Activity I: Hacking Password

1.

```python
import hashlib
SUBSTITUTE = {'o': '0', 'l': '1', 'i': '1'}


def permute(word: str):
    result = []
    search(word, "", 0, result)
    return result


def search(word: str, cur_word: str, i: int, result: list):
    if i < len(word):
        search(word, cur_word + word[i].lower(), i + 1, result)
        search(word, cur_word + word[i].upper(), i + 1, result)
        if word[i] in SUBSTITUTE.keys():
            search(word, cur_word + SUBSTITUTE[word[i]], i + 1, result)
    else:
        result.append(cur_word)


print("Enter input file path: ", end='')
input_path = input().strip()
hashed = 'd54cc1fe76f5186380a0939d2fc1723c44e8a5f7'

fr = open(input_path, 'r')
read_str = fr.read()
fr.close()

lines = read_str.split('\n')
for password in lines:
    permuted_result = permute(password)
    for result in permuted_result:
        if hashlib.sha1(result.encode('utf8')).hexdigest() == hashed:
            print("Found password match: " + result)
            exit(0)
print("No password match")
```

Results:

```
PS C:\Users\User\Documents\Works\Comp security\ComputerSecurity\01-password> python 01.py
Enter input file path: 10k-most-common.txt
Found password match: ThaiLanD
```

2.

```python
import hashlib
import time

SUBSTITUTE = {'o': '0', 'l': '1', 'i': '1'}


def permute(word: str):
    result = []
    search(word, "", 0, result)
    return result


def search(word: str, cur_word: str, i: int, result: list):
    if i < len(word):
        search(word, cur_word + word[i].lower(), i + 1, result)
        search(word, cur_word + word[i].upper(), i + 1, result)
        if word[i] in SUBSTITUTE.keys():
            search(word, cur_word + SUBSTITUTE[word[i]], i + 1, result)
    else:
        result.append(cur_word)


OUTPUT_PATH = "output.txt"
total_entries = 0

print("Enter input file path: ", end='')
input_path = input().strip()

fr = open(input_path, 'r')
read_str = fr.read()
fr.close()

fw = open(OUTPUT_PATH, 'w')
lines = read_str.split('\n')

time_start = time.perf_counter()
for password in lines:
    permuted_result = permute(password)
    total_entries += len(permuted_result)
    for result in permuted_result:
        fw.write(
            f"{hashlib.sha1(result.encode('utf8')).hexdigest()} {result}\n")
time_end = time.perf_counter()
fw.close()

print(f"Total time elapsed: {time_end - time_start} second(s)")
print(f"Total entries: {total_entries}")
```

Results:

```
4339063   1972a122cc27e87af569a8574254035ede3f4890 eYPHEd
4339064   7be3de413f44e1e7052ff3cb3d537ddc5d0a7baa eYPHED
4339065   c0f264b96d16b0a4de19dbccc523ac7f1ba19513 Eyphed
4339066   ea92cf9505eb16439c570b1396dfaeaec60a7edf EypheD
4339067   19423925fc0e4aa1f1ef74141d6a2ba756449d80 EyphEd
4339068   c88e831bc95e2029185934fdf0c2d8fe12b54d85 EyphED
4339069   80221b9f8559588b55aa6cbc19edbeac61a3f896 EypHed
4339070   6f1d250c26e242a0820eb98aeefb0447f48ceeb8 EypHeD
4339071   2751c85157a6851a9c86c6f9ea301a57334051c5 EypHEd
4339072   bb1a71ec00d7620feddd763783704f93e405bd0b EypHED
4339073   871ec474817259566acf66923cfe1acc620f90a2 EyPhed
4339074   56a6a4776926708267c3c534ed18f3f423b73408 EyPheD
4339075   b8ea31349901b90debf32ea211acc079fe3ccf05 EyPhEd
4339076   79b9377203a78ddffe9d758933249737137989 75 EyPhED
4339077   0ca042a63b90111619c1aaae002219496eb4b26b EyPHed
4339078   c809814f75027d5c6bde573f83b5bc1aebb40d50 EyPHeD
4339079   e0abe469be69403cf1fd761960a5e5f32a065b57 EyPHEd
4339080   cc8f042a6ac162a104b37378bcbc9632ba91a346 EyPHED
4339081   3d91ccb6bae26eef4fbacb82db552f22fbc57765 EYphed
4339082   71c67a8ea46d345116fe5deff665c1e7fe4e66d0 EYpheD
4339083   366cf61b8046ceb245be307179a14a9fca58c625 EYphEd
4339084   5a95a80903b73ae93b87482b116f2d5a3d188d80 EYphED
4339085   35dd809356b30b1a6c07f5a23c4fd33737b14a8a EYpHed
4339086   f9d92973a927293b72a80ae37b0c72047c263357 EYpHeD
4339087   7fa67dbd519608b44a6aad1863299dd14c32d8cd EYpHEd
4339088   aa1e838c1c3f0866ab6265d97d55351027401454 EYpHED
4339089   dd16f401df9f8a5b31f2368680dc901ffe8d8274 EYPhed
4339090   83eb183612262b070c493e5dec8848ba775d67e7 EYPheD
4339091   175d527e1533ac12b34a4c7637f5d1b440fd1383 EYPhEd
4339092   27b36f75c36e0e7a5dec641af8ac242b7fdd474a EYPhED
4339093   716643bb1fc5d2723fddfe940f6a4ea2b592bc4e EYPHed
4339094   0f8d6db08a2b47aefd092502a940584f8e2c7a17 EYPHeD
4339095   467dba3df712086914b78f695e0ca17c5780d1e7 EYPHEd
4339096   808fd952e5b15b297a39c86a26f44387784185d6 EYPHED
4339097
```

```
1035770   e5dd1ff3b46b756471118390976dd152646fb429 bluEDOG
1035771   17f2da5a3ad6e09021f412ae9b1d331d279e4803 bluED0g
1035772   fda38ace16b62756315281b9e3258298721947d1 bluED0G
1035773   1e94f799d12e08250a600bf38762b66a41ec18d1 blUedog
1035774   cb8d3c5426113f9741f8127950708c427f0f5230 blUedoG
1035775   488cc54f2e4518551e9f74b3b92644bb0c768204 blUedOg
1035776   7d8e24db4a6875c1e06514a784f644169ade4560 blUedOG
1035777   3652d4fae56281a594c2a4d494d4c6abb018996c blUed0g
1035778   3ddc586672e5d9cf33584d5c115c92c038e2e3e6 blUed0G
1035779   bcd5a0ef369c6d8657b8e47e1de0de41e47b8eb1 blUeDog
1035780   eda01f840921b93fac4dbda32bd7ff5514dfbd9e blUeDoG
1035781   aaf751ac7c73878edc3c5c1028cbda71b3c4e94b blUeDOg
1035782   9f28422767f0393b16644bbe18942d9b60280bf3 blUeDOG
1035783   05e08c5acebcafe504f1433fa5af007f368c7514 blUeD0g
1035784   5fc779bf3abd16316e1f3e20db26eea969e9ca56 blUeD0G
1035785   7f89b67edaecb61427931b68b13c843a401048fc blUEdog
1035786   f2a7b3800240f4db380b547b081d137916218fc1 blUEdoG
1035787   6fae350737d09c63fef11de4ac24e2ab93980547 blUEdOg
1035788   cc17d2fe164b5e36e20ddec0fea1fd5d11bfe600 blUEdOG
1035789   979cf0ee083a98de54ce7c1d6b78dfd5ad4c659f blUEd0g
1035790   4d12dc322c3444660224addf05dfc9ac93b1a86f blUEd0G
1035791   b6e4a28abe10127d9a22899a85de7f48a51fd2f2 blUEDog
1035792   bf757f57eb37c68bf69e0ef441767ca4d41b32fc blUEDoG
1035793   b6556f0fa04d3f219d7d323892aa7db4a4a8b1c2 blUEDOg
1035794   fac321bfb61d62b52281f3465cc39bb48af76164 blUEDOG
1035795   4d6c395ec4a480326e96cb78b3100a82a1288ef9 blUED0g
1035796   a0ef9bd0ef6152ae374d94024caed9a4f791202e blUED0G
1035797   91bc9d8364db416436aa6eeae3176d2450f45994 bLuedog
1035798   f99c93fe47d0171d8377eebbecbb12979e69fa65 bLuedoG
1035799   6121b63b256ca8db22c8627178a5d8e3713c30dc bLuedOg
1035800   9abe657683169af436490ac0d9dae43dc117b2ec bLuedOG
1035801   40a2427e45f3c4d0a3184a6e9bbb587eb99e428e bLued0g
1035802   fa670fa3ebb1b8d5fe3e095c0f00c99ba819313f bLued0G
1035803   611a89dfe7c4e502e3c4331f7030cbc863e8fb44 bLueDog
1035804   935fb020b611574d6271869970c460dfe40e3c66 bLueDoG
1035805   84c982bfb470480480557544f32bdd2fceee15d5 bLueDOg
1035806   550358f18d727fbfd36e5a9893861d0fbe95dce9 bLueDOG
```

```
PS C:\Users\User\Documents\Works\Comp security\ComputerSecurity\01-password> python 02.py
Enter input file path: 10k-most-common.txt
Total time elapsed: 18.6177202 second(s)
Total entries: 4339096
```

3.

```python
import hashlib
import time

SUBSTITUTE = {'o': '0', 'l': '1', 'i': '1'}


def permute(word: str):
    result = []
    search(word, "", 0, result)
    return result


def search(word: str, cur_word: str, i: int, result: list):
    if i < len(word):
        search(word, cur_word + word[i].lower(), i + 1, result)
        search(word, cur_word + word[i].upper(), i + 1, result)
        if word[i] in SUBSTITUTE.keys():
            search(word, cur_word + SUBSTITUTE[word[i]], i + 1, result)
    else:
        result.append(cur_word)


total_time = 0
total_entries = 0

print("Enter input file path: ", end='')
input_path = input().strip()

fr = open(input_path, 'r')
read_str = fr.read()
fr.close()
lines = read_str.split('\n')

for password in lines:
    permuted_result = permute(password)
    total_entries += len(permuted_result)
    for result in permuted_result:
        time_start = time.perf_counter()
        hashlib.sha1(result.encode('utf8')).hexdigest()
        time_end = time.perf_counter()
        total_time += time_end - time_start

print(
    f"Total time elapsed for {total_entries} entries: {total_time} second(s)")
print(f"Time per one SHA1 hash : {total_time / total_entries} second(s)")
print(
    f"Average SHA1 hash calculated in 1 second: {total_entries / total_time}")
```

Result:

```
PS C:\Users\User\Documents\Works\Comp security\ComputerSecurity\01-password> python 03.py
Enter input file path: 10k-most-common.txt
Total time elapsed for 4339096 entries: 5.212048199999324 second(s)
Time per one SHA1 hash : 1.2011829652995288e-06 second(s)
Average SHA1 hash calculated in 1 second: 832512.6386975015
```

4.

Assuming the password has $n$ characters

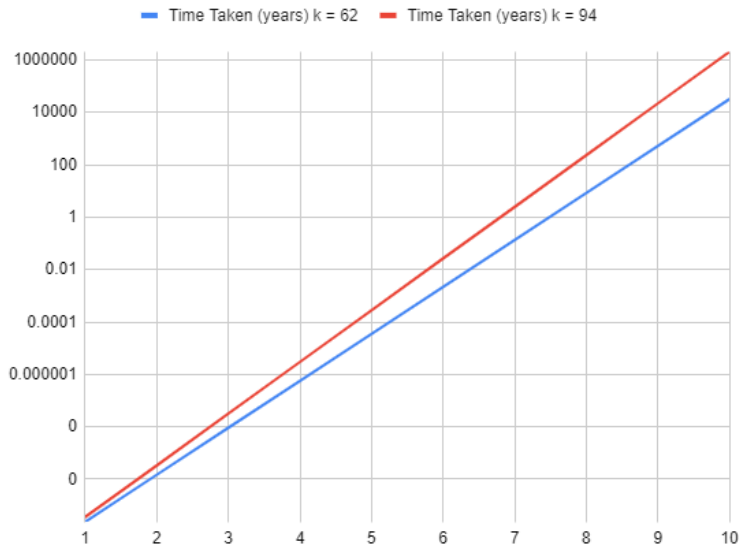
With $k$ types of character to choose for each character

There would be $k^n$ possibilities of password to brute force through

My computer can hash $832512$ entries per second

So, it would take $\frac{k^n}{832512}$ seconds to break the password

5.




According to the chart

With 62 characters in the set, it would take 8 characters long password to have a hash time more than one year.

With 94 characters in the set, it would take 6 characters long password to have a hash time more than one year.

6.

Salt is a random value that is added to password before hashing, preventing rainbow-table attacks and any kind of precomputed-hash attacks. It can also protect weak/commonly used password as the salt value for each weak password is randomly generated, making the hashed value completely different from the original hash value without salt.

| Username | Salt value | String to be hashed | Hashed value = SHA256 (Password + Salt value) |
|---|---|---|---|
| user1 | E1F53135E559C253 | **password123**E1F53135E559C253 | 72AE25495A7981C40622D49F9A52E4F1565C90F048F59027BD9C8C8900D5C3D8 |
| user2 | 84B03D034B409D4E | **password123**84B03D034B409D4E | B4B6603ABC670967E99C7E7F1389E40CD16E78AD38EB1468EC2AA1E62B8BED3A |

Source: https://en.wikipedia.org/wiki/Salt_(cryptography)