Team 23: Ressa Reneth Sarreal & Dominic Schellin
CMSC426 - Dr. Marron
Lab 3: Password Recovery Lab
Due: 05/01/2018

---

1 Introduction

This report details the work done for the Password Recovery Lab. The objective of the lab was to obtain experience with recovering passwords from hash values using various methods such as dictionary, and time memory trade off (TMTO) attack methods. The code must be run with Python 2.7. The dictionary attack runs for about 2.5 hours; the TMTO attack runs for about 30 minutes.

2 Approach

A known variety of passwords must be recovered: unsalted hashes of passwords based on the RockYou dictionary, salted hashes of passwords based on the RockYou dictionary, and hashes of five-character random passwords. The approaches for recovering these different passwords are described below.

2.1 Unsalted RockYou Passwords

For Unsalted RockYou passwords, the number of possible passwords is finite, and all potential passwords are contained in a dictionary. Each password is individually hashed with the SHA-256 hash method, and the obtained value is compared with the hash value of the unknown password. If the two hash values match, then the corresponding RockYou password is the actual password. The run time is on the scale of O(N).

2.2 Salted RockYou Passwords

Similar to Unsalted RockYou passwords, the number of possible passwords is finite; however, each potential salt value must be appended to the front of each password before hashing. For d salt values, and N passwords, the run time is on the scale of O(dN). The obtained hash value is compared to the hash value of the unknown

password, and if the hash values match, then the password is the corresponding value that was hashed after removing the salt.

2.3 Appended Numeric Value to RockYou Passwords

Applying the methods for the unsalted and salted passwords appropriately, once the attempt to hash a basic unsalted or salted password was made, if the hash values never matched, then an additional 20 checks are made for appending a numeric value to the front or the back of a RockYou password. The run time is still on the scale of O(N) and O(dN) still for unsalted and salted passwords respectively.

2.4 Combined RockYou Passwords

After having performed the dictionary-look-up and the appended-numeric-value approaches for the unsalted and salted passwords, it was noted that the last password on the hash list for the unsalted passwords and the last two passwords for the salted hash list for salted passwords did not work well. A separate approach was made for these passwords (still handling the unsalted and salted values with the methods explained in 2.1 and 2.2 respectfully). A nested for loop is necessary for checking all combinations of two RockYou passwords. Separating these three passwords that required this method cut down the actual python execution run time significantly because this would take over four days to perform on all the unsalted and salted passwords. The run time is on the scale of $O(N^2)$.

2.5 5-Character Passwords

A TMTO-attack approach was used for all 5-character passwords. While running the TMTO-attack on all of the blocks, checking the first twelve bits of the hashes on each stream to see if it was between hex values 000 - 003 was performed to cut down run time because the hashes were guaranteed to start with these values. Performing this checking step avoids having to access the dictionary each time when trying to find the final actual password using TMTO.

Team 23: Ressa Reneth Sarreal & Dominic Schellin
CMSC426 - Dr. Marron
Lab 3: Password Recovery Lab
Due: 05/01/2018

---

3 Conclusion and Work Distribution

After completing the lab, the team has a greater understanding of why creating strong passwords is important. Recovering passwords is really difficult. Ressa worked on picking out initial code snippets from the provided code and planning out how to implement the attacks and Dominic primarily wrote the actual python code. Additionally Ressa primarily wrote the brief write-up.

4 Appendix A | tmto_atk.py Code

```
from Crypto.Hash import SHA256
from pickle import load
from binascii import b2a_hex

# Hashes of Five-character Random Passwords
hash_rand = [\
   '5d7af6be8e7ab47467dc87b2470e16cc8fe6986c40537936d5cdc868b3c2d51b',\
   '10ff81984bb4fd2f62cae3859fe7d74fb7293b5ef7198042428a1d0482763f29',\
   '605f5e4d916e03a87347aa9ca40b398251a59b2a689632fffca74ec9a6c95c5e',\
   'bb40fbbce3740533f3491da37239a2dcc4c062f39c781baf0932369e967b3998',\
   'fb741c071bb5ed5cab838bf3f46a50cff17f23ecdebcbd7b5a8458b8d421f71b',\
   '1604d5ebd17cad1fc79a1206473429796c8d90b76f07b1eb5b04869d83c9a44f',\
   'dea2e0a21e0d90e63e8a2a02586bc0e9e1e8feb64e975d835aabdd9d550a48e6',\
   '185db25b849a2535541c219f9a9df3c8d8a8f9a141f9d20f0902f15faa7fd6a3',\
   '3c521d6e4870c3783d0bcc49a68c30fb726816aeb8b72af6a44328d1a0e8d7a4',\
   'e37ce1fdde23ec6b8187caed23efcec3483215d10caa5cd488ecc89fba29445a',\
   '1d9304443d89fc8804e4fcb45a4292a5c8ecd6e7522de0c65c4a4c583c0e989b',\
   'b4de597fa7ee6e5a1aafdbe4f610f44e5917f24555bd134e2dc3b54926fc5688',\
   '658082bfbc45e61d6d078170f9b09c6f7f04b8f5e4a87c3199bba7ff64891a20',\
   '220e3de048e58de267a0ce45388cc51b7006fa359267278b1e46290452ebcd87',\
   'e70fceb6c4dacced90d61ed6b7ea5aede791855ef126c213551034410aa7c052',\
   '09b590c61eb30adbb2316e1e7187d9b6be8022792a8fac852082ac0e34e71bdb',\
   '040cfac289fffeb5aa2b0202da0851371e99f4433b58d81063a526d20e761720',\
   '7f5b2827fab52b8e6d99d6e5f1a63eab26a0c45930a7553a914999a24a6c9fa8',\
   '9189c838f9628e089fc202409e6e7e724a2e2d8305fd12575ba2ab606b253801',\
   '287e899c2d582c5e97b9d292263a555c3f7ca091211d6d6cbdd8a40c2c388b28',\
   'f71d6bb069f4da310cd50c1bf3aa72430a1c337743080b276b9feaa2f2fa6d93',\
   'a28608138cb932b81f2c4af1b313419ce1290366195eea6bc2ee7835919600d1',\
   '6e22c6426697b85c84f54431e057b416e3a081b8917b9cc72cbc87a6642f0165',\
   'ea21f3ffb10c32ffb5f4b51c4d38f4bc6c9c0275cdb45a2cca30f14e1c3e81c3',\
   '5b289f88ef6a94e01b3afc19689069461dc481099fbb21de7caeac082cf72b20',\
   'a40407b85632243158739b6f9b04d453d2f8ad5b1f2a70fe8127eaf0c6411d1b',\
```

Team 23: Ressa Reneth Sarreal & Dominic Schellin
CMSC426 - Dr. Marron
Lab 3: Password Recovery Lab
Due: 05/01/2018

```
   'bf710df7c70f0d4ee06506ed28ea3ac610ae8953eaa1d8d45e968c4d6d9a0955',\
   '5783d38e7c1bfff74255bd781f81a1d24346320984701a7ffec24008ffd01205',\
   'ba204a52bd99db092c80b3597d5c8c44d51db73e85b3d41b0353d209752ace7f',\
   '2f9296215b8f7b1de1926be39650136bbea07d5d334faa8799fbb25f9233d3bf',\
   '2c8fa639a865fdcfa8a343d5bd254d8f2dcb17756e63c6db9dc5ac07d3f152fa',\
   '63e50a725f6ee805110ee2b39b1d020532745349a4d9bcb07f921e25792fb92b',\
   '24a06619390dc056b06a7b26e488573dee2c73998fdaae4257bcfcc58533b422',\
   'f08a3b8da4b7e0c2a202f7fb852a9bd5102d3f173e2af5536e9ab249663efb5f',\
   'ea6260f448b42dd56b01a042ad0e62bee3981253a82cf6358d9b2987ab4e61b1',\
   '260882a1e0e0ef85216bb7f8360e2380cf50c9d049da293b758e6f448a8089dc',\
   '13c70f5ae0d3f77de84ce80fed9d2394d739c2d7fc44cbc4fec9fb8c8842806f',\
   '0980b7f78f23311f39051ae446e50ec6edf70b0214a77f26d93737b17898313f',\
   '356d6cf11a472f330d7459af79d3b6432bcea42dc1470c6de5f6fd7432669c4a',\
   '141a2f41176f165be0cf194835f092e4c2ff1524a27726b34d2021d1ff881bb3',\
   'ae85e8ef36c1a8d57a20a32b110812921d2dd067b5086c3e292a43fefe158ddb',\
   '0642505800901ac234c02599ec6e4a956e1c38e24739f0125cd87bd1ed8c7072',\
   'b132e84763654848033d7416db001badc607579fe3fda9ecb2e6e7bfeb18d306',\
   '4e117e1a696f9f431cc9f4d9db9a74d788185505a1c33f17cfcf8c781cf8180a',\
   'a839f30eecb8d5841f1a41d9697fa178e20b1c6c891247337b9d9dc73138df4c',\
   '3cd82f0526c1c19cd4a6b894d96e6ec16d48bfdd1e95497b247b0b4846501f08',\
   'ebe7e734801aea6741bfc7a3cfb960b7530faaa172418290c02c908f5637a68d',\
   '4fcf027c0220d5db40587732beeaaf30f63e98333821ff2f7d23ab9c50131c0c',\
   '8eabfcc69c38718e739c28c4ad968024b766a49b7499570336c4fb9d826b0ec8',\
   'eb28d9f20cd59f954b7808c04dea061e68e65d30534c33b4f344aa20935718e7',\
]

# List of valid password characters
alpha = list("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789./")

# gen_Rmask() - generate masks for return functions
#   R(i, hash) = R0(hash ^ Rmask[i])
def gen_Rmask(num_masks):
      Rmask = []
      sha = SHA256.new()
      for i in range(num_masks):
        sha.update(str(i))
        Rmask.append(sha.hexdigest())
      return Rmask

# R() - return function
def R(hash, Rmask, alpha, length):

      alpha_mask = pow(2, 6) - 1
      pw = ""

      # Resolves improper hex input
      hash = hash.strip('L')
      if len(hash) % 2 == 1:
        hash = "0" + hash

      # xor the Rmask with the hash
      bhash = bytearray.fromhex(hash)
```

Team 23: Ressa Reneth Sarreal & Dominic Schellin
CMSC426 - Dr. Marron
Lab 3: Password Recovery Lab
Due: 05/01/2018

```
      bRmask = bytearray.fromhex(Rmask)
      for i in range(len(bhash)):
        bhash[i] = bhash[i] ^ bRmask[i]
      hash = b2a_hex(bhash)

      # generate the password
      for i in range(length):
        index = alpha_mask & ord(hash[(2 * i):(2 * (i + 1))].decode('hex'))
        pw = pw + alpha[index]

      return pw

if __name__ == '__main__':

      pw_length = 5 # password length
      num_passwords = 0 # number of passwords found

      # prefix set for distinguished endpoint test
      prefix_set = set(["000", "001", "002", "003"])
      prefix_len = 3

      Rmask = gen_Rmask(1024) # Mask used to get different R functions (based on block number)
      max_string_len = 3072 # Maximum length a TMTO chain can be

      outfile = open('tmto_atk_results.txt', 'w')
      print ('Results will be written to ' + outfile.name + '.')

      for block_num in range(1024):

        infile1 = open('catalog_30bit_v2.block' + str(block_num), 'r')
        infile2 = open('catalog_30bit_v2.block' + str(block_num) + '_2', 'r')
        dict1 = load(infile1) # TMTO dictionary for set 1
        dict2 = load(infile2) # TMTO dictionary for set 2
        infile1.close()
        infile2.close()

        for hash in hash_rand[:]:
                digest = hash
                string_length = 0
                # Iterate through R function/hash chain until a prefix is found or chain length is
exceeded
                while (digest[:prefix_len] not in prefix_set) and (string_length <= max_string_len):
                        pw = R(digest, Rmask[block_num], alpha, pw_length)
                        digest = SHA256.new(pw).hexdigest()
                        string_length = string_length + 1
                # If a matching endpoint is found in dictionary 1
                if digest in dict1:
                        # iterate through the chain from the beginning
                        string_length = 0
                        pw = dict1[digest]
                        digest = SHA256.new(pw).hexdigest()
                        while string_length <= max_string_len:
```

Team 23: Ressa Reneth Sarreal & Dominic Schellin
CMSC426 - Dr. Marron
Lab 3: Password Recovery Lab
Due: 05/01/2018

```
                                # If a password is found
                                if digest == hash:
                                        outfile.write(hash + ' ' + pw + '\n')
                                        outfile.flush() # Write to file immediately
                                        hash_rand.remove(hash) # Remove password hash from the list of
hashes to check
                                        num_passwords = num_passwords + 1
                                        print ('Found password ' + str(num_passwords) + ': ' + pw + ' - '
+ hash)
                                        break
                                pw = R(digest, Rmask[block_num], alpha, pw_length)
                                digest = SHA256.new(pw).hexdigest()
                                string_length = string_length + 1
                # If a matching endpoint is found in dictionary 2
                elif digest in dict2:
                        # iterate through the chain from the beginning
                        string_length = 0
                        pw = dict2[digest]
                        digest = SHA256.new(pw).hexdigest()
                        while string_length <= max_string_len:
                                # If a password is found
                                if digest == hash:
                                        outfile.write(hash + ' ' + pw + '\n')
                                        outfile.flush() # Write to file immediately
                                        hash_rand.remove(hash) # Remove password hash from the list of
hashes to check
                                        num_passwords = num_passwords + 1
                                        print ('Found password ' + str(num_passwords) + ': ' + pw + ' - '
+ hash)
                                        break
                                pw = R(digest, Rmask[block_num], alpha, pw_length)
                                digest = SHA256.new(pw).hexdigest()
                                string_length = string_length + 1
        outfile.close()
        print ('Finished.')
```

---

5 Appendix B | tmto_atk_results.txt

```
4e117e1a696f9f431cc9f4d9db9a74d788185505a1c33f17cfcf8c781cf8180a FcgUQ
ebe7e734801aea6741bfc7a3cfb960b7530faaa172418290c02c908f5637a68d ZIDgN
040cfac289fffeb5aa2b0202da0851371e99f4433b58d81063a526d20e761720 VGBBL
220e3de048e58de267a0ce45388cc51b7006fa359267278b1e46290452ebcd87 sspui
260882a1e0e0ef85216bb7f8360e2380cf50c9d049da293b758e6f448a8089dc 5dAjH
658082bfbc45e61d6d078170f9b09c6f7f04b8f5e4a87c3199bba7ff64891a20 Qfe6M
f71d6bb069f4da310cd50c1bf3aa72430a1c337743080b276b9feaa2f2fa6d93 Yb3ye
5783d38e7c1bfff74255bd781f81a1d24346320984701a7ffec24008ffd01205 2TOgW
6e22c6426697b85c84f54431e057b416e3a081b8917b9cc72cbc87a6642f0165 ObcZM
2c8fa639a865fdcfa8a343d5bd254d8f2dcb17756e63c6db9dc5ac07d3f152fa QvE2N
605f5e4d916e03a87347aa9ca40b398251a59b2a689632fffca74ec9a6c95c5e 5/V1T
1604d5ebd17cad1fc79a1206473429796c8d90b76f07b1eb5b04869d83c9a44f qUKWX
5d7af6be8e7ab47467dc87b2470e16cc8fe6986c40537936d5cdc868b3c2d51b 77UlD
4fcf027c0220d5db40587732beeaaf30f63e98333821ff2f7d23ab9c50131c0c /qB1z
ae85e8ef36c1a8d57a20a32b110812921d2dd067b5086c3e292a43fefe158ddb GYp5X
e70fceb6c4dacced90d61ed6b7ea5aede791855ef126c213551034410aa7c052 NmUl9
3c521d6e4870c3783d0bcc49a68c30fb726816aeb8b72af6a44328d1a0e8d7a4 NIAo1
24a06619390dc056b06a7b26e488573dee2c73998fdaae4257bcfcc58533b422 Gm6L5
a28608138cb932b81f2c4af1b313419ce1290366195eea6bc2ee7835919600d1 p6.pa
a40407b85632243158739b6f9b04d453d2f8ad5b1f2a70fe8127eaf0c6411d1b 7j.MK
185db25b849a2535541c219f9a9df3c8d8a8f9a141f9d20f0902f15faa7fd6a3 RGzLM
5b289f88ef6a94e01b3afc19689069461dc481099fbb21de7caeac082cf72b20 8TOg0
f08a3b8da4b7e0c2a202f7fb852a9bd5102d3f173e2af5536e9ab249663efb5f rPwBN
141a2f41176f165be0cf194835f092e4c2ff1524a27726b34d2021d1ff881bb3 kk/o3
1d9304443d89fc8804e4fcb45a4292a5c8ecd6e7522de0c65c4a4c583c0e989b eTG7d
3cd82f0526c1c19cd4a6b894d96e6ec16d48bfdd1e95497b247b0b4846501f08 yQI.O
a839f30eecb8d5841f1a41d9697fa178e20b1c6c891247337b9d9dc73138df4c nQhF8
287e899c2d582c5e97b9d292263a555c3f7ca091211d6d6cbdd8a40c2c388b28 0RWG3
13c70f5ae0d3f77de84ce80fed9d2394d739c2d7fc44cbc4fec9fb8c8842806f E1C.S
10ff81984bb4fd2f62cae3859fe7d74fb7293b5ef7198042428a1d0482763f29 CA2RQ
```

Team 23: Ressa Reneth Sarreal & Dominic Schellin
CMSC426 - Dr. Marron
Lab 3: Password Recovery Lab
Due: 05/01/2018

```
C:\Users\nick\Documents\CMSC426>tmto_atk.py
Results will be written to tmto_atk_results.txt.
Found password 1: FcgUQ - 4e117e1a696f9f431cc9f4d9db9a74d788185505a1c33f17cfcf8c781cf8180a
Found password 2: ZIDgN - ebe7e734801aea6741bfc7a3cfb960b7530faaa172418290c02c908f5637a68d
Found password 3: VGBBL - 040cfac289fffeb5aa2b0202da0851371e99f4433b58d81063a526d20e761720
Found password 4: sspui - 220e3de048e58de267a0ce45388cc51b7006fa359267278b1e46290452ebcd87
Found password 5: 5dAjH - 260882a1e0e0ef85216bb7f8360e2380cf50c9d049da293b758e6f448a8089dc
Found password 6: Qfe6M - 658082bfbc45e61d6d078170f9b09c6f7f04b8f5e4a87c3199bba7ff64891a20
Found password 7: Yb3ye - f71d6bb069f4da310cd50c1bf3aa72430a1c337743080b276b9feaa2f2fa6d93
Found password 8: 2TOgW - 5783d38e7c1bfff74255bd781f81a1d24346320984701a7ffec24008ffd01205
Found password 9: ObcZM - 6e22c6426697b85c84f54431e057b416e3a081b8917b9cc72cbc87a6642f0165
Found password 10: QvE2N - 2c8fa639a865fdcfa8a343d5bd254d8f2dcb17756e63c6db9dc5ac07d3f152fa
Found password 11: 5/V1T - 605f5e4d916e03a87347aa9ca40b398251a59b2a689632fffca74ec9a6c95c5e
Found password 12: qUKWX - 1604d5ebd17cad1fc79a1206473429796c8d90b76f07b1eb5b04869d83c9a44f
Found password 13: 77UlD - 5d7af6be8e7ab47467dc87b2470e16cc8fe6986c40537936d5cdc868b3c2d51b
Found password 14: /qB1z - 4fcf027c0220d5db40587732beeaaf30f63e98333821ff2f7d23ab9c50131c0c
Found password 15: GYp5X - ae85e8ef36c1a8d57a20a32b110812921d2dd067b5086c3e292a43fefe158ddb
Found password 16: NmUl9 - e70fceb6c4dacced90d61ed6b7ea5aede791855ef126c213551034410aa7c052
Found password 17: NIAo1 - 3c521d6e4870c3783d0bcc49a68c30fb726816aeb8b72af6a44328d1a0e8d7a4
Found password 18: Gm6L5 - 24a06619390dc056b06a7b26e488573dee2c73998fdaae4257bcfcc58533b422
Found password 19: p6.pa - a28608138cb932b81f2c4af1b313419ce1290366195eea6bc2ee7835919600d1
Found password 20: 7j.MK - a40407b85632243158739b6f9b04d453d2f8ad5b1f2a70fe8127eaf0c6411d1b
Found password 21: RGzLM - 185db25b849a2535541c219f9a9df3c8d8a8f9a141f9d20f0902f15faa7fd6a3
Found password 22: 8TOg0 - 5b289f88ef6a94e01b3afc19689069461dc481099fbb21de7caeac082cf72b20
Found password 23: rPwBN - f08a3b8da4b7e0c2a202f7fb852a9bd5102d3f173e2af5536e9ab249663efb5f
Found password 24: kk/o3 - 141a2f41176f165be0cf194835f092e4c2ff1524a27726b34d2021d1ff881bb3
Found password 25: eTG7d - 1d9304443d89fc8804e4fcb45a4292a5c8ecd6e7522de0c65c4a4c583c0e989b
Found password 26: yQI.O - 3cd82f0526c1c19cd4a6b894d96e6ec16d48bfdd1e95497b247b0b4846501f08
Found password 27: nQhF8 - a839f30eecb8d5841f1a41d9697fa178e20b1c6c891247337b9d9dc73138df4c
Found password 28: 0RWG3 - 287e899c2d582c5e97b9d292263a555c3f7ca091211d6d6cbdd8a40c2c388b28
Found password 29: E1C.S - 13c70f5ae0d3f77de84ce80fed9d2394d739c2d7fc44cbc4fec9fb8c8842806f
Found password 30: CA2RQ - 10ff81984bb4fd2f62cae3859fe7d74fb7293b5ef7198042428a1d0482763f29
Finished.
```

Team 23: Ressa Reneth Sarreal & Dominic Schellin
CMSC426 - Dr. Marron
Lab 3: Password Recovery Lab
Due: 05/01/2018

---

6 Appendix C | dict_atk.py Code

```
from Crypto.Hash import SHA256

# The following two lists are provided

# Unsalted Hashes
hash_nosalt = [\
   '2055c34950486290b60268aefb586edc8877e2294edd52a86614a8984f40a8fb',\
   'ca2e89b1f92c980df25543a423500466e08888485711de352f4f892c704198c1',\
   '8c414fd826c961685aeacb57aec8e1154a029a58bab88ab33a14751c3b8386f2',\
   '94309e4772adec972000e4b28efc83208eea13e7b5dddcbb27ff88a2d3da7248',\
   'bdc326dbf5c0ab5238db6e208a4537f7ab6b548723fc0bc945cbc8fcd26a8d6a',\
]

# Salted Hashes
hash_salted = [\
   '38087:e0b869e99b57dec36a2a013d3b04eb286e4ccf1a054a0a029811f25b5cd75ca5',\
   '81933:d531515735375e11c536d6de95d8b781d68b72fe1714f3f802f4eec1a22b004e',\
   '54342:b061e304a9e8534b7dfe7dd75d12abcd72d1e8bb730cc8d0ec772ff6c6bb4d62',\
   '22517:848b261753883417d4a291288ee38507abcabd7a7e21f280653ea0ba0f8e047f',\
   '57281:08301510d6bc7bfea68272bacbaf5ce120366d17d0b0aa9bcf9c1245f03abd97',\
   '92148:c260cfe88a1ca4d5edac7850fdc5d2c7bc67130f788a5b311e05c7f57ff27fb8',\
]

# file that contains passwords and their respective SHA256 hashes
hash_dict_file = open('rockyou.100000.hash', 'r')

pwtable = {} # dictionary for passwords indexed by their hashes
pw_list = [] # list of passwords loaded from hash_dict_file
hashes = [] # list of hashes to crack. built from hash_nosalt and hash_salted
salts = [''] # list of salts used in hash_salted (the first element is for no salt)

# build list of relevant salts and hashes to crack
for element in hash_nosalt:
      hashes.append(element)
for element in hash_salted:
      partitioned_element = element.partition(':')
      salts.append(partitioned_element[0])
      hashes.append(partitioned_element[2])

# build dictionary of possible passwords
for line in hash_dict_file:
      fields = line.rstrip('\n').partition(' ')
      pw_list.append(fields[2])
hash_dict_file.close()
print ("Imported " + hash_dict_file.name + ".")

# Build the password hash dictionary to include:
#  1. the passwords loaded from file
```

Team 23: Ressa Reneth Sarreal & Dominic Schellin
CMSC426 - Dr. Marron
Lab 3: Password Recovery Lab
Due: 05/01/2018

```
#  2. passwords from file w/ 1 digit appended
#  3. passwords from file w/ 1 digit prepended
#  4. passwords from file with salts added to their hashes AND at most one of the previous types
for salt in salts[:-2]: # The last two salts will not be found by the above criteria (they're tested for
later)
      print ("Building dictionary for salt: '" + salt + "'")
      for password in pw_list:
        pwtable[SHA256.new(salt + password).hexdigest()] = password
        # all appended & prepended digits
        pwtable[SHA256.new(salt + password + '0').hexdigest()] = password + '0'
        pwtable[SHA256.new(salt + password + '1').hexdigest()] = password + '1'
        pwtable[SHA256.new(salt + password + '2').hexdigest()] = password + '2'
        pwtable[SHA256.new(salt + password + '3').hexdigest()] = password + '3'
        pwtable[SHA256.new(salt + password + '4').hexdigest()] = password + '4'
        pwtable[SHA256.new(salt + password + '5').hexdigest()] = password + '5'
        pwtable[SHA256.new(salt + password + '6').hexdigest()] = password + '6'
        pwtable[SHA256.new(salt + password + '7').hexdigest()] = password + '7'
        pwtable[SHA256.new(salt + password + '8').hexdigest()] = password + '8'
        pwtable[SHA256.new(salt + password + '9').hexdigest()] = password + '9'
        pwtable[SHA256.new(salt + '0' + password).hexdigest()] = '0' + password
        pwtable[SHA256.new(salt + '1' + password).hexdigest()] = '1' + password
        pwtable[SHA256.new(salt + '2' + password).hexdigest()] = '2' + password
        pwtable[SHA256.new(salt + '3' + password).hexdigest()] = '3' + password
        pwtable[SHA256.new(salt + '4' + password).hexdigest()] = '4' + password
        pwtable[SHA256.new(salt + '5' + password).hexdigest()] = '5' + password
        pwtable[SHA256.new(salt + '6' + password).hexdigest()] = '6' + password
        pwtable[SHA256.new(salt + '7' + password).hexdigest()] = '7' + password
        pwtable[SHA256.new(salt + '8' + password).hexdigest()] = '8' + password
        pwtable[SHA256.new(salt + '9' + password).hexdigest()] = '9' + password

print "Comparing password hashes to dictionary..."
# use the built-up dictionary to match hashes with passwords
results_file = open('dict_atk_results.txt', 'w')
num_passwords = 0
for hash in hashes:
      if hash in pwtable:
        num_passwords = num_passwords + 1
        results_file.write(hash + ' ' + pwtable[hash] + '\n')
        print('Found ' + str(num_passwords) + ' total password(s).')
results_file.flush() # write results immediately in case program is terminated early
                                # (the rest takes a long time)

# Look for password hashes matching the last unsalted hash
# (which should be a concatenation of two passwords by process of elimination)
print "The next three steps will take a while."
print "Guessing unsalted concatenated passwords..."
for password in pw_list:
      for password2 in pw_list:
        if SHA256.new(password + password2).hexdigest() == hashes[4]:
                num_passwords = num_passwords + 1
                print('Found ' + str(num_passwords) + ' total password(s).')
                results_file.write(hashes[4] + ' ' + password + password2 + '\n')
```

Team 23: Ressa Reneth Sarreal & Dominic Schellin
CMSC426 - Dr. Marron
Lab 3: Password Recovery Lab
Due: 05/01/2018

```
                break
        else:
          continue
        break

# Look for password hashes matching the last two salted hashes
# (which should be a concatenation of two passwords by process of elimination)
for salt in salts[-2:]:
        print ("Guessing concatenated passwords for salt: '" + salt + "'")
        for password in pw_list:
          for password2 in pw_list:
                  guess = SHA256.new(salt + password + password2).hexdigest()
                  if guess == hashes[9]:
                          num_passwords = num_passwords + 1
                          print('Found ' + str(num_passwords) + ' total password(s).')
                          results_file.write(hashes[9] + ' ' + password + password2 + '\n')
                          break
                  if guess == hashes[10]:
                          num_passwords = num_passwords + 1
                          print('Found ' + str(num_passwords) + ' total password(s).')
                          results_file.write(hashes[10] + ' ' + password + password2 + '\n')
                          break
          else:
                  continue
          break
results_file.close()
print ('Passwords written to ' + results_file.name + '.')
```

Team 23: Ressa Reneth Sarreal & Dominic Schellin
CMSC426 - Dr. Marron
Lab 3: Password Recovery Lab
Due: 05/01/2018

---

7 Appendix D | Unsalted and Salted Password Attack Results

```
2055c34950486290b60268aefb586edc8877e2294edd52a86614a8984f40a8fb khristine
ca2e89b1f92c980df25543a423500466e08888485711de352f4f892c704198c1 insatiable
8c414fd826c961685aeacb57aec8e1154a029a58bab88ab33a14751c3b8386f2 babytiger
94309e4772adec972000e4b28efc83208eea13e7b5dddcbb27ff88a2d3da7248 freeride3
e0b869e99b57dec36a2a013d3b04eb286e4ccf1a054a0a029811f25b5cd75ca5 pinkpanter
d531515735375e11c536d6de95d8b781d68b72fe1714f3f802f4eec1a22b004e hocuspocus
b061e304a9e8534b7dfe7dd75d12abcd72d1e8bb730cc8d0ec772ff6c6bb4d62 4Merlin
848b261753883417d4a291288ee38507abcabd7a7e21f280653ea0ba0f8e047f sauron1
bdc326dbf5c0ab5238db6e208a4537f7ab6b548723fc0bc945cbc8fcd26a8d6a bermudatriangle
08301510d6bc7bfea68272bacbaf5ce120366d17d0b0aa9bcf9c1245f03abd97 qwertyfamily
c260cfe88a1ca4d5edac7850fdc5d2c7bc67130f788a5b311e05c7f57ff27fb8 barbiehaters
```

```
C:\Users\nick\Documents\CMSC426>python dict_atk.py
Imported rockyou.100000.hash.
Building dictionary for salt: ''
Building dictionary for salt: '38087'
Building dictionary for salt: '81933'
Building dictionary for salt: '54342'
Building dictionary for salt: '22517'
Comparing password hashes to dictionary...
Found 1 total password(s).
Found 2 total password(s).
Found 3 total password(s).
Found 4 total password(s).
Found 5 total password(s).
Found 6 total password(s).
Found 7 total password(s).
Found 8 total password(s).
The next three steps will take a while.
Guessing unsalted concatenated passwords...
Found 9 total password(s).
Guessing concatenated passwords for salt: '57281'
Found 10 total password(s).
Guessing concatenated passwords for salt: '92148'
Found 11 total password(s).
Passwords written to dict_atk_results.txt.
```