PSEC Open Assessment

Exam no: Y0076159

13th February 2017

Contents

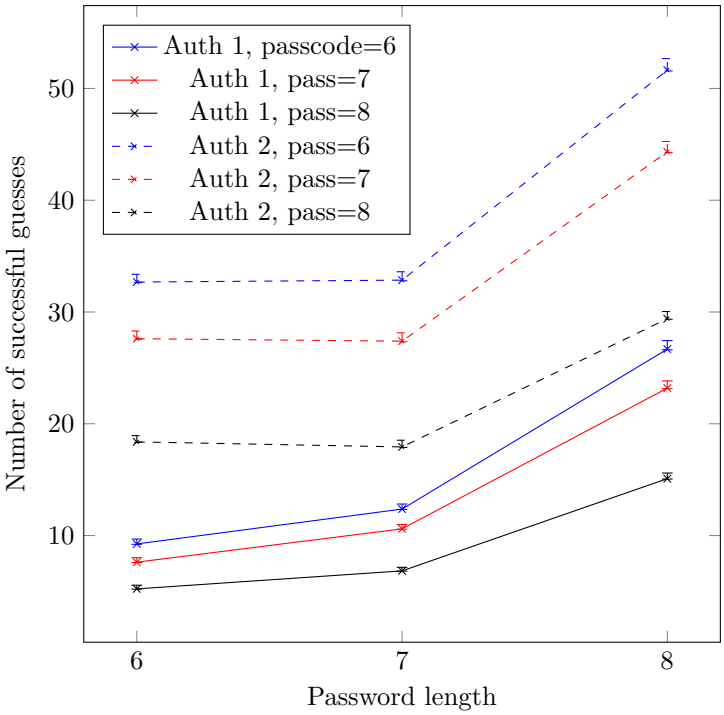

Figure 1: Successful guesses from (50×10^6) attempts using varying passcode length

1 Question 1

i.

2 Question 2

i. Design & Implementation

ii. Results

Our implementation was ran 150 times for $n1, n2 \in \{6, 7, 8\}$, resulting in data with extremely precise confidence intervals as seen in Figure 1. The results demonstrate that for all combinations of password and passcode length, the second authentication system is more vulnerable

iii. Analysis

iv. Counter-measures

A Mainswitch.py

```
import argparse as ap
import random
import multiprocessing

dictionary_filename = "data/500-worst-passwords-processed.txt"
password_frequency_filename = "data/rockyou-withcount-processed.txt"

results_file = "data.csv"


# Method to pick a key from a dictionary based on the frequency of the key
# (frequency of a key is the key's value in the dict)
def weighted_pick(passwords, maxkey):
    r = random.randint(0, maxkey - 1)
    return passwords[r].v


def match(password, guess):
    if len(password) == 0:
        return len(guess) == 0

    indices = random.sample(range(0, len(password)), 3)
    for a in range(0, 3):
        x = indices[a]
        try:
            if password[x] != guess[x]:
                return False
        except IndexError:
            return False
    return True


def matchfull(password, guess):
    return password == guess


# Class to store passwords for guesses
class PasswordAttackList(object):
    # Create using file with line-seperated passwords
    def __init__(self, filename, n1, n2):
        self.passwords = []
        self.valid_passwords = []
        self.valid_passcodes = []
        f = open(filename, 'r')
        line = f.readline()
        while line:
            self.passwords.append(line.strip())
            line = f.readline()
        f.close()

        self.valid_passwords = filter(lambda x: len(x) >= n1, self.passwords)
        self.valid_passcodes = filter(lambda x: len(x) == n2, self.passwords)

        if len(self.valid_passwords) == 0:
            raise ValueError('Invalid␣password␣lengths,␣no␣passwords␣of␣this␣length␣
                exist␣in␣the␣dictionary␣file.')
        if len(self.valid_passcodes) == 0:
            raise ValueError('Invalid␣passcode␣lengths,␣no␣passcodes␣of␣this␣length␣
                exist␣in␣the␣dictionary␣file.')

    # Method to randomly pick a password/passcode of valid lengths from the list of
        common passwords
    def pick_password_passcode(self, n1, n2):
        password = random.choice(self.valid_passwords)
        passcode = random.choice(self.valid_passcodes)
        return password, passcode


class Value:
    def __init__(self, v=None):
        self.v = v


# Class to store passwords to be guessed
class PasswordPicker(object):
```

```
    # Create using file with frequency / password tuples on each line
    def __init__(self, filename, n1, n2):
        f = open(filename, 'r')
        password_dict = dict()
        line = f.readline()

        self.valid_passwords = []
        self.valid_passcodes = []

        self.totalpasswords = 0
        self.totalpasscodes = 0
        while line:
            try:
                v, k = line.strip().split(' ', 1)
            except ValueError:
                v = line.strip()
                k = ''
            obj = Value(k)
            v = int(v)
            if len(k) >= n1:
                for x in range(0, v):
                    self.valid_passwords.append(obj)
                self.totalpasswords += v
            if len(k) == n2:
                for x in range(0, v):
                    self.valid_passcodes.append(obj)
                self.totalpasscodes += v

            password_dict[k] = int(v)
            line = f.readline()
        f.close()

        # self.valid_passwords = dict((k, password_dict[k]) for k in password_dict.keys
            () if len(k) >= n1)
        # self.valid_passcodes = dict((k, password_dict[k]) for k in password_dict.keys
            () if len(k) == n2)
        if len(self.valid_passwords) == 0:
            raise ValueError('Invalid password lengths, no passwords of this length
                exist in the dictionary file.')
        if len(self.valid_passcodes) == 0:
            raise ValueError('Invalid passcode lengths, no passcodes of this length
                exist in the dictionary file.')

    # Method to randomly pick a password/passcode combination from the list
    def pick_password_passcode(self):
        return weighted_pick(self.valid_passwords, self.totalpasswords), weighted_pick(
            self.valid_passcodes,
                                                                                self
                                                                                    .
                                                                                totalpasscod
                                                                                )



def main():
    random.seed()
    # Parse program arguments
    parser = ap.ArgumentParser(description="Calculate probability of dictionary attack
        for each authentication system")
    parser.add_argument("m", metavar="M", help="Number of guesses", type=int)
    parser.add_argument("n1", metavar="N1", help="Min password length", type=int)
    parser.add_argument("n2", metavar="N2", help="Exact passcode length", type=int)
    parser.add_argument("-its", metavar="ITS", help="Number of runs to undertake",
        default=1, type=int)
    parser.add_argument("-save_file", metavar="F", help="Filename to save to", default=
        results_file)

    args = parser.parse_args()

    if args.n2 < 3:
        raise ap.ArgumentTypeError("N2 minimum value is 3")

    print "Loading guess passwords"
    # Initialise list of passwords to sample from during attack
    password_list = PasswordAttackList(dictionary_filename, args.n1, args.n2)

    # Pick user password/passcode (to be guessed)
```

```
    password_distrib = PasswordPicker(password_frequency_filename, args.n1, args.n2)

    for i in range(0, args.its):
        full_guesses = 0
        char_guesses = 0
        for x in range(0, args.m):
            guess_password, guess_passcode = password_list.pick_password_passcode(args.
                n1, args.n2)
            password, passcode = password_distrib.pick_password_passcode()
            if match(passcode, guess_passcode):
                if matchfull(password, guess_password):
                    full_guesses += 1
                if match(password, guess_password):
                    char_guesses += 1
            if x % 1000000 == 0:
                print float(x) / float(args.m), full_guesses, char_guesses
        row = (full_guesses, char_guesses, "\r")
        fd = open("results/" + args.save_file, 'a+')
        fd.write(",".join(map(str, row)))
        fd.close()


if __name__ == "__main__":
    main()
```

B Additional statistics

i. Individual statistics

		Auth system and password length					
Mean	**Password len**	**Auth1, 6**	**Auth1, 7**	**Auth1, 8**	**Auth2, 6**	**Auth2, 7**	**Auth2, 8**
	6	9.25	7.62	5.23	32.68	27.61	18.38
	7	12.37	10.60	6.84	32.85	27.39	17.93
	8	26.68	23.19	15.09	51.64	44.33	29.40
Median							
	6	9.00	7.00	5.00	33.00	28.00	18.00
	7	12.00	10.50	6.50	33.00	28.00	18.00
	8	27.00	23.00	15.00	52.00	44.50	29.00
STD							
	6	3.26	2.96	2.45	5.17	5.14	4.16
	7	3.35	2.95	2.38	5.61	5.58	4.46
	8	5.65	4.80	3.75	7.64	6.86	4.80
Min							
	6	2.00	2.00	1.00	21.00	16.00	9.00
	7	5.00	4.00	2.00	21.00	10.00	6.00
	8	12.00	11.00	3.00	32.00	27.00	20.00
Max							
	6	18.00	15.00	14.00	46.00	43.00	29.00
	7	21.00	20.00	13.00	48.00	43.00	31.00
	8	43.00	38.00	24.00	75.00	64.00	44.00
0.1 confidence							
	6	0.033	0.030	0.025	0.053	0.053	0.043
	7	0.034	0.030	0.024	0.058	0.057	0.046
	8	0.058	0.049	0.039	0.078	0.070	0.049
0.9 confidence							
	6	0.438	0.398	0.329	0.695	0.690	0.559
	7	0.450	0.397	0.319	0.753	0.749	0.599
	8	0.759	0.645	0.504	1.026	0.921	0.645

Table 1: Individual statistics for each authentication system and password/passcode length combination

ii. Data significance

Password/Passcode length	Auth 1 median	Auth 2 median	Mann whitney Z-score	p value
6,6	9	33	-14.974	p<0.01
6,7	7	27.5	-14.974	p<0.01
6,8	5	18	-15.100	p<0.01
7,6	12	33	-14.974	p<0.01
7,7	10.5	28	-14.793	p<0.01
7,8	6.5	18	-14.596	p<0.01
8,6	27	52	-14.898	p<0.01
8,7	23	44.5	-14.801	p<0.01
8,8	15	29	-14.794	p<0.01

Table 2: Statistical tests between authentication systems for equivalent password/passcode length