McKinley & Rice

Develop web application which will help the user to authenticate application and encrypted messaging

|  |
| --- |
| import os |
|  | try: |
|  | import pyaes |
|  | except ImportError: |
|  | print("Install pyaes library!") |
|  | print("windows : python -m pip insatll pyaes") |
|  | print("linux : pip install pyaes ") |
|  | exit() |
|  | import sys |
|  | import socket |
|  | import threading |
|  | import hashlib |
|  | import json |
|  | from datetime import datetime |
|  |  |
|  | HOST = '0.0.0.0' |
|  | if(len(sys.argv)==1): |
|  | PORT = 5555 |
|  | elif(len(sys.argv)==2): |
|  | PORT=int(sys.argv[1]) |
|  |  |
|  | print("[+] Server Running ") |
|  | print("[+] Allowing All Incoming Connections ") |
|  | print("[+] PORT "+str(PORT)) |
|  | print("[+] Waiting For Connection...") |
|  |  |
|  | s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) |
|  | s.bind((HOST, PORT)) |
|  | s.listen(1) |
|  | conn, addr = s.accept() |
|  | print('[+] Connected by ', addr) |
|  |  |
|  | key = str(input('[+] AES Pre-Shared-Key for the Connection : ')) |
|  | hashed = hashlib.sha256(key.encode()).digest() |
|  | aes = pyaes.AES(hashed) |
|  |  |
|  | def verify\_and\_display(recv\_dict): |
|  | timestamp = recv\_dict['timestamp'] |
|  | recv\_hash = recv\_dict['hash'] |
|  | message = recv\_dict['message'] |
|  | mess\_hash = hashlib.sha256(str(message).encode('utf-8')).hexdigest() |
|  | SET\_LEN = 80 |
|  | if (mess\_hash == recv\_hash): |
|  | tag = str('☑') |
|  | else: |
|  | tag = str('☒') |
|  | spaces = SET\_LEN - len(str(message)) - len('Received : ') - 1 |
|  | if spaces > 0 : |
|  | space = ' '\*spaces |
|  | sentence = 'Received : ' + str(message) + space + tag + ' ' + timestamp |
|  | print(sentence) |
|  |  |
|  | def process\_bytes(bytess): |
|  | ret = [] |
|  | while(len(bytess)>=16): |
|  | if(len(bytess)>=16): |
|  | byts = bytess[:16] |
|  | ret.append(byts) |
|  | bytess = bytess[16:] |
|  | else: |
|  | print("Block Size Mismatch ") |
|  | return ret |
|  |  |
|  | def process\_text(data): |
|  | streams = [] |
|  | while (len(data)>0): |
|  | if(len(data)>=16): |
|  | stream = data[:16] |
|  | data = data[16:] |
|  | else: |
|  | stream = data + ("~"\*(16-len(data))) |
|  | data = '' |
|  | stream\_bytes = [ ord(c) for c in stream] |
|  | streams.append(stream\_bytes) |
|  | return streams |
|  |  |
|  | class myThread(threading.Thread): |
|  | def \_\_init\_\_(self,id): |
|  | threading.Thread.\_\_init\_\_(self) |
|  | self.threadID = id |
|  |  |
|  | def stop(self): |
|  | self.is\_alive = False |
|  |  |
|  | def run(self): |
|  | print("[+] Listening On Thread "+str(self.threadID)) |
|  | while 1: |
|  | try: |
|  | data = conn.recv(1024) |
|  | if(data!=""): |
|  | mess = '' |
|  | processed\_data = process\_bytes(data) |
|  | for dat in processed\_data: |
|  | decrypted = aes.decrypt(dat) |
|  | for ch in decrypted: |
|  | if(chr(ch)!='~'): |
|  | mess+=str(chr(ch)) |
|  | try: |
|  | data\_recv = json.loads(mess) |
|  | #message = str(data\_recv['message']) |
|  | verify\_and\_display(data\_recv) |
|  | except: |
|  | print('Unrecognised Data or Broken PIPE ') |
|  | except ConnectionResetError: |
|  | print('Broken PIPE !') |
|  | exit(0) |
|  | self.stop() |
|  |  |
|  |  |
|  | Listening\_Thread = myThread(1) |
|  | Listening\_Thread.daemon = True |
|  | Listening\_Thread.start() |
|  |  |
|  | while 1: |
|  | try: |
|  | sending\_data = str(input("")) |
|  | except KeyboardInterrupt: |
|  | conn.close() |
|  | exit(-1) |
|  | if(sending\_data=="quit()"): |
|  | Listening\_Thread.stop() |
|  | conn.close() |
|  | exit() |
|  | timestamp = str(datetime.now())[11:19] |
|  | mess\_hash = hashlib.sha256(str(sending\_data).encode('utf-8')).hexdigest() |
|  | send\_data = { |
|  | "timestamp" : timestamp, |
|  | "message" : sending\_data, |
|  | "hash" : mess\_hash |
|  | } |
|  | send\_json\_string = json.dumps(send\_data) |
|  | sending\_bytes = process\_text(send\_json\_string) |
|  | enc\_bytes = [] |
|  | for byte in sending\_bytes: |
|  | ciphertext = aes.encrypt(byte) |
|  | enc\_bytes += bytes(ciphertext) |
|  |  |
|  | conn.send(bytes(enc\_bytes)) |
|  | conn.close() |