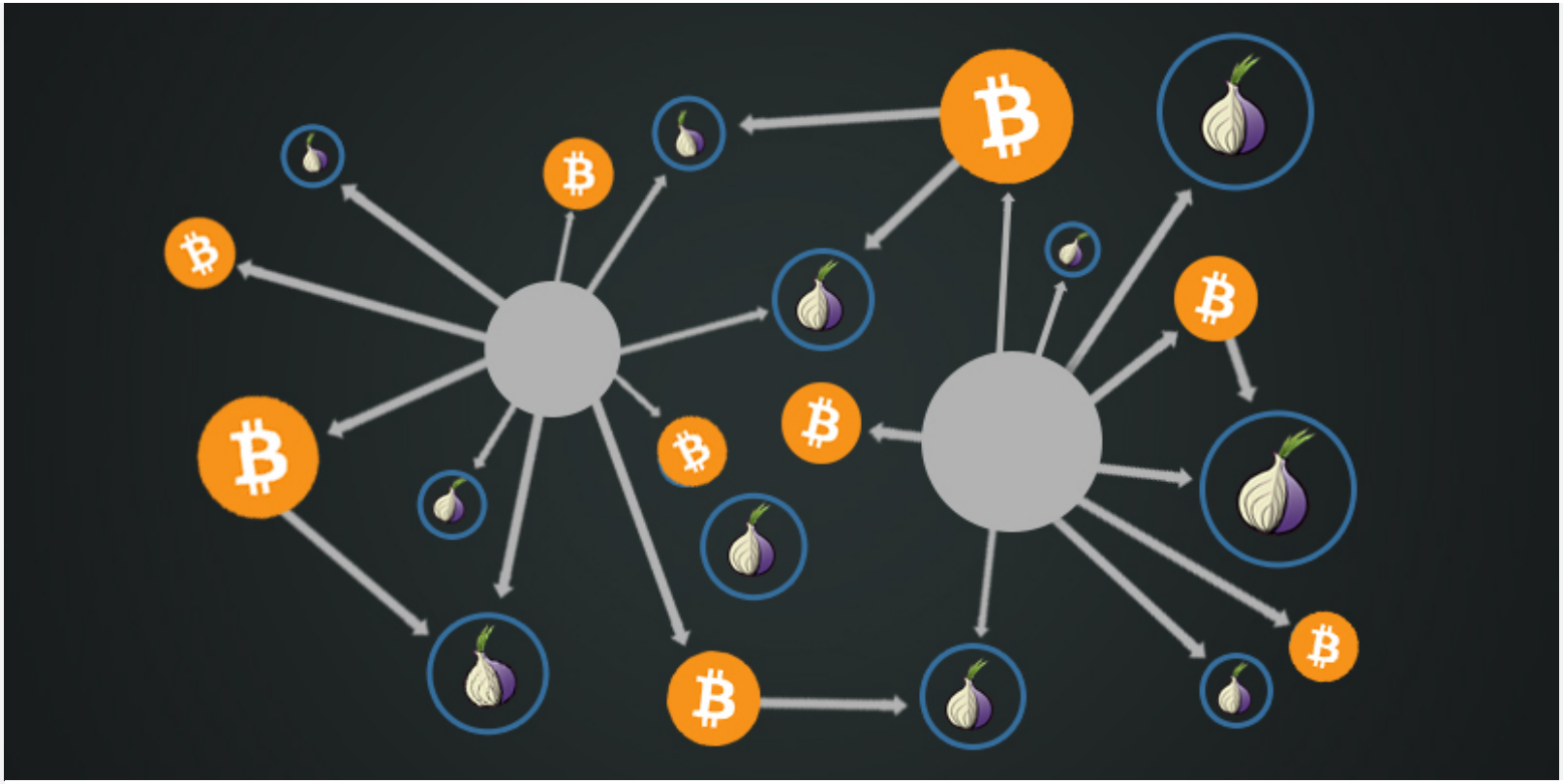
Open source intelligence techniques & commentary



Follow the Bitcoin With Python, BlockExplorer and Webhose.io

Written by **Justin,** September 12th, 2017

More and more investigations are being conducted on Tor and many of them can also include investigating Bitcoin transactions. The nature of Bitcoin is such that the transactions themselves are designed to be anonymous but there are many other factors that can dictate whether the owner of a Bitcoin wallet is protecting their identity correctly. By performing secondary searches for Bitcoin addresses you can typically find relationships or interesting trails of information that you can follow.

In this blog post we are going to develop a tool where we can target a particular Bitcoin address, visualize the transactions flowing in and out of it and then perform secondary dark web searches using Webhose.io to see if we can find hidden services where the bitcoin wallets have been mentioned. We will of course visualize all of this so that we can explore the data when we are finished.

Let's get started!

Prerequisites

First make sure you have Python installed, and then install the requests library: pip install requests. If you are unsure of how to do this then check out the videos on this page under the Setup section.

Next head to Webhose.io and request an API access key.

You will also require a few Python libraries to be installed. So do the following:

```
pip install requests networkx
```

Now let's get coding!

Coding It Up

Crack open a new Python script, name it bitcoindarkweb.py (you can download the full source here) and start hammering out the following code:

```
import argparse
import requests
import networkx

webhose_access_token = "WEBHOSE API TOKEN"

blacklist = ["4a6kzlzytb4ksafk.onion","blockchainbdgpzk.onion"]

webhose_base_url    = "http://webhose.io"
```

```
webhose_darkweb_url = "/darkFilter?token=%s&format=json&q=" % webhose_access_token

block_explorer_url  = "https://blockexplorer.com/api/addrs/"
#?from=0&to=50

parser = argparse.ArgumentParser(description='Collect and visualize Bitcoin transactions and any related hidden services.')

parser.add_argument("--graph",help="Output filename of the graph file. Example: bitcoin.gexf",default="bitcoingraph.gexf")
parser.add_argument("--address", help="A bitcoin address to begin the search on.",)


args = parser.parse_args()

bitcoin_address = args.address
graph_file      = args.graph
```

All of this is pretty straightforward but we'll talk about a few items of importance first:

- **Line 5:** this is the access token you need from Webhose.io in order for the dark web searches to work.
- **Line 7:** this is a blacklist of Tor hidden services that we will not include in our searches or our graph. The reason we have this list is because there are various Bitcoin hidden services that might list piles of addresses, and we aren't really interested in having them come back as results. Feel free to add to this list as well.

The rest of the code deals with setting up some placeholder variables, and setting up the commandline parsing for the script. Now let's continue adding code:

```
#
# Retrieve all bitcoin transactions for a Bitcoin address
#
def get_all_transactions(bitcoin_address):

    transactions = []
    from_number  = 0
    to_number    = 50

    block_explorer_url_full = block_explorer_url + bitcoin_address + "/txs?from=%d&to=%d" % (from_number,to_number)

    response = requests.get(block_explorer_url_full)

    try:
        results  = response.json()
```

```
41      except:
42          print "[!] Error retrieving bitcoin transactions. Please re-run this script."
43          return transactions
44
45      if results['totalItems'] == 0:
46          print "[*] No transactions for %s" % bitcoin_address
47          return transactions
48
49      transactions.extend(results['items'])
50
51      while len(transactions) < results['totalItems']:
52
53          from_number += 50
54          to_number   += 50
55
56          block_explorer_url_full = block_explorer_url + bitcoin_address + "/txs?from=%d&to=%d" % (from_number,to_number)
57
58          response = requests.get(block_explorer_url_full)
59
60          results  = response.json()
61
62          transactions.extend(results['items'])
63
64      print "[*] Retrieved %d bitcoin transactions." % len(transactions)
65
66      return transactions
```

- **Line 29:** we setup our **get_all_transactions** function to take in a Bitcoin address. This function will be responsible for downloading all transactions for this address which we will use later.
- **Line 35:** here we are building up a BlockExplorer URL in the format that they require in order for us to download chunks of transactions from their service.
- **Lines 37-47:** we send off the request to BlockExplorer.com (37) and then attempt to parse the JSON result (40). If the parsing fails we ouput a message (42) and return an empty list of transactions (43). If the parsing was successful but there are no transactions for that address (45) we output a message (46) and return an empty list (47).
- **Line 49:** we take the list of transactions and add them to our main transaction list.
- **Lines 51-62:** we setup a loop to continue grabbing transactions in blocks of fifty at a time (51) and then we continually increase the **from** (53) and **to** (54) variables to “page” through the transactions. We pass these variables along to the BlockExplorer API (56) and then add the results to our main transactions list (62) before continuing to the top of the loop again.
- **Line 66:** now that we are done collecting all of the transactions, we return the transaction list.

Ok cool, we now have a method to collect all transactions flowing into and out of a particular Bitcoin address. Now let's create a function that can loop through all of the Bitcoin transactions and extract all of the unique Bitcoin addresses. This will give us a unique list of Bitcoin addresses to search Webhose.io for later on. Punch in the following code:

```
68 #
69 # Simple function to return a list of all unique
70 # bitcoin addresses from a transaction list
71 #
72 def get_unique_bitcoin_addresses(transaction_list):
73
74     bitcoin_addresses = []
75
76     for transaction in transaction_list:
77
78         # check the sending address
79         if transaction['vin'][0]['addr'] not in bitcoin_addresses:
80             bitcoin_addresses.append(transaction['vin'][0]['addr'])
81
82         # walk through all recipients and check each address
83         for receiving_side in transaction['vout']:
84
85             if receiving_side['scriptPubKey'].has_key("addresses"):
86
87                 for address in receiving_side['scriptPubKey']['addresses']:
88
89                     if address not in bitcoin_addresses:
90
91                         bitcoin_addresses.append(address)
92
93     print "[*] Identified %d unique bitcoin addresses." % len(bitcoin_addresses)
94
95     return bitcoin_addresses
```

- **Line 72:** we setup the function to take in our list of transactions that we previously have retrieved.
- **Lines 79-80:** we check the sending address to see if it is an address we haven't seen before and if it is new, we add it to our **bitcoin_addresses** list (80).
- **Lines 83-91:** we loop through the receiving side of the transaction (83), and then walk through all of the addresses that are contained (87). For each address we encounter we check to see if it is in our list already (89) and if it is a new address we add it to our **bitcoin_addresses** (91) list.

- **Lines 93-95:** once we have retrieved all of the unique addresses that are related to our target address, we return this list so that we can use it in our Webhose.io searches.

Cool at this point we have all of the Bitcoin transactions for an address, and we have nailed down all of the unique addresses that have sent or received Bitcoin from that address. Now let's implement the Webhose.io searching so we can pull on some additional intelligence threads to determine potential hidden services where these addresses have been mentioned. Continue adding code to your script:

```
98  #
99  # Search Webhose.io for each bitcoin address
100 #
101 def search_webhose(bitcoin_addresses):
102
103     bitcoin_to_hidden_services = {}
104     count = 1
105
106     for bitcoin_address in bitcoin_addresses:
107
108         print "[*] Searching %d of %d bitcoin addresses." % (count,len(bitcoin_addresses))
109
110         # search for the bitcoin address
111         search_url = webhose_base_url + webhose_darkweb_url + bitcoin_address
112
113         response   = requests.get(search_url)
114
115         result     = response.json()
116
117         # loop continually until we have retrieved all results at Webhose
118         while result['totalResults'] > 0:
119
120             # now walk each search result and map out the unique hidden services
121             for search_result in result['darkposts']:
122
123                 if not bitcoin_to_hidden_services.has_key(bitcoin_address):
124                     bitcoin_to_hidden_services[bitcoin_address] = []
125
126                 if search_result['source']['site'] not in bitcoin_to_hidden_services[bitcoin_address]:
127
128                     bitcoin_to_hidden_services[bitcoin_address].append(search_result['source']['site'])
129
130             # if we have 10 or less results no need to ding the API again
```

```
131             if result['totalResults'] <= 10:
132                 break
133 
134             # build a filtering keyword string
135             query = "%s" % bitcoin_address
136 
137             for hidden_service in bitcoin_to_hidden_services[bitcoin_address]:
138                 query += " -site:%s" % hidden_service
139 
140             # use the blacklisted onions as filters
141             for hidden_service in blacklist:
142                 query += " -site:%s" % hidden_service
143 
144             search_url = webhose_base_url + webhose_darkweb_url + query
145 
146             response     = requests.get(search_url)
147 
148             result     = response.json()
149 
150         if bitcoin_to_hidden_services.has_key(bitcoin_address):
151             print "[*] Discovered %d hidden services connected to %s" % (len(bitcoin_to_hidden_services[bitcoin_address]),bitco
152 
153         count += 1
154 
155     return bitcoin_to_hidden_services
```

- **Line 101:** we define our **search_webhose** function to take in the list of Bitcoin addresses that we want to search for.
- **Lines 106-115:** we begin looping over our list of Bitcoin addresses (106) and then add it to the Webhose search URL (111), before sending off the request (113) and processing the JSON response (115).
- **Lines 118-128:** we setup a loop to download all results (118) and then start walking through each search result (121). We test where we already have the current Bitcoin address in our **bitcoin_to_hidden_services** dictionary (123) and if not we add it and initialize an empty list (124). We then do a secondary check to make sure that the current hidden service result is not in the list associated with the current Bitcoin address (126) and if not then we add it (128). This whole dance is so that we can associate a Bitcoin address to a list of unique hidden services.
- **Lines 130-131:** if we have less than 10 results (130) we break out of the loop (131) and return to the top of the function to check the next Bitcoin address in the list.
- **Lines 135-142:** we want to now continue grabbing additional search results from Webhose, and how we do this is by using our original query, and then adding some blacklisted sites that we do not was results for. This will also lower the total number of API calls required to get all of our results. We do this by looping over the discovered hidden services (137) and build a negative query for each of them (138). We then loop over the blacklist we have at the top of the script (141) and also add those hidden services to our negative query (142).

- **Lines 144-148:** we send off the new query (144) and then parse the JSON response (148) before returning to the top of the loop on line 118 to continue processing newly discovered hidden services.

Oof! That's a lot of code. At this point in our script we can get all Bitcoin transactions, all unique Bitcoin addresses related to those transactions and any mentions in the dark web that Webhose.io has stored. Now what we need to do is tie this all together into a tidy visualization that will allow us to review these results easily. Time to add our magical graphing function, so keep adding code to the script:

```
157 #
158 # Graph all of the Bitcoin transactions
159 #
160 def build_graph(source_bitcoin_address,transaction_list,hidden_services):
161
162     graph = networkx.DiGraph()
163
164     # graph the transactions by address
165     for transaction in transaction_list:
166
167         # check the sending address
168         sender = transaction['vin'][0]['addr']
169
170         if sender == source_bitcoin_address:
171             graph.add_node(sender,{"type":"Target Bitcoin Wallet"})
172         else:
173             graph.add_node(sender,{"type":"Bitcoin Wallet"})
174
175
176         # walk through all recipients and check each address
177         for receiving_side in transaction['vout']:
178
179             if receiving_side['scriptPubKey'].has_key("addresses"):
180                 for address in receiving_side['scriptPubKey']['addresses']:
181
182                     if address == source_bitcoin_address:
183                         graph.add_node(address,{"type":"Target Bitcoin Address"})
184                     else:
185                         graph.add_node(address,{"type":"Bitcoin Address"})
186
187                     graph.add_edge(sender,address)
188
189     for bitcoin_address in hidden_services:
```

```
190
191         for hidden_service in hidden_services[bitcoin_address]:
192
193             if hidden_service not in blacklist:
194                 graph.add_node(hidden_service,{"type":"Hidden Service"})
195                 graph.add_edge(bitcoin_address,hidden_service)
196
197
198     # write out the graph
199     networkx.write_gexf(graph,graph_file)
200
201     return
```

- **Line 160:** we define our **build_graph** function to take in the source Bitcoin address, the list of Bitcoin transactions and our hidden services dictionary.
- **Line 162:** we create a NetworkX graph object. Note that this is a DiGraph or directional graph which will show us directions for the Bitcoin transactions (in and out).
- **Lines 165-173:** we loop through the list of transactions (165), and pluck out the sender address (168). If we detect the sender is our target Bitcoin address (170) we add a node to the graph with the special ***type*** attribute set to "Target Bitcoin Wallet" (171). Otherwise we just set the attribute to "Bitcoin Address". The reason we have the different attributes is so that we can color the dots on the graph in Gephi based on these attributes.
- **Lines 177-187:** now walk through the receiving side of the transaction (177) and then walk through each of the addresses that are found within the transaction (180). We do the same check for the target Bitcoin wallet (182) and apply the same attributes as we did previously. The last thing we do is connect the sender to the receiver on the graph by adding an edge (a line) between them. Notice that the sender is first in the **add_edge** function which will draw the arrow out to the receiver address when we visualize the graph.
- **Lines 189-195:** now we need to add our hidden services that were discovered. We do this by walking through the dictionary (189) and then walking through each hidden service in stored in with each dictionary key (191). If the hidden service is not in the blacklist (193) we add a new node to the graph with the ***type*** attribute set to "Hidden Service" (194) and then add an edge between the associated Bitcoin address and the hidden service (195).
- **Line 199:** the last thing we do is output the entire graph object to a GEXF file that we can open in Gephi.

Sweet! Now the last step is to tie together all of these functions so that we can take the whole thing for a test drive! We are almost finished, just a little more code to go:

```
204 # get all of the bitcoin transactions
```

```
205 print "[*] Retrieving all transactions from the blockchain for %s" % bitcoin_address
206
207 transaction_list = get_all_transactions(bitcoin_address)
208
209 if len(transaction_list) > 0:
210
211     # get all of the unique bitcoin addresses
212     bitcoin_addresses = get_unique_bitcoin_addresses(transaction_list)
213
214     # now search Webhose for all addresses looking
215     # for hidden services
216     hidden_services   = search_webhose(bitcoin_addresses)
217
218     # graph the bitcoin transactions
219     build_graph(bitcoin_address,transaction_list, hidden_services)
220
221     print "[*] Done! Open the graph file and happy hunting!"
```

- **Line 207:** we first call our **get_all_transactions** function to retrieve our transaction list for the specified target Bitcoin wallet.
- **Lines 209-212:** if we have a valid list of transactions (209) we then get all of the unique Bitcoin addresses from the transaction list (212).
- **Line 216:** we leverage Webhose.io to try to find any hidden services that have references to our discovered Bitcoin addresses.
- **Line 219:** we build the graph so that we can visually explore our results!

Nice work! If you have your Webhose.io token setup and a target Bitcoin address, you should be ready to take this script for a spin.

Let It Rip

Now give your script a run, in my case I used the following. You can download the resulting GEXF file here.

```
1 python bitcoindarkweb.py --address 19m9yEChBSPuzCzEMmg1dNbPvdLdWA59rS
```

[*] Retrieving all transactions from the blockchain for 19m9yEChBSPuzCzEMmg1dNbPvdLdWA59rS
[*] Retrieved 594 bitcoin transactions.
[*] Identified 1998 unique bitcoin addresses.
[*] Searching 1 of 1998 bitcoin addresses.

[*] Searching 2 of 1998 bitcoin addresses.
[*] Searching 3 of 1998 bitcoin addresses.
[*] Discovered 2 hidden services connected to 19m9yEChBSPuzCzEMmg1dNbPvdLdWA59rS
[*] Searching 4 of 1998 bitcoin addresses.
...
[*] Searching 1997 of 1998 bitcoin addresses.
[*] Searching 1998 of 1998 bitcoin addresses.
[*] Done! Open the graph file and happy hunting!

Cool, now you can open up the graph in Gephi (I cover Gephi usage here) and begin exploring some of the transactions and hidden services that you have discovered. Another technique you could do to enhance your results would be to include normal web searches through Searx to find additional results. I have a supporting post here that can help!

Online Python Course

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.

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)