

COMP.SEC.300 Secure Programming

Individual Project Report

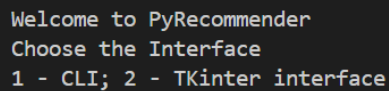
UnoriginalOrigi

1 Program Functionality

The program, PyRecommender¹, provides two implementations of similar functionalities with different graphical interfaces, depending on the systems capabilities. The implementation is written in Python and makes use of 2 libraries:

1. **Cryptodome** – a library which implements standard encryption schemes.
2. **SpotiPy** – a wrapper library which implements the Spotify API functions into Python. The implementation requires a valid Spotify account for the full functionality.

The first implementation is a Command Line Interface (CLI) implementation, the other one is a graphical interface implementation using Tkinter. On start, the program prompts the user to choose the implementation type (Figure 1) through the command line:

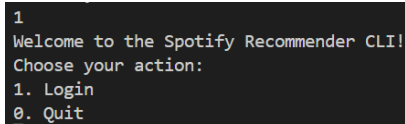


```
Welcome to PyRecommender
Choose the Interface
1 - CLI; 2 - Tkinter interface
```

Figure 1: Interface selection

1.1 CLI Implementation

The CLI implementation provides the user with the ability to look up specific songs or find related artists to an inputted artist or band name. The CLI implementation begins by prompting the user by either logging in using the access tokens or quitting out of the program (Figure 2):



```
1
Welcome to the Spotify Recommender CLI!
Choose your action:
1. Login
0. Quit
```

Figure 2: CLI initialisation

Selecting Login, starts the login process, where a user can input their Client ID and their Client Secret into the CLI (Figure 3a). A successful login will allow the user to make use of the search functionalities (Figure 3b).

Additionally, if a previous login has been detected, the program will query the user if they wish to use previously saved details (Figure 4).

¹https://github.com/UnoriginalOrigi/PyRecommender_SePro_Project

```

Input your Client ID token:
fe04daff59d142db8b5d59dfeaaa766d
Input your Client Secret token:
Password:

```

```

1 - Song/Artist search, 2 - Recommend similar artists

```

(a) CLI Login

(b) CLI Search Select

Figure 3: Menu Selection

```

Welcome to the Spotify Recommender CLI!
Choose your action:
1. Login
0. Quit
1
Saved credentials found, use saved credentials? [y]/n

```

Figure 4: CLI Saved Login

Song/Artist Search Selecting "Song/Artist Search" allows a user to input a query of either song names (Figure 5a) or an artist (Figure 5b) and find songs from said specific author. These songs are the listed out in order of similarity as provided by the Spotify API. Afterwards a user can request to do another search.

```

1 - Song/Artist search, 2 - Recommend similar artists
1
Enter search query: Smells Like teen
1 Smells Like Teen Spirit - Nirvana
2 Uneasy Hearts Weigh The Most - Dance Gavin Dance
3 Smells Like Teen Spirit - Witchz
4 Wild Boy - mgk
5 Smells Like Teen Spirit - Alt Mix - Witchz
6 Smells Like Teen Spirit - Naeleck
7 Pimpin' All Over The World - Ludacris
8 Smells Like Teen Spirit - Malia J
9 I Replied To Tyler With Three Blue Cars - Hot Mulligan
10 RISE - Remix - League of Legends
Another search? ([y]/n)

```

```

Enter search query: Bring me the Horizon
1 Can You Feel My Heart - Bring Me The Horizon
2 Throne - Bring Me The Horizon
3 turn it up - skypebf
4 Sleepwalking - Bring Me The Horizon
5 Kool-Aid - Bring Me The Horizon
6 Kingslayer (feat. BABYMETAL) - Bring Me The Horizon
7 LoST - Bring Me The Horizon
8 Can You Feel My Heart - Remix - Bring Me The Horizon
9 CODE MISTAKE - CORPSE
10 DArkSide - Bring Me The Horizon
Another search? ([y]/n)

```

(a) Finding Specific Songs

(b) Finding Specific Artist Songs

Figure 5: Artist Lookup Functionality

Recommend Similar Artist Selecting "Recommend Similar Artist" allows a user to type in an artist name and find related artist with a similar music style or emotion. The related artists are listed in order, as decided by the Spotify API. PyRecommender can then continue with either another search or by finding songs from one of the listed artists (Figure 6b). This searches functionality is a modified version of the Artists Search discussed previously.

```

2
Enter an Artist: Linkin Park
1 Papa Roach
2 Limp Bizkit
3 Three Days Grace
4 Breaking Benjamin
5 Disturbed
6 System Of A Down
7 Green Day
8 Korn
9 3 Doors Down
10 Slipknot
11 Hoobastank
12 Evanescence
13 The Offspring
14 Sum 41
15 Skillet
16 blink-182
17 Seether
18 Nickelback
19 Bullet For My Valentine
20 P.O.D.
Lookup songs from a related artist? ([y]/n)

```

```

Lookup songs from a related artist? ([y]/n) y
Type in the ID of the artist: 17
1 Fake It - Seether
2 Remedy - Seether
3 Broken - Seether
4 Fine Again - Seether
5 Nobody Praying For Me - Seether
6 Country Song - Seether
7 Careless Whisper - Seether
8 Gasoline - Seether
9 Words As Weapons - Seether
10 Same Damn Life - Seether
Another? ([y]/n)

```

(a) Related Artist

(b) Related Artists Song

Figure 6: Related Artist Functionality

1.2 Tkinter Implementation

The alternative implementation makes use of Tkinter to provide a graphical user interface. The two main windows created with it are the login window (Figure 7) and the main program window (Figure 8).

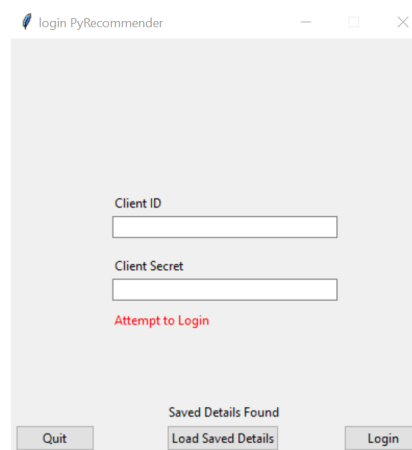


Figure 7: Tkinter Login Screen

The login window functions similarly to the login of the CLI interface. The user is prompted for their client ID and client Secret. Correctly entering the values, the user is logged into the main window. Additionally, if a previous login has been detected and saved, then the user can

use the previously saved login details to speed up the login process.

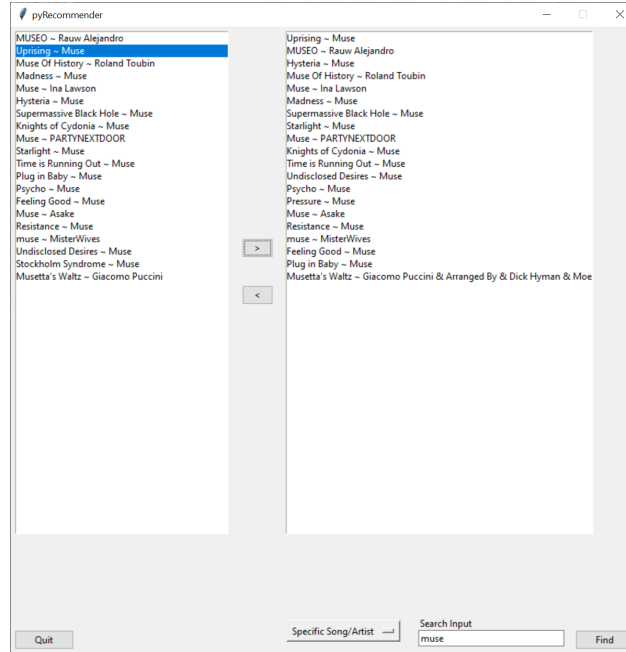


Figure 8: Main Tkinter PyRecommender Window

The main window is a simplified version of the CLI implementation. The user can enter their search query in the bottom right corner. The user can switch between searching for related artists and searching for specific songs or artists. The left list shows the search results, while the right list shows songs from a specific author found in the left list (These lists can be interacted through the use of the "<" and ">" buttons).

2 Program Structure

The program structure is broken down into 2 main implementation files as well as 3 utility programs. The overall structure is:

- **main.py** – core execution of the project.
- **/util** – a folder containing utilities for credential input, loading as well as secure storage and communication.
 - **credential_manager.py** – code for inputting and saving new credential data as well as loading prior credentials.
 - **MyAES.py** – utilities for key generation/loading, encryption and decryption with AES-256-CBC.

- **general_func.py** – interchangeable functions for both CLI and Tkinter implementations.
- **CLI_implemenatation.py** – CLI implementation of PyRecommender.
- **tkinter_implementation.py** – Tkinter implementation of PyRecommender.

3 Secure Programming Solutions

The code contains multiple Secure Programming solutions, regarding input validation, secure file creation and secure login detail storage. Following OWASP Top Ten as well as OWASP Vulnerabilities to check which solutions have been made. The following have been addressed:

- **Vulnerable and Outdated Components** – The code is constructed using Python 3.9 as well as uses the newest available updates of the requirements.
- **Identification and Authentication Failures** – Invalid credentials are ignored after testing them, only specific symbols and lengths are allowed into the input fields.
- **Security Misconfiguration** – the public client ID is available for others to see, but the secret parameter is encrypted with AES-256-CBC. It is assumed the secret key is never shared with the server and the connection is made securely through HTTPS/TLS.
- **Cryptographic Failures** – Strong symmetric encryption is used with connections made online being trusted to be sent securely through HTTPS/TLS.
- **Buffer Overflow (off-by-one, memory leakage)** – Built-in security in Python. Additionally, the code checks input sizes as well as inputted symbols to avoid going over specified lengths.

3.1 Input Validation

Throughout the implementation there are various efforts made to reduce invalid or dangerous inputs being inputted throughout the program. As such the following code snippets in Figure 9, Figure 10 and Figure 12 showcase some of the implemented code solutions for input validation.

Figure 9 showcases how the client login details are validated. Firstly the input should always be 32 characters long as such inputs which are shorter or longer than 32 characters are seen as invalid and will throw and catch an error. Additionally, the input must contain only Base62 characters, which include uppercase and lowercase alphanumerical symbols. Symbols that are outside the valid Base62 symbols are discovered through the use of Regular Expression

```

try:
    print("Input your Client ID token:")
    client_id_input = input()
    if len(client_id_input) != EXPECTED_INPUT_LENGTH:
        raise OverflowError
    if re.findall(BASE62_INVALID_SYMBOLS,client_id_input):
        raise ValueError
    print("Input your Client Secret token:")
    client_secret_input = getpass()
    if len(client_secret_input) != EXPECTED_INPUT_LENGTH:
        raise OverflowError
    if re.findall(BASE62_INVALID_SYMBOLS,client_secret_input):
        raise ValueError

```

Figure 9: Client Detail Validation

and throw and catch an error for the invalid input.

```

def __login_window(self):
    if len(self.entry1.get()) < EXPECTED_INPUT_LENGTH or len(self.entry2.get()) < EXPECTED_INPUT_LENGTH:
        self.label4.config(text="Invalid Credentials. Input too short.") #Credentials wrong length

```

Figure 10: Login Validation in Tkinter

```

def __character_limit(self, txt_entry):
    if len(txt_entry.get()) > EXPECTED_INPUT_LENGTH:
        txt_entry.set(txt_entry.get()[:EXPECTED_INPUT_LENGTH])

```

Figure 11: Character Limit

Figure 10 and Figure 11 showcases how the Tkinter implementations login details are checked. Namely, as the character limit function does not allow for inputs above a set amount of character, the input validation only checks if the length is lower than the expected amount as explained previously. Going over the char limit will not update the input field.

As discussed with the previous input validations, the search in both implementation, make sure that the search input is less than 64 characters long to avoid invalid or malicious inputs. As songs can contain special character from varying regions further input validation is entrusted by the Spotify API to not limit what artists can be found.

3.2 Secure File Operations

As the implementations make use of saving some details to a file, it is important to implement file opening and closing securely to avoid creating race conditions or reading insecure files.

```

if len(search_query) > INPUT_SIZE: # Input Validation
    search_query = ""
    print("Input too long. Input size <=",INPUT_SIZE)
elif len(search_query) == 0:
    print("No Input given")

```

Figure 12: Search Input Validation

```

def loadKey():
    with open("key.json","r") as file_i:
        params = json.load(file_i)
    return json.loads(params)

client_secret_enc = encryptText(data = client_secret, params=params)
with open("client_info.txt","w") as f:
    f.write("{}\n{}".format(client_id,client_secret_enc))

```

(a) Secure File Read

(b) Secure File Write

Figure 13: Secure File Operations

As such Figure 13a and Figure 13a shows the best practices of reading and writing to files in Python.

3.3 Secure Storage

To speed up the login process for a reoccurring user, the program also saves the client ID and client secret after a successful login. However, as the client secret contains sensitive information, which should not be openly available, a solution is provided by encrypting the value with AES-256-CBC into a JSON file.

```

sp = spotipy.Spotify(auth_manager=auth_manager)
client_secret_enc = encryptText(data = client_secret, params=params)

def encryptText(data, params):
    data = Padding.pad(bytes(data,"utf-8"), BLOCK_SIZE)
    cipher = AES.new(b64decode(params["key"].encode('utf-8')), AES.MODE_CBC)
    ct = cipher.encrypt(data)
    iv = cipher.iv
    ct = "".join((b64encode(iv).decode('utf-8'),b64encode(ct).decode('utf-8')))
    return ct

```

(a) Client Secret Encryption

(b) AES-256-CBC Encryption

Figure 14: Secure Storage Solutions

Figure 14a shows that after a correct login is completed the client secret is encrypted with the function defined in Figure 14b. The encryption function provides padding and generates a new Initialisation Vector (IV) for each generated ciphertext. The IV is provided with the ciphertext encoded in Base64 for transferable consistency.

4 Security Testing

To ensure that secure programming practices were followed, various security tests were run to guarantee programs functionality. This includes Manual Testing as well as Specialized Testing

through fuzzing and code analysis.

4.1 Manual Testing

To ensure the robustness of the implementation against a variety of invalid inputs, various tests were ran to ensure that there were no uncaught exceptions or unexpected behaviours. Figure 15 shows how different functions handle invalid inputs as well as what error messages they show in the case of an invalid input.

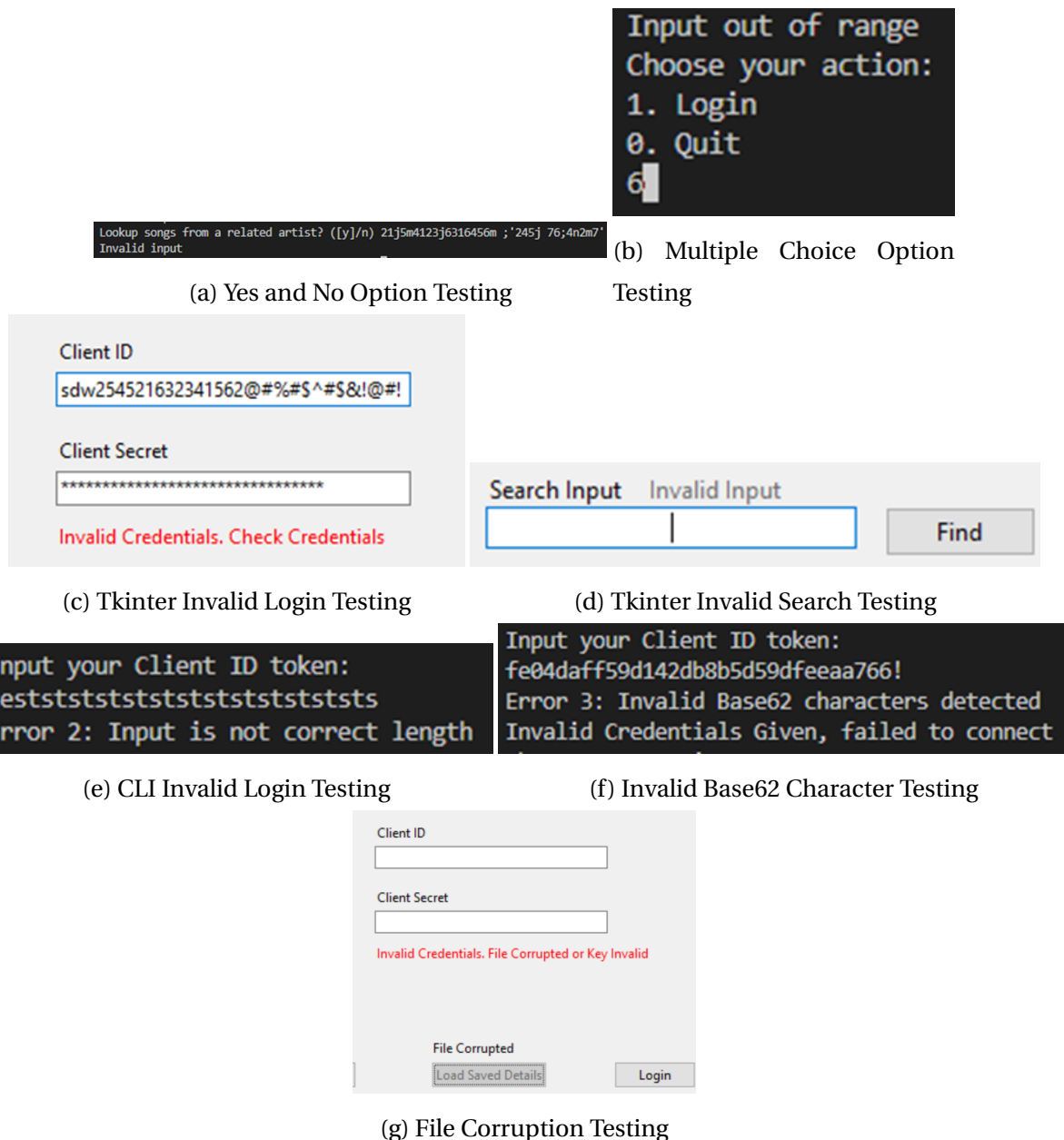


Figure 15: Manual Testing Examples

These tests include inputting possibly invalid symbols or values into each individual field and seeing if they cause an unexpected or uncaught error. Figure 15g showcases what happens if a file corruption happens during the login process, when trying to use saved credentials. Figure 15f, Figure 15e and Figure 15c show how the program reacts during the login process when invalid symbols are provided, or a successful connection cannot be made. Figure 15a and Figure 15b present how out of range inputs are handled, when the selection is limited. Figure 15d shows how an invalid search request is handled and what a user sees if he searches for an empty field or a field filled with blank spaces.

4.2 Specialized Testing

To perform a wider range of tests as well as automate the testing process, two tools were used to ensure the security of the implementation. These two tools are a code vulnerability scanner, called Snyk² and a coverage-guided Python fuzzer by Google, named Atheris³.

Snyk From their website "Snyk is a developer security platform. Integrating directly into development tools, workflows, and automation pipelines, Snyk makes it easy for teams to find, prioritize, and fix security vulnerabilities in code, dependencies, containers, and infrastructure as code. Supported by industry-leading application and security intelligence, Snyk puts security expertise in any developer's toolkit." As such the tool was used to test the code for possible vulnerabilities and improper implementation. The tool makes use of AI to test the code, as such this work has **made use of AI to test the code for possible vulnerabilities**. After scanning the two individual implementations, the program found no noticeable issues with either the CLI or Tkinter implementation (Figure 16).

We couldn't find any security issues in your code!

Figure 16: CLI and Tkinter Implementation Vulnerability Scan with Snyk

Atheris From their repository "Atheris is a coverage-guided Python fuzzing engine. It supports fuzzing of Python code, but also native extensions written for CPython. Atheris is based off of libFuzzer. When fuzzing native code, Atheris can be used in combination with Address Sanitizer or Undefined Behavior Sanitizer to catch extra bugs." The fuzzer was used on the functions in the *credential_manager.py* file as it contained the majority of different possible user inputs. From the fuzzing test certain uncaught exceptions were found, namely a possible error where

²<https://snyk.io/code-checker/python/>

³<https://github.com/google/atheris>

the input would be delivered as a binary input instead of a plaintext, leading to a `TypeError` exception. This was rectified and further fuzzing did not provide any crashes after 134 million runs (Figure 17).

```
(pyrec) wsl_base@DESKTOP-K5F6C4S:~/PyRecommender_SePro_Project$ python3 credential_manager.py fuzzer
INFO: Instrumenting functions: [7953/7953] 100%
INFO: Using built-in libfuzzer
WARNING: Failed to find function "__sanitizer_acquire_crash_state".
WARNING: Failed to find function "__sanitizer_print_stack_trace".
WARNING: Failed to find function "__sanitizer_set_death_callback".
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 530910819
INFO:      6 files found in fuzzer
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: seed corpus: files: 6 min: 1b max: 82b total: 208b rss: 62Mb
#7      INITED cov: 157 ft: 157 corp: 1/32b exec/s: 0 rss: 64Mb
#524288 pulse  cov: 157 ft: 157 corp: 1/32b lim: 4096 exec/s: 262144 rss: 64Mb
#1048576      pulse  cov: 157 ft: 157 corp: 1/32b lim: 4096 exec/s: 262144 rss: 64Mb
#2097152      pulse  cov: 157 ft: 157 corp: 1/32b lim: 4096 exec/s: 262144 rss: 64Mb
#4194304      pulse  cov: 157 ft: 157 corp: 1/32b lim: 4096 exec/s: 262144 rss: 64Mb
#8388608      pulse  cov: 157 ft: 157 corp: 1/32b lim: 4096 exec/s: 262144 rss: 64Mb
#16777216     pulse  cov: 157 ft: 157 corp: 1/32b lim: 4096 exec/s: 258111 rss: 64Mb
#33554432     pulse  cov: 157 ft: 157 corp: 1/32b lim: 4096 exec/s: 258111 rss: 64Mb
#67108864     pulse  cov: 157 ft: 157 corp: 1/32b lim: 4096 exec/s: 257122 rss: 64Mb
#134217728    pulse  cov: 157 ft: 157 corp: 1/32b lim: 4096 exec/s: 254682 rss: 64Mb
```

Figure 17: Atheris Fuzzing Outputs

5 Limitations

Listed below are the main limitations (to the best of my knowledge) of the current implementations iteration:

- The implementation has not been tested on actual users to guarantee a good user experience with the graphical user interface. As such, some interface parts may remain unclear or not straightforward for new users;
- Limited search functionality, as the implementation does not allow searching for a specific genre, mood or locale. All of which are possible options in the Spotify API;
- Cannot access users playlists or demo songs through Spotify, due to requiring different access right and additional methods of authentication;
- Secure storage is implemented only in a local sense, more as a possible demonstration of how it could be saved, however a practical solution has not been implemented using databases;
- Key renewal and revocation was not implemented due to time constraints;
- Due to the implementation of Spotipy, the cached login file cannot be redirected without updating the libraries implementation. As such the cache file can still remain on the

computer if there is a power outage as it cannot be redirected to a temporary directory.

6 Future Directions

The project managed to achieve a usable and securely implemented music recommendation system using the Spotify API through the use of Spotipy. The project used best programming practices for Python and implemented a variety of different secure programming solutions to ensure no unhandled exceptions and a smooth experience for a user. The work has implemented a way for users to search for specific songs and artists as well as find related artists to those specific results. Additionally, the implementation provides a light-weight implementation through the use of the CLI as well as a more visually clear implementation, through the use of the built in Tkinter wrapper in Python.

The possible future directions for the project include:

- Implementing more extensive program capabilities, such as music demonstrations, playlist recommendations, genre and mood search options. This requires extended login details and access control with the Spotify API;
- Full database storage for better login detail storage for other parties;
- Improving the Graphical User Interface;
- Additional security testing;
- Fixing cached access token issue;
- Implementing different encryption schemes which could be used for login detail saving as well as providing key renewal.