МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«**Вятский государственный университет**»

**(«ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Отчет по лабораторной работе №7-8

по дисциплине «Разработка программных систем»

Выполнил студент группы ИВТ-32 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Щесняк Д. С./

Проверил доцент кафедры ЭВМ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Чистяков Г. А./

Киров 2016

1. Задание на лабораторную работу

Необходимо разработать клиент-серверное приложение на языке программирования Python, которое реализует сеть «Tor» состоящую из 2-х узлов используя xml-rpc, реализовать конфигуратор в виде графического интерфейса с использованием библиотеки python Tkinter.

1. Разработка конфигуратора

Для более удобной настройки как клиента, так и сервера узла был разработано графическое приложение, которое позволяет создавать файлы настройки как для клиента, так и для сервера. Исходный код данной программы представлен на рисунке 1.

|  |
| --- |
| #!/usr/bin/python3  import tkinter as tk  from tkinter import filedialog  from tkinter import messagebox  import json  def getClientConfig(root):  key = b'\xf4r:\xe3\xaf\xca\x1d\xcc%\xe1\xe4).#=\xe1\x1eF\x9d\x94\x83\xd24\xa7\xa0\xc6\x15\xfb\xc0$\xe6H'  iv = b'u\xba\xea\x8d\x969x5;\x0e\x17be\xaf\x9c\x16'  return {  "clientIP": tk.StringVar(root, "127.0.0.1"),  "clientPort": tk.StringVar(root, "8080"),  "serverIP": tk.StringVar(root, "127.0.0.1"),  "serverPort": tk.StringVar(root, "8081"),  "key": tk.StringVar(root, "".join(map(lambda x: str(hex(x))[2:].zfill(3), bytearray(key)))),  "iv": tk.StringVar(root, "".join(map(lambda x: str(hex(x))[2:].zfill(3), bytearray(iv))))  }  return clientConf  class ClientConfigurator(tk.Tk):  """docstring for ClientConfigurator."""  def \_\_init\_\_(self):  super(ClientConfigurator, self).\_\_init\_\_()  self.title('Конфигуратор')  self.geometry("350x475")  menu = tk.Menu(self)  menu.add\_command(label = "Открыть", command = self.onOpenMenu)  menu.add\_command(label = "Сохранить", command = lambda: self.onSaveButClick(None))  self.config(menu = menu)  self.clientConf = getClientConfig(self)  clientAddressFrame = tk.Frame(self)  clientAddressFrame.pack(side = "top")  tk.Label(clientAddressFrame, text = "Адрес клиента").pack(pady = 10, fill = "both")  tk.Label(clientAddressFrame, text = "IP:").pack(side = "left", padx = 10)  self.clientIPEntry = tk.Entry(clientAddressFrame, width = 15, textvariable = self.clientConf["clientIP"])  self.clientIPEntry.pack(side = "left")  tk.Label(clientAddressFrame, text = "Port:").pack(side = "left", padx = 10)  self.clientPortEntry = tk.Entry(clientAddressFrame, width = 5, textvariable = self.clientConf["clientPort"])  self.clientPortEntry.pack(side = "left")  serverAddressFrame = tk.Frame(self)  serverAddressFrame.pack(side = "top")  tk.Label(serverAddressFrame, text = "Адрес сервера").pack(pady = 10, fill = "both")  tk.Label(serverAddressFrame, text = "IP:").pack(side = "left", padx = 10)  self.serverIPEntry = tk.Entry(serverAddressFrame, width = 15, textvariable = self.clientConf["serverIP"])  self.serverIPEntry.pack(side = "left")  tk.Label(serverAddressFrame, text = "Port:").pack(side = "left", padx = 10)  self.serverPortEntry = tk.Entry(serverAddressFrame, width = 5, textvariable = self.clientConf["serverPort"])  self.serverPortEntry.pack(side = "left")  cryptoFrame = tk.Frame(self)  cryptoFrame.pack(side = "top", pady = 20)  cryptoKeyFrame = tk.Frame(cryptoFrame)  cryptoKeyFrame.pack(side = "top")  tk.Label(cryptoKeyFrame, text = "Криптография").pack(fill = "both", pady = 10)  tk.Label(cryptoKeyFrame, text = "Ключ:", width = 7).pack(side = "left", padx = 5)  self.keyEntry = tk.Entry(cryptoKeyFrame, width = 32, textvariable = self.clientConf["key"])  self.keyEntry.pack(side = "left")  cryptoIVFrame = tk.Frame(cryptoFrame)  cryptoIVFrame.pack(side = "top")  tk.Label(cryptoIVFrame, text = "Вектор:", width = 7).pack(side = "left", padx = 5)  self.ivEntry = tk.Entry(cryptoIVFrame, width = 32, textvariable = self.clientConf["iv"])  self.ivEntry.pack(side = "left")  whitelistFrame = tk.Frame(self)  whitelistFrame.pack(side = "top", pady = 20)  #  tk.Label(whitelistFrame, text = "Белый список").pack(fill = "both")  self.whitelistText = tk.Text(whitelistFrame, height = 10)  self.whitelistText.pack(side = "top", padx = 10)  self.saveBut = tk.Button(self, text = "Сохранить")  self.saveBut.bind("<Button-1>", self.onSaveButClick)  self.saveBut.pack(side = "top")  def onSaveButClick(self, e):  r = {}  for key in self.clientConf:  r[key] = self.clientConf[key].get()  r["whitelist"] = list(filter(lambda x: x != "", self.whitelistText.get("1.0", tk.END).split("\n")))  f = filedialog.asksaveasfile(mode='w')  if f:  json.dump(r, f, sort\_keys=True, indent=4)  def onOpenMenu(self):  f = filedialog.askopenfile()  if f:  try:  r = json.load(f)  except:  messagebox.showerror("Ошибка", "Выбран ошибочный файл")  return  for key in r:  try:  self.clientConf[key].set(r[key])  except:  continue  self.whitelistText.delete("1.0", tk.END)  try:  self.whitelistText.insert(tk.END, "\n".join(r["whitelist"]))  except:  pass  m = ClientConfigurator()  m.mainloop() |

Рисунок 1 – Исходный код конфигуратора

1. Экранные формы конфигуратора

Экранные формы конфигуратора представлены на рисунке 2.

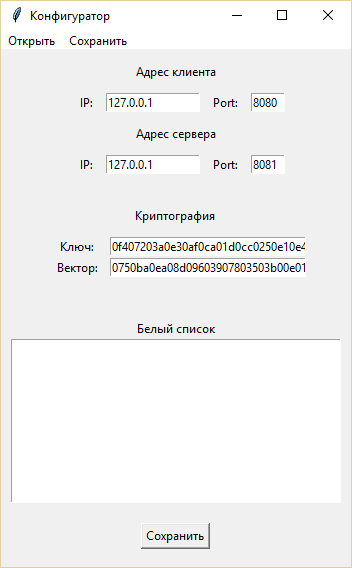


Рисунок 2 – Экранные формы конфигуратора

1. Разработка клиент-серверного приложения

Основная логика приложения заключается в следующем:

1. На клиент приходит в бинарном формате http запрос к сайту
2. Этот запрос шифруется при помощи алгоритма AES n количеством ключей.
3. Шифротекст отправляется на сервер приложения
4. На стороне сервера расшифровывается текст заданным ключем.
5. Если в конфигурации указан IP и порт сервера, то сервер отправляет на сервер шифротекст
6. Если не указан, то запрос идет на ip адрес хоста по 80-му порту. В бинарном виде отправляется запрос.
7. При получении ответа, ответ шифруется и отправляется обратно серверу\клиенту, который отправлял данному серверу http запрос.
8. На клиентской части производится расшифровка текста и отправка приложению, которое первоначально отправило запрос

Таблица маршрута движения пакета представлена в таблице 1

Таблица 1

|  |  |  |  |
| --- | --- | --- | --- |
| Адрес отправителя | Маска | Шлюз | Адрес получателя |
| 192.168.0.40 | 255.255.255.0 | 192.168.0.1 | 192.168.0.27 |
| 192.168.0.27 | 255.255.255.0 | 192.168.0.1 | 192.168.0.3 |
| 192.168.0.3 | 0.0.0.0 | 192.168.0.1 | 0.0.0.0 |

Исходный код клиентской части представлен на рисунке 3. Исходный код серверной части представлен на рисунке 4.

|  |
| --- |
| import socket  from Crypto.Cipher import AES  import xmlrpc.client  import re  import json  import sys  import urllib.request  def sendData(d, host):  url = d.split("\r\n")[0]  print(d)  if ("http://" in url):  req = urllib.request.Request(url.split(" ")[1])  print(url.split(" ")[1])  else:  req = urllib.request.Request("http://" + host + url.split(" ")[1])  print("http://" + host + url.split(" ")[1])  for line in d.split("\r\n")[1:]:  if (":" in line) and (not "Accept-Encoding:" in line):  h = line.split(":")  # print(h[0], h[1:])  req.add\_header(h[0], "".join(h[1:]))  data = urllib.request.urlopen(req).read()  return data  def printError(errorText):  print(errorText)  exit(0)  def readConfig(path):  cfg = {}  keyWords = ["key", "iv", "serverIP", "serverPort", "clientIP", "clientPort", "whitelist"]  try:  with open(path, "r") as f:  cfg = json.load(f)  except:  printError("Ошибочный файл")  f.close()  for key in keyWords:  if not(key in cfg):  printError("В файле конфиругации не определенно поле `%(keyName)s`"%{"keyName": key})  return cfg  def hexToBinStr(h):  try:  return bytes([int(i, 16) for i in h[::3]])  except:  printError("Не удалось получить данные из файла, возможно они поврежденны")  if len(sys.argv) > 1:  cfg = readConfig(sys.argv[1])  else:  printError("client.py <config file>")  KEY = hexToBinStr(cfg["key"])  IV = hexToBinStr(cfg["iv"])  DESTIP = cfg["serverIP"]  DESTPORT = cfg["serverPort"]  WHITELIST = cfg["whitelist"]  targetServer = xmlrpc.client.ServerProxy("http://" + DESTIP + ":" + str(DESTPORT))  s = socket.socket()  try:  s.bind((cfg["clientIP"], int(cfg["clientPort"])))  except:  printError("Не удалось запустить сервер. Возможно неверно заданы поля `clientIP` и `clientPort`")  s.listen(1)  print("Программа успешно запущена")  print("IP:", "0.0.0.0" if cfg["clientIP"] == "" else cfg["clientIP"])  print("Port:", cfg["clientPort"])  while True:  conn, addr = s.accept()  print("Пришли данные от", addr)  req = b''  data = conn.recv(1024)  conn.settimeout(0.1)  while data:  req += data  try:  data = conn.recv(1024)  except socket.error:  break  try:  host = re.search("(?<=Host: )(\w|\.)\*", data.decode("UTF-8")).group(0)  except:  continue  if host in WHITELIST:  ans = sendData(data.decode("UTF-8"), host)  else:  for k, iv in zip(KEY, IV):  aes = AES.new(k, AES.MODE\_CFB, iv)  msgEncrypt = aes.encrypt(data)  try:  ans = targetServer.sendData(msgEncrypt, host).data  except:  print("ERROR: Сервер не доступен")  continue  for k, iv in zip(KEY, IV):  aes = AES.new(k, AES.MODE\_CFB, iv)  ans = aes.decrypt(ans)  conn.send(ans) |

Рисунок 3 – исходный код клиентской части

|  |
| --- |
| #!/usr/bin/python3  from Crypto.Cipher import AES  from xmlrpc.server import SimpleXMLRPCServer  from xmlrpc.server import SimpleXMLRPCRequestHandler  import xmlrpc.client  import socket  import sys  import json  import urllib.request  def printError(errorText):  print(errorText)  exit(0)  def readConfig(path):  cfg = {}  keyWords = ["key", "iv", "clientIP", "clientPort", "serverIP", "serverPort"]  try:  with open(path, "r") as f:  cfg = json.load(f)  except:  printError("Ошибочный файл")  f.close()  for key in keyWords:  if not(key in cfg):  printError("В файле конфиругации не определенно поле `%(keyName)s`"%{"keyName": key})  return cfg  def hexToBinStr(h):  try:  return bytes([int(i, 16) for i in h[::3]])  except:  printError("Не удалось получить данные из файла, возможно они поврежденны")  class RequestHandler(SimpleXMLRPCRequestHandler):  rpc\_paths = ('/RPC2',)  def sendData(data, host):  d = aes.decrypt(data.data)  if SERVER\_PORT and SERVER\_IP:  try:  targetServer = xmlrpc.client.ServerProxy("http://" + SERVER\_IP+ ":" + str(SERVER\_PORT))  except:  printError("Неверно задана конфигурация serverIP и serverPort")    return aes.encrypt(targetServer.sendData(d, host))  d = d.decode("UTF-8")  req = ""  for line in d.split("\r\n"):  if not "Accept-Encoding:" in line:  req += line + "\r\n"  print(req)  s = socket.socket()  s.connect((host, 80))  s.send(req.encode("UTF-8"))  req = b''  data = s.recv(1024)  s.settimeout(1)  while data:  req += data  try:  data = s.recv(1024)  except socket.error:  break  return aes.encrypt(req)  if len(sys.argv) > 1:  cfg = readConfig(sys.argv[1])  else:  printError("server.py <config file>")  KEY = hexToBinStr(cfg["key"])  IV = hexToBinStr(cfg["iv"])  SERVER\_IP = cfg["serverIP"]  SERVER\_PORT = cfg["serverPort"]  aes = AES.new(KEY, AES.MODE\_CFB, IV)  try:  server = SimpleXMLRPCServer((cfg["clientIP"], int(cfg["clientPort"])),  requestHandler=RequestHandler,  allow\_none=True)  except:  printError("Неверно заданые поля `ServerIP` и `ServerPort`")  server.register\_introspection\_functions()  server.register\_function(sendData)  print("Сервер успешно запущен")  print("IP:", "0.0.0.0" if cfg["serverIP"] == "" else cfg["serverIP"])  print("Port:", cfg["serverPort"])  server.serve\_forever() |

Рисунок 4 – исходный код серверной части

1. Вывод

В ходе выполнения лабораторной работы были полученные необходимые знания базового синтаксиса языка программирования Python. Был изучен протокол вызова удаленных процедур xml-rpc и в результате было разработано клиент-серверное приложение эмулирующий анонимную сеть «Tor» с 2-мя узлами. Данные знания являются фундаментальными и необходимы для дальнейшего продолжения изучения языка Python.