
PAPI AND NOVELL NETWARE

Document:	PAPI AND NOVELL NETWARE
Category:	Project Deliverables (6.3a)
Version:	1.0
Author:	Ian Stuart
Date:	22 January 2003

Contents:

PAPI AND NOVELL NETWARE	1
PAPI and Novell Netware, an investigation.	2
Task:	2
Preliminary work:	2
Basic concept:	2
The response is passed to the user, including a sessional-cookie.Set up:	2
Set up:	3
Report:	3
Appendix 1: Creating your first PAPI-AS	4
Step 0: Preperation.....	4
Step 1: Build a standard apache 1.x server.....	4
Step 2: install the PAPI-AS perl packages into your Perl library tree	4
Step 3: Edit the AuthServer.cf file.....	4
Step 4: prepare the database	6
Step 5: Start the server.....	6
Enabling logging for your PAPI-AS	7
Converting your PAPI-AS to a Secure Server	7
Converting your PAPI-AS to use IMAP for authentication	7
Appendix 2: Creating a PAPI-PoA	9
Step 0: Preparation.....	9
Step 1: preping the PAPI-AS:<.dt>	9
Step 2: Create the server:	9
Step 3: Install the PAPI-PoA perl packages into your Perl library tree	9
Step 4: Enable the basic PAPI-Poa control programs.....	9
Appendix 3: Getting the PAPI-PoA to do Authorisation	12

PAPI and Novell Netware, an investigation.

Task:

I was asked to look at PAPI, specifically in relation to using LDAP for Authentication, and Novell Netware servers as an LDAP source.

Preliminary work:

PAPI (Point of Access to Providers of Information: <http://www.rediris.es/app/papi/index.en.html>) is a concept for separating the authentication of a user from the actual information resource.

The main work has been done by a group at Red Iris, in Spain.

Basic concept:

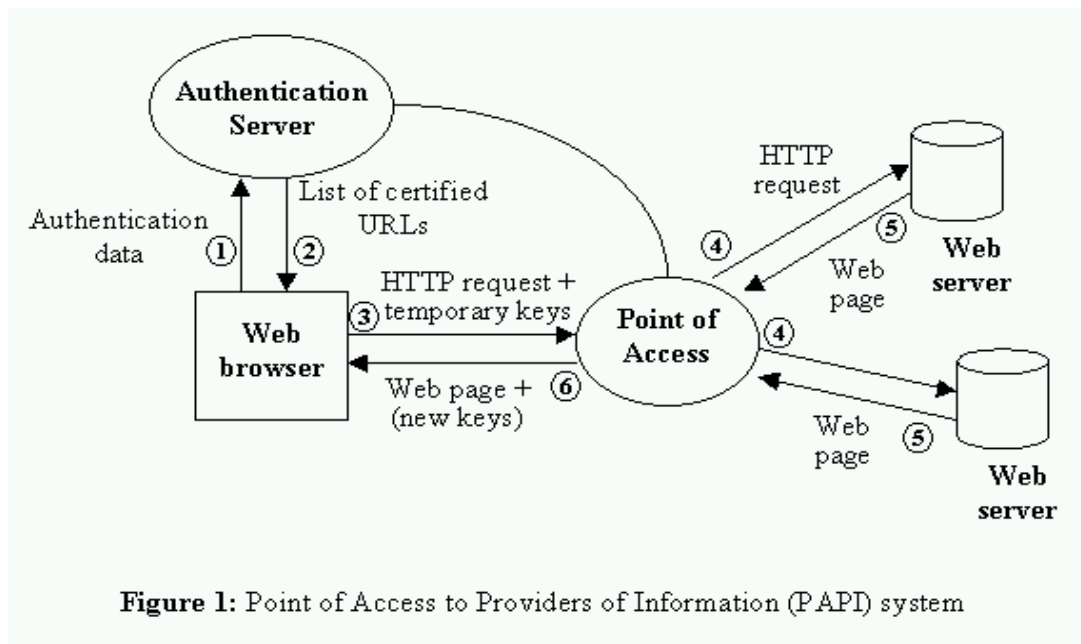


Figure 1: Point of Access to Providers of Information (PAPI) system

The basic concept of how PAPI works is shown in this diagram:

- 1) The user fills in a form, and the details are passed to an Authentication server.
- 2) The Authentication server returns a list of URLs that the user is, in theory, allowed access to (possibly with an Authentication flag)
- 3) The user selects one URL, which passes the ID of the Authentication server, the service requested, and a temporary key to the Point-of-Access server.
- 4) The Point-of-access server compares the identifying data from the href, and passes the request to a web server (either local or proxied)
- 5) ... and the response is read.

The response is passed to the user, including a sessional-cookie.

Set up:

I had a think about the academic environment, and decided that I would set up something that would reflect the Real World environment of distributed (institution-based) Authentication and centralised (data-centre-based) information points. This dictated creating seperate Authentication and Point-of-access servers.

Setting up an Authentication Server is fairly easy (see Appendix 1) and this can simply authenticate a user, and pass them to an open web-server.

Setting up a Point-of-Access server is slightly trickier (see [Appendix 2](#)), however it's all fairly straight forward.

Finally, getting the Point-of-Access to authorize the Authentication server's requests (see [Appendix 3](#)).

Report:

So, how easy is it to achieve the stated objective: To use a Novell Netware server to provide LDAP authentication for user authentication using PAPI.?

It does not seem to be possible, however the difficulty lies with the Netware Server, not the PAPI system.

PAPI works fine with LDAP (I tested it using an openLDAP server on a Linux workstation)

With LDAP, PAPI looks up an attribute called 'papiSiteId' (or 'papiGroupId' if that fails) in the user record. It does not seem possible to extend the user object in Netware NDS to include this attribute (even though the PAPI ObjectClasses and Attributes can be created in the NDS).

The closest I was able to do was to do a double-lookup: using the Netware server to do Authentication, and then a seperate (openLDAP) server to access the PAPI details. This is not a satisfactory solution as it requires two datasets to be maintained, and kept in sync.

Appendix 1: Creating your first PAPI-AS

Step 0: Preperation.

Ensure that you have the basic requirements for building the Authentication Server:

- ◆ Perl >= 5.004
- ◆ Openssl >= 0.9.5
- ◆ The following Perl packages: DB_File; MIME::Base64; URI::Escape; Convert::ASN1
- ◆ Plus, specifically for the PAPI-AS: CGI_Lite; Sys::Syslog (if Unix syslog-based logging is required); Net::LDAP (if LDAP-based authentication is required); Net::POP3 (if POP3-based authentication is required); Net::IMAP::Simple (if IMAP-based authentication is required).
- ◆ The PAPI-AS tarball.

Step 1: Build a standard [apache 1.x server](#)

Step 2: install the PAPI-AS perl packages into your Perl library tree

- 1 Get the code (ie, untar the tarball).
- 2 Configure: perl Makefile.PL
- 3 Supply the filesystem directory for the cgi-scripts of the server
- 4 Supply the filesystem directory for the OpenSSL header files (ie /usr/local/include/openssl)
- 5 run make and make install
- 6 Go and check that the Perl packages are in your Perl library tree (they weren't in my [admitably unusual] installation).
- 7 Go and check that the AuthServer script and associated packages are in the CGI directory, and that the config stuff is in the appropriate working directory (see [below](#)). Again, they weren't - for my installation.

Step 3: Edit the AuthServer.cf file

uncomment the authentication type (basic for these tests)

```
my $authType = "basic";
```

You need define the following...

```
$$cfg{workingDirectory} = '/foo/bar';  
$$cfg{asLocation} = 'http://lucas.ucs.ed.ac.uk:123/cgi-bin/AuthServer';  
$$cfg{serverID} = 'foo-as';
```

```
 #(for basic authentication...)
```

```
$$cfg{basicAuthDB} = 'test.database';

# Default values for the PoA(s)
#
$$cfg{defTimeToLive} = 1800;
$$cfg{defLocation} = '/';
$$cfg{defService} = 'bar-poa';
$$cfg{defPoA} = 'http://lucas.ucs.ed.ac.uk:456/';
$$cfg{defDescription} = 'Test Lucas PAPI PoA';
```

You can also comment-out the following:

```
# Comment these if you do not require split (thus SSL-capable) mode
#
##$cfg{splitModeURL} = 'http://as.papi.dom.ain/';
##$cfg{splitModeParamList} = 'username';
```

At this point, we are not using any keys or cookies, so the following three lines can be ignored:

```
$$cfg{defAuthURI} = '/papi/cookie_handler.cgi';

$$cfg{privateKey} = 'privatekey.pem';
$$cfg{publicKey} = 'publickey.pem';
```

Things to note in the AuthServer.cf file:

The value in `$$cfg{basicAuthDB}` must match the name of the database ([see below](#)) created with the `pdimport` command.
The value in `$$cfg{serverID}` must match the site ID in the database source file.

Additionally, the AuthServer.cf file can contain a default PAPI-PoA, assuming no additional servers are defined via the Authentication process.

At this point, we only need to det the default locations stuff, the *Authorisation* bits are not used.

The value in `$$cfg{defPoA}` is the URL of the default PAPI-PoA server to use. Set to a known working web site.
The value in `$$cfg{defService}` is name of the default service to use within the PAPI-PoA
The value in `$$cfg{defDescription}` is the long text that the PAPI-AS displays for the href for the service.
The value in `$$cfg{defLocation}` is the location on the to request in the href.

Step 4: prepare the database

Sample Basic Authentication source file

```
# Users (group GROUP1)
# <class>::<id>::<crypt'd_passwd>::<alt_name>::<grp_ids>::<list_of_site_ids>
user::user1::asdjkfhkdfhasd::Fred1::group1::Lucas.ucs
#

# Groups
# <class>::<id>::<alt_name>::<list_of_site_ids>
group::group1::Group One::Lucas.ucs

# Sites
#
# <class>::<id>::<description>::<PoA URL>::<PoA token request
URI>::TimeToLive::<service_id>::<PoA URI>
site::Lucas.ucs::main Lucas web
server::http://lucas.ucs.ed.ac.uk::/token_request::1800::lucas-poa::/
```

(Note: asdkjfhkdfhasd is the crypted password for the user. You can generate the by using the Apache htpasswd cammand - ie: `htpasswd -nb user1 password_text`) create the database from the source file: `...../pdimport -n -s source.file test.database`.

Things to note in the database source file:

- ◆ The `group_ids` listed for a user must match the ids for a group. This information can be missed out, and the user will be allowed access to only the site(s) listed for that user.
- ◆ The `site_ids` listed for each user/group must match the ids for a site. The `site_ids` for a user can be missed out, and the ids will be taken from the group data.
- ◆ The PoA URL is the URL the Authentication server communicates with, and the server the client is directed to.
- ◆ The PoA token request URI is the "page" at the PoA URL that the PAPI-AS calls to get a cookie
- ◆ The `service_id` is the name of the name of the server in the PAPI-PoA server configuration file.
- ◆ The PoA URI is the "page" that the client is directed to if they select the server.

For this demo, make the PoA URL any server, the PoA token request URI something that won't resolve, and the PoA URI the root of the server.

Step 5: Start the server

This is the standard Apache server starting command...

Assuming the server starts up, run the CGI script (as defined in the `$$cfg{asLocation}` variable). If you have set it up right, you should get a login page.

Try logging in. You should get a page that gives you two hrefs: one to the PoA server defined in the database file, and one to the default PoA as defined in the AuthServer.cf file. Try selecting one.. it should jump to the appropriate server...

Enabling logging for your PAPI-AS

It is, of course, wise to track who is being authenticated, so the Authentication Server can log who has been Authenticate (and who has not been Authenticated).

To enable logging, edit the AuthServer.cf file, and make the following changes:

Add in the logging package

```
use PAPI::BasicLog;
```

... and define the logging details in the configuration hash:

(an option for logging..)

```
$$cfg{logHook} = \&PAPI::BasicLog::FileLog;
```

```
$$cfg{logFile} = 'queries_log';
```

Converting your PAPI-AS to a Secure Server

If you are using any form of password that is not generated **by you** specifically **for this service alone**, you should rebuild your underlying PAPI-AS with an https server. See

<http://www.apache-ssl.org/>.

Remember, however, that you will need to edit the AuthServer.cf file to reflect the changed URL for the AS:

```
$$cfg{asLocation} = 'https://lucas.ucs.ed.ac.uk:123/cgi-bin/AuthServer';
```

You will need to set up the public and private keys for the Authentication Server to use with the PoA servers.

NOTE: These are **not** the same as the certificates that are used for the *https* server.

Both keys should be created in the working directory.

For details, see step one in the [create a PAPI-PoA](#) file

Converting your PAPI-AS to use IMAP for authentication

To move away from the basic authentication, and to test remote authentication, I set up IMAP authentication.

Alter the AuthServer.cf file:

Add in the PAPI::IMAPAuth package

Change the \$authType to be imap

Ensure that the \$\$cfg{IMAPserver} is set to a valid IMAP mail server

Re-running the AuthServer script should now only list a single available PoA - the default one defined in the AuthServer.cf file.

This is because you are no longer getting the additional details from the database file. This is the downside to both POP & IMAP authentication.

Appendix 2: Creating a PAPI-PoA

To get any functionality about accessing services, you now need to have a Point of Access server running. This is more complicated, and (once built) requires tweaks to the PAPI-AS to get the two servers to talk to each other.

Step 0: Preparation.

Ensure that you have the basic requirements for building the Authentication Server:

- ◆ Perl >= 5.004
- ◆ Openssl >= 0.9.5
- ◆ The following Perl packages: DB_File; MIME::Base64; URI::Escape; Convert::ASN1
- ◆ Plus, specifically for the PAPI-PoA: Apache >= 1.3.8; mod_perl >= 1.20; Data::Dumper; HTML::TokeParser; HTTP::Cookies; LWP::UserAgent; MLDBM; and MLDBM::Sync

Step 1: preping the PAPI-AS:<.dt>

Generate a public-/private-key pair for the PAPI-AS
Move to the PAPI-AS working directory
Create a private key: openssl genrsa 1024 > bar.pem
Generate the public key from the private key: openssl rsa -in bar.pem -pubout -out foo.pem
Edit the *AuthServer.cf* file to reflect the new keys

Step 2: Create the server:

Build a standard [mod_perl-enabled apache 1.x server](#)

Step 3: Install the PAPI-PoA perl packages into your Perl library tree

- 1 Get the code (ie, untar the tarball).
 - 2 Configure: perl Makefile.PL
 - ◆ Supply the filesystem directory for the OpenSSL header files (ie /usr/local/include/openssl)
 - 3 run make and make install
- Go and check that the Perl packages are in your Perl library tree (No problem for me).

Step 4: Enable the basic PAPI-Poa control programs

- Enable the main Perl module: PerlModule PAPI::Conf.
- ◆ This needs to be fairly early in the .conf file, somewhere before the any aliases...

Configure the PAPI_Main block (see <http://www.rediris.es/app/papi/dist/Conf.html> and possibly <http://www.rediris.es/app/papi/dist/Main.html> for further details)

```
<PAPI_Main>
HKEY_File /some/path/hkey
LKEY_File /some/path/lkey
Hcook_DB /some/path/hcookdb
Server bar-poa
PAPI_AS foo-as The PAPI AS at Dom.ain
Pubkeys_Path /some/path/KEYS
Lcook_Timeout 1800
CRC_Timeout 1800
URL_Timeout 1800
Debug 1
Domain dom.ain.com
Accept_File /some/other/path/image1
Reject_File /some/other/path/image2
Auth_Location /papi/cookie_Handler.cgi
</PAPI_Main>
```

Most of the lines here require clarification:

- ◆ The HKEY_File and LKEY_File files are md5sum strings, and can be generated with something akin to: `md5sum < a_file > key_file_name` (Obviously you should use two different files for the two keys, or one file that has rapidly changing contents - so that the two keys are different)
- ◆ The Hcook_DB is the name of a DBM database that the PoA server records information about the tokens that it grants. This file needs to be readable **and** writeable by the web server, so you may need to alter the permissions on the appropriate directory to allow this.
- ◆ The Pubkeys_Path is a directory that the PoA looks in to get the public keys for the PAPI-AS'.
You need to copy the public key from your PAPI-AS into this directory, and rename it.
The format for the filenames is as follows:
PAPI_AS *server name*_pubkey.pem, and must be in .pem format, (ie, foo-as_pubkey.pem, based on the example details shown above - the _pubkey.pem fragment of the name is fixed, and is irrespective of the file name used by the PAPI-AS)
- ◆ The server is the name of the server, and must be the same as the name used by PAPI-AS' configuration (either `$$cfg{defService}` value in the AuthServer.cf file, the site <id> string in the basic Authorisation file, or the papiSiteId value in the LDAP server)
- ◆ The Auth_Location value must be ASCII-equal to the `$$cfg{defAuthURI}` setting, and must also be ASCII-equal to any aliases set in the 'PoA server's .conf file.
- ◆ The PAPI_AS can exist multiple times, and each one is the `$$cfg{serverID}` name (as defined in the AuthServer.cf of a PAPI-AS that

the PAPI-PoA will recognise.)

The format is serverID, space, Descriptive name of server, end of line.

- ◆ The domain is the domain of the server, and the domain that will be used in the cookies.
- ◆ The various `_Timeout` figures are times, in seconds.

Reconfigure the PAPI-AS to load the homepage of the new server (the `$$cfg{def****}` stuff.)

Now start the PAPI-PoA server, and try to connect to it. As we have set up no *Authorisation* requirements, there should be no problems.

Return to the *Authentication* server, and log in. The link should now be shown with an icon next to it (as defined by the `$$cfg{acceptURL}` and `$$cfg{rejectURL}` values, as defined in the `AuthServer.cf` file). If you use the link, and you should again go to the PoA server.

Appendix 3: Getting the PAPI-PoA to do Authorisation

- ◆ Edit the *Authentication* server's (the PAPI-AS) AuthServer.cf file to change the default service details.

Change the `$$cfg{defLocation}` to some subdirectory (I used `/manual`¹, as it's part of the standard distribution).

- ◆ Edit the *Authorisation* server's (the PAPI-PoA) `httpd****.conf` file:
- ◆ Create an alias to access the PAPI-PoA keys:
this is needed for access to the keys.

```
alias /papi /home/cpan/httpd/share/http_papi.poa/PAPI.working
<Location /papi >
    PerlSendHeader On
    PerlAccessHandler PAPI::Main
</Location>
```

- ◆ Define the restricted location:

```
<Location /manual >
    PerlSendHeader On
    PerlAccessHandler PAPI::Main
    <PAPI_Local>
        Server local
        PAPI_Filter default
    </PAPI_Local>
</Location>
```

You may need to precede this with:

```
Alias /manual "/path/to/pages/manual"1
```

```
<Directory "/path/to/pages/manual">
    YADA, Yada, yada
</Directory>
```

- 1 Remember that Apache treats aliases oddly: If you define the alias *without* the trailing slash then apache will automatically supply it after the redirection, whereas defining the alias *with* the slash will require the slash to be present to "activate" the redirection. Personally, I think that the trailing slash should be left of directories that customers may visit, but present for ones they shouldn't. Don't ask me why, it's just something I've evolved as a semi-standard.
- 2 Restart the PAPI-PoA server, and log in via the PAPI-AS. Again, the link should be displayed, along with the *Authorised* icon. Selecting the link should now take you to the appropriate page (the Apache 1.3 documentation, in this example).

- 3 Either start up a different browser (or wait the required time for the local tokens to expire), and try accessing the "restricted" site directly. You should get a "Forbidden" error. This shows that the PAPI-PoA is denying access to *unauthorised* access.