Seguridad

José Luis Cánovas Sánchez - 48636907A
Ezequiel Santamaría Navarro - 20096517Z

11 de abril de 2016

Resumen

En esta memoria se describe el desarrollo del apartado 2.1 de las prácticas de Seguridad: Despliegue de servicios de gestión de identidad avanzados mediante biometría y tecnologías SAML y OAuth.

Índice

1. Introducción

Como grupo 3 de prácticas, desplegaremos las organizaciones 31 y 32. El trabajo en grupo ha consistido en que ambos hemos desarrollado y configurado todo conjuntamente, lo que se refleja en nuestro repositorio git de GitHub, donde subimos los scripts de instalación con los que automatizamos la práctica, pues preferimos no usar máquinas virtuales, por los problemas de fluidez y tamaño de las máquinas virtuales. Así podemos trabajar con unos pocos megas de scripts y capturas en cualquier ordenador.

2. Autenticación biométrica

Desarrollamos una aplicación para **comparación**, **identificación** y **verificación** de huellas dactilares.

2.1. Funcionamiento de la aplicación

Primero vamos a ver **cómo funciona** la aplicación desarrollada, y una vez conocemos todas las funcionalidades que ofrece, veremos cómo lo hemos implementado.

Cuando abrimos la aplicación nos encontramos con 4 paneles, dos superiores donde se mostrarán las imágenes de las huellas dactilares, dos inferiores, el izquierdo con los botones de las funcionalidades del programa, además del menú superior, y un panel que muestra en texto la descripción de los resultados mostrados.

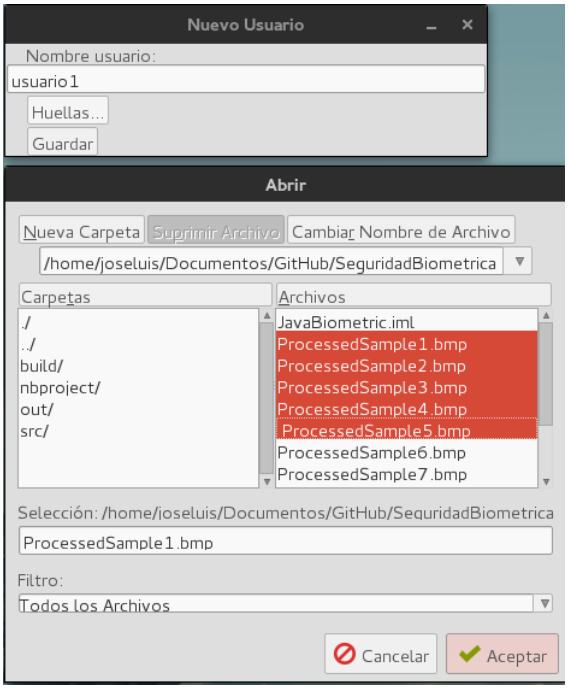
Los dos primeros botones, *Calcular 1 huella* y *Comparar 2 huellas* no dependen de la base de usuarios del programa.

Al pulsar el primero se abre un explorador de ficheros para abrir la foto de una huella dactilar. El programa entonces muestra la imagen seleccionada a la izquierda y la analizada a la derecha (se indica así en el cuarto panel), y su matriz en el cuadro de texto correspondiente del tercer panel.

El botón de *Comparar 2 huellas* abre dos veces el explorador de ficheros, y muestra las imágenes analizadas, sus matrices y el porcentaje de similitud entre las huellas.

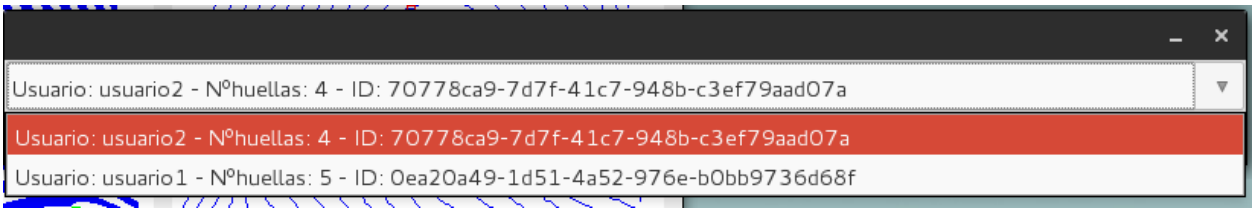
Ventana de la aplicación

Los siguientes botones sí usan la funcionalidad de usuarios del programa. Cuando pulsamos *Nuevo usuario* se abre una ventana nueva donde introducimos su nombre y seleccionamos tantas huellas como queramos para dicho usuario.



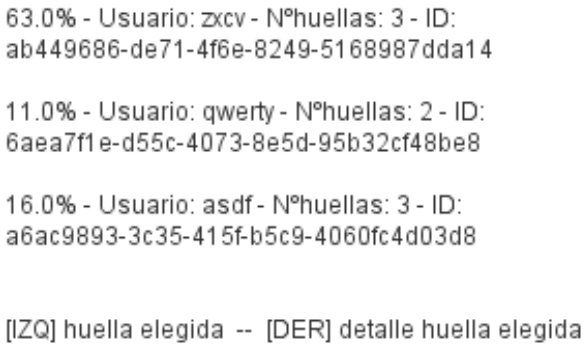
Ventana para registrar un nuevo usuario

Al pulsar en *1 to 1 Match* se realiza una **verificación** entre un fichero que selecciones en el explorador de ficheros y un usuario de entre los registrados, que se puede elegir de una lista desplegable en una nueva ventana.



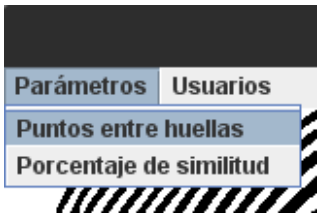
Listado de usuarios para la verificación

Al pulsar en *1 to N Match* se realiza la comparación de un nuevo fichero que se elija, y todos los usuarios registrados, mostrando la información de los usuarios con las huellas que superan el porcentaje de coincidencia. Se puede probar bajando a un 10 % dicho porcentaje, se mostrará una lista donde cada usuario aparece una sola vez, si alguna de sus huellas supera el porcentaje, y se indica el porcentaje de coincidencia de su huella más coincidente.

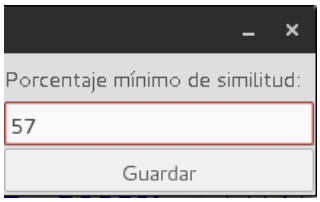


Listado de usuarios con huellas que superan el matching

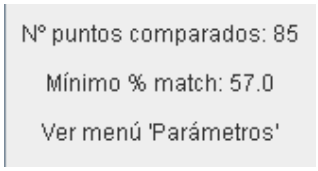
Por último tenemos las opciones de configuración del menú superior. Tenemos un menú de *Usuarios* con las mismas opciones que ya hemos visto, y un menú de *Parámetros* donde podemos modificar el número de puntos que usará el programa para comparar dos huellas dadas, y el porcentaje mínimo de similitud entre dos huellas para considerar que son de la misma persona.



Menú Parámetros



Ventana emergente para configurar el porcentaje de similitud



Cuadro de texto en el tercer panel con la información de los parámetros actuales

2.2. Implementación

Ahora veamos cómo lo hemos **implementado**.

En la carpeta *src* del proyecto encontramos el fichero *CatalogoUsuarios.java*, donde por un *HashMap* guardamos los usuarios registrados, haciendo uso de la clase *Usuario*, y ofreciendo métodos para la **identificación**, método *findMatch()*, que devuelve el conjunto de los usuarios que superan el porcentaje de similitud, como se ha descrito en el apartado de la interfaz de usuario.

El fichero *Usuario.java* implementa la representación de cada usuario registrado, mediante un identificador único (asignado automáticamente), un nombre y un conjunto de huellas, representadas por la clase *Huella*. La clase ofrece dos métodos para la **verificación**, uno que devuelve su huella que mejor coincidencia tiene con otra huella pasada como parámetro, y el otro método devuelve el *double* con el porcentaje de coincidencia.

El fichero *Huella.java* guarda la representación de las huellas dactilares, creándolas a partir del fichero pasado, ofreciendo los métodos para recuperar las imágenes, matriz de la huella calculada (como matriz o string) y el método *coincidencia()* que devuelve el **porcentaje** de coincidencia entre la huella y otra pasada por parámetro, usando tantos **puntos** para comparar como configure el usuario.

Con estos tres ficheros y la biblioteca *BiometrikSDK* tenemos lo necesario para manipular las huellas y usuarios.

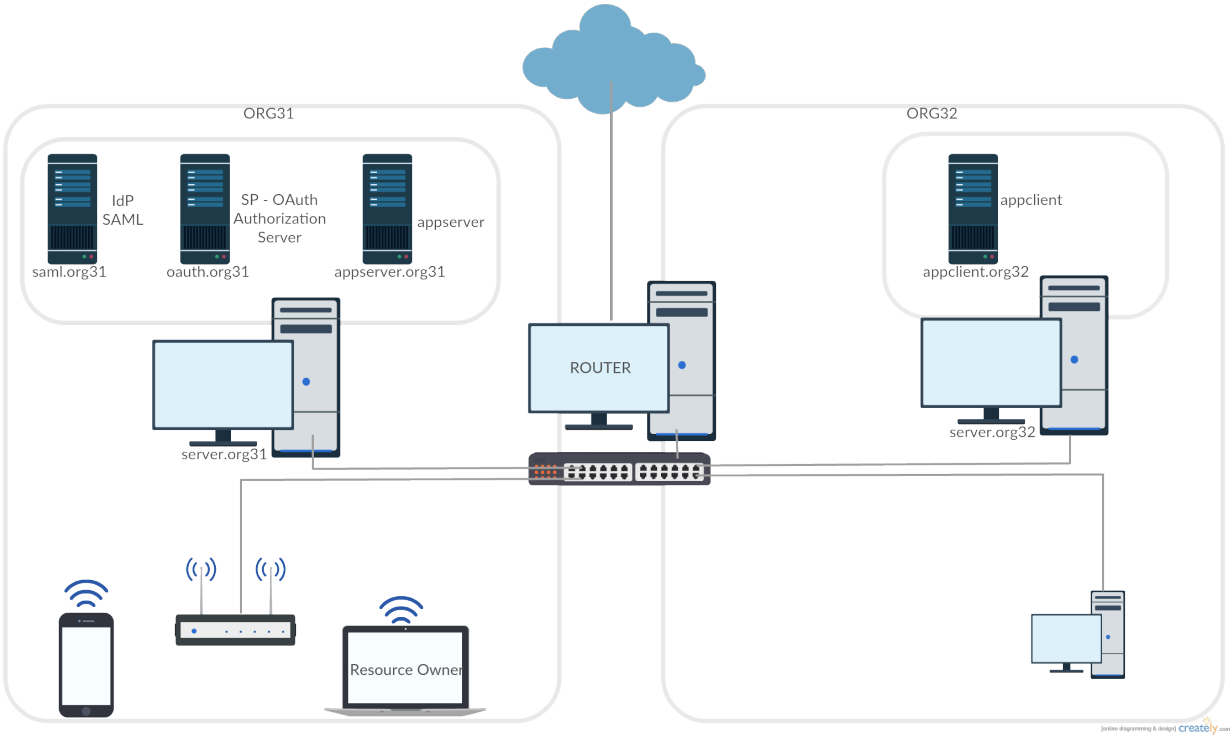
En el fichero *ProgramVariables.java* tenemos como parámetros globales los valores que puede ajustar el usuario, número de puntos de comparación entre huellas, y porcentaje mínimo de éxito.

En el fichero *NuevoUsuarioForm.java* se implementa la ventana de registro de un nuevo usuario, y en la acción del botón *Guardar...* se crea el nuevo objeto *Usuario* y se guarda en el catálogo de usuarios.

En el fichero *CEntityForm.java* se implementa el resto de la funcionalidad, la ventana principal del programa, junto a los métodos de los botones, que usan las clases del *CatalogoUsuarios*, *Usuario* y *Huella* para realizar los cálculos y muestran los resultados en el cuarto panel, *panelTexto*.

3. Escenario de gestión de identidad

Como grupo 3 desplegaremos las organizaciones 31 y 32. Partimos del escenario inicial de LEGO propuesto en el enunciado, con algunos cambios. En la siguiente imagen se muestra una imagen con la arquitectura a desarrollar.



Escenario LEGO de gestión de identidad

Se nos pide desarrollar un escenario donde el *Resource Owner*, el usuario por medio de su explorador web, se conecta a *appclient.org32*, donde quiere iniciar sesión. Al pinchar en *Login* la web *appclient* le redirige al servidor de **autorización**, donde pedirá poder acceder a los datos del usuario para iniciar sesión. Esto se realizará por OAuth con la máquina *oauth.org31*, estando aquí el principal cambio con respecto a la arquitectura propuesta, que comentaremos por qué más adelante. En este punto *oauth.org31* aún no sabe de qué usuario se trata, por eso necesita que iniciemos sesión, pero como *oauth.org31* no maneja los usuarios y credenciales, actúa como un *SP* usando SAML con el IdP en la máquina *saml.org31*, la cual sí tiene configurados los usuarios y contraseñas. Una vez nos **autenticamos** en *saml.org31*, como *Resource Owner* damos permiso en *oauth.org31* para dar los datos de inicio de sesión a *appclient.org32*.

Hemos cambiado *oauth* de la Organización 32 a la Organización 31 debido a las complicaciones de separar la implementación del proyecto de Phillip Shipley en varias máquinas. Consideramos que la estructura actual sigue representando una situación de organizaciones realista. Poniendo de ejemplo Google y Doodle, Doodle sería *appclient* en la Organización 32, y Google sería *oauth* como el servidor de autorización, el que muestra qué cosas compartirás con Doodle, como tu nombre y correo, también será *saml*, el IdP con el que iniciarás sesión, y puede añadir características extra como el *second factor of authentication*, y también el *appserver* por ejemplo con la lista de contactos u otro elemento del que hayas dado permisos en la autorización de *oauth*.

Como *saml.org31* y *oauth.org31* pertenecerían ambos a Google, pueden estar en la misma máquina.

De querer separar el SP y el IdP en 2 máquinas partiendo del proyecto de Shipley, el procedimiento sería, para la máquina del IdP, seguir los pasos de instalación referentes al servicio *saml*, para la máquina del SP, seguir los pasos de instalación referentes a los servicios *php-oauth* y *saml*, y ahora editar en los directorios *simplesamlphp/* instalados, los ficheros *config/authsources.php*, *metadata/saml20-idp-remote.php*, *metadata/saml20-idp-hosted.php* y *metadata/saml20-sp-remote.php*, eliminando de la máquina SP lo referente a la configuración del IdP, y de la máquina del IdP lo referente a la configuración del SP, donde podemos ayudarnos de las diapositivas de la práctica 2.

Repetimos que preferimos no hacerlo por falta de tiempo y porque hemos encontrado otras dificultades que no se podían dejar pasar en el diseño de un escenario realista.

3.1. Scripts de instalación

Para automatizar la configuración del escenario en cualquier máquina, hemos escrito una serie de scripts que no necesitan de interacción manual una vez se inicia la instalación (excepto algún *Enter* ocasional). Utilizamos la herramienta *make* para facilitar aún más el despliegue. La organización de los scripts corresponde a una jerarquía de directorios como la siguiente:

```
ROUTER
        Makefile
        network
        test

ORG1
        SERVER
                Makefile
                network
                test
                ESCENARIO
                        install.sh
                        php-oauth-saml-demo-master
                                appserver
                                php-oauth
                                simplesamlphp
                                vagrant

ORG2
        SERVER
                Makefile
                network
                test
                ESCENARIO
                        install.sh
                        php-oauth-saml-demo-master
                                appclient
                                vagrant
```

Donde ESCENARIO sirve como carpeta contenedora de los scripts específicos de esta práctica, mientras que otros ficheros como *network* nos permiten configurar LEGO automáticamente, configurando la red, el forwarding, la resolución de DNS o fichero *hosts*, etc.

En la práctica, para ejecutarlos, basta con irse al directorio correspondiente, por ejemplo, para configurar la Organización 31, navegamos a ORG1/SERVER/, y desde ahí ejecutamos la orden:

```
sudo make [network] [escenario] [saml] [test] [...]
```

Si no ponemos argumentos, por defecto aplicaría la configuración de la red y test. De querer instalar la práctica 2, deberíamos indicar *make saml*, y, para el escenario a entregar, *make escenario*.

Es importante notar que la configuración de la red (*make network*) no sólo configura las IP de cada máquina, así como el *forwarding* de la que haga de Router, sino que además configura el DNS de Google para navegar por internet, y en el fichero */etc/hosts* añade las entradas de *server.org31*, *server.org32*, *oauth.org31*, *appclient.org32*, etc., que se pueden ver en la imagen del escenario al inicio de esta sección. Esto último es de gran importancia porque así pasamos de usar los nombres **.local* del escenario de la práctica 3, a direcciones cualquiera que podrían estar en la jerarquía DNS, y al forzarnos a cambiarlos, nos daremos cuenta de cómo se podría mejorar este escenario de prácticas.

3.2. Configuración de la Organización 31: appserver, php-oauth y simplesamlphp

En el script de instalación en *ORG1/SERVER/ESCENARIO/* automatizamos la instalación de *apahce2*, *php*, la activación de **HTTPS**, y la copia de los ficheros de configuración de cada servicio.

Los servicios web se configuran al modificar el fichero *vagrant/vhost-local.conf*, eliminando la entrada de *appclient*, cambiando los *.local* por *.org31* y duplicándolo en *vagrant/vhost-secure.conf*, donde añadimos la configuración para **HTTPS**. Los certificados emitidos para *appserver.org31*, *oauth.org31* y *saml.org31* están en el directorio *CACERT/* y se copiarán en */etc/apache2/cert* en la instalación. Son autofirmados, pero al menos van emitidos a los nuevos nombres de dominio de cada servicio.

En la configuración de *appserver*, *php-oauth* y *simplesamlphp* hemos buscado en todos los ficheros de sus directorios cualquier referencia a *appserver.local*, *appclient.local*, *php-oauth.local* y *simplesamlphp.local*, para cambiarlos por *appserver.org31*, *appclient.org32*, *php-oauth.org31* y *simplesamlphp.org32*, o cualquier otro nombre de dominio que les hubiéramos asignado para poder acceder desde fuera de la máquina donde están instalados.

Sin embargo, esto no fue suficiente, porque en las pruebas durante el desarrollo del script, una vez autorizábamos en *oauth.org31* el uso de nuestros datos para *appclient.org32*, nos redireccionaba a *appclient.local*, y buscando incluso en los ficheros ya configurados por el *composer.phar* y en toda la documentación que había disponible, no salían más referencias a *appclient.local*.

El problema radicaba en el fichero de base de datos de sqlite en *php-oauth/data/oauth2.sqlite*, donde el binario, usando un explorador de base de datos, vimos que en la tabla *Client*, tenía 3 entradas para la configuración de *oauth*, donde había referencias a las direcciones *.local*.

Una vez modificamos la base de datos con las nuevas direcciones, el escenario funcionó correctamente.

En cuanto a **HTTPS**, hemos añadido su configuración y si accedemos a cada página con *https://...*, aparte del aviso de que es un certificado autofirmado, todo funciona perfectamente. El problema es, en cada fichero que hemos modificado para eliminar los *.local*, algunas líneas eran parte de las órdenes *redirect*, y llevan explícitamente *http://*, de modo que para que el escenario de pruebas funcione con HTTPS automáticamente, deberíamos localizar los *redirect* y modificarlos por *https://* y volver a aplicar el script de instalación y reiniciar el servicio.

No lo hacemos porque entonces no podríamos tomar las muestras de las trazas con wireshark, pues ahora se usaría forzadamente HTTPS.

Ahora, en cuanto a la configuración de SAML para *saml.org31* como IdP y para *oauth.org31* como SP, debemos modificar algunos ficheros más, varios ya modificados al cambiar los *.local*.

En *simplesamlphp/metadata/* modificamos:

- *saml20-idp-hosted.php* para indicar que los nuevos ficheros del certificado de firma son *saml.org31.pem* y *saml.org31.crt*, que son los mismos que usa HTTPS para *saml.org31*, autofirmados, pero que usará SAML para firmar los mensajes, no para cifrar la comunicación.
- *saml20-idp-remote.php* para modificar los datos criptográficos que tiene el SP sobre el IdP (pues le hemos dado un nuevo certificado al IdP). Lo hacemos copiando y pegando el *$metadata* que da la web de *simplesamlphp*, como hicimos en la práctica 2.

En *simplesamlphp/config/authsources.php* añadimos la configuración de **Facebook**, la única aplicación externa de las que ofrece *simplesamlphp* que no tiene una configuración obsoleta, es decir, que al añadir su *api_key* y su *secret*, consigue conectarse a Facebook e identificarse como la app que hemos generado previamente en el área de desarrolladores. También probamos con Twitter, que al probarlo se queda una web en blanco, pues la URL de llamada de vuelta no parecía poder configurarse correctamente. Google hace tiempo que ya no soporta Open-ID, por lo que tampoco podíamos configurarlo. Windows Live tampoco existe ya, el servicio pasó a Onedrive y *simplesamlphp* tiene todas la URL mal configuradas. Y Yubikey tampoco funcionaba al configurar su *id* y *key*, pero ignoramos qué fallo tendrá *simplesamlphp* para no funcionar.

Para terminar de configurar **Facebook**, debemos crear el fichero *simplesamlphp/modules/authfacebook/enable*, que puede estar vacío, para que el módulo se active.

Si nos conectamos a *saml.org31*, en la pestaña *Autenticación*, en el enlace *Probar las fuentes para la autentificación ya configuradas*, podemos comprobar que funciona la identificación por Facebook, y que una vez autorizamos a usar nuestros datos, Facebook funciona como IdP y servidor de autorización, y nuestro test como *appclient*.

3.3. Configuración de la Organización 32: appclient

Si nos vamos al directorio *ORG2/SERVER/ESCENARIO/* nos encontramos el script de instalación que automatiza el proceso. Nos instala *apache2*, activa el cifrado para usar **HTTPS**, instala las dependencias de *php5*, y copia los ficheros de *appclient*, modificados como ahora indicaremos, de los *virtual hosts* de Apache2 y los certificados.

Configura el directorio de *appclient* con php y reinicia el servicio de Apache2.

Todo muy sencillo y sin intervención humana.

Ahora bien, hemos tenido que modificar algunos ficheros de como estaban en el repositorio original. En los ficheros *protected/config/main.php* y *protected/controllers/AuthController.php* tenemos que modificar todas las referencias de *oauth.local*, *appserver.local* y *appclient.local* por las direcciones *oauth.org31*, *appserver.org31* y *appclient.org32*, o el nombre real del DNS que tuviera cada máquina.

Por supuesto, hemos cambiado también el fichero *vagrant/vhost-local.conf* para borrar los *virtual host* de *oauth*, *saml* y *appserver*, y cambiar *appclient.local* por *appclient.org32*.

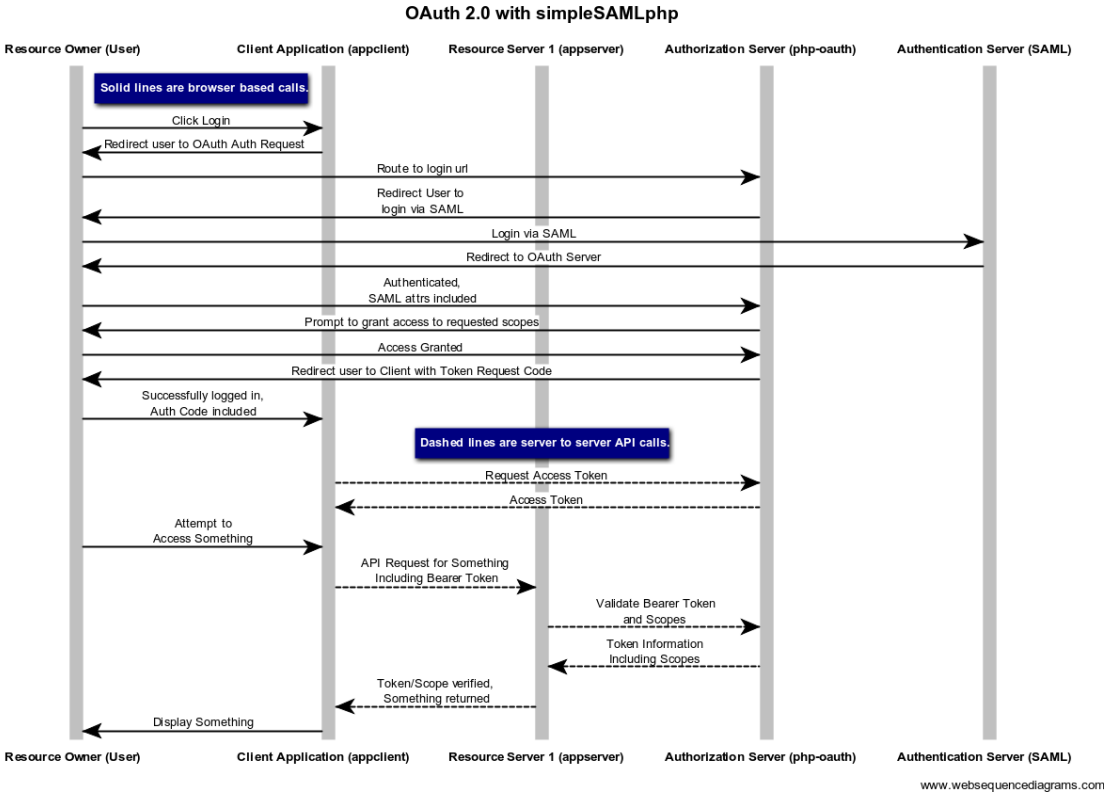
El certificado generado para *CN=appclient.org31* se encuentra en el directorio *CACERT/* del proyecto, y es el que se copia en */etc/apache2/cert* para que la configuración de los *virtual host* con SSLEngine los utilice en HTTPS.

Con estos cambios y el script la máquina *server.org32* sirve la web *appclient.org32* correctamente a cualquier otra máquina de la red. **De haber dejado *appclient.local* fijo**, sin cambiarlo por *appclient.org32*, o por cualquier otro nombre resoluble por DNS, **el escenario sólo funcionaría si el cliente, *Resource***

Owner* se conectase desde la misma máquina donde está instalado *appclient, y consideramos ese caso mucho menos realista que tener los servicios *saml*, *php-oauth* y *appserver* en la misma máquina.

4. Análisis de trazas

Para entender el intercambio de mensajes podemos fijarnos en la siguiente imagen:



4.1. SAML

Normalmente capturaríamos las trazas con Wireshark, y las interpretaríamos. Sin embargo, existe una herramienta llamada SAML Tracer (es un complemento de Firefox) que, si se utiliza mientras hacemos el login Single Sign On, muestra los intercambios y permite extraer el intercambio SAML. Con este intercambio podemos ahorrar tiempo ya que SAML viaja a través de HTTP, y no tenemos intención de analizar las trazas HTTP.

Si entramos en appclient.org32, hacemos login (que nos redirige a saml.org31, el IdP), y nos autenticamos como **user1**, se realizan dos intercambios SAML. El primero, es un intercambio de appclient de org32 para saml de org31, indicando que alguien quiere autenticarse, y el segundo, después de haber hecho login en saml.org31, va desde saml en org31 para appclient en org32, indicando información del usuario.

Para ver la traza hay que abrir **anexos/traza.txt**, ya que el documento tiene líneas demasiado largas para aparecer correctamente en el documento.

De esta traza destacamos los siguientes apartados XML:

- <samlp:AuthnRequest ...> indica una petición de autenticación, con ID=_49881caa8878bdd61c79da8c16d70a77d493e2e3
  Este id es importante, es al que tendrá que responder el IdP.

- Dentro del samlp:AuthnRequest el Destination=http://saml.org31/saml2/idp/SSOService.php indica a donde tiene que dirigirse el cliente a completar la petición.
- Dentro también del samlp:AuthnRequest se encuentra AssertionConsumerServiceURL=http://oauth.org31/module.php acs.php/oauth-sp, a donde habrá que volver una vez obtenida la respuesta.
- <samlp:Response ...> es la etiqueta XML para la respuesta a la petición. Se sabe que es respuesta a la petición anterior porque el campo InResponseTo=_49881caa8878bdd61c79da8c16d70a77d493e2e30e coincide exactamente con el ID de la petición.
- En ds:Signature y en las etiquetas hijas, está la firma de la respuesta. La respuesta la firma digitalmente el IdP para que los servicios puedan fiarse de la información que el cliente está transportando. Contiene información de como se firma el mensaje, así cómo el certificado del IdP.
- El StatusCode [...] Success indica que el intento de login fue correcto.
- Al final de la traza, hay unos atributos que indican el nombre de usuario, los grupos a los que pertenece (person, member) y sus roles (applications, administration).

4.2. Oauth

Para esta parte sí que hemos usado Wireshark. Sin embargo, como viaja por HTTP, adjuntamos el payload HTTP, ignorando las cabeceras TCP, direcciones (porque van implícitos los hosts en HTTP, etc...):

```
POST /token.php HTTP/1.1
Host: oauth.org31
User-Agent: Guzzle/3.9.1 curl/7.35.0 PHP/5.5.9-1ubuntu4.14
Accept: application/json
Authorization: Basic YXBwY2xpZW50OmNyYXp5c2VjcmV0
Content-Length: 152

HTTP/1.1 200 OK
Date: Sun, 10Apr 2016 17:40:04 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Cache-Control: no-store
Pragma: no-cache
Content-Length: 180
Content-Type: application/json

{"access_token":"c33869b8f9da339633282556832 5910a","expires_in":3600,"scope":"documents profile tokenin

GET /api/profile HTTP/1.1
Host: appserver.org31
User-Agent: Guzzle/3.9.1 curl/7.35.0 PHP/5.5.9-1ubuntu4.14
Accept: application/json
Authorization: Bearer c33869b8f9da3396332825568325910a

HTTP/1.1 200 OK
Date: Sun, 10Apr 2016 17:40:04 GMT
Server: Apache/2.4.7 (Ubuntu)
Vary: Authorization
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Content-Length:45
```

```
Content-Type: application/json

{"name":"testing","email":"testing@test.com"}

GET /api/tokenInfo HTTP/1.1
Host: appserver.org31
User-Agent: Guzzle/3.9.1 curl/7.35.0 PHP/5.5.9-1ubuntu4.14
Accept: application/json
Authorization: Bearer c33869b8f9da3396332825568325910a

HTTP/1.1 200 OK
Date: Sun, 10Apr 2016 17:40:04 GMT
Server: Apache/2.4.7 (Ubuntu)
Vary: Authorization
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Content-Length:454
Content-Type: application/json

{"active":true,"exp":1460313604,"iat":1460310004,"scope":"documents profile tokeninfo","client_id":"app

GET /api/profile HTTP/1.1
Host: appserver.org31
User-Agent: Guzzle/3.9.1 curl/7.35.0 PHP/5.5.9-1ubuntu4.14
Accept: application/json
Authorization: Bearer c33869b8f9da3396332825568325910a



HTTP/1.1 200 OK
Date: Sun, 10Apr 2016 17:40:04 GMT
Server: Apache/2.4.7 (Ubuntu)
Vary: Authorization
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Content-Length:45
Content-Type: application/json

{"name":"testing","email":"testing@test.com"}

GET /api/documents HTTP/1.1
Host: appserver.org31
User-Agent: Guzzle/3.9.1 curl/7.35.0 PHP/5.5.9-1ubuntu4.14
Accept: application/json
Authorization: Bearer c33869b8f9da3396332825568325910a



HTTP/1.1 200 OK
Date: Sun, 10Apr 2016 17:40:04 GMT
Server: Apache/2.4.7 (Ubuntu)
Vary: Authorization
X-Powered-By: PHP/5.5.9-1ubuntu4.14
```

```
Content-Length:214
Content-Type: application/json

{"name":"testing","email":"testing@test.com","documents":{"1":{"name":"Document 1","type":"PDF","last_m

GET /api/notAllowed HTTP/1.1
Host: appserver.org31
User-Agent: Guzzle/3.9.1 curl/7.35.0 PHP/5.5.9-1ubuntu4.14
Accept: application/json
Authorization: Bearer c33869b8f9da3396332825568325910a



HTTP/1.1 403 Forbidden
Date: Sun, 10 Apr 201617:40:04 GMT
Server: Apache/2.4.7 (Ubuntu)
Vary: Authorization
X-Powered-By:PHP/5.5.9-1ubuntu4.14
Content-Length: 88
Content-Type: application/json

{"success":"false","error":"You donot have the required scope: notallowed.","code":403}

GET /api/profile HTTP/1.1
Host: appserver.org31
User-Agent: Guzzle/3.9.1 curl/7.35.0 PHP/5.5.9-1ubuntu4.14
Accept: application/json
Authorization: Bearer invalidtoken



HTTP/1.1 401 Unauthorized
Date: Sun, 10 Apr 2016 17:40:04 GMT
Server: Apache/2.4.7 (Ubuntu)
Vary: Authorization
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Content-Length: 108
Content-Type: application/json

{"success":"false","error":"invalid_token, theaccess token isnot active (Bearer invalidtoken)","code":4
```

El primer intercambio lleva el Authorization: Basic YXBwY2xpZW50OmNyYXp5c2VjcmV0, que en base64 es appclient:crazysecret. La respuesta a esa petición al servidor OAuth lleva la información de los datos a los que se cede el acceso, la fecha de caducidad y el propio token que da el acceso. El resto de los intercambios llevan la cabecera Authorization: Bearer c33869b8f9da3396332825568325910a, porque ese es el token que ha dado el servidor OAuth al servidor appclient. En esos intercambios se piden distintos recursos (y también se prueban los errores, como por ejemplo, un documento al cual no se tiene autorización o una petición sin incluir el token).

Estos datos que se intercambian son los que aparecen en la web después de acabar con el Login y la autorización.

5. Conclusiones

El escenario a configurar en el proyecto LEGO creemos que es bastante realista y complejo, sin pasarse en cantidad de servicios a configurar, como para entender bien la diferencia entre SAML para **autenticación** y de OAuth para **autorización**, y de cómo pueden trabajar conjuntamente, siendo cómodos de usar para el usuario final, y transparentes en cómo está realmente desplegado, permitiendo, en un escenario real, complicarlo lo que se quiera, de cara a añadir prestaciones y seguridad.

Sin embargo, siendo el escenario muy correcto, no creemos que el punto de partida del proyecto de Phillip Shipley sea el idóneo. Sí lo es para la práctica 3, donde con un script de *vagrant* se tiene el escenario de una máquina totalmente desplegado. Pero en la parte de llevarlo al mundo real, no tiene tan bien definidos en de los ficheros, qué configuración pertenece a qué servicio, y es muy difícil descubrir que hay referencias a los nombres **.local* incluso en ficheros binarios de bases de datos.

De cara al siguiente curso que deba desplegar esta práctica, recomendaríamos que se modificara el proyecto de Shipley. Por medio de un fichero global de configuración, se mapearían las direcciones de cada servicio con su nombre de dominio final, por ejemplo con el formato:

```
$appclient = cliente.um.es
$appserver = servidor.com
$oauth = authZ.edu
$saml = idp.org
```

Y con un script (bash, python, ...) que lo leyera y modificara cada fichero que hemos mencionado en esta memoria (donde hemos cambiado un *.local* por un *.org3N*).

Un modo sería aprovechar que son ficheros php, cambiar las referencias a *appclient.local*, etc., por variables, y que el script modifique lea los nombres de dominio y modifique las variables.

Todavía quedaría el tema de la base de datos de *php-oauth*, la cual modificaríamos por sólo la definición, y que en la instalación, al igual que hacemos un *php composer.phar install*, añadir un script que rellene la base de datos.

Al final para el alumno quedaría un proyecto tan cómodo como el de Shipley, pero donde por un fichero global puede separarlo en varias máquinas, por medio de una mejor documentación del apartado de *simplesamlphp*, puede separar *oauth* como SP de *saml* como IdP. Quedaría entonces un escenario más realista, donde el alumno aún debe pelearse con OAuth y SAML, identificar sus *roles*, pero no se pelea con partes del proyecto que no tienen relevancia real en la parte de autenticación y autorización.