

Open source intelligence techniques & commentary



Building a Keyword Monitoring Pipeline with Python, Pastebin and Searx

Written by **Justin**, April 18th, 2017

Having an early warning system is an incredibly useful tool in the OSINT world. Being able to monitor search engines and other sites for keywords, IP addresses, document names, or email addresses is extremely useful. This can tell you if an adversary, competitor or a friendly ally is talking about you online. In this blog post we are going to setup a keyword monitoring pipeline

so that we can monitor both popular search engines and Pastebin for keywords, leaked credentials, or anything else we are interested in.

The pipeline will be designed to alert you whenever one of those keywords is discovered or if you are seeing movement for a keyword on a particular search engine.

We will build all of this out into a virtual machine so that you can self-host everything on your local box. If you are comfortable managing your own server, or you just want to have your monitoring run 24×7 without having to keep your virtual machine running feel free to get some free Digital Ocean time by using my referral link [here](#).

Let's get started!

Setting up Searx

Searx is this really cool project that provides you a self-hosted interface to search multiple search engines at once. This is called a meta-search engine and was all the rave in the late 90s. Yeah, I am that old.

One thing that Searx also provides is the ability to query it and receive the results back in JSON. This gives us the ability to write Python code to talk to it and to process the results without having to use web scraping techniques or paying for an expensive API key.

Setting up Searx

Watch the video below to setup Searx if you have never setup a VirtualBox VM or done any work in Ubuntu before. It will help you walk through everything from start to finish.



Now that you are logged in to your server you can run the following commands. These steps have been adapted from the Searx setup guide [here](#). Feel free to copy and paste these steps.

```
1 sudo apt-get update
2
3 sudo apt-get install virtualbox-guest-dkms
4
5 sudo apt-get install git build-essential libxslt-dev python-dev python-virtualenv python-pybabel zlib1g-dev libffi-dev libssl-dev
6
7 cd /usr/local
8
9 git clone https://github.com/asciimoo/searx.git
10
11 cd searx
12
13 ./manage.sh update_packages
14
15 sed -i -e "s/ultrasecretkey/`openssl rand -hex 16`/g" searx/settings.yml
```

Now we just need to edit the settings.yml file so that we can access the Searx search interface across the Internet. Do the following:

```
1 nano -w searx/settings.yml
```

Now find the **bind_address** entry and change it to **127.0.0.1** which will cause Searx to listen only on the machine it is installed on.

It should now look like this:

bind_address : "127.0.0.1"

Once you have edited that setting hit CTRL+O and then CTRL+X to exit nano.

Now you are ready to test it out! Run the Searx web application like so:

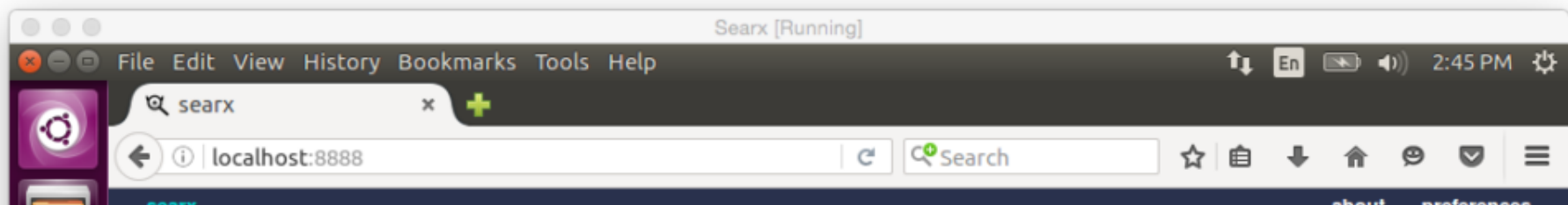
```
1 cd ~/searx/searx/searx
2
3 python webapp.py
```

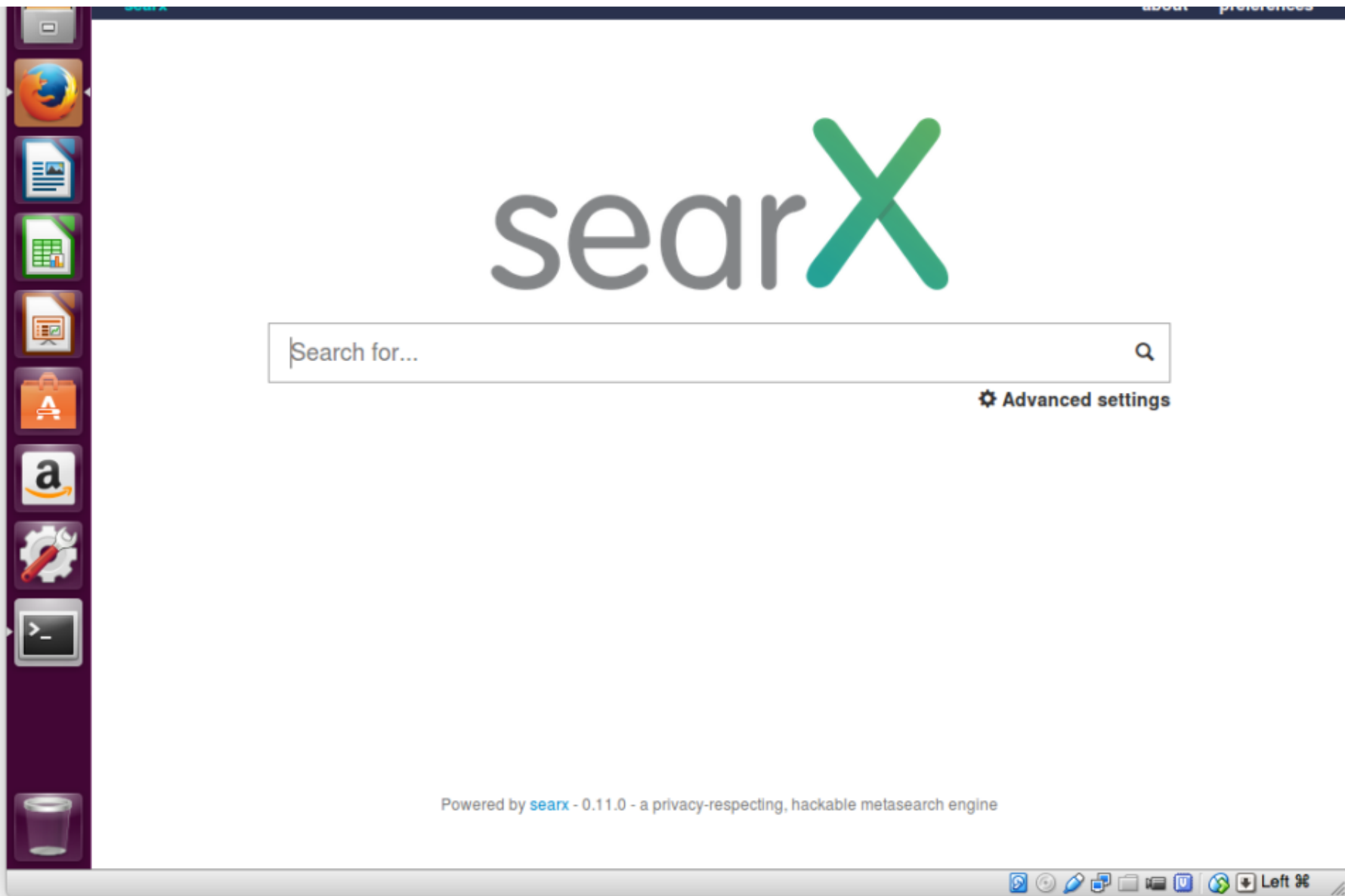
Testing Your Searx Installation

If you now use your browser to browse to:

http://127.0.0.1:8888

When you browse to it you should see a web page appear as shown below, and you can test running some searches to make sure it is all working.





Awesome. Now let's move on to getting setup with a Pastebin Pro account.

Pastebin Pro

Pastebin is the go-to site for data dumps after a breach, ominous messages from Anonymous or for bits of code that people have posted. While you can certainly attempt to scrape data from Pastebin pages, you will find that they will rate limit you by your IP address or potentially ban you.


The best thing to do is to spend the \$29 for a lifetime Pastebin Pro account which gives you the ability to whitelist your IP address and then using their API to pull down pastes to your heart's content. If you can afford to do it, go spend the money now and come back to this blog post.

Sign up [here](#).

Once you have done that you will need to head to [here](#) to whitelist your IP address. If you are using Digital Ocean or another provider make sure to put the public IP address of your server. Otherwise you can search Google for: what is my IP and then enter that address into the text field:

pastebin.com/api_scraping_faq

Bookmarks

 **PASTEBIN** [+ new paste](#) [trends](#) [API](#) [tools](#) [faq](#)

Scraping API

This is the Pastebin scraping API documentation page. Here you can find all the information you need to get started with our scraping API to [contact us](#).

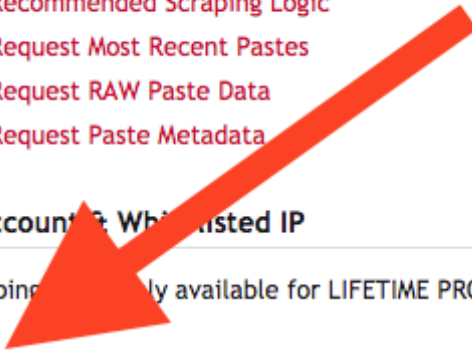
1. [Your Whitelisted IP](#)
2. [Request Limits](#)
3. [Recommended Scraping Logic](#)
4. [Request Most Recent Pastes](#)
5. [Request RAW Paste Data](#)
6. [Request Paste Metadata](#)

Your Account & Whitelisted IP

Our scraping API is only available for LIFETIME PRO members, and only for those who have their IP whitelisted!

Your account status is: **PRO LIFETIME**

Your whitelisted IP is: [REDACTED]



You are now setup to start writing code for the Pastebin Scraping API (documentation [here](#)). Let's get started!

Building A Keyword Monitor

Now let's start writing some code that will continually monitor search results from Searx and Pastebin. Then it will connect to your GMail account and send you an alert email when there are hits. If you don't have a GMail account I recommend you get one, or modify the code to connect to your own mailserver.

Crack open a new Python script (full source code [here](#)), call it *keywordmonitor.py* and start punching in the following code:

```
1 import os
2 import requests
3 import smtplib
4 import time
5
6 from email.mime.text import MIMEText
7
8 alert_email_account = "YOUREMAIL@GMAIL.COM"
9 alert_email_password = "YOURPASSWORD"
10 searx_url = "http://localhost:8888/"
11 max_sleep_time = 120
12
13 # read in our list of keywords
14 with open("keywords.txt", "r") as fd:
15     file_contents = fd.read()
16     keywords = file_contents.splitlines()
17
18 if not os.path.exists("keywords"):
19     os.mkdir("keywords")
```

This is pretty straightforward. We are importing the necessary libraries, and then setting up some variables for our email account and for the path to Searx. We then pull in the stored keywords we want to track and create a directory where we can store all URLs so that we don't have duplicate results.

Now let's build our email alert function. Keep adding code to your script:

```
21 #
22 # Send email to you!
23 #
24 def send_alert(alert_email):
25
26     email_body = "The following are keyword hits that were just found:\r\n\r\n"
27
```

```

28     # walk through the searx results
29     if alert_email.has_key("searx"):
30
31         for keyword in alert_email['searx']:
32
33             email_body += "\r\nKeyword: %s\r\n\r\n" % keyword
34
35             for keyword_hit in alert_email['searx'][keyword]:
36
37                 email_body += "%s\r\n" % keyword_hit
38
39     # walk through pastebin results
40     if alert_email.has_key("pastebin"):
41
42         for paste_id in alert_email['pastebin']:
43
44             email_body += "\r\nPastebin Link: https://pastebin.com/%s\r\n" % paste_id
45             email_body += "Keywords:%s\r\n" % ", ".join(alert_email['pastebin'][paste_id][0])
46             email_body += "Paste Body:\r\n%s\r\n\r\n" % alert_email['pastebin'][paste_id][1]
47
48
49     # build the email message
50     msg = MIMEText(email_body)
51     msg['Subject'] = "AutomatingOSINT.com Keyword Alert"
52     msg['From'] = alert_email_account
53     msg['To'] = alert_email_account
54
55     server = smtplib.SMTP("smtp.gmail.com", 587)
56
57     server.ehlo()
58     server.starttls()
59     server.login(alert_email_account, alert_email_password)
60     server.sendmail(alert_email_account, alert_email_account, msg.as_string())
61     server.quit()
62
63     print "[!] Alert email sent!"
64
65     return

```

Let's break this code down a bit:

- **Line 24:** we define our **send_alert** function and it takes a single parameter **alert_email**. This parameter is a dictionary that we build from our keyword hits.

- **Lines 29-37:** here we are testing if there are results from Searx (29) and if so we walk through each of the keywords that had hits (31) and then add each hit for that keyword (35-37).
- **Lines 40-46:** we test if there are results from Pastebin (40) and if we have some we walk through each result (42) and print out the information from Pastebin including the original text of the paste. We do this because people tend to delete pastes, and this way we'll always have a record of it.
- **Lines 50-54:** we build our email object and set the Subject, From and To fields of the email.
- **Line 55:** here we are initializing our connect to the GMail servers.
- **Lines 57-61:** these lines of code deal with connecting to the mail server (57,58), logging in to the server (59) and then sending the email off (60).

Now we are going to build another support function that will check URLs discovered from Searx against our stored list of URLs. This will determine whether a hit for a keyword is a new URL or something that we have encountered already.

Let's add this code now:

```
66 #
67 # Check if the URL is new.
68 #
69 def check_urls(keyword,urls):
70
71     new_urls = []
72
73     if os.path.exists("keywords/%s.txt" % keyword):
74
75         with open("keywords/%s.txt" % keyword,"r") as fd:
76
77             stored_urls = fd.read().splitlines()
78
79             for url in urls:
80
81                 if url not in stored_urls:
82
83                     print "[*] New URL for %s discovered: %s" % (keyword,url)
84
85                     new_urls.append(url)
86
87     else:
88
89         new_urls = urls
90
```

```

91     # now store the new urls back in the file
92     with open("keywords/%s.txt" % keyword,"ab") as fd:
93
94         for url in new_urls:
95             fd.write("%s\r\n" % url)
96
97
98     return new_urls

```

- **Line 70:** we define our function to take in the **keyword** that the matches correspond to and the list of **urls** that we are going to check.
- **Lines 74-78:** we check to see if we have a file with results for the current keyword (74), and if not we open the file (76) and read each line into a list (78).
- **Lines 80-86:** we walk through our list of results (80) and if we don't already have the URL stored (82) we add it to our **new_urls** list (86) so that we can store it later.
- **Line 90:** if we don't have a record of any hits for the current keyword, we take all of the results and put them in our **new_urls** list.
- **Lines 93-99:** we crack open the keywords file that has all of the results stored (93) and then add each of the new URLs that we have discovered (95,96). We then return the list of newly discovered URLs (99) so that we can include them in our email alert.

Alright so we have the plumbing in place to deal with alerting and managing new results from our Searx results. Now let's start putting the functions in place to actually do the searching piece. First we'll start with Searx, so start adding the following code:

```

101 #
102 # Poll Searx instance for keyword.
103 #
104 def check_searx(keyword):
105
106     hits = []
107
108     # build parameter dictionary
109     params = {}
110     params['q'] = keyword
111     params['categories'] = 'general'
112     params['time_range'] = 'day' #day,week,month or year will work
113     params['format'] = 'json'
114
115     print "[*] Querying Searx for: %s" % keyword
116
117     # send the request off to searx
118     try:

```

```

119     response = requests.get(searx_url,params=params)
120
121     results = response.json()
122
123     except:
124         return hits
125
126     # if we have results we want to check them against our stored URLs
127     if len(results['results']):
128
129         urls = []
130
131         for result in results['results']:
132
133             if result['url'] not in urls:
134
135                 urls.append(result['url'])
136
137         hits = check_urls(keyword,urls)
138
139     return hits

```

Let's examine this function a little more closely:

- **Line 105:** we define our **check_searx** function that accepts a single **keyword** parameter.
- **Lines 110-114:** we setup the parameters for our query to our Searx instance. Note that you can set the **time_range** value on line 113 to a time period of your choosing.
- **Lines 119-122:** we send off our request (120) passing in our parameters dictionary we have created. We then parse the JSON result from Searx (122). If either the request or the JSON parsing fails, we return an empty list (125).
- **Lines 128-140:** if we have a list of good results (128) we walk through each of the results (132) and then add each unique result to our **urls** variable (136). Once we have walked through all of the results we use our previously developed **check_urls** function (138) to determine which of the URLs are new hits. Lastly we return the hits (140) so we can send them out in our alert email.

Alright, now that the Searx portion of the script is finished we can move on to working with Pastebin which has a different bit of logic involved.

```

142 #
143 # Check Pastebin for keyword list.
144 #

```

```

145 def check_pastebin(keywords):
146
147     new_ids    = []
148     paste_hits = {}
149
150     # poll the Pastebin API
151     try:
152         response = requests.get("http://pastebin.com/api_scraping.php?limit=500")
153     except:
154         return paste_hits
155
156     # parse the JSON
157     result = response.json()
158
159     # load up our list of stored paste ID's and only check the new ones
160     if os.path.exists("pastebin_ids.txt"):
161         with open("pastebin_ids.txt", "rb") as fd:
162             pastebin_ids = fd.read().splitlines()
163     else:
164         pastebin_ids = []
165
166     for paste in result:
167
168         if paste['key'] not in pastebin_ids:
169
170             new_ids.append(paste['key'])
171
172             # this is a new paste so send a secondary request to retrieve
173             # it and then check it for our keywords
174             paste_response = requests.get(paste['scrape_url'])
175             paste_body_lower = paste_response.content.lower()
176
177             keyword_hits = []
178
179             for keyword in keywords:
180
181                 if keyword.lower() in paste_body_lower:
182                     keyword_hits.append(keyword)
183
184             if len(keyword_hits):
185                 paste_hits[paste['key']] = (keyword_hits, paste_response.content)
186
187                 print "[*] Hit on Pastebin for %s: %s" % (str(keyword_hits), paste['full_url'])
188
189     # store the newly checked IDs

```

```

190     with open("pastebin_ids.txt","ab") as fd:
191
192         for pastebin_id in new_ids:
193
194             fd.write("%s\r\n" % pastebin_id)
195
196     print "[*] Successfully processed %d Pastebin posts." % len(new_ids)
197
198     return paste_hits

```

Whew that's a lot of code! Let's break this down a little:

- **Line 145:** we define our **check_pastebin** function that accepts a **keywords** parameter that is a list of keywords that we will check against each Paste we discover.
- **Line 147:** when users post content (Pastes) to Pastebin, each Paste gets its own unique identifier. We setup a list to track any new Paste IDs that we have encountered (147) so that we can track them. We don't want to continually check old Pastes that we have already looked at.
- **Lines 151-154:** now we send off our request to the Pastebin scraping API. If you have not properly put your IP address into their whitelist, this will fail and the function will return an empty list of hits (154).
- **Line 157:** we parse the JSON response so that we can walk through the results.
- **Lines 160-164:** as mentioned previously, we want to be able to track all of the Pastebin IDs that we have previously checked. We check to see if this file exists (160) and if it does we open it up (161) and then read all of the Pastebin IDs into a list (162). If we don't have any previously tracked IDs we create a new empty list to hold all of the IDs we are about to process (164).
- **Lines 166-170:** we begin walking through the results (166) and if the Paste ID has not been seen yet (168) we add it to our list of new Paste IDs (170).
- **Lines 174-175:** we send a secondary request (174) to retrieve the full body of the Paste and then normalize it to all lowercase (175).
- **Lines 177-182:** we setup a list to hold our keyword hits (177), and then walk through the list of keywords (172) and check to see if they are found in the Paste body (181). If a keyword matches we add it to our **keyword_hits** list (182).
- **Lines 184-187:** if there were any hits (184) for the current paste we add the hits and the full content of the Paste (185) to our **paste_hits** dictionary, which is keyed by the Paste ID.
- **Lines 190-198:** now that we have finished checking the keywords we open our log file of Paste IDs (190) and then write out each of the Paste IDs (192,194). We then return the hits so that we can use them in our alert emails (198).

If you have survived thus far, don't worry we are nearly finished! We are going to create a wrapper function that will handle calling our other functions and handing back the results for alerting. Let's do this now:

```

201 def check_keywords(keywords):
202
203     alert_email = {}
204
205     time_start = time.time()
206
207     # use the list of keywords and check each against searx
208     for keyword in keywords:
209
210         # query searx for the keyword
211         result = check_searx(keyword)
212
213         if len(result):
214
215             if not alert_email.has_key("searx"):
216                 alert_email['searx'] = {}
217
218                 alert_email['searx'][keyword] = result
219
220     # now we check Pastebin for new pastes
221     result = check_pastebin(keywords)
222
223     if len(result.keys()):
224
225         # we have results so include it in the alert email
226         alert_email['pastebin'] = result
227
228
229     time_end = time.time()
230     total_time = time_end - time_start
231
232     # if we complete the above inside of the max_sleep_time setting
233     # we sleep. This is for Pastebin rate limiting
234     if total_time < max_sleep_time:
235
236         sleep_time = max_sleep_time - total_time
237
238         print "[*] Sleeping for %d s" % sleep_time
239
240         time.sleep(sleep_time)
241
242     return alert_email

```

This is pretty straightforward code, but let's break the major points down:

- **Line 203:** we setup our **alert_email** dictionary that our **send_alert** function will use to send email alerts.
- **Line 205:** we note the current time so that we can measure how long it takes for our keyword searches to occur. This is important later in the function.
- **Lines 207-218:** this little block is responsible for walking through our list of keywords (207) and then checking our Searx instance (211) for hits. If there are any (213) we add them to our **alert_email** dictionary (218).
- **Lines 221-226:** we check Pastebin for any results (221) and if we have results (223) we add them to our **alert_email** dictionary (226).
- **Lines 229-240:** we take another timestamp (229) and then subtract the **time_start** we collected above (230) to determine the total execution time. If we have completed our searches in less than the **max_sleep_time** (234), we determine how long we need to sleep for (236) and then sleep (240). This is so that we are always staying within the suggested rate limiting times that Pastebin recommends. My recommendation is to leave the 120 seconds setting in place.

Alright! The last thing we need to do is write a little loop that will handle calling our **check_keywords** and **send_alert** functions repeatedly. Let's add this little bit of code now:

```
245 # execute your search once first to populate results
246 check_keywords(keywords)
247
248 # now perform the main loop
249 while True:
250
251     alert_email = check_keywords(keywords)
252
253     if len(alert_email.keys()):
254
255         # if we have alerts send them out
256         send_alert(alert_email)
```

Pretty straightforward. We just run the **check_keywords** function once to set a baseline (this won't send any alerts), and then run an infinite loop to keep checking keywords, and sending alerts if there are results. It will do this over and over again until you stop the script from executing.

Let It Rip

Now it's time to shine! Open a new file in the same directory as your *keywordmonitor.py* script and name it *keywords.txt*. Add the keywords you want to monitor, one per line to the file. In my case it would look like:

```
jms_dot_py
justin@automatingosint.com
```

Save this file and run the keyword monitor from within Wing (Debug Menu -> Set Current as Main Debug File and then click the Play button) or from the command line. Make sure you have your keywords loaded into a keywords.txt file that lives in the same directory as your Python script.

You will start to see results showing up in your terminal like so:

```
searx@searx:~$ python keywordmonitor.py
[*] Querying Searx for: jms_dot_py
[*] New URL for jms_dot_py discovered: https://twitter.com/jms_dot_py/status/768559761818324992
[*] New URL for jms_dot_py discovered: https://twitter.com/jms_dot_py/status/826116048541859841
[*] New URL for jms_dot_py discovered: https://twitter.com/jms_dot_py/status/791704263999295488
[*] Querying Searx for: automatingosint.com
[*] Querying Searx for: justin@automatingosint.com
[*] New URL for justin@automatingosint.com discovered: https://computercrimeinfo.com/wp/?cat=9&paged=2
[*] Querying Searx for: hunchly
[*] New URL for hunchly discovered:
https://www.reddit.com/r/craftofintelligence/comments/5otcmz/bulk_extracting_exif_metadata_with_hunchly_and/
[*] Successfully processed 48 Pastebin posts.
[*] Sleeping for 97 s
[!] Alert email sent!
```

And there you have it, you would have an email in your inbox that has the keywords and associated links. Leaving your virtual machine running or setting up Searx on a Digital Ocean droplet (get 2 free months [here](#)) will give you 24/7 monitoring of those

keywords.

As a quick test, head to Pastebin.com and enter a paste that has one of your keywords in it. You should receive an email in a few minutes showing you the hit. How cool is that?



Need to Learn
Python First?

START NOW \$49.99

Online Python Course



Want more Python and OSINT?

Join my mailing list now and don't miss out!

SUBSCRIBE



Learn Everything
You Need to Automate
Your OSINT Tasks.

START NOW

Online OSINT Course

RECENT POSTS

Follow the Bitcoin With Python, BlockExplorer and Webhose.io
New! Automatically Discover Website Connections Through Tracking Codes
Building a Keyword Monitoring Pipeline with Python, Pastebin and Searx
Vacuuming Image Metadata from The Wayback Machine
Dark Web OSINT Part Four: Using Scikit-Learn to Find Hidden Service Clones

RECENT COMMENTS

Justin on Automatically Discover Website Connections Through Tracking Codes
Justin on Gaming Meets OSINT: Using Python to Help Solve Her Story
OSINTDude on Automatically Discover Website Connections Through Tracking Codes
shinrahunter on Gaming Meets OSINT: Using Python to Help Solve Her Story
Harvey on Automatically Discover Website Connections Through Tracking Codes

CATEGORIES

API (11)
Bitcoin (1)
Dark Web (5)
Facebook (1)
Forensics (1)
Geolocation (7)
Gephi (4)
Google Maps API (1)

Imagga (1)
Imagga API (1)
OpenCorporates API (1)
OSINT (28)
Pastebin API (1)
Photography (6)
Python (24)
Shodan (1)
Spyonweb API (1)
Text Analysis (10)
TinEye API (2)
Twitter API (1)
Uncategorized (3)
Vimeo API (1)
Wayback Machine (1)
Web Scraping (3)
Webhose.io API (1)
Wikimapia API (1)
YouTube API (1)