users and themselves? One obvious solution to username and password proliferation is to somehow allow a single account access to multiple services. Traditionally, this process was done with federated login services, such as LDAP, Kerberos [1], and so on. This approach works well for specific organizations, but it's unlikely that sites will grant random people the ability to create and use accounts.

OpenID

OpenID was covered in an excellent article [2] back in 2008, and since then has gained widespread adoption. At this point, OpenID looks like the only real game in town for providing distributed authentication, but, since 2008, much has changed. For one thing, support is much better and is included by default in many Linux distributions, and OpenID has benefited from several years of operational use.

Advantages of Outsourcing Authentication

One of the biggest advantages of outsourcing authentication is you can pick a provider who can offer services that you cannot. Two-factor authentication, for example, requires additional servers, licensing of software, along with the purchase and shipping of hardware devices to customers. Several OpenID providers support two-factor authentication, allowing you to piggyback on their systems. With this approach, you can reduce the amount of sensitive information you have to store and manage. Also, an SQL injection attack against your authentication service is much less problematic when there are no password hashes to steal and crack offline.

Also, you can keep your own identity management and checks internally. For example, you can allow users to use an OpenID to log in and then map that to an internal account with their real name and so on. Outsourcing authentication management doesn't mean you have to give up your own internal checks and balances.

Disadvantages of Outsourcing Authentication

A major disadvantage of outsourcing authentication is that you lose control of the back end. A provider might be using a weak hash function to store passwords (e.g., MD5) or might have other problems (e.g., SQL injection vulnerabilities) within their site. One example is a classic timing attack found in one version of an OpenID library.

By using this attack, an attacker could guess passwords one letter a time, checking the first character for a to z, A to Z, etc. and then moving on to the second character. Thus, even a 20-character password would only take a few thousand guesses (rather than 100 to the power of 20 assuming a-z, A-Z, 0-9, etc.). Libraries for older protocols, such as LDAP and Kerberos, have generally corrected such problems, thus leaving fewer unknowns than newer protocols like OpenID. Another problem with external authentica-

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

tion sources is establishing a secure path from you to them. Some OpenID providers do not use HTTPS, so the initial request may be susceptible to a man-in-the-middle attack (although built-in protections can help).

This need for external authentication also means that your web servers will need to make outgoing web requests to other sites. Thus, you'll need to poke outgoing holes in your firewall, and you might want to either proxy outgoing web traffic (so you can log it) or use an IDS system to spot outgoing attacks.

What's in a Name?

Another issue with distributed authentication systems like OpenID concerns the shared namespace. For example, the *kurtseifried* at WordPress is me, but the *kurtseifried* at Yahoo! is not me. So, when you use OpenID account names, you must also use the provider portion of the name or you could end up with collisions.

Alternatively, you can use the OpenID account for authentication purposes and then map it to a local account. This approach has the added benefit of not letting an attacker insert strange characters into the username or domain part of the account name, which could cause problems for you later (e.g., SQL injection attacks or buffer overflows).

Plugging In OpenID

So, how do you actually plug OpenID into an existing web application? MediaWiki is a great example, because you often want to require people to log in before they can edit documents to minimize vandalism or simply to track who changed what. To begin, you'll need OpenID support for PHP (the language in which MediaWiki and most of its extensions are written). To do so, you can use "OpenID Enabled" [3], which you can find as a package in Debian, Fedora, and several other mainstream Linux distributions, or you can get the source and install it yourself. Note that under Fedora, the package is called `php-pear-Auth-OpenID.noarch` and was broken at the time of writing.

Figure 1: The MediaWiki OpenID login link.

Once the package is installed, you will need the MediaWiki OpenID [4] extension, which is also available as a package on Debian, Fedora (called `mediawiki-openid.noarch`, also broken as of this writing), and so on. Enabling the MediaWiki extension is trivial; just insert the following include line into your `LocalSettings.php`:

```
require_once("extensions/OpenID/
                OpenID.setup.php");
```

Then, you can create another table (using the `openid_table.sql` SQL script) in the MediaWiki database, which will be used to map OpenID accounts to local usernames. Note that not all OpenID usernames are compatible with MediaWiki's account requirements. Then you can click on the link to log in and set your local MediaWiki username.

Widespread support exists for OpenID in other programming languages; see the references at the end of this article for a list of libraries [5].

Providing OpenID Services

Using OpenID is easy enough, but what if you want to provide OpenID services so your users can take advantage of having a single account for multiple services? Several hosting providers, such as Google, now provide OpenID support for business and education users, but, sadly, not for the free version of Google Apps for domains. If you have a Yahoo! account, you can go to *http://openid.yahoo.com/* to enable your account for OpenID usage. (Your best bet is to plug your provider's name + OpenID into Google.) If you want to set up your own OpenID server, you won't be spoiled for choice [6].

OAuth and PAPE

You might not have heard of OAuth [7] or PAPE [8] yet – neither had I. OAuth provides an interface layer between OpenID and actual authentication for services. Thus, you can, for example, give access to photos hosted on a sharing site in a protected location without having to hand out your username and password to the application. Basically, OAuth provides an easy way to interact with OpenID servers that should be almost transparent to users.

PAPE is the OpenID Provider Authentication Policy Extension, which is a fancy way of saying that a site using OpenID also can set rules like "only allow OpenID accounts that have two-factor authentication using a physical token." This capability reduces the chances that a compromised account will be able to access your service. See "A Note on Password Cracking." ■■■

A NOTE ON PASSWORD CRACKING

As a matter of policy, most sites now hash passwords using a one-way function; thus, if an attacker steals the password file, he won't easily figure out the passwords.

This theory, however, has a couple of wrinkles: GPUs and cheap storage. GPUs like NVidia's Tesla (to which you can now rent access on Amazon EC2) have several hundred cores that, although not great for general-purpose computing, are ideal for highly parallelizable tasks like password cracking. If you add cheap storage to that mix, the attacker can now generate huge lists of encrypted passwords. Multiple online MD5 lookup sites [9] allow you to enter an MD5 hash, wait a few seconds, and receive an input that will create that hash.

So, when choosing a hashing algorithm, remember that older ones (like MD5) and fast ones (like SHA1) are less than ideal.

INFO

[1] "Taming the Dogs of Hell" by Walter Neu, *Linux Magazine*, Issue 96, pg 28

[2] "ID Check" by Nils Magnus, *Linux Magazine*, Issue 96, pg 35

[3] OpenID Enabled: *http://www.janrain.com/openid-enabled*

[4] MediaWiki OpenID extension: *http://www.mediawiki.org/wiki/Extension:OpenID*

[5] OpenID libraries: *http://wiki.openid.net/w/page/12995176/Libraries*

[6] Run your own identity server: *http://wiki.openid.net/w/page/12995226/Run-your-own-identity-server*

[7] OAuth: *http://oauth.net/*

[8] PAPE: *http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.html*

[9] Reverse MD5 hash lookup: *http://tools.benramsey.com/md5/*