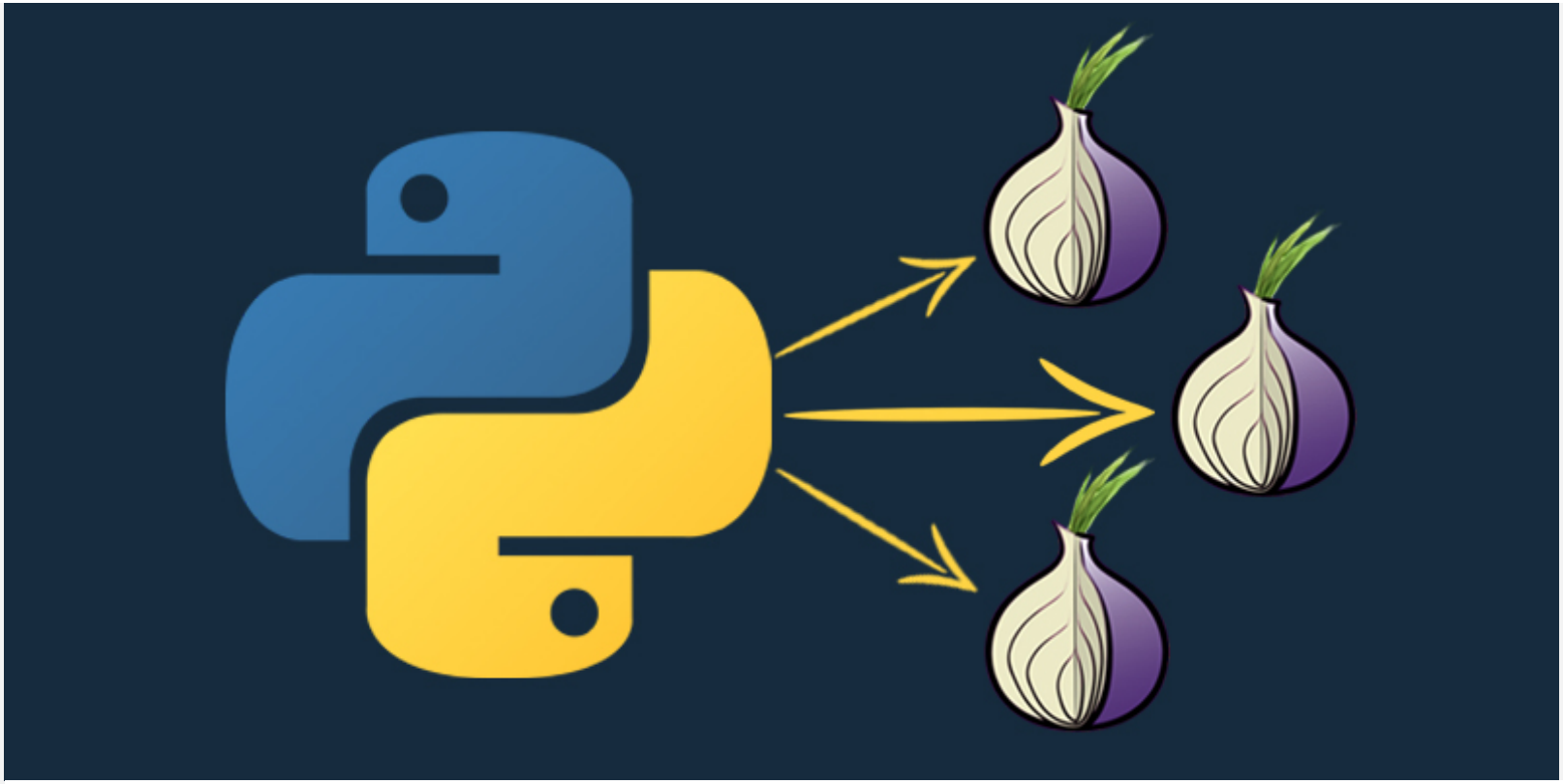
Open source intelligence techniques & commentary



Dark Web OSINT Part Four: Using Scikit-Learn to Find Hidden Service Clones

Written by **Justin,** September 30th, 2016

Welcome back to the fourth and final instalment in this series. If you haven't read part one, two or three definitely feel free to go and do so. This will be much shorter than the others

The original inspiration for this post was from a @krypti3a blog post called: Counterfeiting on the Darknet: USD4U. If you aren't already following his blog you should, there's a lot of interesting stuff there. He details a counterfeiting hidden service at usd4you5sa237ulk.onion that seems to display counterfeit currency. The first thing you see however, is the big warning at the top of the page warning against clone sites and that what you are viewing is the ***real*** site.

This got me wondering how we could leverage our OnionScan results to try to find cloned hidden services so that we could examine the differences between them or just use them as a jumping off point for an investigation. A subsequent conversation with Scot, where he also suggested that finding perfect mirrors would be a good thing as well since that could indicate a site backing itself up or preparing to move to a new hidden service address. The counterfeiting post gives us a great opportunity to try this out. Let's get started.

Getting Scikit-Learn Installed

Scikit-Learn is a machine learning library for Python that has all kinds of cool bits for data analysis and high powered machine learning tasks. Full disclosure: I know precisely nothing about machine learning. Now the cool thing is that there are a number of supporting classes and functions in scikit-learn that can be used for other tasks, such as what we are going to be doing.

This all being said, the installation of scikit-learn can be a bit of a pain but just follow these steps carefully.

Windows

We need to download and install **scipy**, **numpy** and then **scikit-learn**. Each of them has a binary download called a "wheel" file that we can grab from the links below.

How you choose the right download is like so, using the following example link:

scipy-0.18.1-cp27-cp27m-win32.whl

- "cp27" indicates that it is for Python 2.7 (this is what I use).
- "win32" indicates that it is for 32-bit Windows.

Now download the appropriate wheel files for each of the required libraries:

http://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy

http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy

http://www.lfd.uci.edu/~gohlke/pythonlibs/#scikit-learn

Once you have them downloaded you can install them using pip. If you have never used pip before you should check out my Python course here. For example do the follow for **numpy**:

pip install numpy-1.11.1+mkl-cp27-cp27m-win32

Mac OSX / Linux

In my experience, installing the prerequisites from pip works perfectly fine but your mileage may vary:

sudo pip install scipy

sudo pip install numpy

sudo pip install scitkit-learn

Once you have the prerequisites installed we can move on to writing some code!

Coding it Up

Before we start pounding out the code, I started this whole research question out by asking Google: “similarity between two text documents”. It landed me on a great StackOverflow.com thread here that explained how to do this in scikit-learn. I do not know a lick of math or machine learning but I am always up for experimenting with snippets of code that much smarter people post and I have verified that this technique works great for finding cloned hidden services.

Let's get started by creating a new Python script called ***clone_finder.py*** and start entering the following code:

```
1  import argparse
2  import glob
3  import json
4  import os
5  import sys
6
7  from sklearn.feature_extraction.text import TfidfVectorizer
8
9  ap = argparse.ArgumentParser()
10 ap.add_argument("-s","--hidden-service",    required=True,help="The hidden service .onion address you are interested in.")
11
12 args = vars(ap.parse_args())
13
14 base_hidden_service = args['hidden_service']
```

We are just setting up our required imports and adding a commandline argument parser. Nothing too fancy quite yet! Let's add some more code.

```
16 # feel free to mess with the score to test the results
17 detect_score        = 0.9
18 path_to_results     = "/tmp/onionscan_results"
19
20 file_list    = glob.glob("%s/*.json" % path_to_results)
21
22 index_pages      = []
23 hidden_services  = []
24
25 if not os.path.exists("%s/%s.json" % (path_to_results,base_hidden_service)):
26     print "[!] Your desired hidden service %s is not found. Go scan it!" % base_hidden_service
27     sys.exit(0)
28 else:
29     print "[*] Target hidden service %s found. Loading data now." % base_hidden_service
30
```

Let's break this code down a little bit:

- **Line 17:** the **detect_score** variable will basically be our sensitivity setting. The higher the score the less tolerant of changes between hidden services, and the lower the score the higher the probability that you will have false positives. The range of values is 0.0 to 1.0 with 1.0 giving you only exact matches. I found that 0.9 is a good score to set but I encourage you to test it out to see what kind of results you get!

- **Line 18:** this is the decompressed location of all of your JSON files that you had from Part 1 in this series. My example dataset can be downloaded here.

Now let's start walking through each JSON file and get it loaded up and ready for scikit-learn to process and analyze them.

```
32 for json_file in file_list:
33
34     with open(json_file,"rb") as fd:
35
36         scan_result = json.load(fd)
37
38         if scan_result['snapshot'] is not None:
39
40             index_pages.append(scan_result['snapshot'])
41             hidden_services.append(scan_result['hiddenService'])
42
```

- **Lines 32-36:** this little chunk of code should look pretty familiar by now, we are just walking through each JSON file, loading it up and parsing the JSON so that we can use it.
- **Lines 38-41:** if there is an HTML snapshot of the hidden service (38) we shovel the HTML into our **index_pages** list (40) and then add the hidden service address into our **hidden_services** list (41).

Now we have all of our data collected we can pass it in to scikit-learn for analysis and then start to examine the results. Let's hand the data to scikit-learn now:

```
45 tfidf = TfidfVectorizer().fit_transform(index_pages)
46 pairwise_similarity = tfidf * tfidf.T
47
48 # get the exact matrix for our hidden service
49 page_similarity_matrix = pairwise_similarity.A[hidden_services.index(base_hidden_service)]
50
```

- **Lines 45-46:** we hand our list of HTML snapshots to the magical TfidVectorizer which handles the magic math to figure out how similar each HTML page is to one another.
- **Line 49:** the result of the TfidVectorizer hands back a matrix and that matrix we then as for a Numpy array using the **.A** attibrute. This array is effectively a list of lists, which we select out our target hidden service based on its position in the list of hidden services because it is the same

position in our Numpy array.

These three lines of code are pretty opaque to me, due to my lack of mathematical and machine learning smarts. Let's get back to stuff I do understand, and add some more code:

```
51 # this gives us the base hidden service
52 compare_counter = 0
53
54 for score in page_similarity_matrix:
55
56     if score >= detect_score:
57
58         if hidden_services[compare_counter] != base_hidden_service:
59
60             if score == 1.0:
61
62                 print "[*] Mirror: %s to %s (Score: %2.2f)" % (base_hidden_service,hidden_services[compare_counter],score)
63             else:
64                 print "[*] Potential Clone: %s to %s (Score: %2.2f)" % (base_hidden_service,hidden_services[compare_counter],sco
65
66     compare_counter += 1
67
68 print "[*] Finished."
```

- **Lines 54-56:** we loop over the array of results and each item in the array is the score that tells us how similar the HTML is to our base hidden service HTML (54). If the score is greater than or equal to the score we set at the top of the script we are going to print it out.
- **Lines 58-64:** we test to see if we are comparing the base hidden service to itself (58) and if not we check for it to be a perfect match (60) which would indicate a mirror or a clone which we print out separately (64).

Ok not let's test this out using Kryptia's counterfeit hidden service.

Let It Rip

You can drop into a terminal or using your development environment to run your script like so:

```
1 python find_clones.py -s usd4you5sa237ulk.onion
```

[*] Target hidden service usd4you5sa237ulk.onion found. Loading data now.
[*] Potential Clone: usd4you5sa237ulk.onion to dollarsfn45wiq4f.onion (Score: 0.96)
[*] Potential Clone: usd4you5sa237ulk.onion to usd4c6cwr467mpto.onion (Score: 0.96)
[*] Potential Clone: usd4you5sa237ulk.onion to usd4cx7otgnx6wtp.onion (Score: 0.96)
[*] Finished.

Awesome it found some hits! Now if you load up Tor Browser and go have a look you will see that the sites are very similar to one another but there are some small subtle differences. As homework you could enhance this Python script to show you the exact differences.

Want more Python and OSINT?

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

Your email...

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)