EAST WEST UNIVERSITY excellence in education

East West University

Homework

🕮
COURSE TITLE

CSE487: Cyber Security, Law and Ethics
Section: 03
Summer 2022

⍰
SUBMITTED TO

Rashedul Amin Tuhin
Senior Lecturer
Department of Computer Science & Engineering
East West University

⍰
SUBMITTED BY

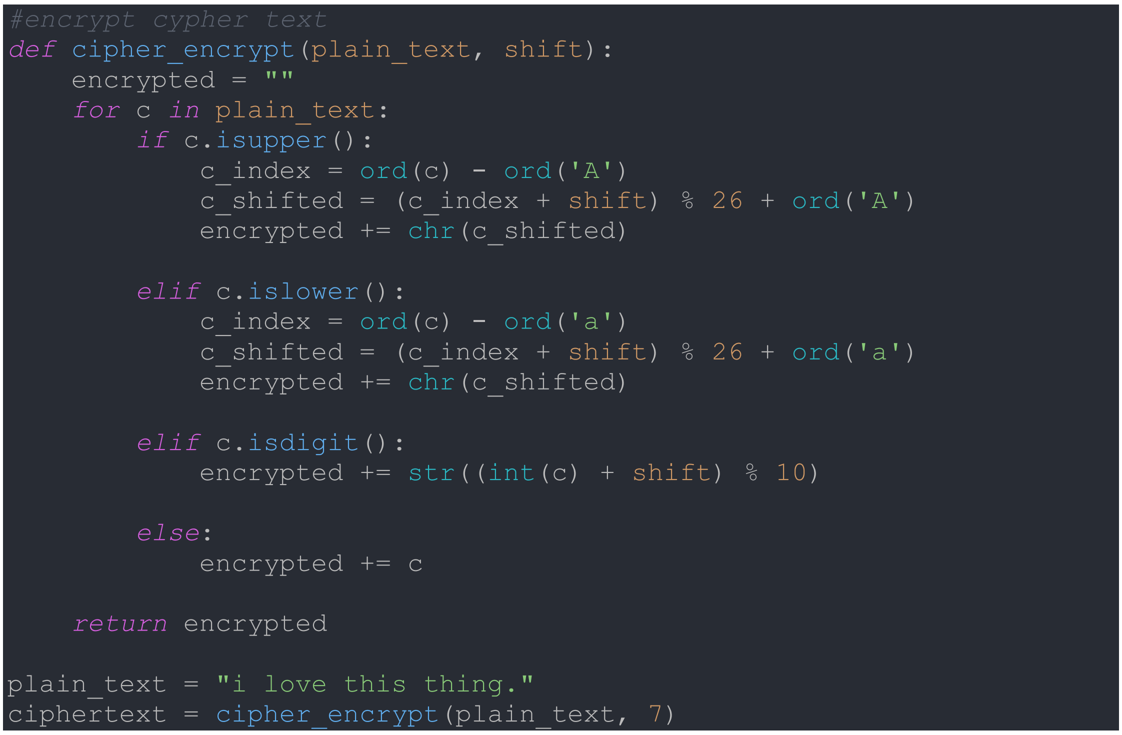
Md. Shahadat Anik Sheikh
ID NO: 2019-1-60-068
Department of Computer Science & Engineering
East West University

⍰
SUBMISSION DATE

November 07, 2021 | Sunday

Week 1 Homework 1. Caesar Cipher Implementation

1. Implement Caesar Cipher in any programming language



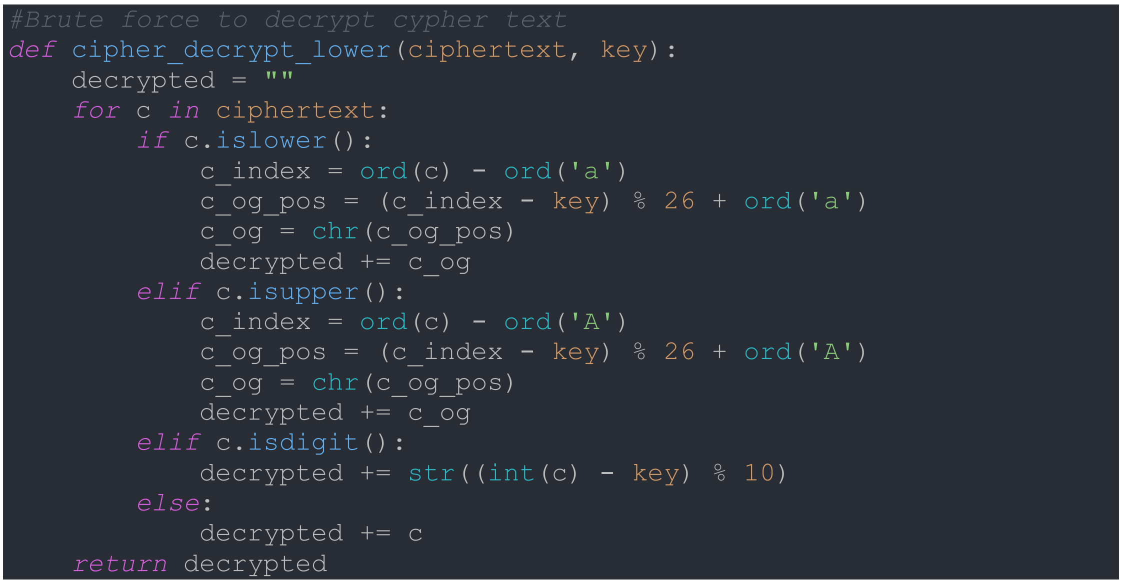
```
#encrypt cypher text
def cipher_encrypt(plain_text, shift):
    encrypted = ""
    for c in plain_text:
        if c.isupper():
            c_index = ord(c) - ord('A')
            c_shifted = (c_index + shift) % 26 + ord('A')
            encrypted += chr(c_shifted)

        elif c.islower():
            c_index = ord(c) - ord('a')
            c_shifted = (c_index + shift) % 26 + ord('a')
            encrypted += chr(c_shifted)

        elif c.isdigit():
            encrypted += str((int(c) + shift) % 10)

        else:
            encrypted += c

    return encrypted

plain_text = "i love this thing."
ciphertext = cipher_encrypt(plain_text, 7)
```

output:

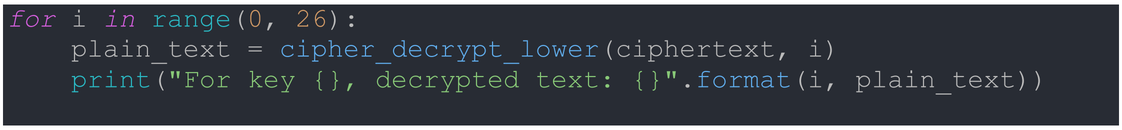
Plain text message: i love this thing.

Encrypted ciphertext: p svcl aopz aopun.

2. Break the cipher using brute force



```
#Brute force to decrypt cypher text
def cipher_decrypt_lower(ciphertext, key):
    decrypted = ""
    for c in ciphertext:
        if c.islower():
            c_index = ord(c) - ord('a')
            c_og_pos = (c_index - key) % 26 + ord('a')
            c_og = chr(c_og_pos)
            decrypted += c_og
        elif c.isupper():
            c_index = ord(c) - ord('A')
            c_og_pos = (c_index - key) % 26 + ord('A')
            c_og = chr(c_og_pos)
            decrypted += c_og
        elif c.isdigit():
            decrypted += str((int(c) - key) % 10)
        else:
            decrypted += c
    return decrypted
```



```python
for i in range(0, 26):
    plain_text = cipher_decrypt_lower(ciphertext, i)
    print("For key {}, decrypted text: {}".format(i, plain_text))
```

output:

```
For key 0, decrypted text: p svcl aopz aopun.
For key 1, decrypted text: o rubk znoy znotm.
For key 2, decrypted text: n qtaj ymnx ymnsl.
For key 3, decrypted text: m pszi xlmw xlmrk.
For key 4, decrypted text: l oryh wklv wklqj.
For key 5, decrypted text: k nqxg vjku vjkpi.
For key 6, decrypted text: j mpwf uijt uijoh.
For key 7, decrypted text: i love this thing. // This is the output
For key 8, decrypted text: h knud sghr sghmf.
For key 9, decrypted text: g jmtc rfgq rfgle.
For key 10, decrypted text: f ilsb qefp qefkd.
For key 11, decrypted text: e hkra pdeo pdejc.
For key 12, decrypted text: d gjqz ocdn ocdib.
For key 13, decrypted text: c fipy nbcm nbcha.
For key 14, decrypted text: b ehox mabl mabgz.
For key 15, decrypted text: a dgnw lzak lzafy.
For key 16, decrypted text: z cfmv kyzj kyzex.
For key 17, decrypted text: y belu jxyi jxydw.
For key 18, decrypted text: x adkt iwxh iwxcv.
For key 19, decrypted text: w zcjs hvwg hvwbu.
For key 20, decrypted text: v ybir guvf guvat.
For key 21, decrypted text: u xahq ftue ftuzs.
For key 22, decrypted text: t wzgp estd estyr.
For key 23, decrypted text: s vyfo drsc drsxq.
For key 24, decrypted text: r uxen cqrb cqrwp.
For key 25, decrypted text: q twdm bpqa bpqvo.
```

3. Attempt to break the cipher using cryptanalysis.

```
print("Cipher Text: ", ciphertext)
stored_letters = {}

for char in ciphertext:
    if char not in stored_letters:
        stored_letters[char] = 1
    else:
        stored_letters[char] += 1

print(stored_letters)
attempt = ciphertext.replace("p", "I")
attempt = attempt.replace("a", "T")
attempt = attempt.replace("o", "H")
attempt = attempt.replace("s", "L")
attempt = attempt.replace("u", "N")
attempt = attempt.replace("z", "S")
attempt = attempt.replace("n", "G")
attempt = attempt.replace("v", "O")
attempt = attempt.replace("c", "V")
attempt = attempt.replace("l", "E")

print("Plain Text: ", attempt)
```

output:

```
Cipher Text:  p svcl aopz aopun.

{'p': 3, ' ': 3, 's': 1, 'v': 1, 'c': 1, 'l': 1, 'a': 2, 'o': 2, 'z': 1, 'u': 1, 'n': 1, '.': 1}

Plain Text:  I LOVE THIS THING.
```

Week 1 Homework 2. MTU is Maximum Transmission Unit

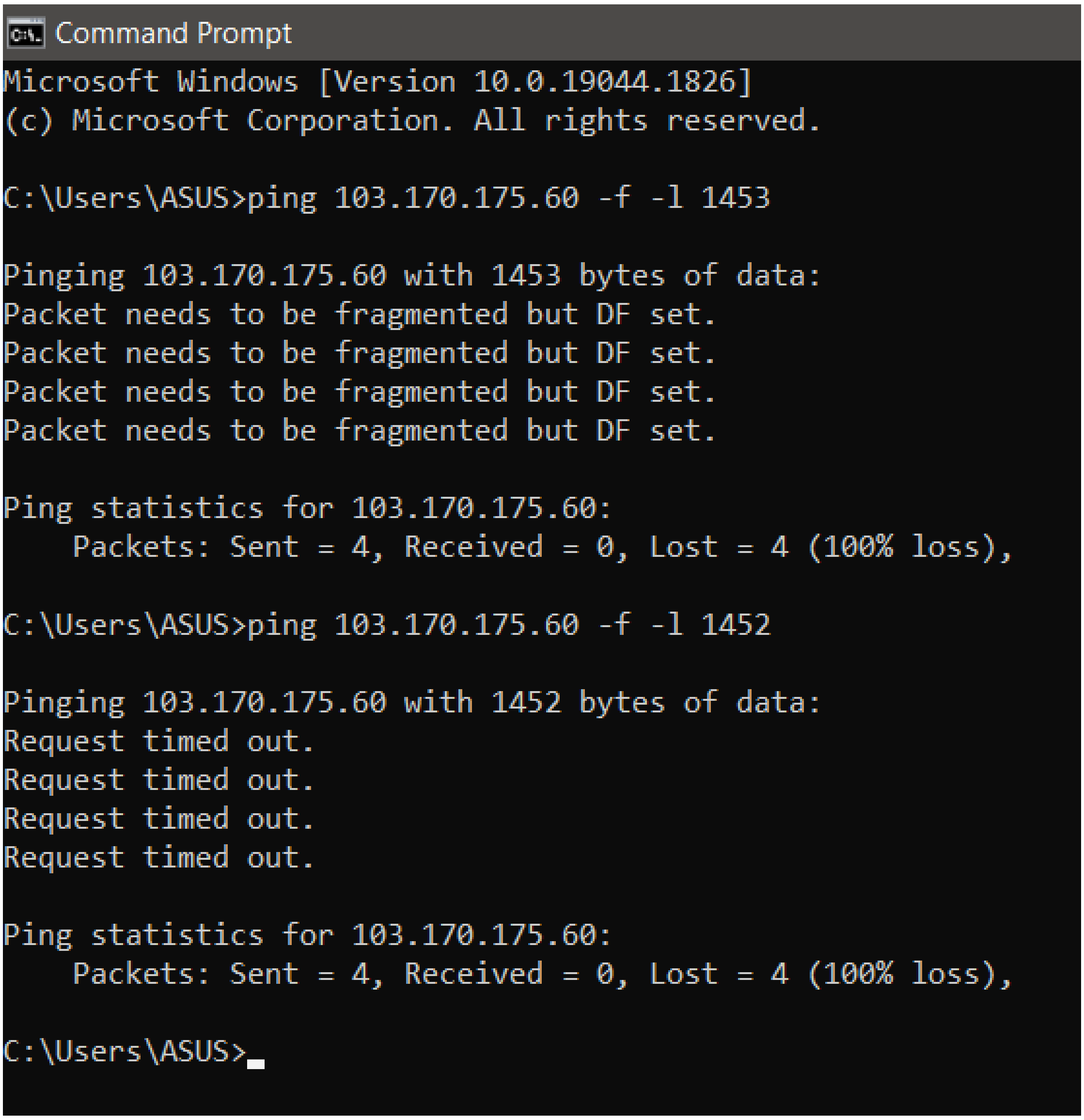
1. Write a computer program to discover the actual MTU size of your communication network.

```
import socket
class IN:
    IP_MTU = 14
    IP_MTU_DISCOVER = 10
    IP_PMTUDISC_DO = 2


s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
hostName = "103.170.175.60"
Port = 9999
s.connect((hostName, Port))
s.setsockopt(socket.IPPROTO_IP, IN.IP_MTU_DISCOVER,
IN.IP_PMTUDISC_DO)
MTU_Size = 1488
try:
    s.send(b'#' * 44 * MTU_Size)
except socket.error:
    print('The message did not make it')
    option = getattr(IN, 'IP_MTU', 14)
    print('MTU:', s.getsockopt(socket.IPPROTO_IP, option))
```

```
else:
    print('My network supports', MTU_Size, 'big packets!')
```

output: My network supports 1488 big packets!

Usuing CMD:



```
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS>ping 103.170.175.60 -f -l 1453

Pinging 103.170.175.60 with 1453 bytes of data:
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.

Ping statistics for 103.170.175.60:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\Users\ASUS>ping 103.170.175.60 -f -l 1452

Pinging 103.170.175.60 with 1452 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 103.170.175.60:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\Users\ASUS>_
```

We have to take into account the size of the TCP/IP header, which can range between 20-60 bytes. The header size varies according to the transmission media.

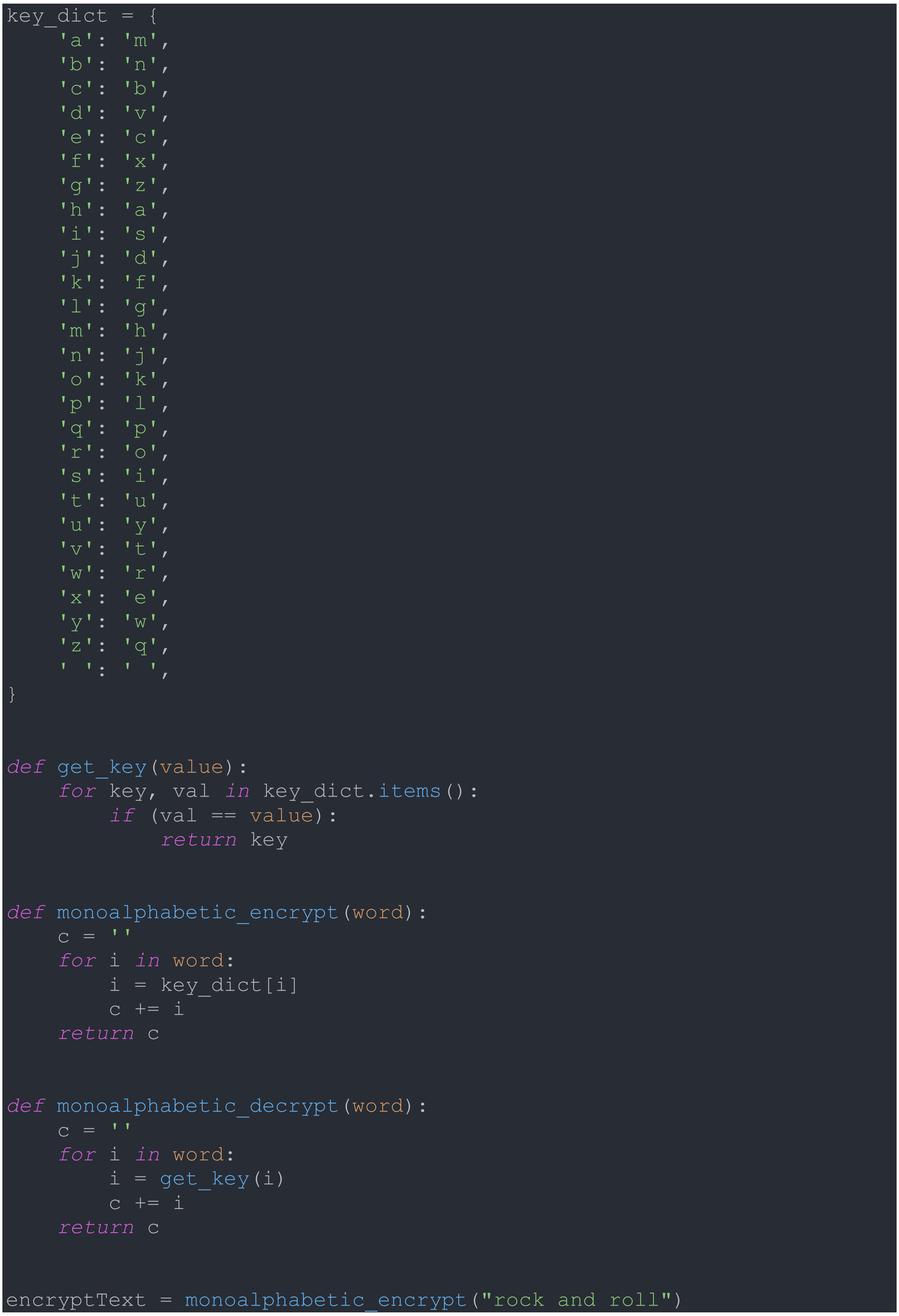
My Point-to-Point Protocol over Ethernet (PPPoE) header takes up 8 bytes in size, while the IPv6 header accounts for 40 bytes in size. So, the size of my TCP/IP header is 48 bytes (40+8).

Let’s take the packet size that gave us the ping reply (1452 bytes) and add it to the header size (48 bytes).

That leads us to the real MTU size, which is 1500 bytes, the common Ethernet MTU.

Week 2 Homework 3. Implement at least five cipher algorithms of different types.

1. Monoalphabetic Cipher



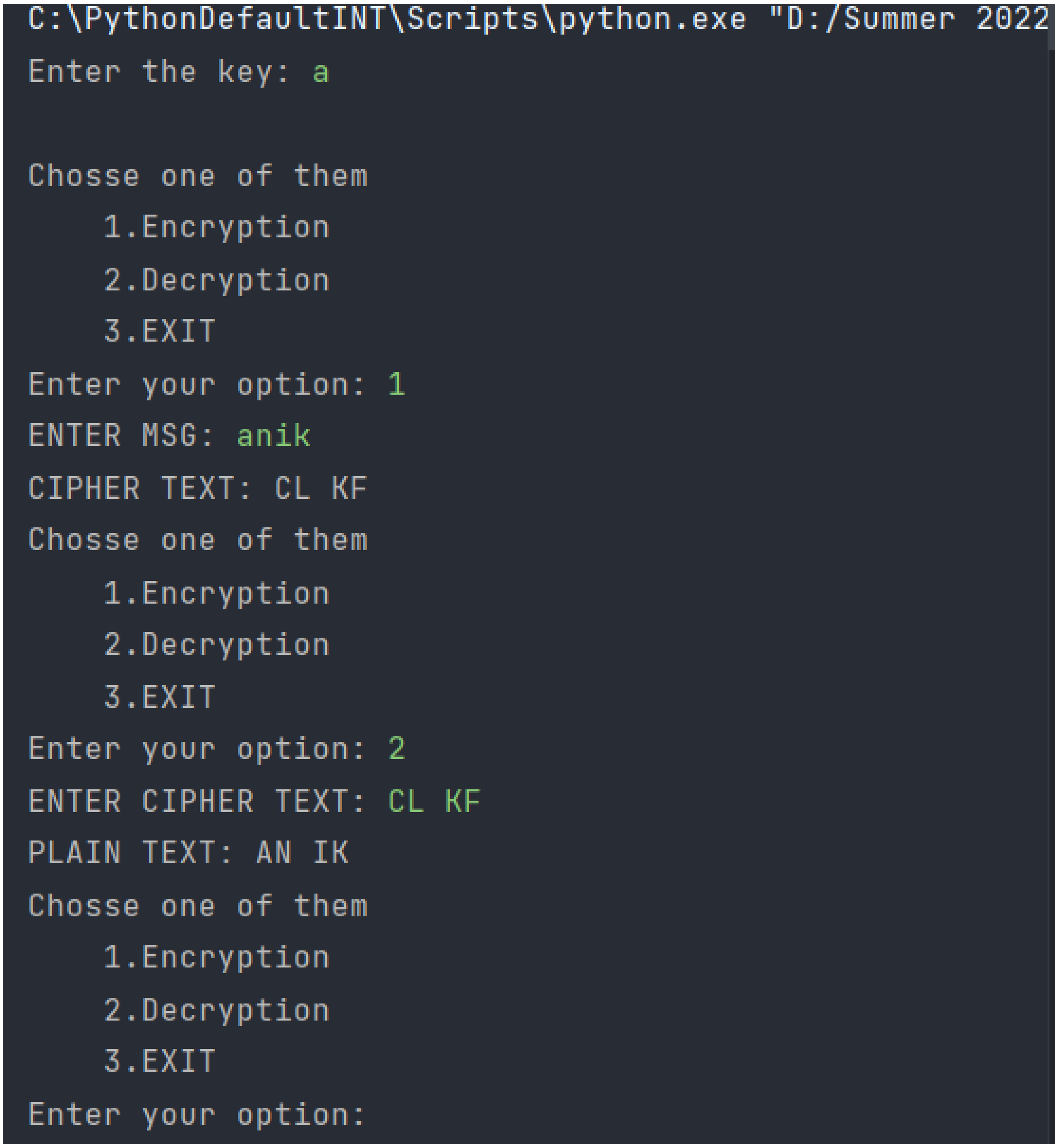
```
key_dict = {
    'a': 'm',
    'b': 'n',
    'c': 'b',
    'd': 'v',
    'e': 'c',
    'f': 'x',
    'g': 'z',
    'h': 'a',
    'i': 's',
    'j': 'd',
    'k': 'f',
    'l': 'g',
    'm': 'h',
    'n': 'j',
    'o': 'k',
    'p': 'l',
    'q': 'p',
    'r': 'o',
    's': 'i',
    't': 'u',
    'u': 'y',
    'v': 't',
    'w': 'r',
    'x': 'e',
    'y': 'w',
    'z': 'q',
    ' ': ' ',
}


def get_key(value):
    for key, val in key_dict.items():
        if (val == value):
            return key


def monoalphabetic_encrypt(word):
    c = ''
    for i in word:
        i = key_dict[i]
        c += i
    return c


def monoalphabetic_decrypt(word):
    c = ''
    for i in word:
        i = get_key(i)
        c += i
    return c


encryptText = monoalphabetic_encrypt("rock and roll")
```

```
print(encryptText)
print(monoalphabetic_decrypt(encryptText))
```

2. Playfair Cipher

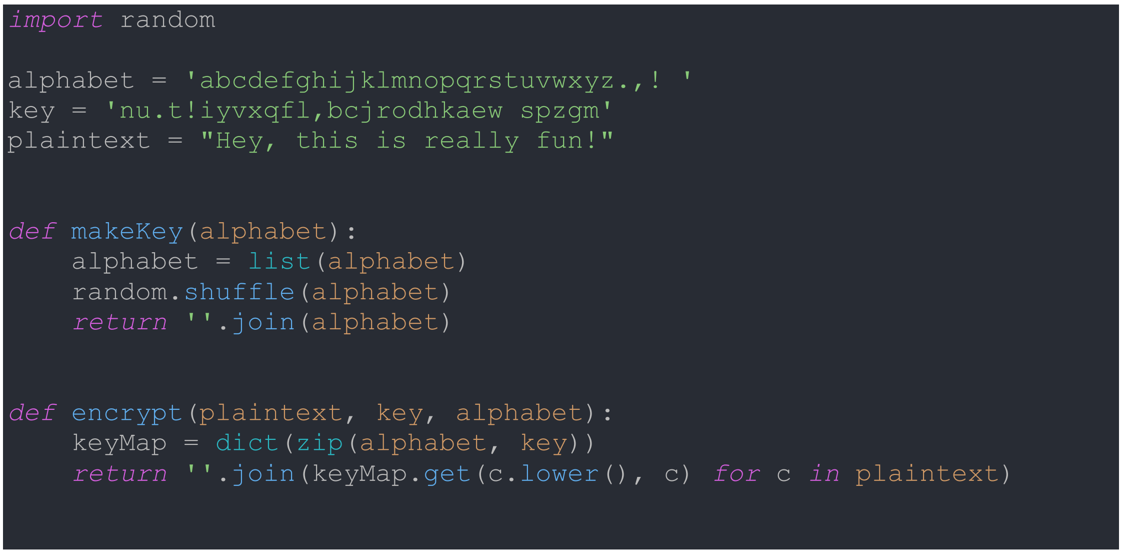
```
key = input("Enter the key: ")
key = key.replace(" ", "")
key = key.upper()


def matrix(x, y, initial):
    return [[initial for i in range(x)] for j in range(y)]


result = list()
for c in key:  # storing key
    if c not in result:
        if c == 'J':
            result.append('I')
        else:
            result.append(c)
flag = 0
for i in range(65, 91):  # storing other character
    if chr(i) not in result:
        if i == 73 and chr(74) not in result:
            result.append("I")
            flag = 1
        elif flag == 0 and i == 73 or i == 74:
            pass
        else:
            result.append(chr(i))
k = 0
my_matrix = matrix(5, 5, 0)  # initialize matrix
for i in range(0, 5):  # making matrix
    for j in range(0, 5):
        my_matrix[i][j] = result[k]
        k += 1


def locindex(c):  # get location of each character
    loc = list()
    if c == 'J':
        c = 'I'
    for i, j in enumerate(my_matrix):
        for k, l in enumerate(j):
            if c == l:
                loc.append(i)
                loc.append(k)
                return loc


def encrypt():  # Encryption
    msg = str(input("ENTER MSG:"))
    msg = msg.upper()
    msg = msg.replace(" ", "")
    i = 0
    for s in range(0, len(msg) + 1, 2):
        if s < len(msg) - 1:
            if msg[s] == msg[s + 1]:
                msg = msg[:s + 1] + 'X' + msg[s + 1:]
    if len(msg) % 2 != 0:
        msg = msg[:] + 'X'
    print("CIPHER TEXT:", end=' ')
    while i < len(msg):
        loc = list()
        loc = locindex(msg[i])
        loc1 = list()
        loc1 = locindex(msg[i + 1])
        if loc[1] == loc1[1]:
            print("{}{}".format(my_matrix[(loc[0] + 1) % 5][loc[1]],
my_matrix[(loc1[0] + 1) % 5][loc1[1]]), end=' ')
        elif loc[0] == loc1[0]:
            print("{}{}".format(my_matrix[loc[0]][(loc[1] + 1) % 5],
my_matrix[loc1[0]][(loc1[1] + 1) % 5]), end=' ')
        else:
            print("{}{}".format(my_matrix[loc[0]][loc1[1]],
my_matrix[loc1[0]][loc[1]]), end=' ')
        i = i + 2


def decrypt():  # decryption
    msg = str(input("ENTER CIPHER TEXT:"))
    msg = msg.upper()
    msg = msg.replace(" ", "")
    print("PLAIN TEXT:", end=' ')
    i = 0
    while i < len(msg):
        loc = list()
        loc = locindex(msg[i])
        loc1 = list()
        loc1 = locindex(msg[i + 1])
        if loc[1] == loc1[1]:
            print("{}{}".format(my_matrix[(loc[0] - 1) % 5][loc[1]],
my_matrix[(loc1[0] - 1) % 5][loc1[1]]), end=' ')
        elif loc[0] == loc1[0]:
            print("{}{}".format(my_matrix[loc[0]][(loc[1] - 1) % 5],
my_matrix[loc1[0]][(loc1[1] - 1) % 5]), end=' ')
        else:
            print("{}{}".format(my_matrix[loc[0]][loc1[1]],
my_matrix[loc1[0]][loc[1]]), end=' ')
        i = i + 2


while (1):
    choice = int(input("\nChosse one of them\n \t1.Encryption \n
\t2.Decryption \n \t3.EXIT\nEnter your option: "))
    if choice == 1:
        encrypt()
    elif choice == 2:
        decrypt()
    elif choice == 3:
        exit()
    else:
        print("Choose correct choice")
```

output:



```
C:\PythonDefaultINT\Scripts\python.exe "D:/Summer 2022
Enter the key: a

Chosse one of them
     1.Encryption
     2.Decryption
     3.EXIT
Enter your option: 1
ENTER MSG: anik
CIPHER TEXT: CL KF
Chosse one of them
     1.Encryption
     2.Decryption
     3.EXIT
Enter your option: 2
ENTER CIPHER TEXT: CL KF
PLAIN TEXT: AN IK
Chosse one of them
     1.Encryption
     2.Decryption
     3.EXIT
Enter your option:
```

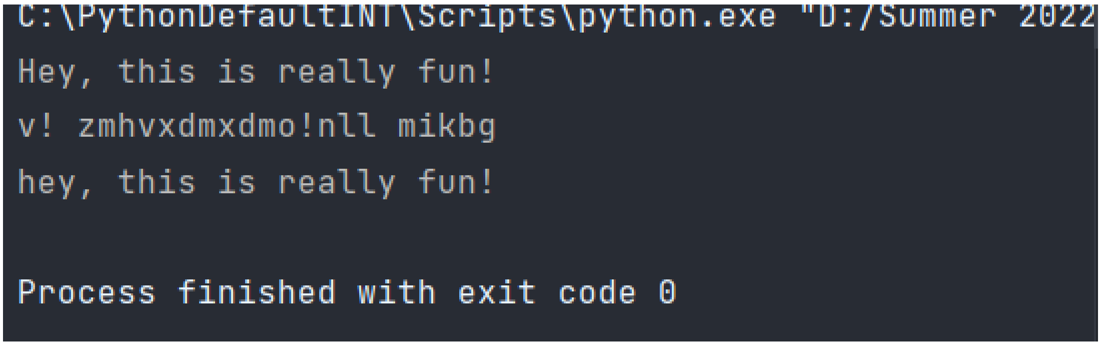
3. Substitution Cipher



```
import random

alphabet = 'abcdefghijklmnopqrstuvwxyz.,! '
key = 'nu.t!iyvxqfl,bcjrodhkaew spzgm'
plaintext = "Hey, this is really fun!"


def makeKey(alphabet):
    alphabet = list(alphabet)
    random.shuffle(alphabet)
    return ''.join(alphabet)


def encrypt(plaintext, key, alphabet):
    keyMap = dict(zip(alphabet, key))
    return ''.join(keyMap.get(c.lower(), c) for c in plaintext)
```

```
def decrypt(cipher, key, alphabet):
    keyMap = dict(zip(key, alphabet))
    return ''.join(keyMap.get(c.lower(), c) for c in cipher)


cipher = encrypt(plaintext, key, alphabet)

print(plaintext)
print(cipher)
print(decrypt(cipher, key, alphabet))
```

output:



```
C:\PythonDefaultINT\Scripts\python.exe "D:/Summer 2022
Hey, this is really fun!
v! zmhvxdmxdmo!nll mikbg
hey, this is really fun!

Process finished with exit code 0
```

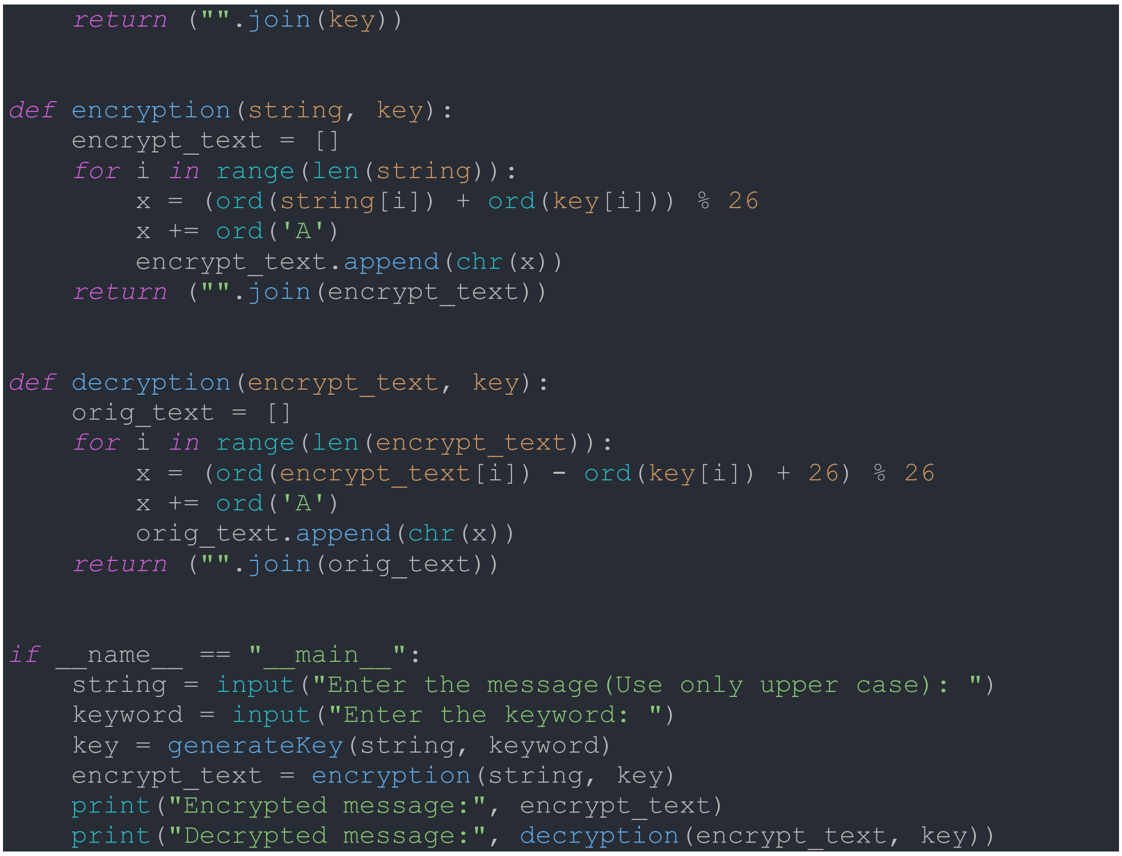
4. Transposition Cipher

```
import pyperclip

def main():
    msg = 'Transposition Cipher'
    key = 10
    ciphertext = encryptMessage(key, msg)

    print("Cipher Text is: ", ciphertext, '|',
pyperclip.copy(ciphertext))

def encryptMessage(key, message):
    ciphertext = [''] * key

    for col in range(key):
        position = col
        while position < len(message):
            ciphertext[col] += message[position]
            position += key
    return ''.join(ciphertext)

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

output: Cipher Text is: Tiroann sCpiopshietr

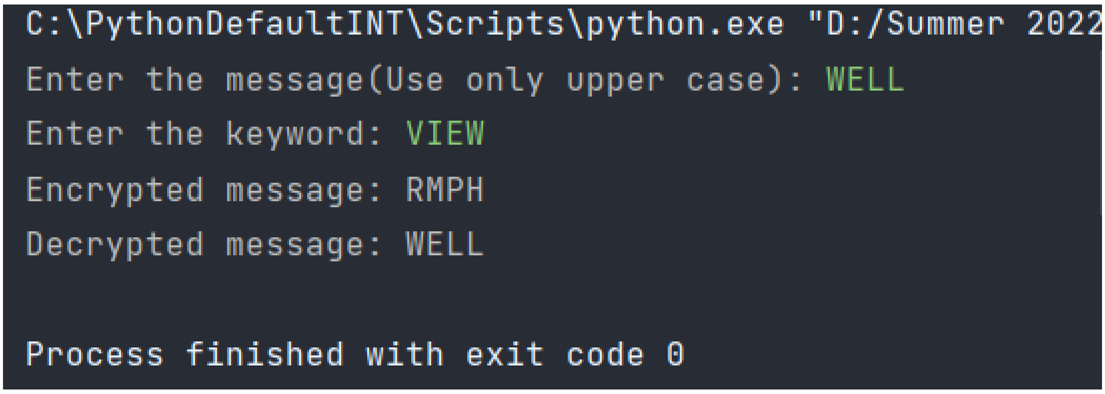
5. Vigenère cipher

```
def generateKey(string, key):
    key = list(key)
    if len(string) == len(key):
        return (key)
    else:
        for i in range(len(string) - len(key)):
            key.append(key[i % len(key)])
```



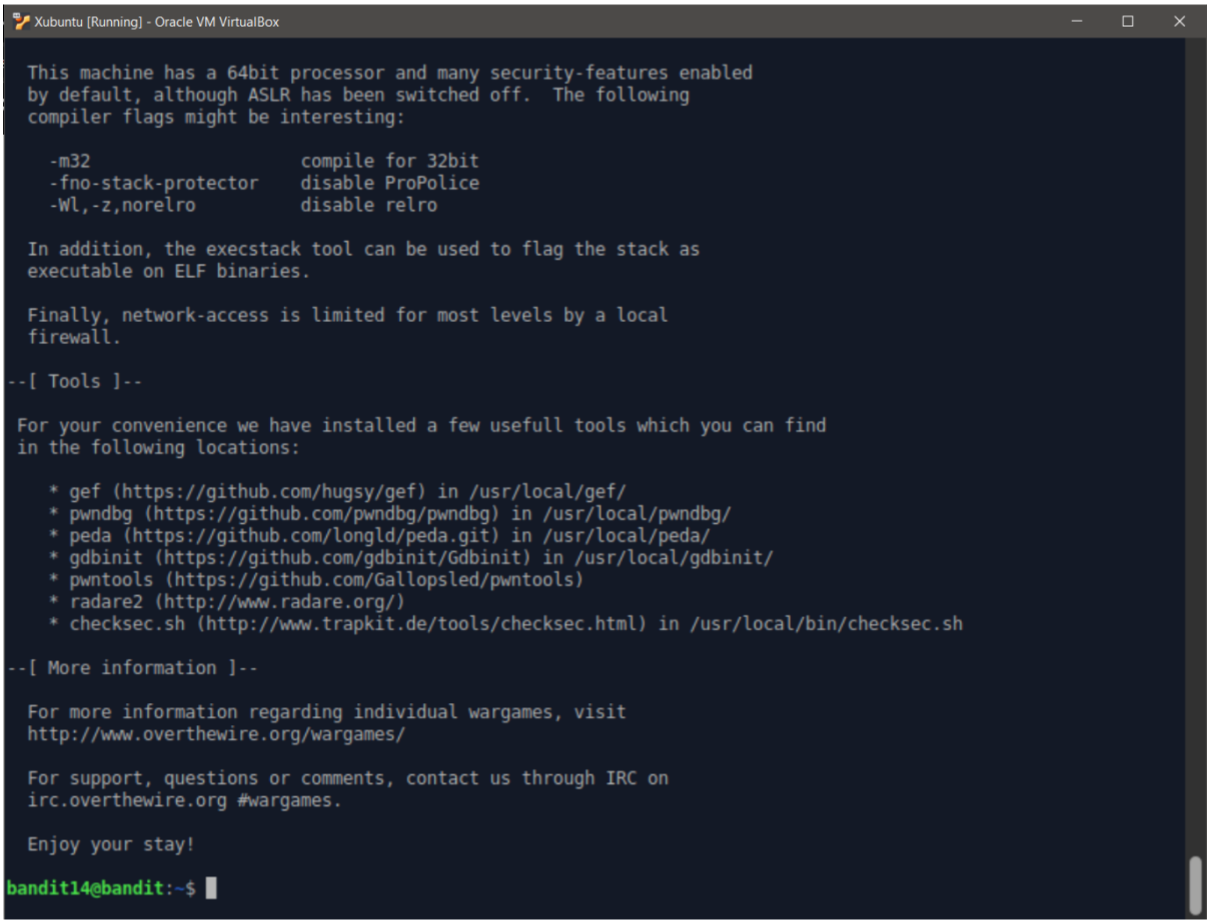
```
    return ("".join(key))


def encryption(string, key):
    encrypt_text = []
    for i in range(len(string)):
        x = (ord(string[i]) + ord(key[i])) % 26
        x += ord('A')
        encrypt_text.append(chr(x))
    return ("".join(encrypt_text))


def decryption(encrypt_text, key):
    orig_text = []
    for i in range(len(encrypt_text)):
        x = (ord(encrypt_text[i]) - ord(key[i]) + 26) % 26
        x += ord('A')
        orig_text.append(chr(x))
    return ("".join(orig_text))


if __name__ == "__main__":
    string = input("Enter the message(Use only upper case): ")
    keyword = input("Enter the keyword: ")
    key = generateKey(string, keyword)
    encrypt_text = encryption(string, key)
    print("Encrypted message:", encrypt_text)
    print("Decrypted message:", decryption(encrypt_text, key))
```

output:



```
C:\PythonDefaultINT\Scripts\python.exe "D:/Summer 2022
Enter the message(Use only upper case): WELL
Enter the keyword: VIEW
Encrypted message: RMPH
Decrypted message: WELL

Process finished with exit code 0
```

Week 2 Homework 4. Pass up to level 14 in Bandit Wargame



```
  This machine has a 64bit processor and many security-features enabled
  by default, although ASLR has been switched off.  The following
  compiler flags might be interesting:

    -m32                    compile for 32bit
    -fno-stack-protector    disable ProPolice
    -Wl,-z,norelro          disable relro

  In addition, the execstack tool can be used to flag the stack as
  executable on ELF binaries.

  Finally, network-access is limited for most levels by a local
  firewall.

--[ Tools ]--

 For your convenience we have installed a few usefull tools which you can find
 in the following locations:

    * gef (https://github.com/hugsy/gef) in /usr/local/gef/
    * pwndbg (https://github.com/pwndbg/pwndbg) in /usr/local/pwndbg/
    * peda (https://github.com/longld/peda.git) in /usr/local/peda/
    * gdbinit (https://github.com/gdbinit/Gdbinit) in /usr/local/gdbinit/
    * pwntools (https://github.com/Gallopsled/pwntools)
    * radare2 (http://www.radare.org/)
    * checksec.sh (http://www.trapkit.de/tools/checksec.html) in /usr/local/bin/checksec.sh

--[ More information ]--

  For more information regarding individual wargames, visit
  http://www.overthewire.org/wargames/

  For support, questions or comments, contact us through IRC on
  irc.overthewire.org #wargames.

  Enjoy your stay!

bandit14@bandit:~$
```