shodan-python Documentation

Release 1.0

achillean

Jun 02, 2018

Contents

This is the official Python wrapper around both the Shodan REST API as well as the experimental Streaming API. And as a bonus it also lets you search for exploits using the Shodan Exploits REST API. If you're not sure where to start simply go through the "Getting Started" section of the documentation and work your way down through the examples.

For more information about Shodan and how to use the API please visit our official help center at:

> https://help.shodan.io

CHAPTER 1

Introduction

1.1 Getting Started

1.1.1 Installation

To get started with the Python library for Shodan, first make sure that you've received your API key. Once that's done, install the library via the cheeseshop using:

```
$ easy_install shodan
```

Or if you already have it installed and want to upgrade to the latest version:

```
$ easy_install -U shodan
```

It's always safe to update your library as backwards-compatibility is preserved. Usually a new version of the library simply means there are new methods/ features available.

1.1.2 Connect to the API

The first thing we need to do in our code is to initialize the API object:

```
import shodan

SHODAN_API_KEY = "insert your API key here"

api = shodan.Shodan(SHODAN_API_KEY)
```

1.1.3 Searching Shodan

Now that we have our API object all good to go, we're ready to perform a search:

```
# Wrap the request in a try/ except block to catch errors
try:
        # Search Shodan
        results = api.search('apache')

        # Show the results
        print('Results found: {}'.format(results['total']))
        for result in results['matches']:
                print('IP: {}'.format(result['ip_str']))
                print(result['data'])
                print('')
except shodan.APIError, e:
        print('Error: {}'.format(e))
```

Stepping through the code, we first call the `Shodan.search()` method on the *api* object which returns a dictionary of result information. We then print how many results were found in total, and finally loop through the returned matches and print their IP and banner. Each page of search results contains up to 100 results.

There's a lot more information that gets returned by the function. See below for a shortened example dictionary that `Shodan.search()` returns:

```
{
        'total': 8669969,
        'matches': [
                {
                        'data': 'HTTP/1.0 200 OK\r\nDate: Mon, 08 Nov 2010 05:09:59␣
↪GMT\r\nSer...',
                        'hostnames': ['pl4t1n.de'],
                        'ip': 3579573318,
                        'ip_str': '89.110.147.239',
                        'os': 'FreeBSD 4.4',
                        'port': 80,
                        'timestamp': '2014-01-15T05:49:56.283713'
                },
                ...
        ]
}
```

Please visit the REST API documentation for the complete list of properties that the methods can return.

It's also good practice to wrap all API requests in a try/ except clause, since any error will raise an exception. But for simplicity's sake, I will leave that part out from now on.

1.1.4 Looking up a host

To see what Shodan has available on a specific IP we can use the `Shodan.host()` function:

```
# Lookup the host
host = api.host('217.140.75.46')

# Print general info
print("""
        IP: {}
        Organization: {}
        Operating System: {}
""".format(host['ip_str'], host.get('org', 'n/a'), host.get('os', 'n/a')))
```

(continues on next page)

(continued from previous page)

```python

# Print all banners
for item in host['data']:
        print("""
                Port: {}
                Banner: {}

        """.format(item['port'], item['data']))
```

CHAPTER 2

Examples

2.1 Basic Shodan Search

```
#!/usr/bin/env python
#
# shodan_ips.py
# Search SHODAN and print a list of IPs matching the query
#
# Author: achillean

import shodan
import sys

# Configuration
API_KEY = "YOUR_API_KEY"

# Input validation
if len(sys.argv) == 1:
        print 'Usage: %s <search query>' % sys.argv[0]
        sys.exit(1)

try:
        # Setup the api
        api = shodan.Shodan(API_KEY)

        # Perform the search
        query = ' '.join(sys.argv[1:])
        result = api.search(query)

        # Loop through the matches and print each IP
        for service in result['matches']:
                print service['ip_str']
except Exception as e:
        print 'Error: %s' % e
```

(continues on next page)

(continued from previous page)

```
        sys.exit(1)
```

2.2 Collecting Summary Information using Facets

A powerful ability of the Shodan API is to get summary information on a variety of properties. For example, if you wanted to learn which countries have the most Apache servers then you would use facets. If you wanted to figure out which version of nginx is most popular, you would use facets. Or if you wanted to see what the uptime distribution is for Microsoft-IIS servers then you would use facets.

The following script shows how to use the *shodan.Shodan.count()* method to search Shodan without returning any results as well as asking the API to return faceted information on the organization, domain, port, ASN and country.

```
#!/usr/bin/env python
#
# query-summary.py
# Search Shodan and print summary information for the query.
#
# Author: achillean

import shodan
import sys

# Configuration
API_KEY = 'YOUR API KEY'

# The list of properties we want summary information on
FACETS = [
    'org',
    'domain',
    'port',
    'asn',

    # We only care about the top 3 countries, this is how we let Shodan know to␣
↪return 3 instead of the
    # default 5 for a facet. If you want to see more than 5, you could do ('country',␣
↪1000) for example
    # to see the top 1,000 countries for a search query.
    ('country', 3),
]

FACET_TITLES = {
    'org': 'Top 5 Organizations',
    'domain': 'Top 5 Domains',
    'port': 'Top 5 Ports',
    'asn': 'Top 5 Autonomous Systems',
    'country': 'Top 3 Countries',
}

# Input validation
if len(sys.argv) == 1:
    print 'Usage: %s <search query>' % sys.argv[0]
    sys.exit(1)

try:
```

(continues on next page)

(continued from previous page)

```
    # Setup the api
    api = shodan.Shodan(API_KEY)

    # Generate a query string out of the command-line arguments
    query = ' '.join(sys.argv[1:])

    # Use the count() method because it doesn't return results and doesn't require a␣
↪paid API plan
    # And it also runs faster than doing a search().
    result = api.count(query, facets=FACETS)

    print 'Shodan Summary Information'
    print 'Query: %s' % query
    print 'Total Results: %s\n' % result['total']

    # Print the summary info from the facets
    for facet in result['facets']:
        print FACET_TITLES[facet]

        for term in result['facets'][facet]:
            print '%s: %s' % (term['value'], term['count'])

        # Print an empty line between summary info
        print ''

except Exception, e:
    print 'Error: %s' % e
    sys.exit(1)

"""
Sample Output
=============

./query-summary.py apache
Shodan Summary Information
Query: apache
Total Results: 34612043

Top 5 Organizations
Amazon.com: 808061
Ecommerce Corporation: 788704
Verio Web Hosting: 760112
Unified Layer: 627827
GoDaddy.com, LLC: 567004

Top 5 Domains
secureserver.net: 562047
unifiedlayer.com: 494399
t-ipconnect.de: 385792
netart.pl: 194817
wanadoo.fr: 151925

Top 5 Ports
80: 24118703
443: 8330932
8080: 1479050
81: 359025
```

(continues on next page)

(continued from previous page)

```
8443: 231441

Top 5 Autonomous Systems
as32392: 580002
as2914: 465786
as26496: 414998
as48030: 332000
as8560: 255774

Top 3 Countries
US: 13227366
DE: 2900530
JP: 2014506
"""
```

2.3 Access SSL certificates in Real-Time

The new Shodan Streaming API provides real-time access to the information that Shodan is gathering at the moment. Using the Streaming API, you get the raw access to potentially all the data that ends up in the Shodan search engine. Note that you can't search with the Streaming API or perform any other operations that you're accustomed to with the REST API. This is meant for large-scale consumption of real-time data.

This script only works with people that have a subscription API plan! And by default the Streaming API only returns 1% of the data that Shodan gathers. If you wish to have more access please contact us at support@shodan.io for pricing information.

```
#!/usr/bin/env python
#
# cert-stream.py
# Stream the SSL certificates that Shodan is collecting at the moment
#
# WARNING: This script only works with people that have a subscription API plan!
# And by default the Streaming API only returns 1% of the data that Shodan gathers.
# If you wish to have more access please contact us at sales@shodan.io for pricing
# information.
#
# Author: achillean

import shodan
import sys

# Configuration
API_KEY = 'YOUR API KEY'

try:
    # Setup the api
    api = shodan.Shodan(API_KEY)

    print 'Listening for certs...'
    for banner in api.stream.ports([443, 8443]):
            if 'ssl' in banner:
                    # Print out all the SSL information that Shodan has collected
                    print(banner['ssl'])
```

(continues on next page)

(continued from previous page)

```
except Exception as e:
    print('Error: {}'.format(e))
    sys.exit(1)
```

2.4 GIF Creator

Shodan keeps a full history of all the information that has been gathered on an IP address. With the API, you're able to retrieve that history and we're going to use that to create a tool that outputs GIFs made of the screenshots that the Shodan crawlers gather.

The below code requires the following Python packages:

- arrow
- shodan

The **arrow** package is used to parse the *timestamp* field of the banner into a Python *datetime* object.

In addition to the above Python packages, you also need to have the **ImageMagick** software installed. If you're working on Ubuntu or another distro using **apt** you can run the following command:

```
sudo apt-get install imagemagick
```

This will provide us with the **convert** command which is needed to merge several images into an animated GIF.

There are a few key Shodan methods/ parameters that make the script work:

1. `shodan.helpers.iterate_files()` to loop through the Shodan data file
2. **history** flag on the *shodan.Shodan.host()* method to get all the banners for an IP that Shodan has collected over the years

```
#!/usr/bin/env python
# gifcreator.py
#
# Dependencies:
# - arrow
# - shodan
#
# Installation:
# sudo easy_install arrow shodan
# sudo apt-get install imagemagick
#
# Usage:
# 1. Download a json.gz file using the website or the Shodan command-line tool␣
↪(https://cli.shodan.io).
#    For example:
#        shodan download screenshots.json.gz has_screenshot:true
# 2. Run the tool on the file:
#        python gifcreator.py screenshots.json.gz

import arrow
import os
import shodan
import shodan.helpers as helpers
import sys
```

(continues on next page)

(continued from previous page)

```
# Settings
API_KEY = ''
MIN_SCREENS = 5 # Number of screenshots that Shodan needs to have in order to make a␣
↪GIF
MAX_SCREENS = 24

if len(sys.argv) != 2:
        print('Usage: {} <shodan-data.json.gz>'.format(sys.argv[0]))
        sys.exit(1)

# GIFs are stored in the local "data" directory
os.mkdir('data')

# We need to connect to the API to lookup the historical host information
api = shodan.Shodan(API_KEY)

# Use the shodan.helpers.iterate_files() method to loop over the Shodan data file
for result in helpers.iterate_files(sys.argv[1]):
        # Get the historic info
        host = api.host(result['ip_str'], history=True)

        # Count how many screenshots this host has
        screenshots = []
        for banner in host['data']:
                # Extract the image from the banner data
                if 'opts' in banner and 'screenshot' in banner['opts']:
                        # Sort the images by the time they were collected so the GIF␣
↪will loop
                        # based on the local time regardless of which day the banner␣
↪was taken.
                        timestamp = arrow.get(banner['timestamp']).time()
                        sort_key = timestamp.hour
                        screenshots.append((
                                sort_key,
                                banner['opts']['screenshot']['data']
                        ))

                        # Ignore any further screenshots if we already have MAX_
↪SCREENS number of images
                        if len(screenshots) >= MAX_SCREENS:
                                break

        # Extract the screenshots and turn them into a GIF if we've got the necessary
        # amount of images.
        if len(screenshots) >= MIN_SCREENS:
                for (i, screenshot) in enumerate(sorted(screenshots, key=lambda x:␣
↪x[0], reverse=True)):
                        open('/tmp/gif-image-{}.jpg'.format(i), 'w').
↪write(screenshot[1].decode('base64'))

                # Create the actual GIF using the  ImageMagick "convert" command
                os.system('convert -layers OptimizePlus -delay 5x10 /tmp/gif-image-*.
↪jpg -loop 0 +dither -colors 256 -depth 8 data/{}.gif'.format(result['ip_str']))

                # Clean up the temporary files
```

(continues on next page)

(continued from previous page)

```
                os.system('rm -f /tmp/gif-image-*.jpg')

                # Show a progress indicator
                print result['ip_str']
```

The full code is also available on GitHub: https://gist.github.com/achillean/963eea552233d9550101

CHAPTER 3

API Reference

3.1 shodan

class shodan.**Shodan**(*key*)

Wrapper around the Shodan REST and Streaming APIs

Parameters **key** (*str*) – The Shodan API key that can be obtained from your account page (https://account.shodan.io)

Variables

- **exploits** – An instance of *shodan.Shodan.Exploits* that provides access to the Exploits REST API.
- **stream** – An instance of *shodan.Shodan.Stream* that provides access to the Streaming API.

class Exploits(*parent*)

count(*query*, *facets=None*)

Search the entire Shodan Exploits archive but only return the total # of results, not the actual exploits.

Parameters

- **query** (*str*) – The exploit search query; same syntax as website.
- **facets** (*str*) – A list of strings or tuples to get summary information on.

Returns dict – a dictionary containing the results of the search.

search(*query*, *page=1*, *facets=None*)

Search the entire Shodan Exploits archive using the same query syntax as the website.

Parameters

- **query** (*str*) – The exploit search query; same syntax as website.
- **facets** (*str*) – A list of strings or tuples to get summary information on.
- **page** (*int*) – The page number to access.

Returns dict – a dictionary containing the results of the search.

alerts(*aid=None*, *include_expired=True*)

List all of the active alerts that the user created.

count(*query*, *facets=None*)

Returns the total number of search results for the query.

Parameters

- **query** (*str*) – Search query; identical syntax to the website
- **facets** (*str*) – (optional) A list of properties to get summary information on

Returns A dictionary with 1 main property: total. If facets have been provided then another property called "facets" will be available at the top-level of the dictionary. Visit the website for more detailed information.

create_alert(*name*, *ip*, *expires=0*)

Search the directory of saved search queries in Shodan.

Parameters **query** – The number of tags to return

Returns A list of tags.

delete_alert(*aid*)

Delete the alert with the given ID.

host(*ips*, *history=False*, *minify=False*)

Get all available information on an IP.

Parameters

- **ip** (*str*) – IP of the computer
- **history** (*bool*) – (optional) True if you want to grab the historical (non-current) banners for the host, False otherwise.
- **minify** (*bool*) – (optional) True to only return the list of ports and the general host information, no banners, False otherwise.

info()

Returns information about the current API key, such as a list of add-ons and other features that are enabled for the current user's API plan.

ports()

Get a list of ports that Shodan crawls

Returns An array containing the ports that Shodan crawls for.

protocols()

Get a list of protocols that the Shodan on-demand scanning API supports.

Returns A dictionary containing the protocol name and description.

queries(*page=1*, *sort='timestamp'*, *order='desc'*)

List the search queries that have been shared by other users.

Parameters

- **page** (*int*) – Page number to iterate over results; each page contains 10 items
- **sort** (*str*) – Sort the list based on a property. Possible values are: votes, timestamp
- **order** (*str*) – Whether to sort the list in ascending or descending order. Possible values are: asc, desc

Returns A list of saved search queries (dictionaries).

queries_search(*query*, *page=1*)

Search the directory of saved search queries in Shodan.

Parameters

- **`query`** (`str`) – The search string to look for in the search query
- **`page`** (`int`) – Page number to iterate over results; each page contains 10 items

Returns A list of saved search queries (dictionaries).

`queries_tags` (*size=10*)
Search the directory of saved search queries in Shodan.

Parameters **`query`** – The number of tags to return

Returns A list of tags.

`scan` (*ips*, *force=False*)
Scan a network using Shodan

Parameters

- **`ips`** (`str or dict`) – A list of IPs or netblocks in CIDR notation or an object structured like: {

  "9.9.9.9": [(443, "https"), (8080, "http")

  ], "1.1.1.0/24": [

  (503, "modbus")

  ]

  }
- **`force`** (`bool`) – Whether or not to force Shodan to re-scan the provided IPs. Only available to enterprise users.

Returns A dictionary with a unique ID to check on the scan progress, the number of IPs that will be crawled and how many scan credits are left.

`scan_internet` (*port*, *protocol*)
Scan a network using Shodan

Parameters

- **`port`** (`str`) – The port that should get scanned.
- **`port`** – The name of the protocol as returned by the protocols() method.

Returns A dictionary with a unique ID to check on the scan progress.

`scan_status` (*scan_id*)
Get the status information about a previously submitted scan.

Parameters **`id`** (`str`) – The unique ID for the scan that was submitted

Returns A dictionary with general information about the scan, including its status in getting processed.

`search` (*query*, *page=1*, *limit=None*, *offset=None*, *facets=None*, *minify=True*)
Search the SHODAN database.

Parameters

- **`query`** (`str`) – Search query; identical syntax to the website
- **`page`** (`int`) – (optional) Page number of the search results
- **`limit`** (`int`) – (optional) Number of results to return

- **`offset`** (*`int`*) – (optional) Search offset to begin getting results from
- **`facets`** (*`str`*) – (optional) A list of properties to get summary information on
- **`minify`** (*`bool`*) – (optional) Whether to minify the banner and only return the important data

Returns A dictionary with 2 main items: matches and total. If facets have been provided then another property called "facets" will be available at the top-level of the dictionary. Visit the website for more detailed information.

`search_cursor`(*query*, *minify=True*, *retries=5*)
Search the SHODAN database.

This method returns an iterator that can directly be in a loop. Use it when you want to loop over all of the results of a search query. But this method doesn't return a "matches" array or the "total" information. And it also can't be used with facets, it's only use is to iterate over results more easily.

Parameters

- **`query`** (*`str`*) – Search query; identical syntax to the website
- **`minify`** (*`int`*) – (optional) Whether to minify the banner and only return the important data
- **`retries`** – (optional) How often to retry the search in case it times out

Returns A search cursor that can be used as an iterator/ generator.

`search_tokens`(*query*)
Returns information about the search query itself (filters used etc.)

Parameters **`query`** (*`str`*) – Search query; identical syntax to the website

Returns A dictionary with 4 main properties: filters, errors, attributes and string.

`services`()
Get a list of services that Shodan crawls

Returns A dictionary containing the ports/ services that Shodan crawls for. The key is the port number and the value is the name of the service.

3.1.1 Exceptions

`exception` `shodan.`**`APIError`**(*value*)
This exception gets raised whenever a non-200 status code was returned by the Shodan API.

Python Module Index

Index