PWS Project

REST API For Remote File Storage and Encryption

Project by:
Arnav Sachdev
179301043
CSE – 7A

Problem Statement: To implement a RESTful web service prototype and show the interaction between the client and the server.

Description:

This project consists of a Python server written using the minimialist Flask framework which allows the user to statelessly transfer and retrieve files over HTTP/S.

The properties of the web service are:-

- Stateless transfer of files
- API endpoints which can be extended in functionality by other web services or client applications
- Files stored in the remote server are protected by AES-128 encryption
- The client and server are competely independent and new client software can easily be written by following the API rules.

Code:-

The code is divided into 2 modules and one client application:

- main: *implements the web server and api*
- crypto : *implements the cryptographic functionality used by main*
- client.sh : *client to interact with the api written in shell script (using curl as the backend)*

```python
#!/usr/bin/env python3
from flask import Flask, request
from crypt import AES
import os
import json
from uuid import uuid4
import base64

app = Flask(__name__)

SAVE_DIR = '/tmp/encfs'

def save_data(data,filename,store_dir):
    if os.path.isdir(store_dir) == False:
        os.makedirs(store_dir)
    else:
        if store_dir[-1] == '/':
            del store_dir[:-1]
        open(store_dir+'/'+filename,'wb').write(data)


@app.route('/')
def index():
    welcome_message = '''file storage API\n1) send data to /e
    return welcome_message

@app.route('/encrypt',methods=['POST'])
def encrypt():
    data = request.get_data()
    secret = os.urandom(16)
```

```python
        self.shared_secret = (self.privkey.exchange(ec.ECDH(

class AES:
    def __init__(self,secret_key):
        self.secret_key = secret_key
        self.iv = None
        self.algorithm = algorithms.AES(self.secret_key)
        self.cipher = None
        self.encryptor = None
        self.decryptor = None


    def encrypt(self,data):
        self.iv = os.urandom(16)
        self.cipher = Cipher(self.algorithm,modes.CBC(self.i
        self.encryptor = self.cipher.encryptor()
        tmp_padder = padding.PKCS7(128).padder()
        padded_data = tmp_padder.update(data)+tmp_padder.fin
        return (self.iv)+(self.encryptor.update(padded_data)


    def decrypt(self,data):
        self.iv = data[0:16]
        crypted_data = data[16::]
        self.cipher = Cipher(self.algorithm,modes.CBC(self.i
        self.decryptor = self.cipher.decryptor()
        tmp_unpadder = padding.PKCS7(128).unpadder()
        try:
            decrypted_data = (self.decryptor.update(crypted_
            final_decryption = tmp_unpadder.update(decrypted_
        except:
```

Fig 1.1: The project directory structure and code

```python
@app.route('/')
def index():
    welcome_message = '''file storage API\n1) send data to /encrypt endpoint\n2) you will get a uuid and secret_key\n3) send the
    return welcome_message
```

Fig 1.2: / index endpoint

```python
@app.route('/encrypt',methods=['POST'])
def encrypt():
    data = request.get_data()
    secret = os.urandom(16)
    uniq = str(uuid4()).replace('-','')
    a = AES(secret_key=secret)
    hidden_data = a.encrypt(data)
    save_data(hidden_data,uniq,SAVE_DIR)
    resp = {'uuid':uniq,'secret_key':base64.b64encode(secret).decode()}
    return resp
```

Fig 1.3: /encrypt endpoint

```python
@app.route('/retrieve',methods=['POST'])
def retrieve():
    try:
        json_data = request.get_json(force=True)
        uniq = json_data['uuid']
        secret_key = json_data['secret_key']
    except:
        return 'invalid json'
    if (os.path.isfile(SAVE_DIR+'/'+uniq) == False):
        return 'file does not exist on server'
    else:
        #sanitize uniq
        uniq.replace('/','')
        uniq.replace('.','')
        #decode secret
        secret = base64.b64decode(secret_key)
        ret_file_enc = open(SAVE_DIR+'/'+uniq,'rb').read()
        a = AES(secret_key=secret)
        ret_file = a.decrypt(ret_file_enc)
        return ret_file
```

Fig 1.4: /retrieve endpoint

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master›
$ curl 127.0.0.1:5000/
file storage API
1) send data to /encrypt endpoint
2) you will get a uuid and secret_key
3) send the uuid and sceret key to /retrieve endpoint and get back your file
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master›
$

glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master›
$ ./run.sh
 * Serving Flask app "main/main.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [07/Nov/2020 17:07:50] "GET / HTTP/1.1" 200 -

[0] 0:zsh*                                                    "predator" 17:07 07-Nov-20
```

Fig 1.5: Runtime screenshot of the webserver index

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master›
$ cat run.sh
export FLASK_APP=main/main.py
flask run
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$
```

Fig 1.6: run.sh to run the application

```python
#!/usr/bin/env python3
from flask import Flask, request
from crypt import AES
import os
import json
from uuid import uuid4
import base64

app = Flask(__name__)

SAVE_DIR = '/tmp/encfs'

def save_data(data,filename,store_dir):
    if os.path.isdir(store_dir) == False:
        os.makedirs(store_dir)
    else:
        if store_dir[-1] == '/':
            del store_dir[:-1]
        open(store_dir+'/'+filename,'wb').write(data)
```

Fig 1.7: Imports and helper functions

```
#!/bin/zsh

upload_file(){
  if [ $# -ne 2 ]
  then
    echo echo 'useage ./client [upload [filename]] [download [secret_key uuid]] url'
    return 1
  else
  file=$(cat $1)
    curl -XPOST --data \'$file\' $2/encrypt
  fi
}

download_file(){
  if [ $# -ne 3 ]
  then
    echo 'useage ./client [upload [filename]] [download [secret_key uuid]] url'
    return 1
  else
    json={\"secret_key\":\"$1\",\"uuid\":\"$2\"}
    #echo \'$json\'
    echo curl -XPOST --data \'$json\' $3/retrieve | zsh
  fi
}

if [ $# -lt 3 ]
then
  echo 'useage ./client [upload [filename]] [download [secret_key uuid]] url'
  return 1
else
  if [ $1 = 'upload' ]
  then
    echo UPLOAD
    upload_file $2 $3
  elif [ $1 = 'download' ]
  then
    echo DOWNLOAD
    download_file $2 $3 $4
  else
    echo $1 'is an invalid option'
    return 1
  fi
fi
```

Fig 1.8: Client application

API Documentation:-

The API contains **three** endpoints:

a) /
Method-Type: GET
Response Content-Type: text/plain

Description: The / or index endpoint returns basic information on the web service (as shown in the screenshot in **figure 1.5**)

b) /encrypt
Method-Type: POST
Content-Type: text/plain
Response Content -Type: application/json

Description: The endpoint accepts arbitrary raw data send via a HTTP POST request to *encrypt. The server reads the data in the POST request, generates a random 128 bit secret key and encrypts the POSTed data. Each file is identified by a UUID4 (universal unique identifier) of 32 bits and stored on the server's filesystem (configureable). The returned data is of type* ***application/json*** and contains two keys:

- uuid
- secret_key

c) /retrieve
Method-Type: POST
Content-Type: application/json
Response Content-Type: text/plain

Description: The endpoint accepts json data containing the following two keys:

- uuid
- secret_key

The server parses the json and if no errors are found and both keys are present, searches for any file on the server matching the requested UUID. If a file is found, then the secret_key is used for it's decryption and resulting data is returned.

Demonstration of Functionality:-

1) Uploading a file for encryption:-

a) A file is ready for upload.

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ cat important.txt
This is a very important file.
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$
```

b) The file is uploaded and we get back a json object containing the secret key used for it's encryption and a uuid to identify our file.

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./client.sh upload important.txt 127.0.0.1:5000
UPLOAD
{"secret_key":"JyPEqgbz8yh1JxyrpT+3eQ==","uuid":"cd26cae9e3aa4984b46f4e6ec3656c41"}
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$
```

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./run.sh
 * Serving Flask app "main/main.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [07/Nov/2020 17:42:27] "POST /encrypt HTTP/1.1" 200 -
```

c) We use the give the client application the secret_key, uuid and url for the web server and get our file back!

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./client.sh upload important.txt 127.0.0.1:5000
UPLOAD
{"secret_key":"JyPEqgbz8yh1JxyrpT+3eQ==","uuid":"cd26cae9e3aa4984b46f4e6ec3656c41"}
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./client.sh download JyPEqgbz8yh1JxyrpT+3eQ== cd26cae9e3aa4984b46f4e6ec3656c41 127.0.0.1:5000
DOWNLOAD
'This is a very important file.'%
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$
```

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./run.sh
 * Serving Flask app "main/main.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [07/Nov/2020 17:42:27] "POST /encrypt HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2020 17:43:14] "POST /retrieve HTTP/1.1" 200 -
```

d) If for example we send an incorrect uuid, the api tells us that the file does not exist

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./client.sh upload important.txt 127.0.0.1:5000
UPLOAD
{"secret_key":"JyPEqgbz8yh1JxyrpT+3eQ==","uuid":"cd26cae9e3aa4984b46f4e6ec3656c41"}
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./client.sh download JyPEqgbz8yh1JxyrpT+3eQ== cd26cae9e3aa4984b46f4e6ec3656c41 127.0.0.1:5000
DOWNLOAD
'This is a very important file.'%
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./client.sh download JyPEqgbz8yh1JxyrpT+3eQ== ef26ty2733a54834b4674e6ec3656c56 127.0.0.1:5000
DOWNLOAD
file does not exist on server%
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$
```

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./run.sh
 * Serving Flask app "main/main.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [07/Nov/2020 17:42:27] "POST /encrypt HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2020 17:43:14] "POST /retrieve HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2020 17:44:31] "POST /retrieve HTTP/1.1" 200 -
```

e) If we send an incorrect decryption key, then the api tells us that the decryption failed

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./client.sh download GTOExgbz8yh1JxyrfH94dF== cd26cae9e3aa4984b46f4e6ec3656c41 127.0.0.1:5000
DOWNLOAD
decryption failed!%
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$
```

```
glitch@predator ~/university_stuff/7sem/pws/pws-project/rest-api-test0 ‹master*›
$ ./run.sh
 * Serving Flask app "main/main.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [07/Nov/2020 17:42:27] "POST /encrypt HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2020 17:43:14] "POST /retrieve HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2020 17:44:31] "POST /retrieve HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2020 17:46:59] "POST /retrieve HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2020 17:47:32] "POST /retrieve HTTP/1.1" 200 -
```

Conclusion:

To conclude, I have demonstrated that the API is stateless since the user's session does not need to be retained. Simply a valid uuid and decryption key is necessary to retrieve a file.

It is also clear that the client and the server are completely indepedant, and all that is necessary for interacting with the API is HTTP. Infact, the server ignores HTTP headers as each endpoint has its own isolated function, thus we can interact with the API with various other tools as well, including netcat and Postman. For the purposes of this project I have written the client in shell script and utilized curl for making the HTTP requests.

Dependancies/Requirements:

- **Python3**
- **Flask framework**
- **python cryptography library (used in crypt.py)**
- **zsh**

Ideally, should be run in a Linux environment. Use ./run.sh to start the server and ./client.sh to interact with the API.