Pastebin Buddy- A Python 3 Interaction With Pastebin API

Dustin Chung

Project: Oct 2017 — Writeup: July 2018

Overview

Pastebin Buddy (PB) is a script that allows users to interact with the Pastebin API using Python 3. The base functionality involves is the ability to scrape and post pastes. The script works even without a Pastebin account, although having one greatly increases the scope of things that can be done (e.g. private pastes). With this script, one can do anything they wouldn normally be able to do on Pastebin, but programatically. This facilitates automation, and potentially, very simple server functionality.

1 Introduction

Pastebin is an online website that allows users to upload text to be stored and retrieved later on without the need for an account. Plain text is uploaded as a ”paste”, after which a link is given which allows the paste to be found again. The platform can upload text about anything: from simple paragraphs of literature that need to be kept for future use, to snippets of code that need to be shared amongst collaborators. The latter is especially convenient, as Pastebin supports syntax highlighting for a range of popular programming languages. Users can also set an ”expiry date” for their pastes: a duration after which their paste is no longer available to be seen by anybody (presumed deleted).

Although basic functionality is available to anybody (Guests), more functionality is provided to users that have accounts (Free), and even more options are open to paid accout users (Pro). For example, Free accouts have a max paste size of 500KB, whilst Pro accounts can have pastes up to sizes of 10MB. Free accounts have their Unlisted pastes (that need a link to be seen) and Private pastes (that can only be seen by the creator) capped at 10 and 2 respectively. Pro users have no cap.

1.1 Pro accounts and scraping

The most important difference between the account types with regards to scraping is that Pro accounts cannot be blacklisted for scraping, whilst scraping with a Free account runs the risk of getting IP blocked. The consequence for this is of course that PB works *best* when used with a Pro account.

2 Practical Uses

Since PB is simply a way to interact with the Pastebin API, its uses are obviously limited to what can be done with Pastebin itself Its uses described below will also apply to other similar scripts or programs that interact with the API. The purpose of PB is to aid automation and make uploading text more convenient for those who use Pastebin often. The following subsections describe one real-life implemented use case and potential uses for PB.

2.1 Actual use case: live data feed

PB has been used previously to create a live data feed accessible from both computers, or mobile phones (via the Pastebin app). A Python script was used to scrape statistics of certain users from Instagram. This data was then analyzed to try and determine whether the users in question were gaining their follower base thorough illegitimate means. The scraping was performed live, with data being gathered once a day.

Due to the nature of the one-day-one-scrape approach to this project, the scraper was left to run for long periods of time. In order to check if the scraper was still running, and to get an idea of the "landscape" of the data being collected, PB was used to upload reports and statistics. These could then be inspected at any time through the Pastebin account on a computer or a mobile phone.

2.2 Potential uses: basic server

PB allows a computer to both post and retrieve text data to and from a single source (the Pastebin account). This can be exploited, making Pastebin a "middleman server" to allow for communication between computers or devices over the internet without the need for any complicated network setups.

The most simple "basic server" use case for PB would be blinking an LED over the internet by giving an instruction via a paste. One could quickly come up with their own syntax of paste that could be read and parsed by a python script into instructions. For example, to make the LED lightup, the instruction paste might say:

```
#PB setup code etc etc
...
...
content = ".instruction1.instruction2"

p.paste(content, title="Turn␣on␣LED", privacy=2)
```

Then, imagine one has an LED connected to a Raspberry Pi. On the RPi side, the paste can be retrieved, parsed, and the instructions executed (note, any strange convention used in the following script is purely aribitrary and simply to illustrate the possible functionality of PB):

```
#PB setup code etc etc
```

```
...
...
def parse_paste(pastes):
  for paste in pastes:
    paste_key = paste["key"]
    raw_data = p.raw(p.uk, paste_key)[2]
    instructions = raw_data.split(".")

    for instruction in instructions:
      if instruction=="instruction1":
        led_on()
      if instruction=="instruction2":
        led_off()

while True:
  pastes = list_pastes(p.uk)[1:]
  time.sleep(sleep_time)
```

Of course, there are more efficient ways to perform this task, or write the code. The objective here was to explore a possible use case. An example such as the one above can be further developed by including parsing rules for paste titles to categorize the type of instructions that should be done, or to distinguish instruction pastes from non-instruction pastes.

It goes without saying that a PB server like the one above should not be used for time-sensitive tasks, as there will be delays in both posting to, and reading from Pastebin.

3 Conclusion

PB is a useful way to programmatically interact with the Pastebin API, making it more convenient to use from the perspective of somebody who would not want to click around the website for ages just to upload some simple text.