HAVE YOU BEEN PWNED

February 20, 2019

Pedro Sanchez Munoz
Santa Clara University
Department of Computer Science & Engineering

Contents

0.1 Introduction

This document provides supporting documentation for the HaveYouBeenPwned software, written in Python.

0.2 Business Problem

Acme Corp has been suffering from breach after breach and continually finding out through third parties that it's data is being leaked on the internet. After reviewing some open source solutions on the internet your manager has found the site *"Have I Been Pwned"* which provides APIs to check whether any of your sensitive data has been seen on common sources across the internet. We also want to keep an eye on all known breaches to ensure none of these impacts my service providers.

0.3 Requirements and Solution Functionality

1. Get a list of all known breaches the site is tracking and store this data in some form of local storage that will allow for easy retrieval/search across the data.
   (a) Solution leverages Python's Requests library for HTTP requests/responses.
   (b) Functionality is provided to request, parse and insert data into a RDBMS
   (c) Solution leverages Python's Sqlite3 library as an RDBMS
   (d) Functionality is provided to select desired data from database.
2. Ensure that any new or updated content, is also consumed by the solution on a daily basis and stored locally with no duplicates.
   (a) Functionality is provided to allow the user to request from the breaches API at a rate of their choice, in compliance with the API.
3. In the event that a breach is reported which matches one of the following domain names (acme.com, acme.net, acme.org), send all the metadata to a syslog server in an appropriate syslog format.
   (a) Functionality is provided to lookup breaches relevant to a user-defined list of domains.

   (b) Solution leverages Python's Logbook library for Logging purposes.

4. We want to know if our CEO's account philip.price@acme.com shows up in any data associated with a breach or is mentioned on pastebin sites. This should be checked on a daily basis, if a match is found this should also be sent to the syslog server in the same way as defined in #3.

   (a) Functionality is provided to allow the user to request and lookup a user-defined list of emails at a rate of their choice.

5. Bonus points... we expect in the next 3 months this list of email addresses to monitor will expand to 1000+ and thus want to ensure our solution scales appropriately. These daily checks should run for all email addresses in parallel. Demonstrate how this would be done with 5 email addresses (you can make the other 4 up).

   (a) Functionality is provided to allow the user to spawn K worker threads that will request from the API and lookup from the database, resulting in log entries for any relevant breaches that are encountered.

   (b) Solution leverages Python's Multiprocessing library to provide process parallelism.

   (c) In order for 5 emails to be checked in parallel, one must set K=5 and provide a list of 5 emails in emails.txt.

0.4 Solution Modules

0.4.1 Code Modules

The following items contain a description of their respective code modules and their functionality. Note that all code modules should exist under modules/.

1. Command_Line.py
   (a) This module provides the Command Line Interface by which the user can interface with the solution. It is the "parent" module that can call-and-return functionality from other modules.
   (b) A more detailed explanation of how the user would use this module can be found in the following section.
   (c) This module is used to satisfy requirements 2 and 4.
2. Breach_Database_Definition.py
   (a) This module provides the table schema for the breaches table.
   (b) It also provides functionality to initalize and de-initialize the table itself.
3. Breach_Handler.py
   (a) This module provides the functionality for requesting breach data from the HIBP API, parsing the data, and inserting it into the breaches table.
   (b) Additionally, it also provides functionality that is able to check domains against the latest breach data and report any relevant breaches via log.
   (c) Note that "latest breach data" means that the user must have requested data from the API in order to receive relevant breach information.
   (d) It also defines specific log handlers for the use cases relevant to breaches.
   (e) It is used to satisfy requirements 1 and 3.
4. Database_Connection.py
   (a) This module is a SQLite utility used to initialize a connection to the database. Loose coupling; high cohesion.
5. Email_Database_Definition.py

   (a) This module provides the table schema for the emails table.

   (b) It also provides functionality to initalize and de-initialize the table itself.

6. Email_Handler.py

   (a) This module provides the functionality for requesting email data from the HIBP API, parsing the data, and inserting it into the emails table.

   (b) Additionally, it also provides functionality that is able to check emails against the latest breach data and report any relevant breaches via log.

   (c) Note that "latest breach data" means that the user must have requested data from the API in order to receive relevant breach information.

   (d) It also defines specific log handlers for the use cases relevant to emails.

   (e) It is used to satisfy requirement 4.

7. Multiprocess_Email_Handler.py

   (a) This module provides the functionality to run various processes that will execute certain functions provided in Email_Handler.py in parallel.

   (b) It is used to satisfy requirement 5.

0.4.2 Non-Code Modules

The following files are not Python code files, but are nevertheless important during system operation.

1. inputs/domains.txt

   (a) This file contains all domains that will be requested and checked against the API and database, respectively.

2. inputs/emails.txt

   (a) This file contains all emails that will be requested and checked against the API and database, respectively.

3. logs/domains.log

   (a) This file contains all log information corresponding to the domains in input/domains.txt when a relevant breach is found.

4. logs/emails.log
    (a) This file contains all log information corresponding to the emails in input/domains.txt when a relevant breach is found.
5. requirements.txt
    (a) This file specifies all of the project dependencies and their versions– it is used by venv to ensure that the solution is extremely portable.
6. db/breaches.db
    (a) This file is the database object created and managed by the SQLite3 module.

0.5 Using the Solution with Command Line

1. Load breaches from API into Breaches.db
   (a) Loads the latest breach data from the HIBP API into the database.
2. Check domains against database (from inputs/domains.txt)
   (a) Will create logs for any breaches relevant to any of the domains in inputs/domains.txt
   (b) Logs will have a blurb show up on screen, for both NOTICES (no breach found) and WARNINGS (breach found).
   (c) WARNING Logs will file a complete report to the breaches log file.
3. Check emails against breaches database (from inputs/emails.txt)
   (a) Will create logs for any breaches relevant to any of the emails in inputs/domains.txt
   (b) Logs will have a blurb show up on screen, for both NOTICES (no breach found) and WARNINGS (breach found).
   (c) WARNING Logs will file a complete report to the breaches log file.
4. Get all known breaches at a user-defined rate and check domains against new data
   (a) Will create logs for any breaches relevant to any of the domains in inputs/domains.txt at a user-defined rate.
   (b) NOTE that time between requests must be >= 1.3 seconds, in compliance with the API.
   (c) NOTE that this is a blocking process. The log files can still be read and refreshed to receive a live feed of log entries.
5. Get all known breaches at a user-defined rate and check emails against new breaches
   (a) Will create logs for any breaches relevant to any of the emails in inputs/emails.txt at a user-defined rate.
   (b) NOTE that time between requests must be >= 1.3 seconds, in compliance with the API.
   (c) NOTE that this is a blocking process. The log files can still be read and refreshed to receive a live feed of log entries.

6. Cat Log Files
    (a) Will show the output of both log files.
7. Clean Databases
    (a) Will wipe and re-initialize both the breaches and emails tables.
8. Clean Breaches Log
    (a) Will wipe the breaches log file.
9. Clean Emails Log
    (a) Will wipe the emails log file.
10. Any other key press will result in a program exit().

0.6 Solution Architecture

The solution uses a call-and-return architecture, giving complete control to the Command Line module. The architecture embraces the philosophy of "loose coupling, high cohesion" by providing standalone functions in each of the handler modules. However, the architecture strives to be user-friendly by giving the end-user a CLI that provides wrappers for these functions.

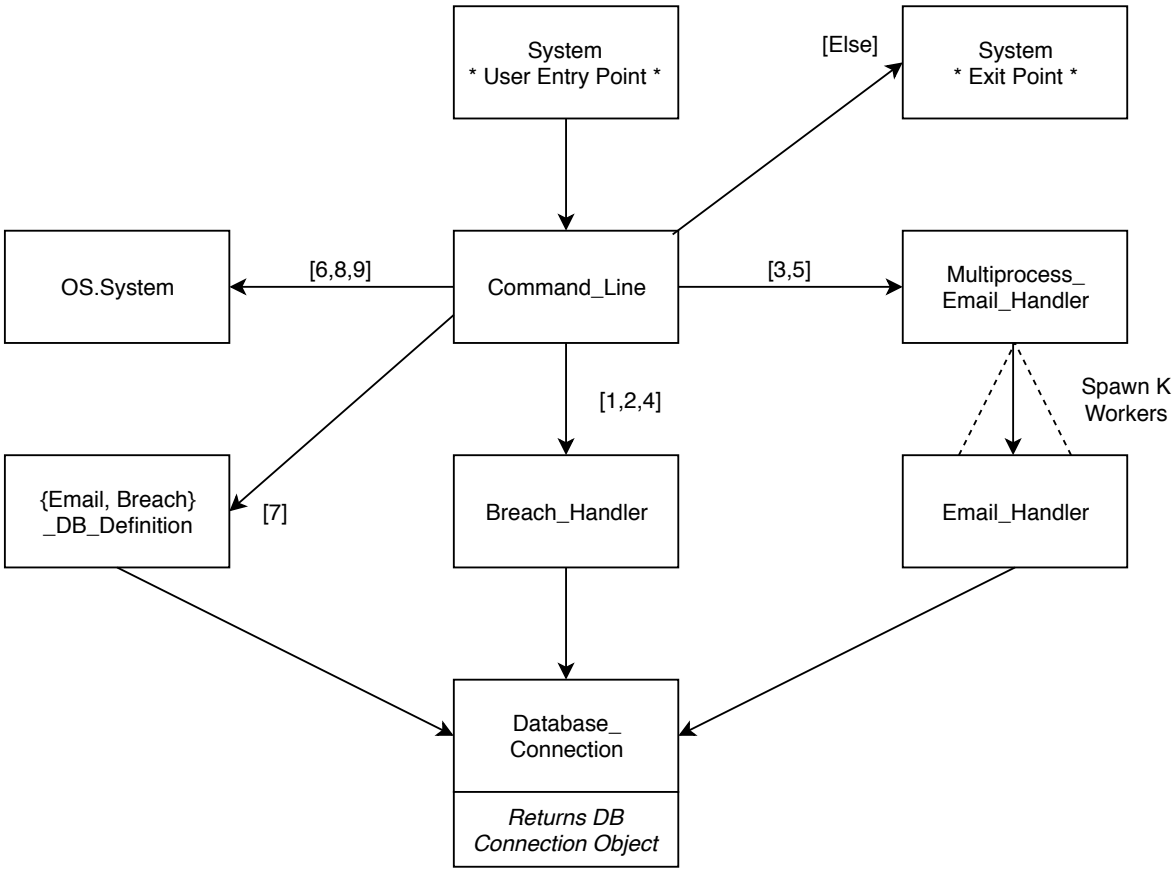

Figure 1: Have You Been Pwned Architecture. Refer to the "Using the Solution with Command Line" section for the enumerations of the different actions.

0.7 Technologies Used

1. Requests
   (a) Python's requests library allows for an interface to HTTP Methods.
   (b) It allows the solution to request the API for both breaches and email data.
   (c) It allows the solution to easily specify the user agent in accordance with the API.
   (d) It provides a text object compatible with Python's JSON library, used for processing and parsing the data.
2. JSON
   (a) Python's JSON library allows for an interface that can take raw response text and convert it into a native Python dictionary.
3. SQLite3
   (a) Python's SQLite3 library provides an interface for interacting with the SQLite RDBMS.
   (b) SQLite is a full-featured SQL implementation with great performance, low overhead, a simple interface, 100% branch test coverage, and an open-source codebase.
4. Multiprocessing
   (a) Python's Multiprocessing library provides the ability for a parent process to spawn worker processes with a target function and collect their results.
   (b) It also natively provides thread and process-safe data structures to facilitate inter process communication.
5. Logbook
   (a) The Logbook library, written for Python, is a replacement for the standard Python Logging library.
   (b) It is designed to be low-overhead for simple applications, but can scale effectively for larger projects.
   (c) It provides a native handler for interacting with Unix's Syslog.

(d) This library was chosen due to its simple interface and low setup overhead.

6. Venv

(a) The Venv library is used to manage Python dependencies and ensure solution portability.

0.8 Design Rationale

0.8.1 Architecture

The call-and-return architecture that was chosen was mainly rationalized by the desire for parallel processes executing tasks on shared resources.

In the case where multiple emails are requested against the API in parallel, the use of native Multiprocessing data structures combined with the SQLite database ensures that all race conditions are handled properly in order to maximize the gains from parallel computing.

An object-oriented architecture was considered, but due to a lack of experience with/opportunities for inheritance, the use of classes was restricted. The files themselves act as pseudo-objects with both dependent and standalone functionality. Some potential areas for inheritance were:

1. In the database connection object– instead this was done with a developer-defined module that all database operations referenced.

2. In the Logbook handlers, specifically for files and status codes. Note that the loggers themselves are unique to the type of request (breaches/emails) so the stream handlers would likely not benefit greatly from inheritance.

0.8.2 Logbook

Logbook is perhaps the most non-standard library chosen and used in the solution. As per the recommendations of various online sources, including StackOverflow, Reddit, and more, the developer chose Logbook instead of the standard Logging library. The developer had not had any experience with any logging libraries in the past.

The key benefits of Logging were proposed to be two-fold:

1. A simple interface with (relatively) little setup required

2. Native handlers provided for interfacing with Syslog

However, as the project went on, the developer was unable to use the Logger.Processor() function to pass arguments to target functions in order to add attributes to log records– this resulted in the use of dynamic scoping for function definition in order to make sure that the attributes could be accessed and aggregated into log records.

0.8.3 Syslog

Although the requirements state that the solution should "... send all the metadata to a syslog server in an appropriate syslog format", the developer chose to define their own logging output format based on the information returned by the HIBP API.

Although the developer later realized that Syslog is one of two RFCs, the developer opted to continue using their own custom output format. This is two-fold: for one, the log formats were already set up. Additionally, development was proceeding on macOs and the developer was unsure if they could support Syslog without switching to a Linux environment.

Instead, the developer created three main handlers. A key benefit is that all of these handlers are OS-independent, as claimed by the Logbook documentation.

1. Status Code Handler
    (a) This handler logged all status codes returned from requests to the HIBP API to stdout.
2. Stream Handler
    (a) This handler directs all logs (of any level) to the stdout.
    (b) The rationale here is that the user, when interacting with the solution, wants to see progress updates. So, even when a domain/email is found to not have any relevant data, the stream is notified.
    (c) However, the outputs on the stream are a concise blurb in order to not clutter up the stdout excessively.
    (d) More information can be found in the corresponding log files.
3. File Handler
    (a) This handler directs logs (of level WARNING and above) to the corresponding log file (breaches.log or emails.log).
    (b) This handler's purpose is to output detailed, well-formatted logs that completely capture all the meaning when a relevant breach has been found.

0.8.4 Multiprocessing

Python's Multiprocessing was chosen over Threading due to the advantages in performance that multiple processes can provide.

That being said, there is an argument for the use of the Threading library, since threads would still see performance gains when used in I/O bound applications.

However, since processes do not have to share memory space, they can operate independently on a machine's cores to achieve very high-performance parallelism. The main drawback of Multiprocessing is slightly more overhead to perform inter process communication– this obstacle was handled using the native constructs provided in the module.