PHPPOA DOCUMENTATION

PAPI Point of Access in PHP

Index

1. Description

PAPI is a system for providing access control to restricted information resources across the Internet. The system consists of two independent elements:

a) Authentication server (AS)

The purpose of the AS is to provide users with a single authentication point and making available to them (in a completely transparent way) all the temporary keys that will let them access the services they are authorized to.

b) Point of access (PoA).

The PoA manages actual access control to a set of web locations for a given organization. PoAs can be hierarchically combined into groups controlled by a Group-wide PoA (a GpoA), that is to say, a trusted source within the group for accessing user data without requiring them directly from ASes. PAPI is programed in Perl and its configuration is integrated in the Apache configuration file. More information about PAPI in http://papi.rediris.es

phpPoA implements the main features of a PAPI PoA using PHP. Therefore, it is not necessary to include its configuration into the web server configuration file: you only have to call PoA methods from any web page you want to be protected. At the beginning of the web page you have to include PHP embedded code to call PoA methods that manage the authentication. It is very important to include this code before everything in the web page to ensure that nothing will be shown before proper authorization. For example:

```
<?php

include 'PoA.php';

$poa = new autoPoA('admin');
$userData = $poa->check_Access();

?>
<html>
        //Web page.
</html>
```

The access control configuration for every location in the web site protected by phpPoA is defined in the phpPoA.ini configuration file. In this file you can set filters, key paths, error web pages, error log file, etc. Since phpPoA implements a simple PoA, you have to set the GpoA or AS as well.

phpPoA has two operation modes:

a) Automatic redirection (autoPoA class)

In this mode the PoA redirects automatically the user to an error page when the authorization fails or when there is some system error. This functionality is implemented in autoPoA class.

b) Simple authentication (PoA class)

In this mode the protected web page has the control and decide what to do when the PoA returns the validation. If the user is authorized in whatever mode, the phpPoA returns a positive code to the web page. This is implemented by PoA class.

This component provides a very easy way to protect a web site with a minimum configuration and installation. By simply including three lines of code in any web page (and building the configuration file), we can be sure that no unauthorized user will access those web pages.

2. Requirements

In order to protect our resources in a web site, we only need a web server with PHP support. The PHP module must be compiled with several security options.

phpPoA has been designed without using any web server specific function, therefore you can use it with any web server, but as most PHP systems run over Apache, we provide information related to Apache.

1. **Web server (tested in Apache 2.0.54)**

Download Apache from your repository and install it.

```
$ ./configure –prefix=apache_directory --enable-module=so
$ make
$ make install
```

Apache will be installed in the apache_directory, it could be /usr/local/apache2 for example. Apache modules will be able to be loaded dinamically thanks to he enable-module=so option.

2. **PHP as Apache module compiled with OpenSSL, Mcrypt and DBA:**

phpPoA is specifically coded for PHP > 5.2.

a) **OpenSSL is** necessary for RSA encryption

b) **Mcrypt** is used for AES functions (“Rindjael”). You can download it from http://mcrypt.sourceforge.net/

c) phpPoA stores original requests in a DB file. You have to compile PHP with **DBA support**. php-PoA has been tested with gdbm and db4. You can download it from http://www.gnu.org/software/gdbm/gdbm.html (Mirror in ftp://ftp.rediris.es/pub/gnu/gnu/gdbm/)

Install php with the following options:

```
$ ./configure -prefix=/usr/local/php   --with-apxs=/usr/local/apache/bin/apxs
 --with-openssl --enable-dba --with-gdbm --with-db4 --with-mcrypt
$ make
$ make install
```

It isn't necessary to recompile Apache when you install PHP but don’t forget to add the PHP module in the Apache config file, httpd.conf like follows:

```
LoadModule php5_module        libexec/libphp5.so
AddType application/x-httpd-php .php .php3 .php4 .phtml
AddType application/x-httpd-php-source .phps

<IfModule mod_dir.c>
    DirectoryIndex index.html index.php
</IfModule>
```

Plus, we have to move the php.ini file where the php module is expecting to find it:

```
$cp  /etc/php5.0/apache/php.ini      /usr/local/php/lib/
```

If you don't know were it should be, execute phpinfo() in your web server and check the value of Configuration File (php.ini) Path.

3. Installation

The phpPoA is distributed in a small tarball and under the GPL License. You only have to move the files to your usual location on your server and set their paths in the phpPoA.ini configuration file. These are the files included in the distribution:

FILE	DESCRIPTION	EXAMPLE PATH
sample_auto.phtml	Example of a protected page with automatic redirection mode (using autoPoA class)	/usr/local/apache2/htdocs/
sample_noauto.phtml	Example of a protected page with simple authentication mode (using PoA class)	/usr/local/apache2/htdocs/
SystemError.html	Example of a system error page	/usr/local/apache2/htdocs/PoA/
CookieError.html	Example of a cookie error page	/usr/local/apache2/htdocs/PoA/
NotAuthorized.html	Example of a not-authorized-user error page	/usr/local/apache2/htdocs/PoA/
ConfigError.html	Example of a configuration error page	/usr/local/apache2/htdocs/PoA/
DB directory	Directory to store the DB file. The uid that the web server is running under must have write permission in it	/usr/local/papi/etc/DB/
PoA.php	PoA class definition	/usr/local/papi/lib/
crypt.php	Encryption/decryption library	/usr/local/papi/lib/
lkey	PoA symmetric key	/usr/local/papi/etc/KEYS/Lcook/PoA/
_GPoA_pubkey.pem	GpoA public key	/usr/local/papi/etc/KEYS/
Logs directory	Directory to store the error log file. The uid that the web server is running under must have write permission in it	/usr/local/papi/logs/
phpPoA.ini	Configuration file	/usr/local/papi/etc/
README	Brief description about the tarball contents	
LICENSE	File containing the GPL license	

3.1 ***Instalation steps.***

1) Extract the files from the tarball

```
$tar xvf phpPoA.1.2.tar.gz
```

2) Copy the files to your file system at the suitable paths (keep in mind you have to set these paths in phpPoA.ini)

3) Change permissions to DB and logs directory in order to the web server could create files inside.

4) Update phpPoA.ini accordingly.

5) PHP must know the path to the PoA code (class PoA in PoA.php, library crypt.php and phpPoA.ini), therefore you have to add the path(s) to these files in the include_path variable inside php.ini. Therefore, add this directive in php.ini:

include_path = ".:/usr/local/php/lib/php:PoA_directory"

where PoA_directory is the directory where PoA.php and crypt.php are in your System.

In order to PHP can find the phpPoA.ini you have to include a new parameter in the php.ini called phpPoA_ini_file with the absolute path to the configuration file. Add the following line in the php.ini file:

phpPoA_ini_file = "/usr/local/php/lib/php/phpPoA.ini"

Take into acount that if PHP is using any module that overloads string functions (str*), then the openssl functions may return errors when encrypting and decrypting data. This is the case of the module mbstring, when it is configured to overload string single byte functions.

mbstring.func_overload = 2

4. Configuration

phpPoA.ini is the configuration file (in PHP ini format), composed by sections and variables. There is a main section called PAPI_Main (whose name must not be changed) and several local sections. The PAPI_Main section sets general rules for all the locations, and the local sections set specific rules for each section. For example:

```
[PAPI_Main]
Not_Auth_Error_File = /PoA/NotAuthorized.html
Cookie_Error_File = /PoA/CookieError.html
System_Error_File = /PoA/SystemError.html
    //Other variables for the general section.
[poA_directory]
Location = /admin
LKEY_File = /usr/local/papi/etc/KEYS/Lcook/PoA/lkey
//Other variables of the local section .
```

These are the variables you can use in each section:

VARIABLE	DESCRIPTION	EXAMPLE
Location	Specific path in the web site for what the policy is defined. This is the location included in the PAPI tokens. Only used in local sections, not in the main section.	/samples
Cookie_Domain	Domain included in the PAPI tokens	poa.dom.ain
LKEY_File	Absolute path to the file containing the PoA symmetric key	/usr/local/papi/etc/KEYS/Lcook/PoA/lkey
GPoA_Pub_Key	Absolute path to the file containing the GpoA public key	/usr/local/papi/etc/KEYS/_GPoA_pubkey.pem
GPoA_URL	Complete URL for the GpoA	http://gpoa.dom.ain/papiGPoA/papiPoA
AS_Pub_Key	Absolute path to the file containing the AS public key	/usr/local/papi/etc/KEYS/MyAS_pubkey.pem
AS_URL	Complete URL for the AS	http://as.dom.ain/papiAuthServer
PAPI_Filter_accept	Regular expression defining the contents of PAPI assertions that will lead to positive authorization. The syntax used is PCRE (The same as Perl 5 with just a few differences)	".*?staff.*?uid=david"
PAPI_Filter_reject	Regular expression defining the contents of PAPI assertions that will lead to negative authorization. The syntax used is PCRE (The same as	".*"

VARIABLE	DESCRIPTION	EXAMPLE
	Perl 5 with just a few differences)	

Pass_Pattern	Regular expresions defining patterns to let direct access to those URLs that matches with them, without previous authentication, It can use Get or Post methods.
	
	Several Pass_Pattern can be defined if they are separated one from each other with an "|".
	
	The syntax used is PCRE (The same as Perl 5 with just a few differences)
		//Two pass patterns
		
		"checkid_sepup=true|staff"
		
		//One pass patter
		
		"staff"

Lcook_Timeout	Time period (in seconds) for which the PAPI tokens are valid	86400
DB_Type	Database type	db4
Request_DB	Database file	/usr/local/papi/etc/DB/request_db.db4
Not_Auth_Error_File	Absolute URL to the error web page sent when the user is not authorized. Only used in automatic redirection mode.	http://poa.dom.ain/PoA/NotAuthorized.html
Cookie_Error_File	Absolute URL to the error web page sent when the cookie has not been correctly updated. Only used in automatic redirection mode.	http://poa.dom.ain/PoA/CookieError.html
System_Error_File	Absolute URL to the error web page sent when a system error is detected. Only used in automatic redirection mode.	http://poa.dom.ain/PoA/SystemError.html
Config_Error_File	Absolute URL to the error web page sent when the variables GpoA_URL and AS_URL in phpPoA.ini aren't set up correctly.	http://poa.dom.ain/PoA/ConfigError.html
Log	Log file	/usr/local/papi/logs/phpPoA.log
Allow_From	A single or an array of IP addresses. The source IP address of the client will be matched against the address or addresses here and if match is positive, then any further authentication will be skipped for that client. IP addresses representing a network class are supported, altough network masks aren't.	127.0.0.0 10.0.1.1
Deny_From	A single or an array of IP	10.0.0.1

VARIABLE	DESCRIPTION	EXAMPLE
	addresses.The source IP address of the client will be matched against the address or addresses here and if match is positive, then the client will be denied. IP addresses representing a network class are supported, altough network masks aren't. Allow_From directive has precedence over Deny_From.	127.0.0.2 192.168.0.0

When configuring the phpPoA.ini file, you have to take into account that for each local section, the value of the variables that will be consider will be those defined in that section. Just in case, there are some variables that are not set up here, then they will take, if define, the values that were set up in the global section (PAPI_Main).

When you configure the parameters in the phpPoA.ini file, you can choose between contacting an AS directly or doing it through a GpoA. Either way, you have to configure one, and only one, of the variables AS_URL or GPoA_URL with it's corresponding public key. If both or none of them are set up, then an error is raised. Be aware that you aren't in the case of having an AS_URL defined in the global section and a GpoA_URL defined in one of the local section (or vice versa).

5. Running

When the PoA has been installed and configured, it is time to test it. You can use the sample pages in the distribution. The first time any protected page is accessed, the PoA should ask for user data (inside a PAPI assertion) to the GPoA /AS you have configured in phpPoA.ini (GpoA_URL, AS_URL). The GPoA can redirect you to an Authentication Server in order to authenticate yourself (or you can contact directly with the AS). If your profile is allowed to access this location you can view the web page content.

Be careful with the filters in phpPoA.ini. We should remember that the parameters in local section of phpPoA_ini_file have priority over parameters in the PAPI_Main section with the same name and the filters too. These are the rules about the filters:

- If you leave the filters empty, by default the PoA lets access to everybody (accept filter = ".*" by default).
- If the reject filter is empty the PoA will not reject anybody and only the accept filter is checked.
- If the accept filter is empty the PoA only checks the reject filter (accept filter = ".*" by default).
- The PoA checks the accept filter and then the reject filter. If the assertion matches with the accept one, the PoA will not check the reject.

On the other hand, if the Pass_Pattern variable isn't set up, it won't let access to anyone without previous Authentication.

The interface of phpPoA to a user application is implemented by means of the method check_Access(), that returns an associative array with the following fixed keys:

- **PAPIAuthValue**, that will be
  - 2 if the request matches any pass pattern.
  - 1 if the request is authorized,
  - 0 if the authorization is denied but user attributes are available
  - -1 in the case of an error.
- **PAPIASName**, that will hold the PAPI Authentication Server identifier that issued the original assertion. If the request matches any pass pattern this field will be absent.
- **PAPIAssertion**, that will contain the whole assertion as received by phpPoA. If the request matches any pass pattern this field will be absent.
- **PAPIPassPattern,** if the request matches any pass-through pattern then it will contain the information related to the pattern, if not, it will be absent.

Apart from these, the array will contain a key-value pair per each attribute contained in the received assertion. PhpPoA assumes that assertions are formatted using "," as separators between attributes and "=" as separators between attribute names and values. As an example, an assertion of the format:

uid=myUserID,group=myGroupID,role=admin

Issued by AS myAuthNServer will produce the following key-value pairs as the return of check_Access(), once the request is authorized:

- PAPIAuthValue => 1
- PAPIASName => myAuthNServer
- PAPIAssertion => uid=myUID,group=myGID,role=admin@myAuthNServer
- uid => myUserID
- group => myGroupID
- role => admin

If the authorization fails the results would be the same, but with PAPIAuthValue => 0.

If an error is received from the GpoA, or an error occurs in processing its response, only the value associated with PAPIAuthValue (that will hold -1) is significant.

5.1 Authorization Process

When an error occurs, the autoPoA class and the PoA class have different behaviours:

Automatic redirection (autoPoA class)

1. If the GPoA_URL and AS_URL variables are incorrect --> Redirect user to Config_Error_File page
2. If the GPoA/AS Response isn't valid --> Redirect user to Not_Auth_Error_File page
3. If the assertion does not match filters --> Redirect user to Not_Auth_Error_File page
4. If the Lcook cookie has any other error --> Redirect user to Cookie_Error_File page.

Simple authentication Automatic (PoA class)

1. If the GPoA_URL and AS_URL variables are incorrect --> Redirect usert to Config_Error_File page
2. If the GPoA/AS Response isn't valid --> Return code 0 and user data
3. If the assertion does not match filters --> Return code 0 and user data
4. If the Lcook cookie has any other error --> Return code 0 and user data

What follows is a brief description of the mechanisms used by phpPoA in order to authorize users in the two operational modes.

In simple authentication mode (using the class PoA):

1) User tries to access the web page
2) The web page calls check_Access() method from PoA object to check authorization
3) If the Pass_Pattern is set up, and it matches with the URL. -> Return code 2 and NO user data.
4) The PoA checks the Lcook cookie sent by the browser:
   a) If there is no cookie:
      - Save request data
      - Ask for assertion to its GPoA. In this step the GPoA can redirect the user to a PAPI Authentication Server in order to sign in (the PoA grants the control to GPoA which will load later the protected web page).
      - If the GPoA has sent a valid assertion and it matches the filters -> Generate Lcook, and return code 1 and user data
      - If the GpPoA has sent a valid assertion an it does not match the filters -> Return code 0 and user data
      - Otherwise -> Return code -1

   a) If there is cookie -> Check if cookie matches with the request data stored in DB (timestamp, location, filters, etc.):
      - If the cookie is expired -> Return code 2 and user data
      - If the assertion does not match filters -> Return code 0 and user data
      - If the cookie has any other error -> Return code -1
      - Otherwise (cookie is valid):
        - Update Lcook
        - Return code 1 and user data

In automatic redirection mode (using the class autoPoA):

1. User tries to access the web page
2. The web page calls check_Access() method from autoPoA object to check authorization
3. If the Pass_Pattern is set up, and it matches with the URL. -> Return code 1 and NO user data.
4. The PoA checks the Lcook cookie sent by the browser:
   a) If there is no cookie:
      - Save request data
      - Ask for assertion to its GPoA. In this step the GPoA can redirect the user to a PAPI Authentication Server in order to sign in (the PoA grants the control to GPoA which will load later the protected web page).

- If the GPoA has sent a valid assertion and it matches the filters:
  - ✗ Generate Lcook
  - ✗ Return code 1 and user data
- Otherwise -> Redirect user to Not_Auth_Error_File page

a) If there is cookie -> Check if cookie matches with the request data stored in DB (timestamp, location, filters, etc.):

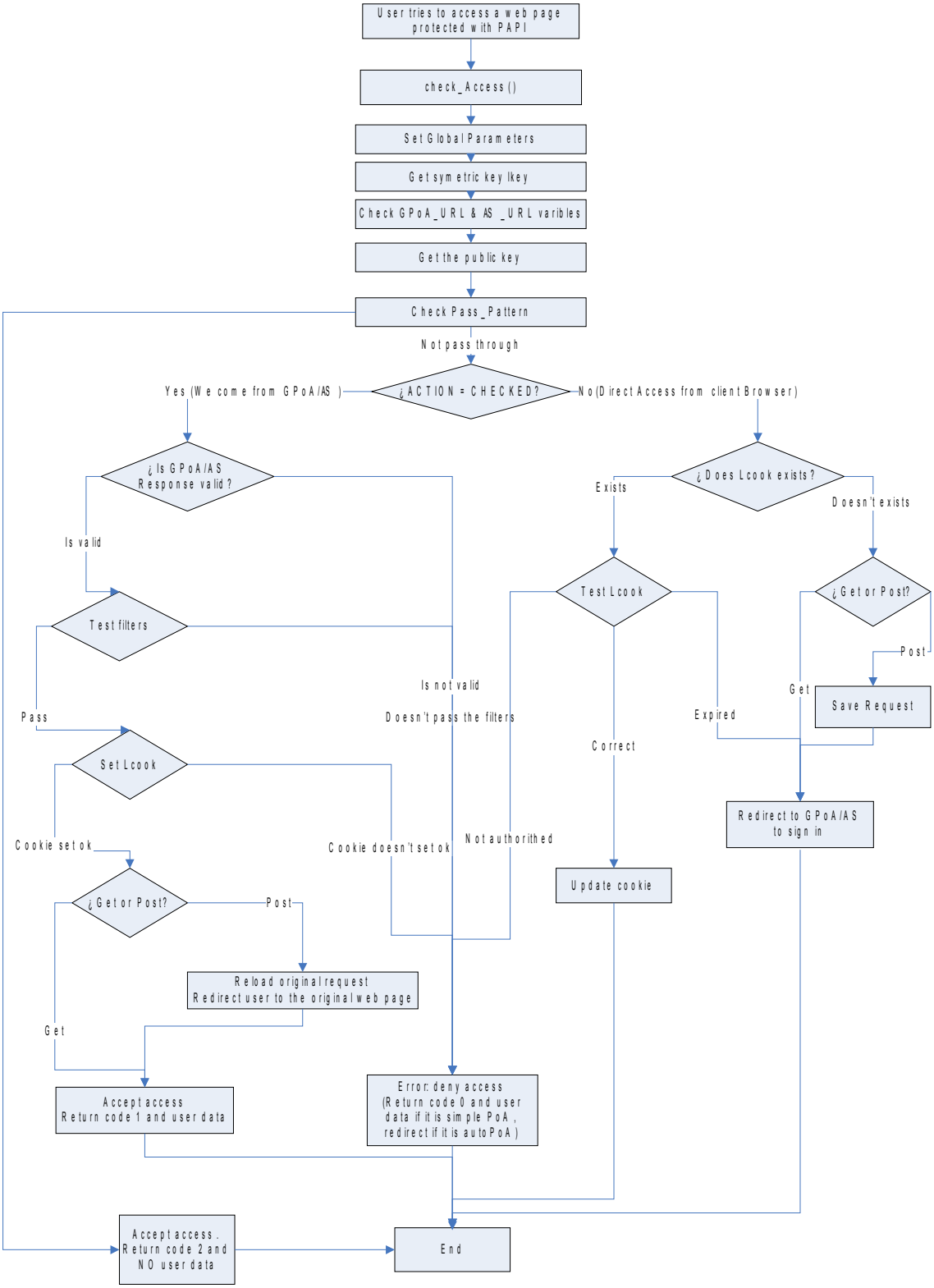
- If the cookie is expired -> Ask GPoA to re-issue assertion
- If the assertion does not match filters -> Redirect user to Not_Auth_Error_File page
- If the cookie has any other error -> Redirect user to Cookie_Error_File page.
- Otherwise (cookie is valid):
  - ✗ Update Lcook
  - ✗ Return code 1 and user data

5.2 PHP Interface

You must bear in mind that phpPoA may return to the application after having redirected the user's browser to a GPoA and got back the GPoA response. To preserve the original request alog these redirections, phpPoA stores the initially received data and restores it once the reply from the GPoA is received.

Up to version 1.9, including it, the original request data is available to the application through the $_REQUEST global variable, nothing else is restored. Nowadays, $_GET, $_POST, $_REQUEST, $_SERVER["QUERY_STRING"] and $_SERVER["REQUEST_METHOD"] are restored.

Up to our knowledge, the entire PHP environment is restablished to match the one at the original request.


User tries to access a web page protected with PAPI
check_Access()
Set Global Parameters
Get symetric key Ikey
Check GPoA_URL & AS_URL varibles
Get the public key
Check Pass_Pattern
Not pass through
¿ACTION = CHECKED?
Yes (We come from GPoA/AS)
No (Direct Access from client Browser)
¿Is GPoA/AS Response valid?
Is valid
Test filters
Pass
Set Lcook
Cookie set ok
¿Get or Post?
Post
Get
Reload original request
Redirect user to the original web page
Accept access
Return code 1 and user data
Cookie doesn't set ok
Is not valid
Doesn't pass the filters
Not authorithed
Error: deny access
(Return code 0 and user data if it is simple PoA, redirect if it is autoPoA)
Accept access.
Return code 2 and NO user data
End
¿Does Lcook exists?
Exists
Doesn't exists
Test Lcook
Correct
Update cookie
Expired
¿Get or Post?
Post
Get
Save Request
Redirect to GPoA/AS to sign in