Information Secutiry - Week 1

Arvid Lindstrom - s2740761, Nil Stolt Anso - s2705338,
Razvan Andrei Poinaru - s2914751

September 19, 2017

1 Exercise 1

1.1 Link

http://www.rug.nl/society-business/centre-for-information-technology/security/aup/

1.2 Questions and Answers

Q1: *What (if any) are the differences between the responsibilities of 'ordinary' users and systems managers? Do systems managers have special privileges and responsibilities (if so, what are they)?*
A1: By 'ordinary' users we refer to both internal and external users, such as students, personnel, guests etc. While the system managers are responsible for the security of the system itself and will take care of the installation and maintanance of the required and available software.
Q2: *What is the ground-rule upon which the RUG's AUP is based?*
A2: The ground-rule on which this AUP is based is similar to the ground-rule on which traffic is based: the users of the university computer systems may not endanger these systems, nor may they hinder other users. Some implications of this ground-rule are that users are not allowed to send unsolicited email or try to obtain or use other users' passwords; neither occasionally, nor 'for fun'.
Q3: *Mention four advices for users of 'RuGnet'.*
A3: 1. Keep your access information secret: don't hand this information over to friends or acquaintances. 2. Don't type your password when somebody watches you type. 3. Do not use personal data about yourself, your friends or relatives when constructing your password 4. Do not use existing words or abbreviations(like rcrug of ppsw)
Q4: *Describe four actions that are prohibited by the RUG's AUP*
A4: 1. To modify or to remove hard- or software without having obtained prior permission from proper authorities 2. To use university computer systems, or to use any software or stored data without having obtained prior permission from proper authorities 3. To send any email using other people's names and/or addresses, or to read or distribute other people's mail without having obtained their consent in advance 4. To alter IP-addresses or other identifying data of university computer systems (e.g., by using spoofing)
Q5: *What sanctions can be applied to those who violate the AUP?*
A5: The access rights of the suspect user may be restricted or suspended.
Q6: *If a sanction is applied to you, where can you go to challenge that sanction?*
A6: The suspect may file an objection to this restriction or suspension with the chair of his/her department.

2 Exercise 2

2.1 Key used

The key: uoieazyxwvtsrqpnmlkjhgfdcb
Maps onto:
abcdefghijklmnopqrstuvwxyz

2.2 Decrypted text

```
9 common security awareness mistakes (and how to fix them)

To err is human, but to err in cyber security can cause major damage to an
organization. It will never be possible to be perfect, but major improvement
is possible, just by being aware of some of the most common mistakes and their
consequences.



1. Falling for phishing: One of the most common mistakes.
2. Unauthorized application or cloud use, known as shadow IT.
3. Weak or misused passwords
4. Remote insecurity: This is the common practice of transferring files
   between work and personal computers
5. Disabling security controls: This is usually done by users with
   administrative privileges, to make things easier for employees
6. Clueless social networking
7. Poor mobile security
8. Too many privileges
9. Failure to update or patch software

See also http://www.csoonline.com/article/2877259/security-awareness/
            nine-common-security-awareness-mistakes-and-how-to-fix-them.html
```

2.3 Source of Program

If the system argument [-o] is not given, the program will ignore non-letters and will not honor letter casing. If the system argument [-d] is not given, the program will encrypt. In the other hand, if [-d] is given, the program will decrypt. The last argument when running the program should be the key. A key should be as long as letters in there are in the alphabet,

```
#!/usr/bin/python

import sys

def readFile(name):
    file = open(name,'r')
    return file.read()

def substitute(text, o, d, key):
    print '------------------------'
    print "(KEY LENGTH SHOULD BE SIZE OF ALPHABET)"
    print "Key length: " + str(len(key)) + "/n"
    if key == None:
        print "Key == Null"
        print "text not converted"
```

```
    else:
        print "Key:"
        print "abcdefghijklmnopqrstuvwxyz"
        print "|"
        print "V"
        print key
    print "\n"
    if d == True:
        print "Decrypt"
    else:
        print "Encrypt"
    print "\n"
    if o == True:
        print "Keep non-letters, honor casing"
    else:
        print "Ignore non lower-case letters"
    print "------------------------\n"
    text2 = ""
    for ch in text:
        if d == False: #If encrypting
            if o == False:
                if ord(ch) >= ord('a') and ord(ch) <= ord('z'): #If lowercase
                    text2 += key[(ord(ch)-ord('a'))%len(key)]
                elif ord(ch) >= ord('A') or ord(ch) <= ord('Z'): #If upper-case
                    text2 += key[(ord(ch)-ord('A'))%len(key)]
            else:
                #Keep non-letters and Honor casing
                if ord(ch) >= ord('a') and ord(ch) <= ord('z'): #If lowercase
                    text2 += key[(ord(ch)-ord('a'))%len(key)]
                elif ord(ch) >= ord('A') or ord(ch) <= ord('Z'): #If upper-case
                    text2 += key[(ord(ch)-ord('A'))%len(key)]
                else: #If non-letter symbol
                    text2 += ch
        else: #If decrypting
            if o == False:
                if ord(ch) >= ord('a') or ord(ch) <= ord('z'): #If lowercase
                    pos = 0 #Count position
                    for l in key:
                        if l == ch:
                            text2 += chr(ord('a') + pos)
                            break
                        pos = pos+1
                elif ord(ch) >= ord('A') and ord(ch) <= ord('Z'): #If upper-case
                    pos = 0 #Count position
                    for l in key:
                        if chr(ord(l) - (ord('a') - ord('A'))) == ch:
                            text2 += chr(ord('a') + pos)
                            break
                        pos = pos+1;
            else:
                if ord(ch) >= ord('a') and ord(ch) <= ord('z'): #If lowercase
                    pos = 0 #Count position
                    for l in key:
                        if l == ch:
                            text2 += chr(ord('a') + pos)
```

```
                            break
                        pos = pos+1
                elif ord(ch) >= ord('A') and ord(ch) <= ord('Z'): #If upper-case
                    pos = 0 #Count position
                    for l in key:
                        if chr(ord(l) - (ord('a') - ord('A'))) == ch:
                            text2 += chr(ord('A') + pos)
                            break
                        pos = pos+1
                else: #If non-letter symbol
                    text2 += ch
    print "Processed text:\n"
    print text2


def main():
    text = readFile(sys.argv[1])
    print "Original text" + text + "\n"
    if len(sys.argv) > 2 and sys.argv[2] == "-o":
        if len(sys.argv) > 3 and sys.argv[3] == "-d":
            converted = substitute(text, True, True, sys.argv[4])
        else:
            converted = substitute(text, True, False, sys.argv[3])
    elif len(sys.argv) > 2 and sys.argv[2] == "-d":
        converted = substitute(text, False, True, sys.argv[3])
    elif len(sys.argv) > 2:
        converted = substitute(text, False, False, sys.argv[2])
    else:
        converted = substitute(text, False, False, None)


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

3 Exercise 3

3.1 Decrypted Text

welcometothecourseaboutinformationsecuritythiscourseisaboutsecuringinformationin
thiscontextwethinkforexampleabouthowtopreventtheunauthorizedreadingofinformation
orabouthowtopreventtheunauthorizedmodificationofinformationmanyencryptionmethods
existsomealreadythousandsyearsoldinitiallywellfocusonsimplemethodstoencryptinfor
mationfollowingthiswellusecharacteristicvaluesidentifyinginformationmakingitdiff
iculttomodifyinformationunnotifiedlaterinthiscoursewellintroducepersonalencrypti
onandwellstudytopicslikebufferoverflowexploitsandscrosssitescriptingihopeyoullen
joythiscourseaboutinformationsecurity

3.2 Human Readable Text

welcome to the course about information security
this course is about securing information in this
context we think for example about how to prevent
the unauthorized reading of information or about
how to prevent the unauthorized modification of
information many encryption methods exist some

```
already thousands years old initially well focus
on simple methods to encrypt information following
this well use characteristic values identifying
information making it difficult to modify information
unnotified later in this course well introduce personal
encryption and well study topics like buffer overflow
exploits and scross site scripting i hope youll enjoy this
course about information security
```

3.3 Shifts used:

The shift applied to obtain the text above was 20.
The smallest postive substitution cipher shift value used to recover the original text was 6.

3.4 Code

```
# 1. Find the encrypted text
cipherText = "qyfwigyninbywiolmyuvionchzilguncihmy\
wolcnsnbcmwiolmycmuvionmywolchachzilguncihchnbcmwi\
hnyrnqynbchezilyrugjfyuvionbiqnijlypyhnnbyohuonbil\
ctyxlyuxchaizchzilguncihiluvionbiqnijlypyhnnbyohuo\
nbilctyxgixczcwuncihizchzilguncihguhsyhwlsjncihgyn\
bixmyrcmnmigyuflyuxsnbiomuhxmsyulmifxchcncuffsqyff\
ziwomihmcgjfygynbixmniyhwlsjnchzilguncihziffiqchan\
bcmqyffomywbuluwnylcmncwpufoymcxyhnczschachzilgunc\
ihguechacnxczzcwofnnigixczschzilguncihohhinczcyxfu\
nylchnbcmwiolmyqyffchnlixowyjylmihufyhwlsjncihuhxq\
yffmnoxsnijcwmfceyvozzylipylzfiqyrjficnmuhxmwlimmm\
cnymwlcjnchacbijysioffyhdisnbcmwiolmyuvionchzilgun\
cihmywolcns"

def solveSubCipher(cipher, key):
    alpha = "abcdefghijklmnopqrstuvwxyz"
    output = ""

    for letter in cipherText:
        letter_idx = ord(letter) % ord('a')
        output += alpha[(letter_idx - key) % 26]
    return output

def findKeyFromAlphaShift(cipher):
    print "-----------------" # <-- helps reading output
    for i in range(26):
        print str(i) + " -- " + solveSubCipher(cipher, i)
    print "-----------------"

findKeyFromAlphaShift(cipherText)
# Key found to be 20!

# 2. Find the smallest postive substitution cipher shift value
# to return the original text

plainText = "welcometothecourseaboutinformationsecurity\
thiscourseisaboutsecuringinformationinthiscontextwethin\
kforexampleabouthowtopreventtheunauthorizedreadingofinf\
```

```
ormationorabouthowtopreventtheunauthorizedmodificationo\
finformationmanyencryptionmethodsexistsomealreadythousa\
ndsyearsoldinitiallywellfocusonsimplemethodstoencryptin\
formationfollowingthiswellusecharacteristicvaluesidenti\
fyinginformationmakingitdifficulttomodifyinformationunn\
otifiedlaterinthiscoursewellintroducepersonalencryption\
andwellstudytopicslikebufferoverflowexploitsandscrosssi\
tescriptingihopeyoullenjoythiscourseaboutinformationsecurity"

def encrypt(cipher, key):
    alpha = "abcdefghijklmnopqrstuvwxyz"
    output = ""

    for letter in cipherText:
        letter_idx = ord(letter) % ord('a')
        output += alpha[(letter_idx + key) % 26]
    return output

def findSmallestPositiveShift():
    shift = 0
    while (encrypt(cipherText, shift) != plainText):
        shift += 1
    return shift

print "Smallest positive shift = " + str(findSmallestPositiveShift())
```

4 Exercise 4

4.1 Decrypted text with spaces

```
i came to security from cryptography and thought of the problem in
a military like fashion most writings about security
come from this perspective and it can be summed up pretty easily
security threats are to be avoided using preventive countermeasures
this is how encryption works the threat is eavesdropping
and encryption provides the prophylactic this
could all be explained with block diagrams alice is communicating
with bob both are identified by boxes and there is a line
between them signifying the communication eve is the eavesdroppers
he also is a box and has a dotted line attached to the
communications line she is able to intercept the communication
the only way to prevent eve from learning what alice a
nd bob are talking about is through a preventive countermeasure
encryption theres no detection theres no response theres
no risk management you have to avoid the threat for decades
we have used this approach to computer security we draw boxes
around the different players and lines between them we define
different attackers eavesdroppers impersonators thieves
and their capabilities we use preventive countermeasures
like encryption and access control to avoid different threats
if we can avoid the threats weve won if we cant weve lost
imagine my surprise when i learned that the world doesnt
work this way some history from the vigenere wiki page the first
well documented description of a polyalphabetic cipher was
formulated by leon battista alberti around and used a metal
cipher disc to switch between cipher alphabets albertis
```

system only switched alphabets after several words and
switches were indicated by writing the letter of the corresponding
alphabet in thecipher text later in johannes trithemius in his
work poligraphia invented the tabula recta a critical component
of the vigenre cipher the trithemius cipher however only
provided a progressive rigid and predictable system for switching
between cipher alphabets what is now known as the vigenere cipher
was originally described by giovan battista bellaso in his book
la cifra del sig giovanbattista bellaso he built upon the tabula
recta of trithemius but added a repeating countersignakey to switch
cipher alphabets every letter whereas alberti
and trithemius used a fixed pattern of substitutions bellasos
scheme meant the pattern of substitutions could be easily
changed simply by selecting a new key keys were typically
single words or short phrases known to both parties in advance
or transmitted out of band along with the message bellasos
method thus required strong security for only the key as it
is relatively easy to secure a short keyphrase say by a
previous private conversation bellasos system was considerably
more secure blaisede vigenere published his description of a
similar but stronger auto key cipher before the court of henry
iii of france in later in the th century the invention of
bellasos cipher was misattributed to vigenere david kahn in
his book the code breakers lamented the misattribution by
saying that his tory had ignored this important contribution
and instead named a regressive and elementary cipher for him
vigenere though he had nothing to do with it the vigenere
cipher gained a reputation for being exceptionally strong
noted author and mathematician charles lutwidg edodgson lewis
carroll called the vigenere cipher unbreakable in his piece
the alphabet cipher in a childrens magazine in scientific
american described the vigenre cipher as impossible of translation
this reputation was not deserved charles babbage is known to have
broken a variant of the cipher as early as however he didnt publish
his work kasiski entirely broke the cipher and published the
technique in the th century even before this though some skilled
cryptanalysts could occasionally break the cipher in the th
century cryptographic slide rule used as a calculation aid by
the swiss army between and the vigenere cipher is simple enough
to be a field cipher if it is used in conjunction with cipher
disks the confederate states of america for example used a
brass cipher disk to implement the vigenere cipher during
the american civil war the confederacys messages were far
from secret and the union regularly cracked their messages
throughout the war the confederate leadership primarily relied
upon three keyphrases manchester bluff complete victory and
as the war came to a close come retribution gilbert vernam
tried to repair the broken cipher creating the vernam vigenere
cipher in but no matter what he did the cipher was still
vulnerable to cryptanalysis vernams work however eventually
led to the one time pad a provably unbreakable cipher

4.2 Printed table of summed std-deviations

Sum of 5 std. devs: 45.0098415026,
Sum of 6 std. devs: 89.133414186,

```
Sum of 7 std. devs: 53.93734543,
Sum of 8 std. devs: 93.867013076,
Sum of 9 std. devs: 77.5771115818,
Sum of 10 std. devs: 61.3767199633,
Sum of 11 std. devs: 68.8687432352,
Sum of 12 std. devs: 141.597799568,
Sum of 13 std. devs: 71.5147082926,
Sum of 14 std. devs: 83.5648082971,
Sum of 15 std. devs: 71.8909640457,
```

4.3 Initial keyword and alternatives

Initially we found the peak of standard deviations at key-size 12. Looking at the letter frequency counts from size-12 and assuming that the maximum counts referred to the letter 'e' we found the keyword `confidential`. We were unable to find any reasonable alternatives to this. We did however manage to sort the letters in each frequency table for keys of size 12 in descending order to find other possible words.

```
Most possible letters: (ranging from most to 3rd most)
Key index: 0    1    2    3    4    5    6    7    8    9    10   11
1st:     ['c', 'o', 'n', 'f', 'i', 'd', 'e', 'n', 't', 'i', 'a', 'l']
2nd:     ['n', 'z', 'y', 'q', 't', 'o', 'p', 'y', 'e', 't', 'l', 'w']
3rd:     ['g', 's', 'r', 'j', 'm', 'h', 'i', 'r', 'x', 'm', 'e', 'p']
4th:     ['u', 'g', 'f', 'x', 'a', 'v', 'w', 'f', 'l', 'a', 's', 'd']
5th:     ['o', 'a', 'z', 'r', 'u', 'p', 'q', 'z', 'f', 'u', 'm', 'x']
```

Looking at this list there are few actual english words that come to mind.

4.4 If you had to generalize the vigenere cipher in such a way that not only the (lowercase) letters were used in the encryption process but all printable ascii characters, what would be your most frequently occurring character in that case?

If all prinatble ascii-characters were used then `space` should be the most frequently occurring character. Not every word contains an 'e' but (almost) every word is separated by two spaces.

4.5 Code

```
import math
import numpy

#### MANAGE FILES ####

#1. Read the file and store as one string
file1 = open("testtext", 'r').read()
cipherText = open("ciphertext", 'r').read()

#1.1    Remove all '\n's from text
file1 = file1.replace('\n', '')
cipherText = cipherText.replace('\n', '')

# Global Variable
stdSumTable = []
globalvecs = []
```

```
# Calculate standard deviation(s) sum for frequency count(s)
# Takes a list of frequency count list(s), calculates the
# std-deviation of each list and then returns the sum of
# all the std-deviations
def standardDeviationSum(vectors):
    stdevSum = 0
    for i in range(len(vectors)):
        stdev = math.sqrt( sum(numpy.multiply(vectors[i], vectors[i]))/26 - \
            numpy.power(sum(vectors[i])/26, 2))
        stdevSum += stdev
    return stdevSum

# Returns a list consisting of 'size' number of
# lists with frequency counts
def makeFreqVects(size):
    # Initialize a list of 'size' lists
    vectors = [([0] * 26) for i in range(size)]
    for i in range(len(cipherText)):
        vectors[i%size][ord(cipherText[i]) % ord('a')] += 1
    globalvecs.append(vectors)
    return vectors


def main():
    ### MAIN PART ###
    #1. Loop through all possible key-lengths
    highestStDev = 0
    highestStDevIndex = 0
    for i in range(5,16): # <-- from 5 to 15
        stdSumTable.append(standardDeviationSum(makeFreqVects(i)))
        print "Sum of " + str(i) + " std. devs: " + str(stdSumTable[i-5]) + ","
        if stdSumTable[i-5] > highestStDev: #determine key length with highest StDev
            highestStDev = stdSumTable[i-5]
            highestStDevIndex = i

    keyList = []
    frequencies = []
    #for frequency vectors of the length with highest StDev
    for i in range(len(globalvecs[highestStDevIndex-5])):
        #Make a tuple with the letter index corresponding to the frequency
        frequencies.append([globalvecs[highestStDevIndex-5][i],range(0,26)])
        #Bubble sort letters, highest frequency first
        for j in range(len(globalvecs[highestStDevIndex-5][i])):
            for k in range(len(globalvecs[highestStDevIndex-5][i])-1-j):
                if frequencies[i][0][k] < frequencies[i][0][k+1]:
                    a = frequencies[i][0][k]
                    frequencies[i][0][k] = frequencies[i][0][k+1]
                    frequencies[i][0][k+1] = a
                    b = frequencies[i][1][k]
                    frequencies[i][1][k] = frequencies[i][1][k+1]
                    frequencies[i][1][k+1] = b
    for i in range(len(frequencies)):
        poss = []
        #Try for 3rd most common letters in English
        poss.append(chr(ord('a') + (frequencies[i][1][0]-(ord('e')-ord('a')))%26))
```

```
        poss.append(chr(ord('a') + (frequencies[i][1][0]-(ord('t')-ord('a')))%26))
        poss.append(chr(ord('a') + (frequencies[i][1][0]-(ord('a')-ord('a')))%26))
        poss.append(chr(ord('a') + (frequencies[i][1][0]-(ord('a')-ord('o')))%26))
        poss.append(chr(ord('a') + (frequencies[i][1][0]-(ord('a')-ord('i')))%26))
        keyList.append(poss)
    print "\n"
    print "Most possible letters: (ranging from most to 3rd most)"
    print "Key index: 0    1    2    3    4    5    6    7    8    9    10   11"
    print "1st:     " + str([i[0] for i in keyList])
    print "2nd:     " + str([i[1] for i in keyList])
    print "3rd:     " + str([i[2] for i in keyList])
    print "4th:     " + str([i[3] for i in keyList])
    print "5th:     " + str([i[4] for i in keyList])

    print "\n"
    #Decript text using key that assumes all highest
    # frequency letters were originally an 'e'
    text = ""
    for i in range(len(cipherText)):
        text += chr( ( (ord(cipherText[i]) - ord('a')) - \
            (ord(keyList[i%len(keyList)][0]) - ord('a')))%26 + ord('a'))
    print "Original text:"
    print cipherText
    print "\n"
    print "Decrypted text:"
    print text


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