Не для собственных нужд, но как-то понадобилось написать простой web-сервер для обработки на нём загруженных данных. В общем-то должен был получиться простой вёб-интерфейс к программе. Он и получился, местами даже слишком простой. В общем суть такова – необходимо получить от пользователя экзешник, на сервере его обработать нужными программами и выдать результат, т.е. обработанный файл для скачивания. Можно было бы написать скрипт на php, что было бы проще, но тогда необходимо с собой таскать какой-нибудь вёб-сервер (apache, denwer и т.п.). На питоне можно его реализовать достаточно просто в самом скрипте используя стандартные заготовки и библиотеки, такие как BaseHTTPServer. Ниже следует листинг с краткими комментариями.

*# -\*- coding: utf-8 -\*-*

**import** cgi

**from** os **import** curdir, sep

**from** BaseHTTPServer **import** BaseHTTPRequestHandler, HTTPServer

**import** subprocess

**class** **MyHandler**(BaseHTTPRequestHandler):

**def** **do\_GET**(self):

**try**:

**if** self.path != "/output.exe":

f = open(curdir+sep+"upload.html")

self.send\_response(200)

self.send\_header("Content-type", "text/html")

self.end\_headers()

self.wfile.write(f.read())

f.close()

**else**:

self.send\_response(200)

self.send\_header("Content-type", "application/octet-stream")

self.end\_headers()

self.wfile.write(open(curdir+sep+"output.exe", "rb").read())

**except** IOError:

self.send\_error(404,"File Not Found: %s" % self.path)

**def** **do\_POST**(self):

**try**:

ctype, pdict = cgi.parse\_header(self.headers.getheader("content-type"))

**if** ctype == "multipart/form-data":

query = cgi.parse\_multipart(self.rfile, pdict)

self.send\_response(200)

self.end\_headers()

upfile = query.get("file")

f = open(curdir+sep+"output.exe", "wb")

f.write(upfile[0])

f.close()

params = " np output.exe"

p = query.get("encryption")

**if** p[0] == "aes":

params += " sf 1"

**elif** p[0] == "rc5":

params += " sf 2"

**elif** p[0] == "xor":

params += " sf 3"

**else**:

params += " sf 0"

p = query.get("hw\_bind")

**if** p[0] == "yes":

p = query.get("hw\_bind\_serial")

**assert** len(p[0]) == 8

params += " sn " + p[0]

**else**:

params += " sn 0"

p = query.get("passwd")

**assert** len(p[0]) > 0

params += " pass " + p[0]

p = query.get("pack")

**if** p[0] == "yes":

params += " pack 1"

**else**:

params += " pack 0"

pipe = subprocess.Popen("processor.exe "+params, shell=**True**, stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

pipe.stdin.close()

pipe.wait()

self.wfile.write('[Download results](http://zetblog.ru/output.exe).')

**except** Exception:

**pass**

**if** \_\_name\_\_ == "\_\_main\_\_":

**try**:

server = HTTPServer(("", 8080), MyHandler)

**print** "started httpserver..."

server.serve\_forever()

**except** KeyboardInterrupt:

**print** "^C received, shutting down server"

server.socket.close()

Итак, для начала импортируем из библиотеки BaseHTTPServer два класса: HTTPserver — собственно, сам вёб-сервер и BaseHTTPRequestHandler — класс, который служит для обработки запросов от пользователей.

Объявляем свой класс MyHandler, который наследуется от BaseHTTPRequestHandler. И перегружаем в нём два метода — do\_GET и do\_POST для обработки данных, переданных методом GET и POST, соответственно.

В первом обработчике проверяем запрос от пользователя (содержится в self.path), если запрашиваемый документ не является /output.exe, то выдаём форму для загрузки файла, иначе — считываем и выдаём output.exe. Так же, перед этим, посылается ответ от сервера с кодом 200, что соответствует успешной обработке запроса от клиента и передаём заголовок, в котором указан тип содержимого (text/html для html-код или application/octet-stream для бинарного файла).

В обработке POST-запроса проверяем какие данные были переданы. Сначала получаем файл и записываем его под именем output.exe. Далее формируем строку параметров на основе переданных от пользователя данных, запускаем хранящуюся на сервере программу для обработки полученных файлов, ждём пока она завершится и выдаём ссылку на скачивание результата.

Далее, в самой программе создаём вёб-сервер, указав при этом порт, на котором он будет работать (в примере это 8080) и класс, который необходимо использовать для обработки запросов от пользователей.

Недостаток заключается в том, что данная реализация не сможет обработать одновременную загрузку файлов от разных пользователей, т.к. в лучшем случае просто один из запросов затрёт файл output.exe, полученный в другом запросе. Но такая задача не стояла. На мой взгляд хороший пример для демонстрации возможностей python’а.

Так же код html-файла upload.html, который должен располагаться в той же директории, что и сервер:

Начало формы

|  |  |
| --- | --- |
| Encryption | None AES RC5 XO |
| Hardware binding | No Yes <="" td="" style="margin: 0px; padding: 0px;"> |
| Password | <="" td="" style="margin: 0px; padding: 0px;"> |
| Pack | No Yes |
| <="" td="" style="margin: 0px; padding: 0px;"> | |

Конец формы