Instalación y configuración de una arquitectura PAPI

Jaime Pérez Crespo <jaime.perez@urjc.es>

8 de junio de 2006

Resumen

Este documento cubre la instalación y configuración de una arquitectura PAPI de servicios web. A lo largo del mismo se utilizarán ejemplos y casos reales de la implantación de PAPI en la Universidad Rey Juan Carlos.

1. Introducción

PAPI es un sistema que permite la autenticación y autorización de usuarios a través de una red de servicios web. Sus principales bazas, en las que nos centraremos aquí, son la implementación del concepto *single-sign-on* (sólo es necesario autenticarse una vez para acceder a todos los servicios ofrecidos) y la capacidad de actuar como proxy de reescritura transparente al usuario, de forma que permite anteponerse a políticas de acceso de filtrado por IP de origen mediante una autenticación previa.

La arquitectura de PAPI utiliza una serie de conceptos que es necesario tener claros en todo momento a la hora de implantar un servicio con él.

1. *Servidor de autenticación* o *AS*. Es el encargado de autenticar a los usuarios en un único punto de la red, de la forma que se desee, y recolectar información sobre los mismos para proporcionársela a los puntos de acceso. Emite *aserciones* con la información necesaria para verificar la identidad de los usuarios. Por lo general, consulta un fichero con usuarios, una base de datos, o un LDAP para autenticar.

2. *Puntos de acceso* o *PoA*. Son los encargados de proteger el acceso a un recurso concreto. Utilizan las *aserciones* de los *servidores de autenticación* en los que confían para verificar la identidad de sus usuarios y obtener información adicional sobre los mismos. Un *punto de acceso* garantizará el acceso al recurso que protege siempre y cuando algún *servidor de autenticación* en el que confía le proporcione una *aserción* indicando que ha autenticado previamente al usuario, y utilizando la información adicional que éste le puede disponer para

autorización. Los *PoA* pueden funcionar de múltiples formas, dependiendo de los requisitos de la aplicación que se quiere proteger:

a) *PoA* convencional. Escrito en Perl, se integra directamente con el servidor web Apache (versiones 1.x, la rama 2.x no está soportada hasta la fecha). Este tipo de *PoA* permite usar los mecanismos propios del servidor web para permitir el acceso a la aplicación que sirve el mismo o denegarla utilizando una simple página de error *HTTP 403 Forbidden*.

b) *PoA* en modo proxy de reescritura. En este caso, el *PoA* protege el acceso a una aplicación remota que no se encuentra en su mismo servidor web. El *PoA* actúa como un proxy web convencional, editando los contenidos HTML que sirve para que los enlaces hagan referencia a él mismo en lugar de al servicio final que se protege. Su principal uso consiste en dar acceso autenticado a aplicaciones con filtrado por IP de origen, por ejemplo, páginas con recursos bibliográficos a las que se puede acceder únicamente desde direcciones IP suscritas.

c) *PoA* con autocompletado de formularios. En caso de un *PoA* que, actuando como proxy de reescritura, necesite un nivel de autenticación más impuesto por la aplicación protegida (típicamente, nombre de usuario y contraseña), el autocompletado de formularios permite obtener la información necesaria de las aserciones realizadas por el *AS*, pudiendo así rellenar de forma automática y transparente los formularios necesarios. Útil en casos en los que se desea proteger una aplicación remota con el proxy de reescritura, a la que no se tiene acceso para integrar directamente con PAPI, y que requiere al usuario identificarse de alguna manera adicional.

d) *PoA* en PHP. Una versión reducida de un punto de acceso, que permite autenticar a un usuario mediante la arquitectura PAPI, desde una aplicación escrita en PHP y mediante una simple llamada a una función. Su uso más generalizado consiste en la delegación (no tiene por qué ser exclusiva) de la autenticación en PAPI en lugar de la propia aplicación.

3. *Agrupaciones de puntos de acceso* o *GPoA*. Se trata de un *PoA* que no protege ningún recurso en concreto, y en su lugar da acceso a un grupo de *PoAs* con características comunes de validación. Permite personalizar la cantidad de información del usuario que se proporciona a los *PoA*, interponiéndose entre éstos y el *AS*, de forma que cada *PoA* reciba exclusivamente la información necesaria para su autenticación y autorización.

Para información más detallada sobre PAPI, el protocolo subyacente e instalación y configuración del sistema, se recomienda consultar el sitio web que *RedIris* dispone a tal efecto en *http://papi.rediris.es/doc/*.

2. Instalación y configuración

A continuación se proporcionan algunas consideraciones básicas para la instalación de todos los elementos que componen un servicio PAPI. Como convención, los comandos se indicarán precedidos del *prompt* del usuario *root* (#).

2.1. Servidor de autenticación

En primer lugar configuramos un servidor web en el que instalaremos nuestro servidor de autenticación. Lo mejor es redirigir todas las peticiones que reciba dicho servidor web a la URL del *AS* y utilizar SSL. En nuestro caso vamos a utilizar un *virtual host* de apache, que configuraremos de la siguiente manera:

```
NameVirtualHost 192.168.84.92:80
NameVirtualHost 192.168.84.92:443

<VirtualHost 192.168.84.92:80>
  ServerName www.papi.urjc.es
  RedirectMatch (.*)
          https://www.papi.urjc.es/cgi-bin/AuthServer
</VirtualHost>

<VirtualHost 192.168.84.92:443>
  ServerName www.papi.urjc.es
  ErrorDocument 404
          https://www.papi.urjc.es/cgi-bin/AuthServer
  RedirectMatch /index.html
          https://www.papi.urjc.es/cgi-bin/AuthServer
  <IfModule mod_ssl.c>
    SSLEngine on
    SSLCertificateFile /etc/apache/ssl.crt/server.crt
    SSLCertificateKeyFile /etc/apache/ssl.key/server.key
    SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown
  </IfModule>
</VirtualHost>
```

Descargamos y descomprimimos el código fuente de la última versión del servidor de autenticación:

```
# cd /root
# wget ftp://ftp.rediris.es/rediris/papi/PAPI-AS.1.4.0.tar.gz
# tar xzvf PAPI-AS.1.4.0.tar.gz
# cd PAPI-1.4.0
```

Procedemos a compilar e instalar el servidor. Antes de nada debemos asegurarnos de que tenemos los siguientes prerrequisitos (e instalarlos si no cumplimos alguno):

1. Perl >= 5.6
2. openssl >= 0.9.5 (recomendado openssl >= 0.9.6g)
3. Perl Convert::ASN1
4. Perl DB_File
5. Perl Digest::MD5
6. Perl MIME::Base64
7. Perl URI::Escape
8. Perl CGI::Lite

```
# apt-get install perl openssl libssl0.9.7 libssl-dev
# apt get install libconvert-asn1-perl liburi-perl
# apt get install libmd5-perl libmime-perl libdb-file-lock-perl
# cd /usr/share/perl/5.8.4/CGI/
# wget http://search.cpan.org/src/SMYLERS/CGI-Lite-2.02/Lite.pm
# cd /root/PAPI-1.4.0/
# perl Makefile.PL
You must provide a location for installing the PAPI
Authentication Server software. It must be a directory
configured to hold CGI programs in your Apache
installation [/usr/local/apache/cgi-bin]
/usr/lib/cgi-bin
The Authentication Server CGI will be in
/usr/lib/cgi-bin/AuthServer
The configuration file will be /usr/lib/cgi-bin/AuthServer.cf
Other data files will be installed under /usr/lib/cgi-bin/etc/
[...]
The program was not able to determine a location for your
OpenSSL installation, required to build PAPI. Please type the
full path to the OpenSSL directory [/usr/local/ssl]
/usr/lib/ssl
[...]
# make
# make install
# mkdir -p /usr/local/PAPI/AS
# cp -r AS/etc /usr/local/PAPI/AS/
```

Una vez hecho esto tendremos instalado el servidor de autenticación en */usr/lib/cgi-bin/AuthServer* y su fichero de configuración en */usr/lib/cgi-bin/AuthServer.cf*. Procedemos por tanto a configurarlo:

```
# This is the PAPI AS configuration file, a Perl source
# file that essentially contains asignments to a hash named
# PAPI::AuthServer::cfgVar.

use LWP::UserAgent;

# Include here the modules referenced by the elements
# inside the configuration
#use PAPI::BasicAuth;
#use PAPI::POPAuth;
#use PAPI::IMAPAuth;
use PAPI::LDAPAuth;
#use PAPI::CertAuth;

# Just convenience
#
$cfg = \%PAPI::AuthServer::cfgVar;

# The definition of the working directory should be the first
# configuration assignment
#
$$cfg{workingDirectory} = '/usr/local/PAPI/AS/etc';

# This variable will allow the selection of the authentication methods
#
# Uncomment for using basic PAPI authentication
#my $authType = "basic";
# Uncomment for using POP-based authentication
#my $authType = "pop";
# Uncomment for using IMAP-based authentication
#my $authType = "imap";
# Uncomment for using LDAP-based authentication
my $authType = "ldap";
# Uncomment for using X509 certificate authentication
#my $authType = "cert";

# Admin data (included in the HTML templates)
#
$$cfg{adminContact} = '<b>papi@urjc.es</b>';
```

```
# HTML templates
#
$$cfg{loginTemplate} = fromFile("login.html");
$$cfg{acceptTemplate} = fromFile("accept.html");
$$cfg{testTemplate} = fromFile("test.html");
$$cfg{logoutTemplate} = fromFile("logout.html");
$$cfg{rejectTemplate} = fromFile("reject.html");
$$cfg{siteInfoTemplate} =
'<tr>
  <td>*</td>
  <td><a href="<papi var="poa"/><papi var="location"/>
     <papi var="accessURI"/>"
      target="<papi var="service"/>">
   <strong><papi var="desc"/></strong></a>
  </td>
 </tr>';

# Properties of this AS, to be sent to the PoA(s)
#
$$cfg{asLocation} = 'https://www.papi.urjc.es/cgi-bin/AuthServer';
$$cfg{serverID} = 'URJCAS';
$$cfg{privateKey} = 'privkey.pem';
$$cfg{publicKey} = 'pubkey.pem';

# Comment these if you do not require split (thus SSL-capable) mode
#
#$$cfg{splitModeURL} = '';
#$$cfg{splitModeParamList} = '';
#
# These parameters make PoAs redirect accept/reject responses to
# locally controlled URLs
#
#$$cfg{acceptURL} = 'http://www.papi.urjc.es/accept-file';
#$$cfg{rejectURL} = 'http://www.papi.urjc.es/reject-file';
#
# Values for the authentication cookie
# Comment them if you do not require its use
#
$$cfg{authCookie} = 'PAPIuid,username,password';
$$cfg{authCookieDB} =
           '/usr/local/PAPI/AS/etc/PAPIAuthenCookies';
$$cfg{authCookieTimeToLive} = 7200;
```

```
#
# Value of the AS symmetric key
# Used in split mode and for en-/de-crypting the authentication cookie
# It is highly advisable that you change this value
#
#$$cfg{symKey} = 'ca1813914a25b12a14ca121516e26180';
#
# The connection variable holding user id (for TEST and LOGOUT)
$$cfg{uidVar} = 'username';

# Default values for the PoA(s)
#
$$cfg{defTimeToLive} = 7200;
$$cfg{defLocation} = '/';
$$cfg{defService}= '';
$$cfg{defPoA} = '';
$$cfg{defDescription} = '';
$$cfg{defAuthURI} = '/cookie_handler.cgi';
$$cfg{defAccessURI} = '';
#
# Default assertion about users to be sent to PoA(s)
$$cfg{defAssertion} = '';

# Hooks and hook config. By default, "basic" authentication is used
#
if ($authType eq "pop") {
[...]
elsif ($authType eq "imap") {
[...]
elsif ($authType eq "ldap") {
   $$cfg{authenticationHook} = \&PAPI::LDAPAuth::VerifyUser;
   $$cfg{credentialHook} = \&PAPI::LDAPAuth::UserCredentials;
   $$cfg{attrRequestHook} = \&PAPI::LDAPAuth::UserAttributes;
   $$cfg{LDAPserver} = "dagobah.urjc.es";
   $$cfg{LDAPport} = 389;
   $$cfg{LDAPUSERtemplate} = 'uid=<papi var="username"/>';
   $$cfg{LDAPAuthSearchBase} = 'ou=gente,dc=urjc,dc=es';
   $$cfg{LDAPAuthScope} = 'sub';
   $$cfg{LDAPsearchBase} = 'ou=papiSite,dc=urjc,dc=es';
# Uncomment these to use simple authentication when binding
# to the LDAP server
   $$cfg{LDAPbindDN} =
         'uid=managerid,ou=admin,dc=urjc,dc=es';
```

```
   $$cfg{LDAPbindPassword} = 'contraseña';
# Uncomment these to use a TLS connection and verify server identity
#   $$cfg{LDAPS} = 1;
#   $$cfg{LDAPSverify} = 'require';
#   $$cfg{LDAPScafile} = '/etc/PAPI/CAs/MyRoot.pem';
}
elsif ($authType eq "cert") {
[...]
else {
[...]
$$cfg{logHook} = undef;

[...]
```

Una vez configurado el servidor de autenticación, generamos su par de claves asimétricas */usr/local/PAPI/AS/etc/pubkey.pem* y */usr/local/PAPI/AS/etc/privkey.pem*, referenciadas en la configuración:

```
# cd /usr/local/PAPI/AS/etc
# openssl genrsa 1024 -out privkey.pem
# openssl rsa -in privkey.pem -pubout -out pubkey.pem
```

Editamos además las plantillas HTML indicadas en la configuración, para que se adecúen a nuestras necesidades, en el directorio */usr/local/PAPI/AS/etc*. Podemos modificar las plantillas de ejemplo:

- *accept.html*
- *login.html*
- *logout.html*
- *reject.html*
- *test.html*

Hecho esto disponemos de nuestro servidor de autenticación configurado, a falta de incluir puntos de acceso y su configuración asociada.

2.2. Puntos de acceso convencionales

La configuración aquí seguida implica que tanto el *AS* como los *PoAs* perl que actúan en modo proxy se instalan en el mismo servidor, al igual que los *GPoAs*. De este modo, nuestros *PoAs* serán *hosts virtuales* del servidor. Lo mejor en este caso es crear un archivo separado de configuración para los puntos de acceso, por ejemplo, */etc/apache/conf.d/papi*. Dicho fichero debe ir encabezado por la configuración general de PAPI para sus puntos de acceso:

```
#
# Configuración general de PAPI
#
PerlModule PAPI::Conf
<PAPI_Main>
  Service_ID poa_generico
  HKEY_File /usr/local/PAPI/PoA/hkey
  LKEY_File /usr/local/PAPI/PoA/lkey
  Lcook_Timeout 86400
  CRC_Timeout 1800
  URL_Timeout 1800
  Debug 0
  Auth_Location /cookie_handler.cgi
# en caso de utilizar el mecanismo de las ``bolitas'',
# debemos descomentar las siguientes líneas
#  Accept_File /usr/local/PAPI/PoA/yes.gif
#  Reject_File /usr/local/PAPI/PoA/no.gif
  Pubkeys_Path /usr/local/PAPI/PoA/KEYS
  Hcook_DB /usr/local/PAPI/PoA/hcook.db
  PAPI_AS URJCAS
     https://www.papi.urjc.es/cgi-bin/AuthServer AuthServer URJC
</PAPI_Main>
```

Descargamos y descomprimimos el código fuente de la última versión del servidor de autenticación:

```
# cd /root
# wget ftp://ftp.rediris.es/rediris/papi/PAPI-PoA.1.4.0.tar.gz
# tar xzvf PAPI-PoA.1.4.0.tar.gz
```

Compilamos e instalamos el punto de acceso. Nos aseguramos de que tenemos los siguientes prerrequisitos (y los instalamos si no cumplimos alguno):

1. mod_perl >= 1.20
2. apache >= 1.3.8
3. Perl SSL
4. Perl MLDBM
5. Perl MLDBM::Sync
6. Perl URI::URL
7. Perl HTML::TokeParser

8. Perl HTTP:Cookies
9. Perl Data::Dumper
10. Perl HTTP::Request::Form
11. Perl Net::LDAP

```
# apt-get install libapache-mod-perl apache
# apt get install libio-socket-ssl-perl libmldbm-perl
# apt get install liburi-perl libmldbm-sync-perl
# apt-get install libhtml-tokeparser-simple-perl
# apt-get install libapache-authcookie-perl
# apt-get install libdata-dumper-simple-perl
# apt-get install libnet-ldap-perl
# cd
# wget http://.../HTTP-Request-Form-0.952.tar.gz
# tar xzvf HTTP-Request-Form-0.952.tar.gz
# cd HTTP-Request-Form-0.952
# perl Makefile.PL
# make
# make install
```

Una vez cumplidos los prerrequisitos, instalamos el PoA:

```
# cd /root/PAPI-1.4.0
# perl Makefile.PL
You must provide a location for installing the PAPI
Authentication Server software. It must be a directory
configured to hold CGI programs in your Apache
installation [/usr/local/apache/cgi-bin]
/usr/lib/cgi-bin
The Authentication Server CGI will be in
/usr/lib/cgi-bin/AuthServer
The configuration file will be /usr/lib/cgi-bin/AuthServer.cf
Other data files will be installed under /usr/lib/cgi-bin/etc/
[...]
The program was not able to determine a location for your
OpenSSL installation, required to build PAPI. Please type the
full path to the OpenSSL directory [/usr/local/ssl]
/usr/lib/ssl
[...]
# make
# make install
```

Estando el PoA instalado, podemos incluir la configuración general de PAPI ya expuesta en la configuración de Apache, y proceder a configurar los puntos de acceso que necesitemos. Lo más sencillo es utilizar *virtual hosts* para cada uno. En nuestro caso, queremos que los accesos se hagan siempre mediante conexiones seguras, por lo que redirigimos todo el tráfico para utilizar SSL:

```
<VirtualHost 192.168.84.92:80>
  ServerName poa.papi.urjc.es
  ErrorDocument 404 https://poa.papi.urjc.es/index.html
</VirtualHost>

<VirtualHost 192.168.84.92:443>
  ServerName poa.papi.urjc.es
  ErrorDocument 403 https://www.papi.urjc.es/cgi-bin/AuthServer
  <Location /app>
    PerlSendHeader On
    PerlAccessHandler PAPI::Main
    <PAPI_Local>
      Service_ID poa
      Req_DB /usr/local/PAPI/PoA/req_poa
      GPoA_URL wayf:built-in
    </PAPI_Local>
  </Location>
</VirtualHost>
```

Este punto de acceso funciona de la forma más simple posible, en nuestro caso, protegiendo todos los accesos que se hagan a *http://poa.papi.urjc.es/app*. Si un usuario trata de acceder a cualquier URL que comience por */app*, será redireccionado a una conexión segura, y una vez hecho esto se comprobará si tiene las credenciales de PAPI en orden. En caso afirmativo se le garantizará acceso, y en caso contrario se le redirigirá al *AS* para que se autentique. Si el acceso se realiza a otra URL que no está bajo la protección del punto de acceso, PAPI no intervendrá de ningún modo.

2.3. Puntos de acceso en modo proxy de reescritura

Los puntos de acceso en modo proxy se configuran exactamente igual que cualquier otro punto de acceso, con la particularidad de que debemos indicar la URL o el dominio al que redirigir las peticiones entrantes. Por ejemplo:

```
<VirtualHost 192.168.84.92:80>
  ServerName proxy.papi.urjc.es
  RedirectMatch /index.html https://proxy.papi.urjc.es
  ErrorDocument 404 https://proxy.papi.urjc.es/index.html
```

```
</VirtualHost>

<VirtualHost 192.168.84.92:443>
  ServerName proxy.papi.urjc.es
  ErrorDocument 403 https://www.papi.urjc.es/cgi-bin/AuthServer
  <Location />
    PerlSendHeader On
    PerlAccessHandler PAPI::Main
    <PAPI_Local>
      Service_ID proxy
      Req_DB /usr/local/PAPI/PoA/req_proxy
      GPoA_URL wayf:built-in
      Remote_URL http://www.urjc.es
      Eval_Proxy_Redirects 1
    </PAPI_Local>
  </Location>
</VirtualHost>
```

De esta forma el punto de acceso en vez de proteger una localización concreta del propio servidor, lo hace respecto a una URL externa comportándose como un proxy. Si en vez de utilizar la directiva **Remote_URL** utilizamos **Remote_Domain** acompañada de un nombre de dominio, todas las peticiones recibidas de la forma *http://proxy.papi.urjc.es /host/pagina* se traducirán en peticiones a *http://host.dominio/pagina*. Para conseguir esto es conveniente usar también la directiva **Strip_Location** con el valor *1*. De este modo las peticiones */host/pagina* se traducen en */pagina* en la URL remota especificada con **Remote_URL**.

Por último, podemos utilizar las directivas **PAPI_Redirect**, **Rewrite_URL** y **Redirect_All** para definir las sustituciones de URLs en los documentos HTML servidos, hacerlo incluso en contenidos no HTML (como código javascript) y evaluar todas las URLs encontradas en el documento en modo agresivo, respectivamente. Para información detallada sobre todas estas y otras directivas adicionales es conveniente consultar la guía para principiantes de PAPI.

2.4. Autocompletado de formularios

El autocompletado de formularios es una vuelta de tuerca sobre la configuración de un *PoA* en modo proxy. Básicamente consiste en utilizar la directiva From_Processor para configurar el módulo de PAPI que se encargará de los formularios. Por ejemplo:

```
<VirtualHost 192.168.84.92:80>
  ServerName proxy.papi.urjc.es
  RedirectMatch (.*) https://proxy.papi.urjc.es/
</VirtualHost>
```

```
<VirtualHost 192.168.84.92:443>
  ServerName proxy.papi.urjc.es
  <Location />
    PerlSendHeader On
    PerlAccessHandler PAPI::Main
    <PAPI_Local>
      Service_ID proxy
      Req_DB /usr/local/PAPI/PoA/req_proxy
      GPoA_URL wayf:built-in
      Remote_URL https://www.urjc.es
      Redirect_All 1
      PAPI_Redirect www.urjc.es(.*?) proxy.papi.urjc.es$1
      Eval_Proxy_Redirects 1
       Form_Processor urle=>^/.* \
         form=>login buttonName=>action_login \
         field=>(password,$r->notes('PAPIAttr-password')) \
         field=>(usuario,$r->notes('PAPIAttr-username'))
    </PAPI_Local>
  </Location>
```

La directiva **Form_Processor** se compone de los siguientes atributos:

- *urle*: una expresión regular que define la URL o URLs en las que se encuentra el formulario que queremos completar automáticamente.
- *form*: el nombre del formulario, en caso de que haya más de uno. Si no se indica este atributo o no existe ningún formulario con el nombre indicado, se elegirá el primer formulario de la página.
- *buttonName*: el nombre del botón de envío del formulario.
- *field*: un campo del formulario, definido por una tupla (*nombre*, *valor*), donde *nombre* es el nombre del campo en el formulario y *valor* el valor con el que queremos rellenarlo. Incluiremos este atributo tantas veces como campos necesitemos enviar, incluyendo aquellos ocultos. Podemos utilizar valores fijos o incluso variables de PAPI para rellenar un campo con, por ejemplo, el nombre de usuario o la contraseña del mismo. Para esto debemos activar la directiva **Eval_Proxy_Redirects**, que permite que Perl evalúe los valores especificados en la configuración. De esta forma, cuando queramos obtener un campo de la aserción de PAPI utilizaremos la sintaxis *$r->notes('PAPIAttr-nombre')*, donde *nombre* es el nombre del campo en la propia aserción. Esto implica que previamente debemos modificar el formato de la aserción de la siguiente forma: *variable1=valor1,variable2=valor2*. Esto lo haremos en la propia configuración del

punto de acceso en el atributo *papiAssertion* del LDAP. Por ejemplo, si queremos que la aserción del PoA sea *usuario=valor,password=valor*, configuraremos *papiAssertion* de la siguiente forma: *usuario=<papi attr="login"/>, password=<papi var="password"/>*, donde *attr* especifica que "*login*" es un atributo localizado en el perfil del usuario en el LDAP, y *var* indica que "*password*" es una variable del formulario de autenticación del AS (y que deberá propagarse en la aserción del mismo).

2.5. Puntos de acceso PHP

En primer lugar debemos descargar la última versión del *phpPoA* del sitio FTP de *RedIris*: *ftp://ftp.rediris.es/rediris/papi/*. Descomprimimos el archivo a algún lugar en nuestro sistema, por ejemplo, */usr/local/phpPoA*:

```
# cd /usr/local
# wget ftp://ftp.rediris.es/rediris/papi/phpPoA.1.1.tar.gz
# tar -xzvf phpPoA.1.1.tar.gz
```

Así, en */usr/local/phpPoA* tendremos los contenidos siguientes:

doc la documentación de la versión del *phpPoA* instalada.

etc la configuración del *phpPoA*. Esto incluye un archivo con extensión *ini*, ficheros html de error, y un directorio para las claves criptográficas utilizadas.

php el *phpPoA* propiamente dicho y su librería criptográfica asociada.

samples ejemplos de uso del *phpPoA*.

LICENSE la licencia de distribución.

README una descripción de los contenidos como ésta.

Debemos asegurarnos de que la instalación de PHP encuentre el script del *PoA*, por lo que incluiremos la ruta completa en el fichero *php.ini*, en debian típicamente */etc/php4/apache/php.ini*. Buscamos la sección de rutas y directorios, y añadimos las siguientes lineas:

```
include_path = ".:/usr/share/php:/usr/local/phpPoA/php"
phpPoA_ini_file = "/usr/local/phpPoA/etc/phpPoA.ini"
```

Dado que PAPI utiliza métodos criptográficos que no suelen venir instalados por defecto, deberemos asegurarnos de tener instalada la librería *mcrypt*. De no estarlo, la instalamos:

```
# apt-get install php4-mcrypt
```

Reiniciamos Apache y colocamos alguno de los ejemplos del directorio *samples* en el directorio raíz de Apache para comprobar que el *PoA* está funcionando correctamente. La configuración del *PoA* se realiza en el archivo *phpPoA.ini*. En dicho archivo indicaremos las opciones necesarias para la autenticación y autorización de los usuarios en la aplicación o aplicaciones que proteja este *PoA*. La sección [PAPI Main] es obligatoria y define la configuración global para todas las aplicaciones. Aparte, es posible definir nuevas secciones que redefinan el comportamiento de PAPI para aplicaciones concretas. Por ejemplo:

```
[PAPI_Main] Lcook_Timeout = 86400
; localización de la base de datos de peticiones
; el directorio DB debe ser propiedad del usuario
; que ejecuta apache y tener permisos de escritura
Request_DB = /usr/local/phpPoA/var/DB/request_db.db4
DB_Type = db4
; al igual que el directorio DB, el directorio logs
; debe tener permisos suficientes para que el usuario
; de apache pueda escribir en él
error_log = /usr/local/phpPoA/var/logs/papi_error.log
; si se usa el modo automático, se debe proporcionar
; la url de los posibles documentos de error
Not_Auth_Error_File = https://foo.urjc.es/NotAuthorized.html
Cookie_Error_File = https://foo.urjc.es/CookieError.html
System_Error_File = https://foo.urjc.es/SystemError.html
; definición de aserciones que se permiten
PAPI_Filter_accept = ".*"
; definición de aserciones que se deniegan
PAPI_Filter_reject = ".*"
; Cookie_Domain is Optional
Cookie_Domain = papi.urjc.es
; clave compartida con el GPoA
LKEY_File = /usr/local/phpPoA/KEYS/lkey
; clave pública del GPoA
GPoA_Pub_Key = /usr/local/phpPoA/KEYS/_GPoA_pubkey.pem
; la url completa del GPoA, debe coincidir con la
; definida por la directiva Auth_Location en el GPoA
GPoA_URL = https://gpoa.papi.urjc.es/cookie_handler.cgi

[admin]
Location = /papi/admin
; Cookie_Domain is Optional
Cookie_Domain = papi.urjc.es
LKEY_File = /usr/local/phpPoA/KEYS/lkey
```

```
GPoA_Pub_Key = /usr/local/phpPoA/KEYS/_GPoA_pubkey.pem
GPoA_URL = https://gpoa.papi.urjc.es/cookie_handler.cgi
PAPI_Filter_accept = ".*"
PAPI_Filter_reject = ".*"
Lcook_Timeout = 86400
Request_DB = /usr/local/phpPoA/var/DB/request_db.db4
DB_Type = db4
error_log = /usr/local/phpPoA/var/logs/papi_error.log
Not_Auth_Error_File = https://foo.urjc.es/NotAuthorized.html
Cookie_Error_File = https://foo.urjc.es/CookieError.html
System_Error_File = https://foo.urjc.es/SystemError.html
```

En particular, debemos tener en cuenta las siguientes consideraciones a la hora de tocar el fichero de configuración:

1. El PoA necesita escribir en dos lugares concretos, tanto una base de datos de peticiones, como un archivo de log. **El proceso Apache debe tener los permisos necesarios** para escribir en ambos sitios.

2. **El modo automático redirige al usuario directamente** a los documentos de error cuando la autenticación no ha sido correcta. Esto es interesante cuando queremos utilizar PAPI como única alternativa de autenticación y autorización. Si por contra es un método adicional, y queremos presentar al usuario alternativas (por ejemplo, un formulario en el que introducir nombre de usuario y contraseña) en caso de que PAPI no pueda autenticar, debemos desactivar el modo automático y controlar el valor devuelto por la llamada a *check_Access()*.

3. *phpPoA* no implementa un *PoA* completo. Por eso **es necesario instalar y configurar un *GPoA*** al que el *phpPoA* pedirá información. Por tanto en la configuración hay que indicar la URL del mismo, incluyendo la ruta absoluta indicada en la directiva *Auth_Location* del *GPoA*.

4. Los filtros permiten determinar la autorización de acceso al *PoA* para usuarios que el *GPoA* define como autenticados. Por ello se especifican filtros mediante expresiones regulares de Perl que definan qué aserciones se aceptan y cuáles se deniegan. Lo habitual es **denegar todas salvo aquellas que cumplan cierta propiedad** (por ejemplo, que el usuario pertenezca a un grupo concreto).

Si estamos utilizando *PAPI* con el mecanismo de autenticación mediante "bolitas", podemos conseguir que las entradas en la página del AS para *PoAs* en PHP también tengan su imagen asociada (y que indique correctamente el estado de las credenciales en dicha aplicación), mediante un sencillo script en PHP:

```
<?php
```

```
$AS = "https://www.papi.urjc.es/cgi-bin/AuthServer";
$BASE = "https://sympa.papi.urjc.es/";
$YES = $BASE."yes.gif";
$NO = $BASE."no.gif";
$_SERVER['HTTPS'] = "on"; // only if we use https
include('PoA.php');

// check PAPI auth
$auto = 0; // do not use papi auto mode
$poa = new PoA('scheme',$auto);
list($res,$userAssertion) = $poa->check_Access();

// show image according to auth status
if ($res == 0)
  // GPoA doesn't know user
  header("Location: $NO");
else
  header("Location: $YES");
?>
```

Guardamos el fichero en el directorio raiz de Apache y configuramos el *AS* para que consulte la URL correspondiente al script en nuestro *phpPoA*.

2.6. Agrupaciones de puntos de acceso

La configuración de un grupo de puntos de acceso o *GPoA* es idéntica al de un *PoA* convencional, salvo por la inclusión de algunas directivas propias del *GPoA*. A continuación se muestra una posible configuración con hosts virtuales de Apache de un *GPoA*:

```
<VirtualHost 192.168.84.92:80>
  ServerName gpoa.papi.urjc.es
  RedirectMatch (.*) https://gpoa.papi.urjc.es/
</VirtualHost>

<VirtualHost 192.168.84.92:443>
  ServerName gpoa.papi.urjc.es
  ErrorDocument 403 https://www.papi.urjc.es/cgi-bin/AuthServer
  DocumentRoot /var/www/gpoa/
  <Location />
    PerlSendHeader On
    PerlAccessHandler PAPI::Main
    <PAPI_Local>
```

```
        Service_ID gpoa
        Req_DB /usr/local/PAPI/PoA/req_gpoa
        GPoA_Priv_Key /usr/local/PAPI/AS/etc/privkey.pem
         GPoA_Hash_User_Data 0
        GPoA_URL wayf:built-in
      </PAPI_Local>
    </Location>
  </VirtualHost>
```

La directiva *GPoA_Priv_Key* indica al *GPoA* dónde localizar la clave privada con la que encriptar los datos asociados a sus respuestas, mientras que la directiva *GPoA_Hash_User_Data* indica si el *GPoA* debe aplicar una función hash sobre los datos del usuario antes de enviarlos. En caso de activarse (**1**) los datos jamás se enviarán en claro, y por tanto los puntos de acceso finales no podrán acceder a ellos, mientras que manteniéndola desactivada (**0**) nos aseguramos de que los clientes puedan acceder a dichos datos (lo cual puede suponer un riesgo de seguridad). Adicionalmente, y mediante la directiva *GPoA_Rewrite*, podemos controlar la información que enviamos a cada *PoA* subordinado. La directiva recibe tres parámetros. El primero, una expresión regular que se comparará con el nombre del *PoA*, el segundo una expresión regular que se aplicará a la aserción emitida por el *GPoA*, y el tercero el patrón con el que se debe sobreescribir dicha aserción.

Por último, si queremos que el *GPoA* no aparezca como opción en la lista de *PoAs* disponibles para un usuario, bastará con que modifiquemos su configuración en el LDAP, dejando el campo *papiSiteService* vacío (con uno o más espacios).

3. Integración con aplicaciones

A continuación se encuentran detalladas las implementaciones concretas para cada aplicación integrada en el sistema *PAPI* de la universidad.

3.1. Safari (publicaciones online)

Safari es un servicio online de *O'Reilly* mediante el cual se pueden consultar publicaciones desde cualquier dirección IP de la universidad. Nuestra intención es poder utilizar dicho servicio desde cualquier ubicación (no sólo desde la universidad) mediante una autenticación previa. Para ello instalamos un servidor que actuará como *proxy PoA*, y lo configuramos de la siguiente manera:

```
<VirtualHost 192.168.84.92:80>
  ServerName safari.papi.urjc.es
  RedirectMatch /index.html
            https://safari.papi.urjc.es/index.html
```

```
  ErrorDocument 404 https://safari.papi.urjc.es/index.html
</VirtualHost>

<VirtualHost 192.168.84.92:443>
  ServerName safari.papi.urjc.es
  ErrorDocument 403 https://www.papi.urjc.es/cgi-bin/AuthServer
  <Location />
    PerlSendHeader On
    PerlAccessHandler PAPI::Main
    <PAPI_Local>
      Service_ID safari
      Req_DB /usr/local/PAPI/PoA/req_safari
      GPoA_URL wayf:built-in
       Remote_URL http://safari.oreilly.com
    </PAPI_Local>
  </Location>
</VirtualHost>
```

3.2. LAPA (aplicación de comunicaciones)

En esta aplicación el método de autenticación consistía en un usuario y contraseña con los que se creaba una sesión que se comprobaba el resto del tiempo. En su lugar, crearemos un *index.php* que compruebe las credenciales de PAPI y haga lo necesario para autenticar al usuario en caso de que no lo esté ya (en este caso, redirigirle al *GPoA* asociado). Cuando las credenciales sean las esperadas, se le creará una sesión que servirá para identificarle hasta que cierre la aplicación, de modo que no es necesario tocar ningún código de la misma, sino que basta con incluir un *index.php* similar al que sigue:

```
<?php
$AS = "https://www.papi.urjc.es/cgi-bin/AuthServer";
session_start();
include('/var/www/include/dbconnectv2.php');
include('/var/www/clases/c_log.php');
include'PoA.php';

//logs
$log->seccion = "autenticacion";
$log->operador = $correo;

$poa = new PoA('lapa');
list($res, $userAssertion) = $poa->check_Access();
```

```
if ($res == 0) {
  // el GPoA no autentica
  header("Location: $AS");
  exit;
}

list($mail,$set) = split(",",$userAssertion,2);
list($login,$domain) = split("@",$mail,2);
$consulta = mysql_query("select * from operadores where login='$login'");
if (mysql_num_rows($consulta) == 0) {
  //No esta en la tabla de operadores
  header("Location: $AS");
  exit;
}

// autenticado con éxito por PAPI, y es operador
if (session_is_registered("SESSION"))
  session_unregister("SESSION");
session_register("SESSION");
$SESSION["login"] = $login;
$log->descripcion = "Exito en la autenticacion";
$log->putlog();

header("Location: /main.php");
exit;
?>
```

En este ejemplo se incluye código que se utilizaba en el antiguo sistema de autenticación para crear la sesión y guardar un histórico de accesos. En **negrita** se encuentra el código más importante desde nuestro punto de vista, que se encarga de consultar al *PoA* las credenciales del usuario, y redirigirle, bien al *AS*, bien a otro fichero PHP previa creación de una sesión, en función de las mismas.

3.3. HORDE (correo web)

Horde es el sistema de correo web utilizado en la universidad. Permite la utilización de diferentes servidores IMAP y por tanto dar acceso a varios dominios de correo. Partiendo de la base de que tenemos *Horde* configurado y que utilizamos *IMP* para la autenticación, lo único que tenemos que hacer es sustituir el fichero *index.php* de *Horde* por uno ligeramente modificado. Encontraremos el original en */usr/share/horde3/index.php*. Realizamos una copia de seguridad y editamos su contenido:

```
# cp /usr/share/horde3/index.php /usr/share/horde3/index.php.bak
# vi /usr/share/horde3/index.php
```

Un *index.php* similar al siguiente servirá para que los usuarios puedan acceder directamente a su buzón si ya están identificados en *PAPI*, y sean redirigidos al *AS* en caso contrario:

```
<?php
/**
 * $Horde: horde/index.php,v 2.105.4.3 2005/03/02 07:10:35 chuck Exp $
 *
 * Copyright 1999-2005 Charles J. Hagenbuch <chuck@horde.org>
 * Copyright 1999-2005 Jon Parise <jon@horde.org>
 *
 * See the enclosed file COPYING for license information (LGPL).  If you
 * did not receive this file, see http://www.fsf.org/copyleft/lgpl.html.
 */

@define('AUTH_HANDLER', true);
@define('HORDE_BASE', '/usr/share/horde3');
$horde_configured =
        (@file_exists(HORDE_BASE . '/config/conf.php') &&
         @file_exists(HORDE_BASE . '/config/mime_drivers.php') &&
         @file_exists(HORDE_BASE . '/config/nls.php') &&
         @file_exists(HORDE_BASE . '/config/prefs.php') &&
         @file_exists(HORDE_BASE . '/config/registry.php'));

if (!$horde_configured) {
    require HORDE_BASE . '/lib/Test.php';
    Horde_Test::configFilesMissing('Horde', HORDE_BASE, 'prefs.php',
      array('conf.php' => 'This is the main Horde configuration file. It\
         contains paths and basic items that apply to the core\
         framework and all Horde applications.',
        'mime_drivers.php' => 'This file controls the global set of\
         MIME  drivers for the Horde framework, allowing applications\
         to make use of programs such as enscript or mswordview to\
         render  content into HTML for viewing in a browser.',
        'nls.php' => 'This file provides localisation support for the\
         Horde framework.',
        'registry.php' => 'The registry is how Horde applications find\
         out how to talk to each other. You should list any installed\
         Horde applications that you have here.'));
```

```
}

require_once HORDE_BASE . '/lib/base.php';
// añadimos la biblioteca criptográfica de Horde
require_once HORDE_BASE . '/lib/Horde/Secret.php';

if ($browser->isMobile()) {
    require HORDE_BASE . '/services/portal/mobile.php';
    exit;
}

$main_page = Util::getFormData('url');
if (!$main_page) {
    $main_page = Horde::initialPage();
}

// autenticamos mediante PAPI
$AS = "https://www.papi.urjc.es/cgi-bin/AuthServer";
$_SERVER['HTTPS'] = "on"; // sólo si usamos https
include('PoA.php');

$poa = new PoA('horde');
list($res,$userAssertion) = $poa->check_Access();

// si no está autenticado, redirigimos al AS
if ($res == 0) {
  header("Location: $AS");
  exit;
}

// parseo de la asercion
list($mail,$other) = split(",",$userAssertion,2);
list($imapuser,$server) = split("@",$mail,2);
list($pass,$AS) = split("@",$other,2);

if ($imapuser == '' || $pass == '') {
    header("Location: $AS");
    exit;
}

// rellenamos las variables post del formulario de login
$_POST['imapuser'] = $imapuser;
$_POST['pass'] = $pass;
```

```
$_POST['autologin'] = "0";
$_POST['mailbox'] = "INBOX";
$_POST['load_frameset'] = "1";
$_POST['actionID'] = "";
$_POST['folders'] = "INBOX.";
$_POST['url'] = "index.php";
$_POST['new_lang'] = "es_ES";

/* codigo específico para los servidores de correo configurados
     habría que cambiarlo! */
if ($server == "alumnos.urjc.es")
  $server = "imapALUMNOS";
if ($server == "urjc.es")
  $server = "imapURJC";
$_POST['server'] = $server;

if (!Util::getFormData('frameset_loaded') &&
    ($conf['menu']['always'] ||
     (Auth::getAuth() && $prefs->getValue('show_sidebar')))) {
    if ($browser->hasQuirk('scrollbar_in_way')) {
        $scrollbar = 'yes';
    } else {
        $scrollbar = 'auto';
    }
    /* nos aseguramos de mantener la password en la variable de
         sesion en caso de que no sea nuestro primer acceso */
    $_SESSION['imp']['pass'] = Secret::write(
                Secret::getKey('imp'), $pass);
    require HORDE_TEMPLATES . '/index/frames_index.inc';
} else {
    // tras rellenar a mano el formulario, lo "enviamos"
    require HORDE_BASE . '/imp/redirect.php';
}
?>
```

3.4. SYMPA (listas de correo)

En el caso del gestor de listas de correo no se puede utilizar el *phpPoA*, pero si un *PoA* convencional, con la particularidad de que es necesario adaptar *Sympa* para que utilice dicho *PoA* como sistema *single-sign-on*. Para ello es necesario modificar el código del propio *Sympa* y crear conectores que se encarguen de enlazar con la funcionalidad proporcionada por *PAPI*.

En nuestro caso, por sencillez, es preferible configurar un proxy de reescritura que

utilice el autocompletado de formularios para rellenar el mismo automáticamente. **Es necesario modificar la plantilla principal de Sympa ya que no incluye nombre en sus formularios**. Para ello, editamos el archivo */usr/share/sympa/wws_templates/loginbanner.es.tpl* y en la línea 17 añadimos el atributo *NAME=login*. Configuramos adicionalmente un *virtual host* de apache para nuestro proxy, del siguiente modo:

```
<VirtualHost 192.168.84.92:80>
  ServerName sympa.papi.urjc.es
  RedirectMatch (.*) https://sympa.papi.urjc.es/
</VirtualHost>

<VirtualHost 192.168.84.92:443>
  ServerName sympa.papi.urjc.es
  <Location />
    PerlSendHeader On
    PerlAccessHandler PAPI::Main
    <PAPI_Local>
      Service_ID sympa
      Req_DB /usr/local/PAPI/PoA/req_sympa
      GPoA_URL wayf:built-in
      Remote_URL https://listas.urjc.es
      Redirect_All 1
      PAPI_Redirect listas.urjc.es(.*?) sympa.papi.urjc.es$1
      Eval_Proxy_Redirects 1
      Form_Processor urle=>^/wws.* form=>login
            buttonName=>action_login
            field=>(passwd,$r->notes('PAPIAttr-password'))
            field=>(email,$r->notes('PAPIAttr-username'))
    </PAPI_Local>
  </Location>
</VirtualHost>
```

3.5. GENTE (directorio)

Gente es la aplicación de directorio utilizada en la universidad, y se utiliza adicionalmente para que el personal modifique sus datos personales, entre los que se encuentran, por ejemplo, el nombre de usuario y la contraseña que utilizará PAPI para autenticar. Es por tanto una aplicación un poco especial, ya que acepta dos formas de autenticación a través del mismo formulario. Cuando un usuario no tiene un nombre de usuario y una contraseña, deberá hacer login con su NIF y su fecha de nacimiento, ambos valores almacenados en una base de datos. Tras identificarse por primera vez y elegir un nombre de usuario y contraseña, esta forma de acceso no volverá a permitirse para este usuario. Por si fuera poco, sólo se permite el acceso autenticado (de

cualquier forma) a través de la propia red de la universidad, lo cual complica aún más la implementación.

Las implicaciones directas de este modelo son que la aplicación debe mantener su formulario, en vez de delegar absolutamente la autenticación en PAPI. Deberá recoger los datos del usuario y realizar los análisis pertinentes (como verificar si se ha introducido un nombre de usuario o un DNI y si se conecta desde la red corporativa), y proceder de forma adecuada. En particular, si se recibe un DNI la aplicación deberá autenticar por su cuenta contra la base de datos, y si recibe un nombre de usuario deberá delegar la autenticación en un *phpPoA*. Al introducir un *phpPoA* como forma de autenticación se elimina la necesidad de acceder exclusivamente desde la red de la universidad, ya que PAPI proporciona las garantías de seguridad necesarias para que los usuarios accedan a la aplicación desde otras localizaciones.

La implementación de este sistema no es en absoluto trivial. Una forma más sencilla de implementar el acceso autenticado a la aplicación a través de PAPI consiste en implementar un fichero PHP aparte, por ejemplo, papi.php, al que se accederá desde la lista de aplicaciones del servidor de autenticación, y que básicamente realizará un POST manual al formulario de gente. De esta forma podemos acceder una vez autenticados en PAPI, pero perdemos la capacidad de hacerlo directamente, viéndonos obligados a hacerlo obligatoriamente a través del *AS*.

```
<?php
$AS = "https://www.papi.urjc.es/cgi-bin/AuthServer";
include'PoA.php';

$poa = new PoA('gente');
list($res, $userAssertion) = $poa->check_Access();

// comprobamos si el usuario está autenticado
if ($res == 0) {
  // el GPoA no autentica
  header("Location: $AS");
  exit;
}

// verificamos la aserción
if (!strpos($userAssertion,',')) {
    header("Location: $AS");
    exit;
}

// obtenemos las variables de la aserción
list($mail,$other) = split(",",$userAssertion,2);
list($usuario,$server) = split("@",$mail,2);
```

```
list($clave,$AS) = split("@",$other,2);

// rellenamos manualmente las variables del formulario
@$_REQUEST['accion'] = "entrar";
@$_REQUEST['usuario'] = $usuario;
@$_REQUEST['clave'] = $clave;
@$_REQUEST['submit'] = "Aceptar";

// invocamos manualmente a la autenticación
require ("login.php");
?>
```

3.6. Campus Virtual (WebCT)

El caso de WebCT es relativamente sencillo, ya que basta con instalar un proxy de reescritura que rellene automáticamente el formulario de autenticación. Ya que la aplicación se distribuye con su propio servidor web y es propietaria, utilizar otros métodos de integración con PAPI se complica considerablemente. A continuación se muestra una configuración de ejemplo:

```
<VirtualHost 192.168.84.92:80>
  ServerName campusvirtual.papi.urjc.es
  RedirectMatch /index.html https://campusvirtual.papi.urjc.es
  ErrorDocument 404 https://campusvirtual.papi.urjc.es/index.html
</VirtualHost>

<VirtualHost 192.168.84.92:443>
  ServerName campusvirtual.papi.urjc.es
  ErrorDocument 403 https://www.papi.urjc.es/cgi-bin/AuthServer
  <Location />
    PerlSendHeader On
    PerlAccessHandler PAPI::Main
    <PAPI_Local>
      Service_ID campusvirtual
      Req_DB /usr/local/PAPI/PoA/req_campusvirtual
      GPoA_URL wayf:built-in
      Remote_URL http://www.campusvirtual.urjc.es
      Eval_Proxy_Redirects 1
       Form_Processor urle=>.*/webct/ticket/ticketLogin\?\
          action=print_login&request_uri=/webct/homearea/homearea.* \
          form=>authenticate buttonName=>submitbutton \
          field=>(Password,$r->notes('PAPIAttr-password')) \
          field=>(WebCT_ID,$r->notes('PAPIAttr-username'))
```

```
    </PAPI_Local>
  </Location>
</VirtualHost>
```

3.7. Secretaría Virtual

El caso de la secretaría virtual es un poco más delicado. Su formulario de identificación utiliza distintos parámetros dependiendo del tipo de usuario. En caso de que el usuario sea PAS o PDI utilizará el nombre de usuario y contraseña que se utiliza en el resto de servicios, mientras que los alumnos deberán utilizar su DNI (sin letra) y su contraseña. Instalaremos un proxy de reescritura con autocompletado de formulario, con la particularidad de que necesitaremos hacer uso de la directiva *Hcook_Generator* para modificar la aserción emitida por el AS atendiendo al tipo de usuario. De este modo comprobaremos si el usuario tiene perfil de alumno, y en tal caso sustituiremos su nombre de usuario por su DNI sin letra.

A continuación se muestra una configuración muy básica de un proxy PAPI que rellenará el formulario correspondiente, **sin** modificar la cookie, de modo que sólo permitirá identificarse al personal universitario:

```
<VirtualHost 192.168.84.92:80>
  ServerName gestion.papi.urjc.es
  RedirectMatch /index.html https://gestion.papi.urjc.es
  ErrorDocument 404 https://gestion.papi.urjc.es/index.html
</VirtualHost>

<VirtualHost 192.168.84.92:443>
  ServerName gestion.papi.urjc.es
  ErrorDocument 403 https://www.papi.urjc.es/cgi-bin/AuthServer
  <Location />
    PerlSendHeader On
    PerlAccessHandler PAPI::Main
    <PAPI_Local>
      Service_ID gestion
      Req_DB /usr/local/PAPI/PoA/req_gestion
      GPoA_URL wayf:built-in
      Remote_URL https://gestion.urjc.es
      Redirect_All 1
      PAPI_Redirect gestion.urjc.es(.*?) gestion.papi.urjc.es$1
       Eval_Proxy_Redirects 1
      Form_Processor urle=>^/Redcampus/acceso.htm$ \
         form=>formAcceso buttonName=>image \
         field=>(TXT_CLAVE,$r->notes('PAPIAttr-password')) \
         field=>(TXT_USUARIO,$r->notes('PAPIAttr-username'))
```

```
      </PAPI_Local>
    </Location>
  </VirtualHost>
```