

UNIVERSIDAD DE MURCIA

GRADO EN INGENIERÍA INFORMÁTICA

4º CURSO

GRUPO 6

CURSO 2016/2017 - JUNIO

---

# Seguridad

---

## Práctica final

---

Alumnos:

Cristian Roche Borja

DNI: 76581531H

Alicia Ruiz Tovar

DNI: 48693813F

Docentes:

Alberto Huertas Celdrán

Gabriel López Millán

Gregorio Martínez Pérez





# Índice

<b>1. Oauth</b>	<b>4</b>
1.1. Introducción . . . . .	4
1.2. Adaptación a la topología . . . . .	4
1.3. Implementación . . . . .	5
1.3.1. URLs y Servidores . . . . .	7
1.3.2. Secuencia de Funcionamiento Interno . . . . .	9
1.3.3. Manual de Uso . . . . .	10
<b>2. NMAP y Metasploit</b>	<b>11</b>
2.1. Víctima . . . . .	11
2.2. Atacante . . . . .	11
2.2.1. NMAP . . . . .	11
2.2.2. NMAP con Metasploit . . . . .	13
2.2.3. Wireshak: trazas . . . . .	13
2.3. Scripts NMAP . . . . .	15
2.3.1. Scripts /usr/share/nmap/scripts . . . . .	15
2.3.2. Realización de script básico . . . . .	15
2.3.3. Comando Portscan . . . . .	18
<b>3. Explotar Vulnerabilidades</b>	<b>21</b>

# 1. Oauth

## 1.1. Introducción

Open Authorization(Oauth) es un mecanismo un mecanismo de autorización, está estandarizado y es abierto, es decir, disponible públicamente para su uso. El protocolo que implementa Oauth se basa en la *autorización delegada*, se hace uso de una tercera entidad para autenticar al usuario, sin embargo, esta es la utilidad más básica, una vez autenticado a un usuario, se pueden solicitar su información asociada.

Desde el punto de vista del usuario final, este sistema supone una gran ventaja, no necesita de múltiples credenciales en cada uno de los sitios web que frecuenta, basta con disponer de una cuenta de usuario válida en un servidor de autenticación, es el caso de Google, Facebook, etc... grandes empresas que ofrecen este servicio a sus usuarios.

Desde el punto de vista de una mediana empresa, resulta muy sencillo beneficiarse de este mecanismo llegando a acuerdos con empresas de autenticación, como las ya mencionadas. La inclusión en los servicios web es muy sencilla y basta con poner un botón opcional en la página que solicita las credenciales de acceso. El usuario final demostrará su identidad e incluso puede permitir el acceso diferentes apartados(scopes) de su información personal, que está siendo gestionada actualmente por Google, Facebook...

## 1.2. Adaptación a la topología

En nuestro grupo de prácticas, trabajamos dando servicio a un bufete de abogados, cubriendo las necesidades de los trabajadores y de los clientes del bufete. Consideremos que Oauth puede facilitar en gran medida las gestiones online de los clientes, permitiendo la compartición de sus datos con diferentes empresas colaboradoras. No supone una violación de la privacidad del cliente, ya que será este en cada momento quien decida que empresas pueden acceder a su información.

Para que Oauth tenga una aplicación real en nuestra topología, es necesario incluir en la escena a una segunda empresa colaboradora, como trabajamos con abogados, es interesante la inclusión de una notaría "Notaría Arrigaga Asociados". Como trabajamos con un bufete con un buen número de abogados, disponemos de una infraestructura informática importante, la información de nuestros clientes ya está almacenada en nuestros servidores, la dirección ha decidido que el bufete implantará el servicio de autenticación con Oauth. Cualquier empresa colaboradora que esté interesada en hacer uso de nuestro servicio tendrá que firmar un acuerdo, en el que se incluye las garantías de confidencialidad y los aspectos económicos.

En el momento de la implantación, solo se ha llegado a un acuerdo con la notaría anteriormente mencionada, asumiendo cada actor los siguientes roles:

- Bufete de abogados: Será el encargado de la identificación(*IDP Server*), autorización(*Authorization Server*) y de servir los recursos(*Resource Server*).
- Notaría: Actuará como cliente (*Client*).

- **Usuario final:** Será quién hace uso del servicio mediante su navegador web y quien facilite el acceso a sus recursos(*ResourceOwner*).

En esta práctica la implementación de Oauth no solo se limitará a la autenticación de los clientes del bufete, sino que también se utilizará para facilitar parte de la información personal de los mismos, facilitando las gestiones.

### 1.3. Implementación

Utilizaremos una implementación basada en el esquema *implicit* como se muestra en la figura 1. Esta modalidad tiene como característica principal, que es el navegador del propietario de los recursos (Resource Owner) el medio utilizado para interactuar con las diferentes entidades. Modificaremos ligeramente el esquema separando la identificación y la autenticación en dos servidores independientes.

Una vez que el *User agent* haya realizado todas las peticiones previas, el *Client*, en este caso corresponde con la *Notaría*, podrá realizar consultas de la información del *Resource owner* de forma directa en el *Client resource*, este permiso tendrá una duración de 30 segundos, expirado ese periodo, el permiso caducará. Aunque la duración de dicho permiso pueda parecer corta, el fin de esta implementación es que la Notaría pueda consultar casi de forma instantánea los datos del usuario, para rellenar un formulario con su información personal, obtener algún documento, etc...

Para la realización de esta práctica, hemos implementado los servicios con Java Rest, que presenta las siguientes ventajas:

1. Alto rendimiento, soportando un gran número de peticiones simultáneas.
2. Claridad en el código. Se ve de forma muy clara el tipo de función implementada y el resultado que genera.
3. Permite la inclusión y lectura de cabeceras HTTP de forma muy clara y en una sola línea de código.
4. Uso dinámico de las URLs, pudiendo usarlas para recibir parámetros que forman parte de la propia URL. (<http://www.notaria.es/userID/userPhone/>)
5. Dispone de un amplio soporte y documentación para la resolución de problemas.
6. La librería de Oauth esta implementada también en Java Rest, por lo que se hace más sencilla la adaptación.

A la hora de resolver las dependencias con otras librerías, hemos optado por el uso de *Maven*, que tiene las siguientes ventajas:

1. Soportado de forma nativa en Eclipse.

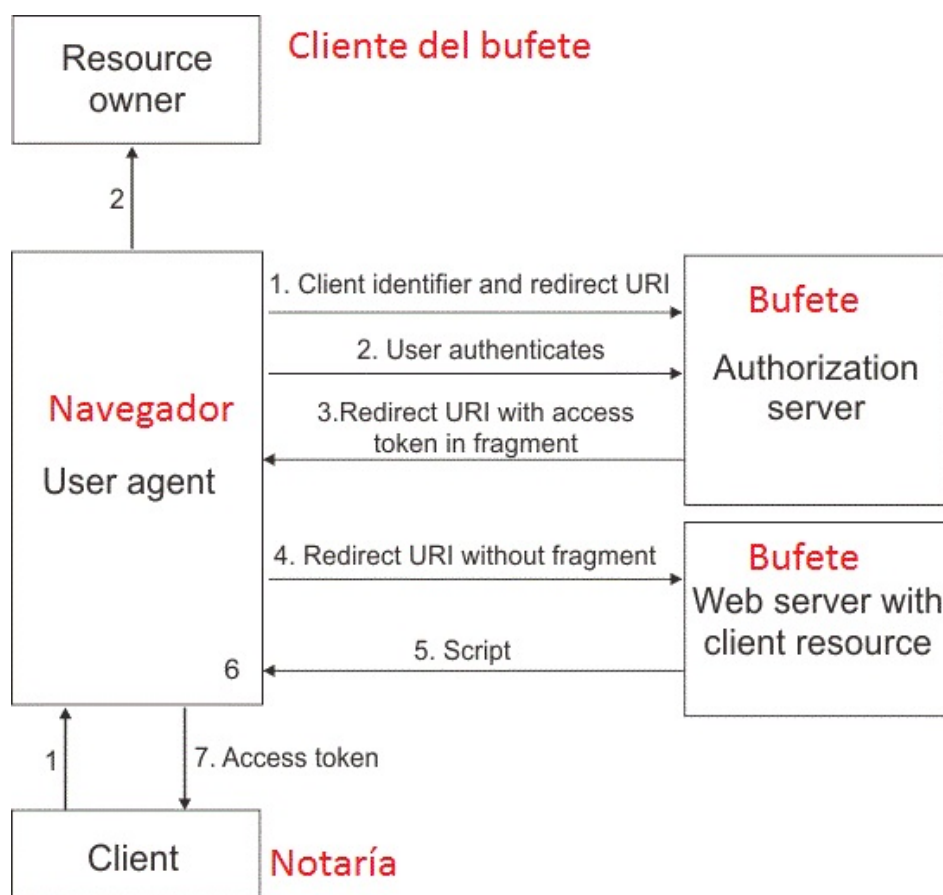


Figura 1: Esquema implicit.

2. Fácil conversión de un proyecto tradicional a uno maven.
3. Amplio soporte y documentación de uso.
4. No es necesario descargar las librerías, solo se incluyen las referencias y se descargan automáticamente.
5. Basándonos en el punto anterior, el tamaño del proyecto final es muy pequeño.
6. Maven dispone de la librería de Oauth en su repositorio.

### 1.3.1. URLs y Servidores

A continuación, se muestra una lista con los diferentes servidores que hemos implementado, así como las URLs de las que dispone cada uno. Así se podrá entender de forma más clara el funcionamiento de nuestro esquema.

- `ServidorWeb Notaría(Client)`
  - `/Seguridad.NotariaWeb/rest/Index GET`  
Página de inicio de la Notaría, en la que se puede visualizar el índice de contenidos.
  - `/Seguridad.NotariaWeb/rest/Formulario GET`  
Muestra un mensaje indicando que el formulario al que se ha accedido necesita consultar datos personales del usuario, después de 5 segundos realiza una redirección. Si la Notaría no se encontraba registrada, realiza el registro antes de redirigir.
  - `/Seguridad.NotariaWeb/rest/Resultado GET`  
Se llama cuando el usuario ya ha obtenido el *AccessToken* del servidor de autorización. Cuando recibe el *AccessToken* realiza las consultas al servidor de recursos. Como resultado, se muestra un formulario al usuario autocompletado con sus datos personales.
- `Servidor Identificación(IDP)`
  - `/Seguridad.IDP/rest/OauthRegistro POST`  
Recibe las peticiones de registro, registra al *Client* y envía la respuesta.
  - `/Seguridad.IDP/rest/Identifier GET`  
Muestra una página web en la que el usuario debe introducir la imagen de su huella digital y su DNI/ Contraseña. Al pulsar enviar, se realiza un POST a la misma URL.
  - `/Seguridad.IDP/rest/Identifier POST`  
Recibe la información de identificación del usuario, valida por niveles cada uno de los datos, la detección de un dato no válido implica no comprobar los

siguientes. En primer lugar el DNI/Contraseña, ya que es el menos costoso desde el punto de vista computacional. Si se superan todos los niveles de identificación, se genera una *Cookie* para el cliente.

- */Seguridad.IDP/rest/Identifier/redirect POST*  
Transforma la petición POST del cliente, que en este momento contiene la imagen de la huella y el usuario y contraseña, en una petición GET dirigida al servidor de autorización.
- */Seguridad.IDP/rest/CheckAuthCookie POST*  
Esta página es invocada por el servidor de autorización para validar la *Cookie* que el *Client* le ha presentado. Recibe la *Cookie* en formato Json y devuelve el *ID* del usuario asociado a esa *Cookie*, en el caso de que no sea válida, mostrará un error 401 *Unauthorized*.
- */Seguridad.IDP/rest/Error GET*  
Página de error mostrada para indicar al usuario el fallo de alguno de los pasos.

■ Servidor Autorización (Authorize)

- */Seguridad.Authorize/rest/Authorize GET*  
Recibe la *Cookie* y los datos de identificación del *Client*. En primer lugar, se verifica que los datos del *Client* sean válidos, posteriormente se valida la *Cookie* presentada (Esta validación se realiza lanzando una consulta al IDP */Seguridad.IDP/rest/CheckAuthCookie*). Si toda la información es correcta, se genera un *AccessToken* y se redirecciona.
- */Seguridad.Authorize/rest/CheckAccessToken POST*  
Esta página es invocada por el servidor de recursos para validar el *AccessToken* que el *Client* le ha presentado. Recibe el *AccessToken* en formato Json y devuelve el *ID* del usuario asociado a ese *AccessToken*, en el caso de que no sea válido, mostrará un error 401 *Unauthorized*.

■ Servidor Recursos (Resource)

- */Seguridad.ResourceServer/rest/DNI GET*  
Consulta al servidor de autorización la validez del *AccessToken* presentado, en esta consulta recibe el ID del usuario asociado. Devuelve el número de DNI en formato Json.
- */Seguridad.ResourceServer/rest/Name GET*  
Consulta al servidor de autorización la validez del *AccessToken* presentado, en esta consulta recibe el ID del usuario asociado. Devuelve el nombre en formato Json.
- */Seguridad.ResourceServer/rest/Surname GET*  
Consulta al servidor de autorización la validez del *AccessToken* presentado,



en esta consulta recibe el ID del usuario asociado. Devuelve el apellido de DNI en formato Json.

- */Seguridad\_ResourceServer/rest/Phone GET*  
Consulta al servidor de autorización la validez del *AccessToken* presentado, en esta consulta recibe el ID del usuario asociado. Devuelve el número de teléfono en formato Json.
- */Seguridad\_ResourceServer/rest/Address GET*  
Consulta al servidor de autorización la validez del *AccessToken* presentado, en esta consulta recibe el ID del usuario asociado. Devuelve la dirección en formato Json.
- */Seguridad\_ResourceServer/rest/Email GET*  
Consulta al servidor de autorización la validez del *AccessToken* presentado, en esta consulta recibe el ID del usuario asociado. Devuelve el email en formato Json.
- */Seguridad\_ResourceServer/rest/BirthDate GET*  
Consulta al servidor de autorización la validez del *AccessToken* presentado, en esta consulta recibe el ID del usuario asociado. Devuelve la fecha de nacimiento en formato Json.

### 1.3.2. Secuencia de Funcionamiento Interno

La imagen 2 muestra un diagrama de flujo, en este diagrama se muestra gráficamente todos los mensajes enviados y recibidos desde el *User Agent* y el *Client*. Por simplificar el diagrama, solo se muestra la obtención del DNI, en la resolución práctica se realizan consultas de cada uno de los datos del usuario: nombre, fecha de nacimiento, DNI, apellido, dirección, email, fecha de nacimiento y número de teléfono.

1. El usuario accede con su navegador a la página web principal de la Notaría.
2. El usuario intenta acceder al formulario de solicitud de hipoteca, este formulario precisa de la autenticación del usuario y su información personal. Como el usuario ya es miembro del bufete de abogados (empresa colaboradora), se delega en ellos la autenticación y la obtención de recursos. Por tanto, redirigimos al usuario *Redirect.html*.
3. La primera parada es el servidor de identificación, este devolverá al cliente una página web con un formulario de indentificación *Identifier.html*. El usuario completará la información requerida y pulsará en el botón *Enviar*.
4. Se recibe la información de identificación en el servidor IDP, se valida, se genera la *CookiePass* y se responde con una redirección *Redirect.html*.

5. Esta última redirección simplemente convierte el mensaje POST que aún contiene la huella y el DNI y contraseña en un mensaje GET, redirigiendo nuevamente al servidor de autorización *Authorize.html*.
6. El servidor de autorización recibe la *CookiePass*, verifica que la petición sea lícita, valida la *CookiePass* (consultando al IDP si es válida), genera un *AccessToken* y realiza una redirección a la web de la Notaría *Resultado.html*.
7. Llegados a este punto, la Notaría ya tiene el código con el que puede realizar las consultas en el servidor de recursos, en cada petición presentará el *AccessToken*. Por ejemplo, *dni.html* para obtener el DNI.
8. El servidor de recursos solo recibe el *AccessToken*, no necesita recibir más información, porque al consultar al servidor de autorización si es válido, este puede contestar o bien, diciendo que no es un código válido *401 Unauthorized* o, con el ID del usuario al que pertenece ese *AccessToken* en formato Json. Teniendo el ID del usuario, el servidor de recursos consulta en su base de datos y responde con los datos solicitados en formato Json.
9. La Notaría está programada de tal forma que realizará tantas peticiones como sean necesarias para obtener la información requerida por el formulario. Como resultado, se mostrará el formulario al usuario final totalmente autocompletado *Resultado.html(Autocompleted)*.

### 1.3.3. Manual de Uso

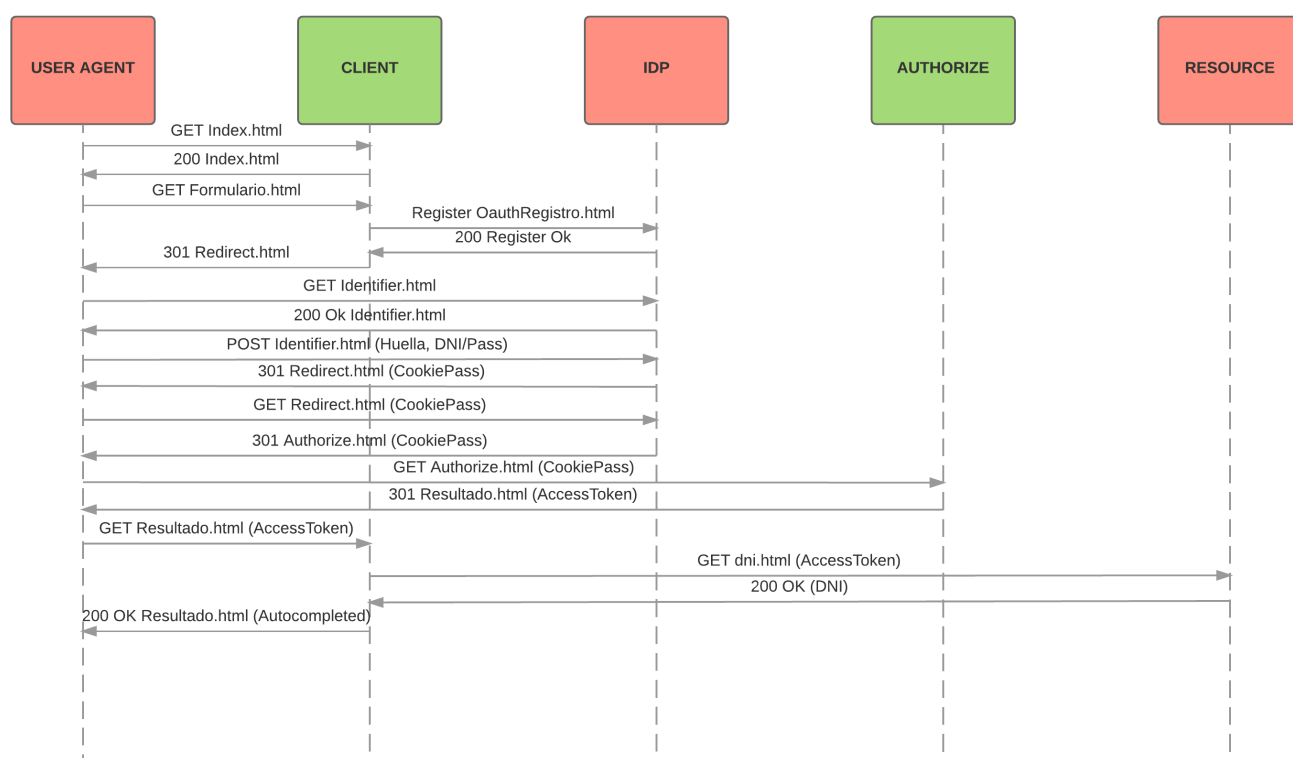


Figura 2: Diagrama User Agent.

## 2. NMAP y Metasploit

### 2.1. Víctima

Utilizaremos una máquina virtual de prueba. Esta máquina ha sido creada con vulnerabilidades para la práctica de ataques. La URL de descarga es la [siguiente](#).

La IP de esta máquina es la 192.168.62.189.

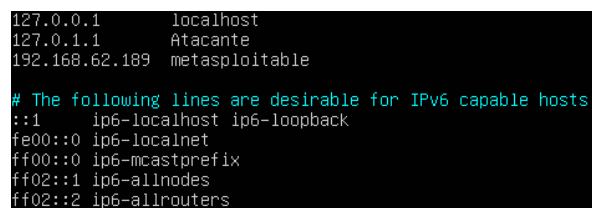
### 2.2. Atacante

#### 2.2.1. NMAP

El equipo que actuará como atacante hace uso de la herramienta NMAP. Para instalarla ejecutamos el siguiente comando:

```
$ sudo apt-get install nmap
```

Establecemos en el archivo `/etc/hosts`, equivalente al DNS local, la IP de la víctima (192.168.62.189) y la denominamos `metasploitable`, como muestra la figura 3.



```
127.0.0.1    localhost
127.0.1.1    Atacante
192.168.62.189 metasploitable

# The following lines are desirable for IPv6 capable hosts
::1         ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
```

Figura 3: Atacante\_dns\_victima.

De esta forma, tenemos dos opciones para hacer referencia a la víctima. En la figura 4 se observa el resultado de este escaneo simple fruto de cualquiera de estas dos opciones.

```
$ nmap 192.168.62.189
$ nmap metasploitable
```

De forma un poco más elaborada, se puede ejecutar el escaneo de puertos haciendo uso de otras técnicas:

- Mediante listado de equipos: `$ nmap 192.168.62.1 192.168.62.10 192.168.62.189`
- Mediante subred: `$ nmap 192.168.62.0/24`
- Mediante un fichero que almacene las IPs (o las expresiones de las mismas) a analizar: `$ nmap -iL hosts.txt`, como muestra la figura 5.

```
Starting Nmap 7.01 ( https://nmap.org ) at 2017-04-23 13:06 CEST
Nmap scan report for 192.168.62.189
Host is up (0.0010s latency).
Not shown: 977 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
8009/tcp  open  ajp13
8180/tcp  open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.27 seconds
alumno@Atacante:~$
```

Figura 4: Atacante\_nmap\_simplescan.

```
alumno@Atacante:~$ cat hosts.txt
192.168.62.189
192.168.62.1
alumno@Atacante:~$ cat hosts2.txt
192.168.61.0/24
metasploitable
192.168.62.1
192.168.62.200-220
alumno@Atacante:~$
```

Figura 5: Atacante\_nmapscan\_filecomplex.

### 2.2.2. NMAP con Metasploit

También hemos de instalar Metasploit para hacer uso de él: <https://github.com/rapid7/metasploit-framework/wiki/Nightly-Installers>. Una vez instalado, con `$ msfconsole` inicializamos Metasploit y la base de datos asociada.

A continuación, realizamos un scanner básico de la red, almacenando el contenido en la base de datos interna y exportándolo completo de la misma a un fichero, para así analizarlo:

```
$ db_nmap -v -sV 192.168.62.0/24
$ db_export out_ejercicio1.txt
```

Como muestra la figura 6, se observa que en dicho fichero encontramos el contenido del escaneo. Por un lado, podemos ver información del usuario que ha invocado el Metasploit. Seguidamente, tenemos el apartado que refiere a los hosts y servicios que se han encontrado en la dirección de subred que se le ha pasado al escaneo. Por último, podemos observar que el grueso del fichero son los módulos del Metasploit.

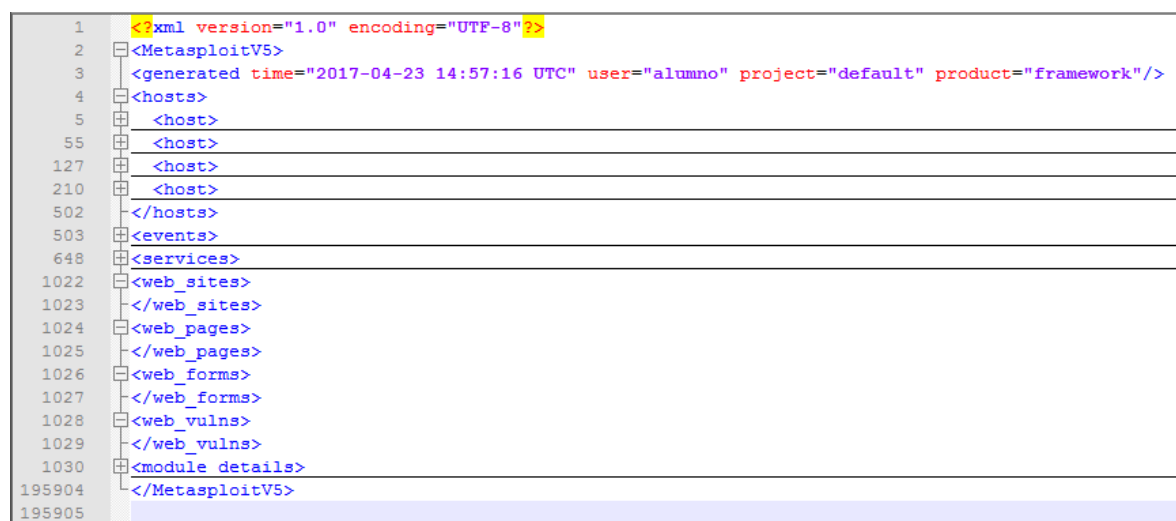


Figura 6: Atacante\_scanner\_y\_BBDD.

### 2.2.3. Wireshak: trazas

A continuación mostramos algunas trazas obtenidas tras ejecutar ciertos comandos con NMAP.

- `$ nmap -sS -sV -p 20-30 192.168.62.0/24`. En el host `metasploitable` se lanza un escaneo de puertos cada segundo a un puerto diferente entre los puertos 20 al 30, como muestra la figura 7. El fin principal de realizar un escaneo de puertos de esta forma es evitar ser detectado por la seguridad que pueda tener la subred.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <MetasploitV5>
3    <generated time="2017-04-23 14:57:16 UTC" user="alumno" project="default" product="framework"/>
4    <hosts>
5      <host>
55     <host>
127    <host>
210    <host>
502    </hosts>
503    <events>
648    <services>
1022   <web_sites>
1023   </web_sites>
1024   <web_pages>
1025   </web_pages>
1026   <web_forms>
1027   </web_forms>
1028   <web_vulns>
1029   </web_vulns>
1030   <module_details>
195904 </MetasploitV5>
195905

```

Figura 7: Atacante\_wireshar\_scaneo\_delay.

- \$sudo nmap -sS -mtu 24 -p 80 metasploitable 192.168.62.102. En el hots metasploitable y en la IP 192.168.62.102 se lanza un escaneo al puerto 80 con el bit SYN activado, como se muestra en la figura 8 Lo que se hace es enviar un paquete SYN, como si se fuera a abrir una conexión real y después se espera una respuesta. Si se recibe un paquete SYN/ACK esto indica que el puerto está abierto, mientras que si se recibe un RST (reset) indica que no hay nada escuchando en el puerto. Si no se recibe ninguna respuesta después de realizar algunas retransmisiones o se recibe un ICMP entonces el puerto se marca como filtrado.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <MetasploitV5>
3    <generated time="2017-04-23 14:57:16 UTC" user="alumno" project="default" product="framework"/>
4    <hosts>
5      <host>
55     <host>
127    <host>
210    <host>
502    </hosts>
503    <events>
648    <services>
1022   <web_sites>
1023   </web_sites>
1024   <web_pages>
1025   </web_pages>
1026   <web_forms>
1027   </web_forms>
1028   <web_vulns>
1029   </web_vulns>
1030   <module_details>
195904 </MetasploitV5>
195905

```

Figura 8: Atacante\_wireshar\_scaneo\_delay.

## 2.3. Scripts NMAP

### 2.3.1. Scripts `/usr/share/nmap/scripts`

En la instalación de NMAP se crea el directorio `/usr/share/nmap/scripts`, este directorio contiene una lista de scripts implementados por otros usuarios y que están diseñados para ser invocados desde el comando `nmap`. A continuación se describen algunos:

- `http-git.nse`: Realiza una conexión al puerto 80 de la víctima en busca de un servidor web activo, si el puerto está abierto, se intenta localizar un directorio `.git`. La existencia de este directorio implica que la víctima está realizando un control de versiones, por tanto, el siguiente paso que realiza el script es la búsqueda de coincidencias en *Github*, si se encuentran coincidencias (por un perfil o proyecto público), se muestran un mensaje al usuario con toda la información que se ha podido extraer de la víctima de su repositorio.
- `smb-server-stats.nse`: Este script explota una vulnerabilidad de *Samba* corriendo sobre sistemas operativos de Windows, esta vulnerabilidad permite que un usuario externo pueda solicitar los datos estadísticos del servicio, recopilando así valiosa información de los archivos que se comparten.
- `ssh2-enum-algos.nse`: Devuelve los algoritmos de cifrado y compresión que tiene implementados la víctima. Esta información puede ser muy útil para reducir considerablemente el tiempo de los ataques por fuerza bruta.
- `dhcp-discover.nse`: Script que recopila información del servidor DHCP de la red, se imprimirá por pantalla al usuario el valor de cada uno de los campos que se obtienen del DHCP (Gateway, máscara de subred, router, nombre de dominio, etc...). Para el funcionamiento del script no es necesario consumir una dirección IP.

### 2.3.2. Realización de script básico

Se pueden crear nuevos scripts adaptados a nuestras necesidades, que automaticen tareas habituales, o repetitivas. En la siguiente url <http://nmap.org/book/nse-tutorial.html> se describe la estructura que debe tener el script. Para poner en práctica este apartado, a continuación, incluye el contenido de un script realizado por nosotros, las acciones que realiza son las siguientes:

- Comprobar si el equipo objeto tiene el puerto 80 abierto (el número de puerto se puede cambiar a la hora de ejecutar el comando)
- En el caso de que se cumpla el paso anterior, se entiende que existe un servidor web en el equipo, por tanto, se solicita la página `index.html`, dicha página se crea por defecto en los navegadores web.



- La página web descargada se almacena en un fichero con el mismo nombre *index.html*

Para ejecutar el script, se debe escribir el siguiente comando:

```
$ nmap -p 80 <ip> --script=http-index
```

```
local http = require "http"
local io = require "io"
local shortport = require "shortport"
local stdnse = require "stdnse"

description = [[
Comprobamos si el host remoto tiene el puerto indicado activo
, en ese caso, obtenemos el /index.html y lo almacenamos
en un fichero "index.html"
]]

---
-- @usage
-- nmap -p 80 <ip> --script=http-index
--
--80/tcp open http
--|_http-index: /index.html Obtenido correctamente!
--
-- Version 0.1
-- Created 23/04/2017 - v0.1 - created by R&R_Asociados
--

author = "R&R_Asociados"
license = "Open_License"
categories = {"discovery"}

portrule=shortport.http

action = function( host, port )

local result
local output = stdnse.output_table()
local request_type
path = "/index.html"
```

```
result = http.get(host, port, path)
request_type = "GET"

if ( not(200 <= result.status and result.status < 210) ) then
    output.error = ("ERROR:_Fallo_al_obtener_la_url_%s"):
        format(path)
    return output,output.error
end

local fname = "index.html"
local f = io.open(fname,"w")

if ( not(f) ) then
    output.error = ("ERROR:_Fallo_al_crear/abrir_el_fichero
        _%s"):format(fname)
    return output,output.error
end

    io.output(f)
    io.write(table.tostring( result ))
    f:close()

if ( 200 <= result.status and result.status < 210 ) then
    output.result = ("%s_Obtenido_correctamente!"):format(
        path)
    return output,output.result
end

return

end

-- Transformacion de tipo table en string
function table.val_to_str ( v )
    if "string" == type( v ) then
        string.gsub( v, "\n", "\\n" )
        if string.match( string.gsub(v,"[^\\"]",""), '^"+$' )
            then
            return "'" .. v .. "'"
        end
        return "'" .. string.gsub(v,'"','\\"') .. "'"
    else
        return "table" == type( v ) and table.tostring( v ) or
```

```
        tostring( v )
    end
end

function table.key_to_str ( k )
    if "string" == type( k ) and string.match( k, "^[_%a][_%a%d
    ]*$" ) then
        return k
    else
        return "[" .. table.val_to_str( k ) .. "]"
    end
end

function table.tostring( tbl )
    local result, done = {}, {}
    for k, v in ipairs( tbl ) do
        table.insert( result, table.val_to_str( v ) )
        done[ k ] = true
    end
    for k, v in pairs( tbl ) do
        if not done[ k ] then
            table.insert( result,
                table.key_to_str( k ) .. "=" .. table.val_to_str( v ) )
        end
    end
    return "{" .. table.concat( result, "," ) .. "}"
end
```

El script contiene un control de errores, por lo que se mostrará uno de los siguientes resultados:

- "ERROR: Fallo al obtener la url index.html"
- "ERROR: Fallo al crear/abrir el fichero index.html"
- "index.html Obtenido correctamente!"

### 2.3.3. Comando Portscan

Accedemos a la consola de Metasploit con el comando *\$msfconsole*, para encontrar las modalidades existentes de Portscan, lanzamos la búsqueda con *\$search portscan*, el resultado se puede ver en la figura 9.

1. Para este ejemplo haremos uso de *auxiliary/scanner/portscan/tcp*, para ello, lo seleccionamos ejecutando *\$use auxiliary/scanner/portscan/tcp*.

```
msf > search portscan

Matching Modules
=====

  Name                                         Disclosure Date  Rank   Description
  ----                                         -
auxiliary/scanner/http/wordpress_pingback_access  normal  Wordpress Pingback Locator
auxiliary/scanner/natpmp/natpmp_portscan          normal  NAT-PMP External Port Scanner
auxiliary/scanner/portscan/ack                    normal  TCP ACK Firewall Scanner
auxiliary/scanner/portscan/ftpbounce              normal  FTP Bounce Port Scanner
auxiliary/scanner/portscan/syn                    normal  TCP SYN Port Scanner
auxiliary/scanner/portscan/tcp                    normal  TCP Port Scanner
auxiliary/scanner/portscan/xmas                    normal  TCP "XMas" Port Scanner
auxiliary/scanner/sap/sap_router_portscanner      normal  SAPRouter Port Scanner
```

Figura 9: Metasploit Portscan Search

2. Visualizamos los diferentes parámetros de configuración que permite con el comando *\$show options*.
3. Ajustamos el número de hilos y la ip de la víctima.
4. Lanzamos el escaneo con el comando *\$run*. Se muestra el resultado de la ejecución en la imagen 10.
5. El funcionamiento del comando *portscan* es muy similar al de Nmap, con la ventaja de poder ajustar el número de hilos que queremos dedicar al escaneo.

```

Name      Current Setting  Required  Description
-----
CONCURRENCY 10              yes       The number of concurrent ports to check per host
DELAY       0               yes       The delay between connections, per thread, in milliseconds
JITTER      0               yes       The delay jitter factor (maximum value by which to +/- DELAY) in milliseconds.
PORTS       1-10000         yes       Ports to scan (e.g. 22-25,80,110-900)
RHOSTS      metasploitable yes       The target address range or CIDR identifier
THREADS     1               yes       The number of concurrent threads
TIMEOUT     1000            yes       The socket connect timeout in milliseconds

msf auxiliary(tcp) > set RHOSTS metasploitable
RHOSTS => metasploitable
msf auxiliary(tcp) >
msf auxiliary(tcp) > set THREADS 50
THREADS => 50
msf auxiliary(tcp) > run

[*] 192.168.62.189: - 192.168.62.189:23 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:22 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:25 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:21 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:53 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:80 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:111 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:139 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:445 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:514 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:513 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:512 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:1099 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:1524 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:2049 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:2121 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:3306 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:3632 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:5432 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:5900 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:6000 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:6667 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:6697 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:8009 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:8180 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:8787 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) >

```

Figura 10: Metasploit Portscan

### 3. Explotar Vulnerabilidades

Para usuarios que no son expertos en el uso de Metasploit, se recomienda el uso de la interfaz gráfica del programa, que se puede descargar desde el [siguiente enlace](#). La instalación es muy simple en Linux, basta con descargarse el programa, convertirlo en ejecutable (`$ chmod 777 metasploit-latest-linux-x64-installer.run`) y ejecutar el fichero (`.run`). Se abrirá un instalador gráfico intuitivo.

Para hacer uso del programa, abrimos un navegador web y accedemos a la URL que se nos indicó durante la instalación, si no hemos modificado las opciones por defecto, será `https://localhost:3790/`. En la figura 11 podemos ver la apariencia.

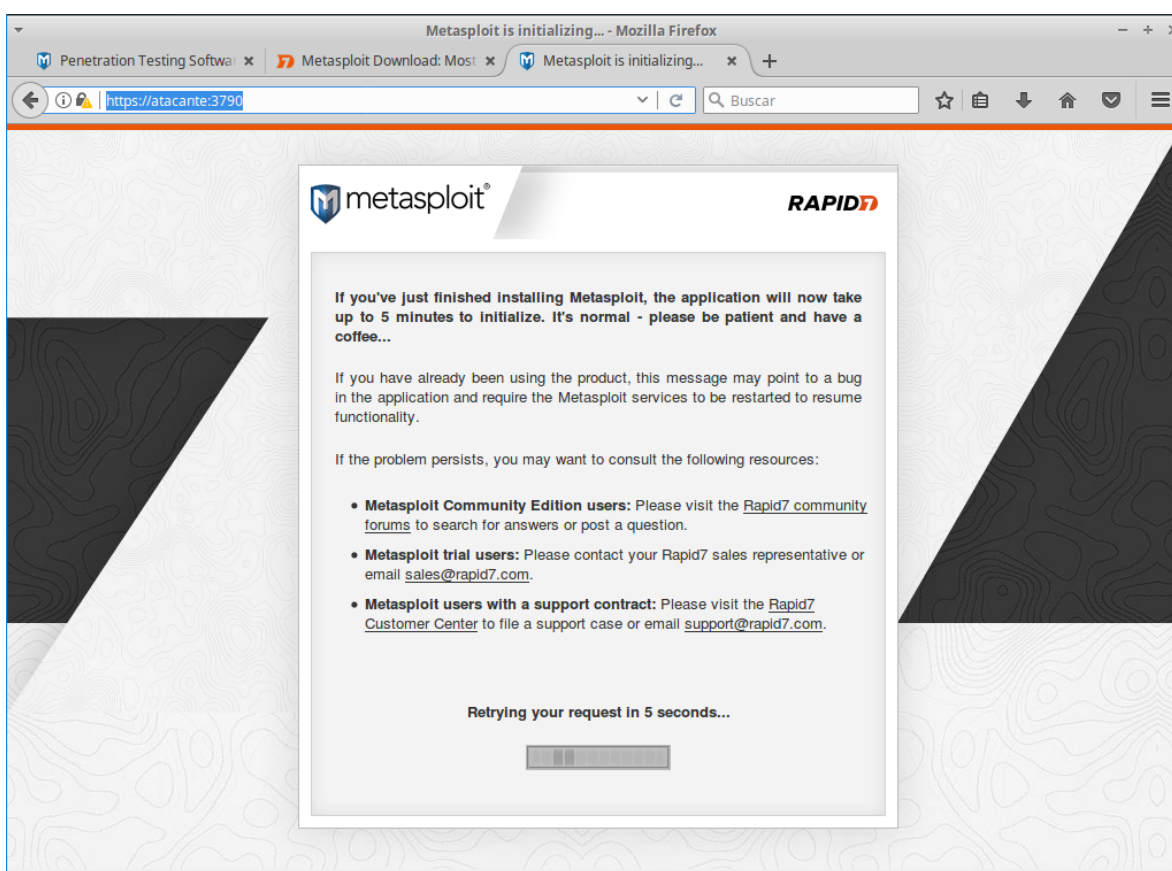


Figura 11: Metasploit Gráfico