CSC 479/579 Computers and Networks Security

Arelly Ragland and Brittany Molenda

06/06/2017

Peer to Peer Remote File Storage

A Conceptual Design Implementation Code

```
import argparse
import hashlib
import getpass
import os, sys
from Crypto import Random
from Crypto.Util import Counter
from Crypto.Cipher import AES
import bluetooth
import time
import os
import ast
import pickle
import os.path


key=b'Sixteen byte key'
server_sock=bluetooth.BluetoothSocket(bluetooth.L2CAP)
client_sock=bluetooth.BluetoothSocket(bluetooth.L2CAP)
bd_addr="B8:27:EB:17:13:B5"
peer_bd_addr="B8:27:EB:29:FA:9B"
sport=0x1001
cport=0x1003
raddr=""
laddr=""

cipher = AES.new(key)
filehash ={}
global toSend
created="False"
sock=bluetooth.BluetoothSocket(bluetooth.L2CAP)

def login():
    status = ""
    users = {}
    with open('Users.txt', 'r') as f:
        for line in f:
            name, pwd = line.strip().split(':')
            users[name] = pwd
    while status != "q":
        status = raw_input("Please press y to login or q to quit: ")

        if status == "y": #login user
            login = raw_input("Enter login name: ")

            if login in users:
                passw = getpass.getpass(prompt="Enter password: ")
                hashpass = hashlib.sha256(passw).hexdigest()
                print

                if login in users and hashpass == users[login]: # login matches password
                    print ("Login successful!\n")
                    return

                else:
                    print
                    print("Incorrect Username/Password or User doesn't exist!\n")

        else:
            print("Not a valid option")

    sys.exit("Quit")


def getTime():
    ticks = time.time()
    print ("TimeStamp:", ticks)
    return ticks

def pad(s):
    return s + ((16-len(s)%16)*'{')

def encrypt(plaintext):
    global cipher
    return cipher.encrypt(pad(plaintext))

def decrypt(ciphertext):
    global cipher
    dec = cipher.decrypt(ciphertext).decode('utf-8')
    l = dec.count('{')
    return dec[:len(dec)-l]

def senddata(text):
    print ("Sending data")
    client_sock.send (text)
    #client_sock.close()
```

```
def calculateHash(text):
    h = hashlib.md5()
    h.update(text)
    return h.hexdigest()

def getMac():
    str = open('/sys/class/bluetooth/hci0/address').read()

    return str[0:17]

def client():
    print raddr
    client_sock.connect((raddr, sport))
    server_sock.bind ( ("",cport))
    server_sock.listen(1)
    sock, address = server_sock.accept()
    while 1:
        print("")
        print("1. Save the file in the Remote Rasperry Pi")
        print("2. List all the available timestamp from the Remote Rasperry Pi")
        print("3. Retrive a particular file from the Remote Rasperry Pi")
        print ("")
        choice = input ("Please enter your choice")
        print ("Choice entered is:  ", choice)
        os.system('clear')
        if (choice == 1):
            print ("We are going to save the file in remote Pi")
            print ""
            file = raw_input ("Please enter your file name   ")
            print "Choice entered is:  " + file
            getTime()
            with open (file,'r') as f:
                plaintext=f.read()
            encrypted = encrypt(plaintext)
            encryptfilename = encrypt (file)
            hashout = calculateHash(plaintext)
            print ("Passing the encrypted file to the server")
            sendText = "S:"+encryptfilename+":"+encrypted+":"+hashout
            #print ("The text to send is " + sendText)
            client_sock.send(sendText)
            # Waiting for acknowledment from server
            hashin=sock.recv(1024)
            print ("Received confirmation from server with hash")
            #print ("data content is " + hashin)
            if (hashin == hashout):
```

```
        # Deleting the Original File
        os.remove(file)
        print ("HASH Matched")
        print ("Original File Removed.")
    else:
        print ("HAST not Matched")
    print ""
    print "Sucessfull!!!"
    getTime()

elif (choice == 2):
    print ("We are going to retrive the list of timestamp")
    print ""
    client_sock.send("L:A:A:A")
    data = sock.recv(1024)
    print ("")
    print ("Below are the timestamps: ")
    timestamps = data.split(":")
    for times in timestamps:
        if (times !=""):
            dt=ast.literal_eval(times)
            print (times+"/"+time.strftime('%Y-%m-%d %H:%M:%s',time.gmtime(dt)))
    print ""
    print "Sucessfull!!!"

elif (choice == 3):
    print ("We are going to retrive a particular file")
    print ""
    timestamp = raw_input("Enter a particular time stamp    ")
    print ("The timestamp entered is " + timestamp)
    toSend=""
    toSend="R:"+timestamp+":A:A"
    login()
    client_sock.send(toSend)
    print ("Retriving the file..")
    rdata = sock.recv(1024)
    typee,filename,encrypted,hashed = rdata.split(":")
    decrypted = decrypt(encrypted)
    hashin = calculateHash(decrypted)
    orgfile = decrypt(filename)

    #Comparing the HASH
    if (hashin == hashed):
        print (" Original Hash matches with Hash from Server")
        # Write to the file
        fullfile = os.path.join (os.environ['HOME'], orgfile)
```

```
                print ("Writing the original file")
                with open (fullfile, 'w') as f:
                    f.write(decrypted)
                    f.flush()
                    f.close()
                # Send a confirmation
                print ("Sending confirmation to server")
                toSend="R:C:A:A"
                client_sock.send(toSend)
                print ""
                print "Sucessfull!!!"
            else:
                print ("Hash failed.")
        else:
            print ("Wrong choice provided")
            exit()
def server():
    server_sock.bind(("",sport))
    server_sock.listen(1)
    sock, address = server_sock.accept()
    print "Accepted connection from", address
    client_sock.connect((raddr, cport))
    filehashfile="filehash.p"
    global filehash
    #myFile = os.path.join (os.environ['HOME'], filehashfile)
    if os.path.isfile('/home/pi/filehash.p'):
        filehash = pickle.load(open(filehashfile, "rb"))
    while 1:
        data = sock.recv(1024)
        #print "received [%s]" % data
        if data:
            type,filename,encrypt,hashed = data.split(":")
            if (type == "L"):
                print ""
                print ("Listing mode")
                toSend=""
                print filehash.keys()
                for k in filehash.keys():
                    toSend = toSend + k
                    toSend = toSend + ":"
                #print ("Staring to send is "+ toSend)
                client_sock.send(toSend)
                print ("Done with Listing")
                print ("Sucessfull!!!")
                print ""
            elif type == "R":
                if (filename =="C"):
                    print ("Confirmation received from client")
                    print ("Removing the encrypted file")
                    os.remove(file)
                else:
                    print ("Fetching mode")
                    file = filehash.get (filename)
                    with open(file,'r') as f1:
                        fdata = f1.read()
                    client_sock.send(fdata)
                    f1.close()
                    del filehash[filename]
                    print ("Sucessfull!!!")
                    print ""
            elif type == "S":
                print ("Saving mode")
                time = getTime()
                filehash[str(time)]=filename
                if data:
                    print ("Sending ACK to client")
                    #print ("Staring to send is "+ hashed)
                    client_sock.send(hashed)
                    #print filehash[time]
                    fullfile = os.path.join (os.environ['HOME'], filename)
                with open (fullfile, 'w') as f:
                    f.write(data)
                    f.flush()
                    f.close()
                    print ("Sucessfull!!!")
                    print ""
            else:
                print ("Wrong protocol message")
        else:
            print ("Data is not received.")
            pickle.dump(filehash, open("filehash.p", "wb"))
            exit()

argparser = argparse.ArgumentParser()
argparser.add_argument ("--server", "-s" ,required=False,action='store_true')
argparser.add_argument ("--client", "-c" ,required=False,action='store_true')
args=argparser.parse_args()

mac = getMac()

if (mac.upper()==bd_addr.upper()):
    raddr=peer_bd_addr
```

```
else:
    raddr=bd_addr


if (args.server):
    login()
    print ("Running as a server")
    server()
elif (args.client):
    login()
    print ("Running as a client")
    client()
else:
    argparser.print_help()
    exit()
```