Lab 2: Password Cracking - The Art of Hash Hunting

University of Texas at San Antonio, IS-6303-001-202510

Garrett Stokes

October 22, 2024

Table of Contents

Executive Summary

This report outlines the setup and execution of password cracking using VirtualBox and Kali Linux, focusing on John the Ripper and Hashcat. I updated VirtualBox and installed Kali (2024.3), configuring the virtual machine to use 4 CPU cores, 4096 MB memory, and a 50 GB hard drive. After resolving initial issues, I prepared my environment for cracking by downloading necessary password files.

Using John the Ripper, I first attempted brute-force cracking on a shadow file, which yielded limited results. However, switching to a dictionary attack with the "rockyou.txt" wordlist allowed me to crack 6 password hashes. Applying the same technique to a SAM dump file resulted in 7 cracked hashes. The passwords cracked were mostly weak and short, highlighting the need for quality wordlists and efficient resources.

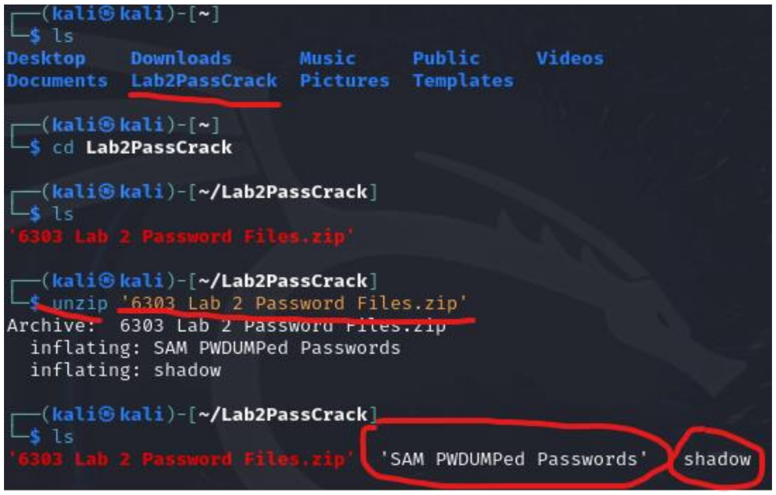
Next, I explored Hashcat, a tool designed for GPU-accelerated password cracking. Although limited to CPU operation, I ran a dictionary attack on the shadow and SAM dump files, achieving slightly lower results than John the Ripper. Hashcat's full potential was not realized due to this limitation, but it holds promise for outperforming John with proper configuration and GPU support.

Lastly, I examined Cewl, a tool for generating custom wordlists by scraping email addresses and modified the username generator tool to convert this data into potential usernames. These tools streamline the process of gathering usernames, facilitating the creation of tailored password wordlists for effective cracking.

1 Setting Up VirtualBox and Kali Linux

To begin cracking passwords with Kali, I first had to set up the environment to do so. I already had VirtualBox and an older version of Kali Linux installed on my Windows 10 host system, but for the sake of staying up to date I decided to update VirtualBox to its latest release and install the newest version of Kali Linux (2024.3). I leave out the screenshots for the initial processes to reduce the amount of bloat in this report.

I configured Kali to have 4 CPU cores, 4096 MB memory, 64 MB Video Memory, and a 50 GB virtual hard drive, as recommended by the lab instructions. After loading into Kali, I encountered some issues with the VirtualBox Guest Additions and Shared Folders, so I decided to download the password files directly from an internet browser on the Kali VM. This worked well, and I was able to easily unzip the files to have them prepared for password cracking. I created a folder called “Lab2PassCrack”, where the password files were unzipped to.

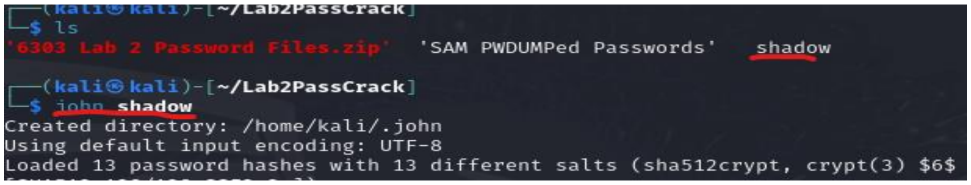


```
┌──(kali㉿kali)-[~]
└─$ ls
Desktop    Downloads      Music     Public     Videos
Documents  Lab2PassCrack  Pictures  Templates

┌──(kali㉿kali)-[~]
└─$ cd Lab2PassCrack

┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ ls
'6303 Lab 2 Password Files.zip'

┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ unzip '6303 Lab 2 Password Files.zip'
Archive:  6303 Lab 2 Password Files.zip
  inflating: SAM PWDUMPed Passwords
  inflating: shadow

┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ ls
'6303 Lab 2 Password Files.zip'  'SAM PWDUMPed Passwords'  shadow
```

Figure 1 VirtualBox lab directory and files

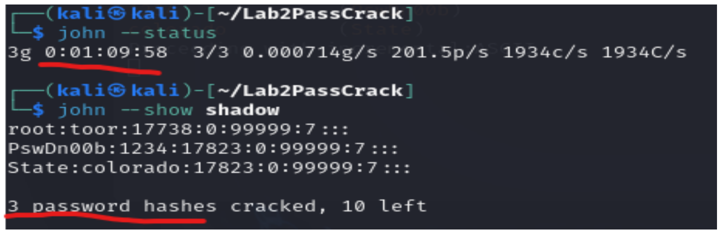
2 Hash Cracking with John the Ripper

I decided to attempt cracking the passwords of the shadow file first with John the Ripper, an open-source tool, which can be utilized to perform password cracks against weak passwords by using dictionary, brute-force, and hybrid attacks. It's more of a CPU focused tool, not utilizing the full capability of GPUs, like the next tool I used can do. To start with John the Ripper after I had the shadow file ready, it was easy to run a simple brute force attempt with the command 'john shadow'. This wasn't very efficient though, as after waiting over an hour and only cracking 3 basic passwords, I was ready to try something new.



```
┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ ls
'6303 Lab 2 Password Files.zip'  'SAM PWDUMPed Passwords'   shadow

┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ john shadow
Created directory: /home/kali/.john
Using default input encoding: UTF-8
Loaded 13 password hashes with 13 different salts (sha512crypt, crypt(3) $6$
```

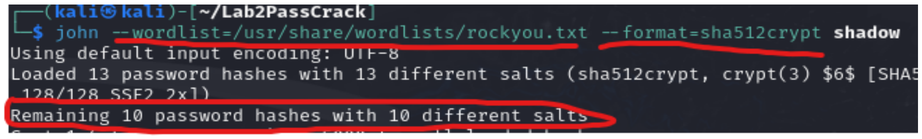
Figure 2 1st john basic execution (shadow)



```
┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ john --status
3g 0:01:09:58  3/3 0.000714g/s 201.5p/s 1934c/s 1934C/s

┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ john --show shadow
root:toor:17738:0:99999:7:::
PswDn00b:1234:17823:0:99999:7:::
State:colorado:17823:0:99999:7:::

3 password hashes cracked, 10 left
```

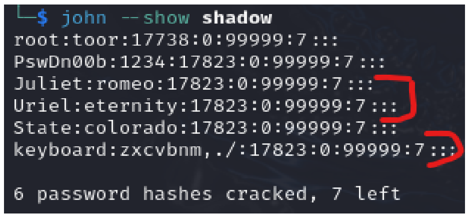
Figure 3 1st john basic results (shadow)

For my next approach to this, I decided to use a wordlist, or in more cybersecurity specific terms a "dictionary" attack. I used the "rockyou.txt." wordlist, which contains around 14 million historically breached passwords. I also specified for john to utilize the format of sha512 with salt, as after analyzing the actual contents of the shadow file and doing some research, this seemed to be the format of the entries within the shadow file, indicated by the 6 in the password hash entries. After executing this new approach with John the Ripper, within 2 minutes, 3 more password hashes were cracked.



```
┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=sha512crypt shadow
Using default input encoding: UTF-8
Loaded 13 password hashes with 13 different salts (sha512crypt, crypt(3) $6$ [SHA5
 128/128 SSE2 2x1])
Remaining 10 password hashes with 10 different salts
```

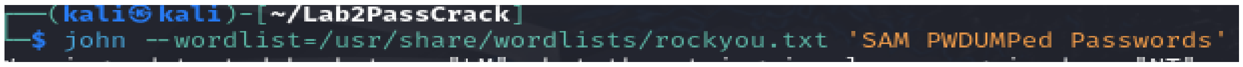
Figure 4 2nd john execution (shadow)



```
└─$ john --show shadow
root:toor:17738:0:99999:7:::
PswDn00b:1234:17823:0:99999:7:::
Juliet:romeo:17823:0:99999:7:::
Uriel:eternity:17823:0:99999:7:::
State:colorado:17823:0:99999:7:::
keyboard:zxcvbnm,./:17823:0:99999:7:::

6 password hashes cracked, 7 left
```

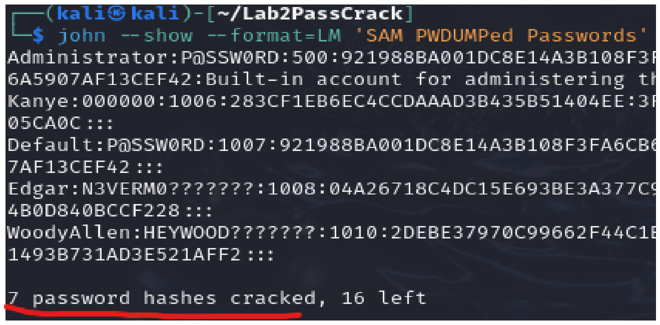
Figure 5 2nd john results (shadow)

I was happy being able to crack 6 of the password hashes on the shadow file and decided to try this same maneuver on SAM dump file, that is, using the dictionary attack with the “rockyou.txt” wordlist. Initially, I tried to utilize the format of “nt”, then “lm”, as the Sam dump files seemed to consist of Windows passwords, but John was throwing errors with these formats. Because of this, I just ran it with no format. This in turn yielded 7 cracked password hashes on the Sam dump file.



```
┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt 'SAM PWDUMPed Passwords'
```

Figure 6 john execution (Sam dump)



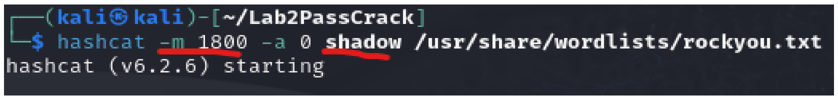
```
┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ john --show --format=LM 'SAM PWDUMPed Passwords'
Administrator:P@SSW0RD:500:921988BA001DC8E14A3B108F3
6A5907AF13CEF42:Built-in account for administering t
Kanye:000000:1006:283CF1EB6EC4CCDAAAD3B435B51404EE:3
05CA0C:::
Default:P@SSW0RD:1007:921988BA001DC8E14A3B108F3FA6CB
7AF13CEF42:::
Edgar:N3VERM0???????:1008:04A26718C4DC15E693BE3A377C
4B0D840BCCF228:::
WoodyAllen:HEYWOOD???????:1010:2DEBE37970C99662F44C1
1493B731AD3E521AFF2:::

7 password hashes cracked, 16 left
```

Figure 7 john results (Sam dump)

It seems that all the passwords I managed to crack were short (1-8 characters) and not complex (only lowercase, or only uppercase, or just easily predictable) which made them easier to crack. To crack passwords as fast as possible, I can imagine its best to combine quality wordlists together like “rockyou.txt”, as well as resorting to brute-force methods after the wordlists are exhausted. Its also paramount that the computer/device used for cracking, whether it be a virtual machine or not, has a great deal of resources allocated to it. A 50GB hard drive, 4096 MB memory, 64mb video memory, and 4 CPU cores are good for learning purposes, but in a real-world scenario today are very weak for this purpose.

3 Hash Cracking with Hashcat

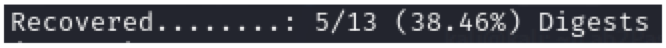
For my second attempt at password cracking, I chose to use the Hashcat tool. Hashcat is different from John, in that it is designed to leverage GPU (Graphic Processing Units) power to perform faster brute-force and dictionary-based password cracking. It also supports a wider range of hash algorithms in comparison to John the Ripper. Unfortunately, since I am running Kali Linux via virtualization on VirtualBox, I couldn't take advantage of my host system GPU to see the full potential of Hash cat's results. Because of this, I had to use Hashcat using just CPU cracking, so it could actually be a fair comparison to my experience using John the Ripper. To begin cracking with Hashcat on the shadow file, I had to specify the hash type as SHA-512 with "-m 1800", and then indicate a straight attack (dictionary attack) with "-a 0", then pointing to the directory of the wordlist to be used.



```
┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ hashcat -m 1800 -a 0 shadow /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting
```

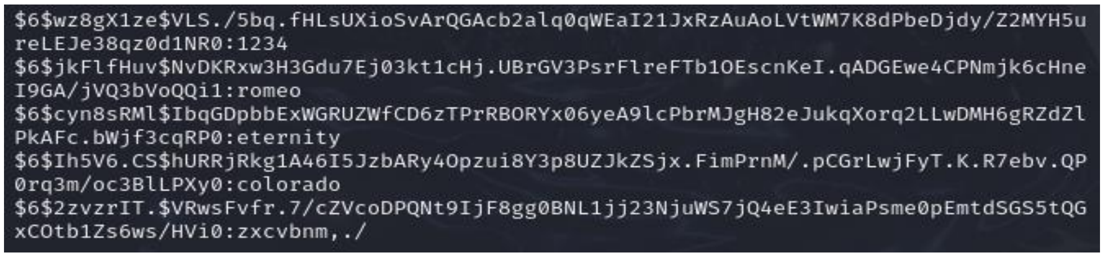
Figure 8 hashcat execution (shadow)

Once this is complete Hashcat will go through its checks and begin doing its work on the file. I will say that Hashcat seemed to provide a much more verbose detailing on the entire process by default and seems to be a more modern password cracking tool. For example, there were various keys you could enter for real-time information, the below screenshot showing the output of "s" key, or status. After letting it run for a couple of minutes, it ended up cracking 5 passwords, 1 less than John the Ripper.



```
Recovered........: 5/13 (38.46%) Digests
```

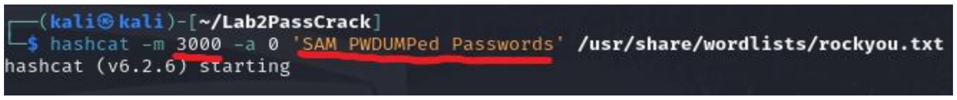
Figure 9 hashcat status (shadow)



```
$6$wz8gX1ze$VLS./5bq.fHLsUXioSvArQGAcb2alq0qWEaI21JxRzAuAoLVtWM7K8dPbeDjdy/Z2MYH5u
reLEJe38qz0d1NR0:1234
$6$jkFlfHuv$NvDKRxw3H3Gdu7Ej03kt1cHj.UBrGV3PsrFlreFTb1OEscnKeI.qADGEwe4CPNmjk6cHne
I9GA/jVQ3bVoQQi1:romeo
$6$cyn8sRMl$IbqGDpbbExWGRUZWfCD6zTPrRBORYx06yeA9lcPbrMJgH82eJukqXorq2LLwDMH6gRZdZl
PkAFc.bWjf3cqRP0:eternity
$6$Ih5V6.CS$hURRjRkg1A46I5JzbARy4Opzui8Y3p8UZJkZSjx.FimPrnM/.pCGrLwjFyT.K.R7ebv.QP
0rq3m/oc3BlLPXy0:colorado
$6$2zvzrIT.$VRwsFvfr.7/cZVcoDPQNt9IjF8gg0BNL1jj23NjuWS7jQ4eE3IwiaPsme0pEmtdSGS5tQG
xCOtb1Zs6ws/HVi0:zxcvbnm,./
```

Figure 10 hashcat results (shadow)

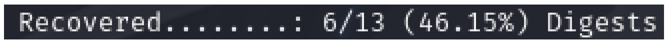
Seeing as Hashcat was able to crack 5 passwords on the shadow file, I decided to move on to the Sam dump file. This time, I specify "-m 3000", which sets the hash type as LM hashes, typically used by older windows systems, which is a very weak and outdated hash method. In my research, it seemed that this is the hashing method the "SAM PWDUMPed Passwords" file is likely using, and for this reason I gave it a shot.



```
┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ hashcat -m 3000 -a 0 'SAM PWDUMPed Passwords' /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting
```

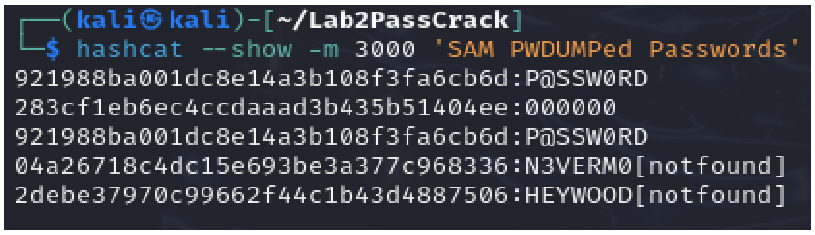
Figure 11 hashcat execution (Sam dump)

Within a fast period, Hashcat once again was seemingly able to garner some results, apparently cracking 6 password hashes according to its output via the status check. Upon check the actual password hashes it cracked with the "—show -m 3000" command, I was a bit puzzled by the results. In the status check, it says it recovered 6 passwords, yet I can count 5 in the output. Also, 2 of those passwords seem to be duplicates with the exact same hash. I was further puzzled by the fact that 2 of the entries were followed by [notfound], indicating that they were not actually able to be cracked by Hashcat. It seems that Hashcat was actually only able to crack 2 password hashes.



```
Recovered........: 6/13 (46.15%) Digests
```

Figure 12 hashcat status (Sam dump)



```
┌──(kali㉿kali)-[~/Lab2PassCrack]
└─$ hashcat --show -m 3000 'SAM PWDUMPed Passwords'
921988ba001dc8e14a3b108f3fa6cb6d:P@SSW0RD
283cf1eb6ec4ccdaaad3b435b51404ee:000000
921988ba001dc8e14a3b108f3fa6cb6d:P@SSW0RD
04a26718c4dc15e693be3a377c968336:N3VERM0[notfound]
2debe37970c99662f44c1b43d4887506:HEYWOOD[notfound]
```

Figure 13 hashcat results (Sam dump)

In my analysis, I do think that Hashcat is a very powerful tool despite inferior results to John the Ripper in my experience. With time-constraints, I was not able to fully configure Hashcat to run in the most efficient manner, especially in the Sam dump file, but I think with proper configuration and enabling GPU capabilities, this tool could potentially blow John the Ripper out in terms of password hash cracking results.

4 Practical Usage of Cewl and Username Generators

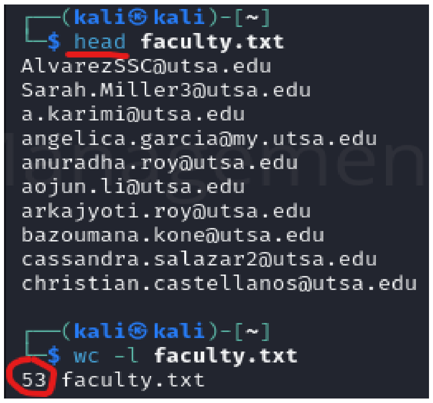
CeWl, aka Custom Word List generator is a ruby program that spiders through given URLs to a specified depth and returns list/s of words that can be used for password cracking. It can follow external links and create a list of email addresses found in mailto links. It's already installed when you use Kali Linux and is particularly useful in finding email addresses that can then be converted to usernames to be tested in brute force attacks with tools like John the Ripper or Hashcat. As most companies' websites have pages where they include people's email address, Cewl is the tool to gather these addresses all out once rather than manually going through and collecting them. It's super-efficient, and you can have it spider through subsequent pages to find further information. I did this myself, using the command "cewl -e -n -d 5 https://business.utsa.edu/management-science-statistics/faculty/ > faculty.txt" The flags I use represent as follows:

"-e" – tells Cewl to look for and extract email addresses from the webpage

"-n" – disables the extraction of non-email words, which makes Cewl only output email addresses.

"-d 5" – sets the crawl depth as 5, which determines how many links Cewl will follow and scrape from the initial page.

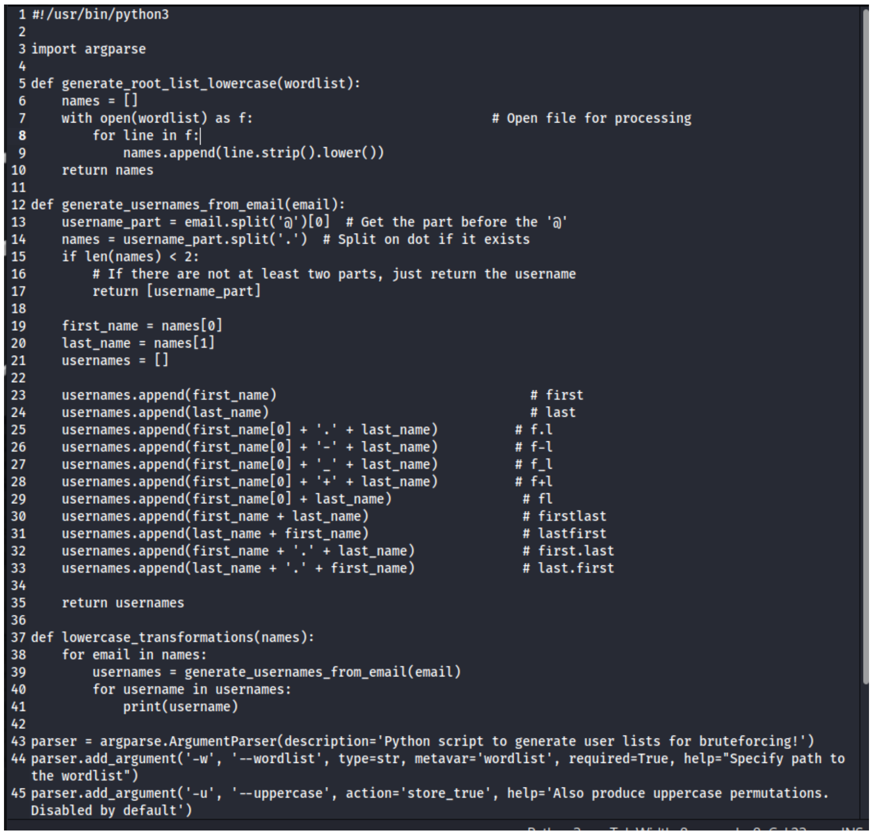
After removing any non-email address lines in the output file, this command yielded 53 total emails. I do not think the "-d 5" flag provided any extra results in this case, but it can work depending on the nature of the website.



```
┌──(kali㉿kali)-[~]
└─$ head faculty.txt
AlvarezSSC@utsa.edu
Sarah.Miller3@utsa.edu
a.karimi@utsa.edu
angelica.garcia@my.utsa.edu
anuradha.roy@utsa.edu
aojun.li@utsa.edu
arkajyoti.roy@utsa.edu
bazoumana.kone@utsa.edu
cassandra.salazar2@utsa.edu
christian.castellanos@utsa.edu

┌──(kali㉿kali)-[~]
└─$ wc -l faculty.txt
53 faculty.txt
```

Figure 14 head of faculty.txt

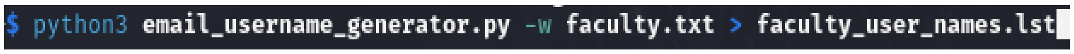
The **username_generator** tool serves the purpose of ingesting data from a source and creating a list of possible usernames from that source from what I can understand. I tried to use this tool based on the output file I had from Cewl, which was just a listing of emails, but it didn't work properly, as the syntax of the file is not what the program expects. The username_generator program seems to expect a syntax per line formatted such as "First Last", like "John Doe". For this reason, I used AI to help alter the program to read data such that the "faculty.txt" file resembles e.g. email addresses each line having characters separated by a dot followed by a domain name, for example "abdulrahman.yash@gmail.com". I renamed it "email_username_generator", and its code looks like the below screenshot



```
#!/usr/bin/python3

import argparse

def generate_root_list_lowercase(wordlist):
    names = []
    with open(wordlist) as f:                               # Open file for processing
        for line in f:
            names.append(line.strip().lower())
    return names

def generate_usernames_from_email(email):
    username_part = email.split('@')[0]  # Get the part before the '@'
    names = username_part.split('.')  # Split on dot if it exists
    if len(names) < 2:
        # If there are not at least two parts, just return the username
        return [username_part]

    first_name = names[0]
    last_name = names[1]
    usernames = []

    usernames.append(first_name)                                 # first
    usernames.append(last_name)                                  # last
    usernames.append(first_name[0] + '.' + last_name)          # f.l
    usernames.append(first_name[0] + '-' + last_name)          # f-l
    usernames.append(first_name[0] + '_' + last_name)          # f_l
    usernames.append(first_name[0] + '+' + last_name)          # f+l
    usernames.append(first_name[0] + last_name)                 # fl
    usernames.append(first_name + last_name)                    # firstlast
    usernames.append(last_name + first_name)                    # lastfirst
    usernames.append(first_name + '.' + last_name)              # first.last
    usernames.append(last_name + '.' + first_name)              # last.first

    return usernames

def lowercase_transformations(names):
    for email in names:
        usernames = generate_usernames_from_email(email)
        for username in usernames:
            print(username)

parser = argparse.ArgumentParser(description='Python script to generate user lists for bruteforcing!')
parser.add_argument('-w', '--wordlist', type=str, metavar='wordlist', required=True, help="Specify path to the wordlist")
parser.add_argument('-u', '--uppercase', action='store_true', help='Also produce uppercase permutations. Disabled by default')
```

Figure 15 "email_username_generator" program code

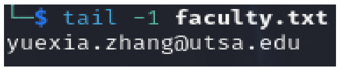
Once it could correctly read the data from the faculty.txt file, it could now generate 10 possible usernames for emails that match the structure. The command to execute it is the same as the original program, seen in screenshot below



```
$ python3 email_username_generator.py -w faculty.txt > faculty_user_names.lst
```

Figure 16 Program execution command with redirect output

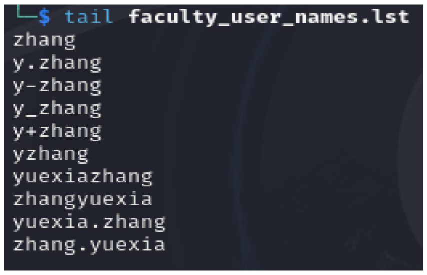
I will show what the results look like. The list is very long, so I'll show one example. For context, I will show the last line (or email) of the faculty.txt from cewl below



```
└─$ tail -1 faculty.txt
yuexia.zhang@utsa.edu
```

Figure 17 last email of faculty.txt

And the last 10 lines of the email_username_generator program output file, “faculty_user_names.lst”. Based on the program code, these are the 10 possible usernames for that email address.



```
└─$ tail faculty_user_names.lst
zhang
y.zhang
y-zhang
y_zhang
y+zhang
yzhang
yuexiazhang
zhangyuexia
yuexia.zhang
zhang.yuexia
```

Figure 18 ten possible usernames of last email

Tools like Cewl and username generators are extremely powerful in password cracking. Cewl is used to retrieve the source data (email addresses or other relevant information), while username generators or similar tools take this source data and convert it to a structure that could match that of an actual username, placing it in a wordlist format. From here, tailored password wordlists can be generated based on these usernames and supplied to tools like John the Ripper or Hashcat, which can seriously speed up the process by using a wordlist that contains password combinations derived from the possible usernames.