yahi Documentation

Release 0.1.0

julien tayon

Oct 26, 2018

Contents

Contents:

help as usual is obtained this way:

```
./speed_shoot --help
```

it spits out:

```
usage: speed_shoot [-h] [-c FILE] [-g GEOIP] [-q SILENT] [-cs CACHE_SIZE]
                   [-d DIAGNOSE [DIAGNOSE ...]] [-in INCLUDE] [--off OFF]
                   [-x EXCLUDE] [-f OUTPUT_FORMAT] [-lf LOG_FORMAT]
                   [-lp LOG_PATTERN] [-lpn LOG_PATTERN_NAME]
                   [-dp DATE_PATTERN] [-o OUTPUT_FILE]
                   ...

Utility for parsing logs in the apache/nginx combined log format
and output a json of various aggregatted metrics of frequentation :
     * by Geolocation (quite fuzzy but still);
     * by user agent;
     * by hour;
     * by day;
     * by browser;
     * by status code
     * of url by ip;
     * by ip;
     * by url;
     * and bandwidth by ip;

Example :
=========

from stdin (useful for using zcat)
**********************************
zcat /var/log/apache.log.1.gz | parse_log.py  > dat1.json

excluding IPs 192.168/16 and user agent containing Mozilla
**********************************************************
use::
    parse_log -o dat2.json -x '{ "ip" : "^192.168", "agent": "Mozill" }'  /var/log/
↪apache*.log

Since archery is cool here is a tip for aggregating data::
    >>> from archery.barrack import bowyer
    >>> from archery.bow import Hankyu
    >>> from json import load, dumps
    >>> dumps(
            bowyer(Hankyu,load(file("dat1.json"))) +
            bowyer(Hankyu,load(file("dat2.json")))
        )

Hence a usefull trick to merge your old stats with your new one


positional arguments:
  files

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  -c FILE, --config FILE
                        specify a config file in json format for the command
                        line arguments any command line arguments will disable
                        values in the config
  -g GEOIP, --geoip GEOIP
                        specify a path to a geoip.dat file
  -q SILENT, --silent SILENT
                        quietly discard errors
  -cs CACHE_SIZE, --cache-size CACHE_SIZE
                        in conjonction with cp=fixed chooses dict size
  -d DIAGNOSE [DIAGNOSE ...], --diagnose DIAGNOSE [DIAGNOSE ...]
                        diagnose list of space separated arguments :
                        **rejected** : will print on STDERR rejected parsed
                        line, **match** : will print on stderr data filtered
                        out
  -in INCLUDE, --include INCLUDE
                        include from extracted data with a json (string or
                        filename) in the form { "field" : "pattern" }
  --off OFF             turn off plugins : geo_ip to skip geoip, user_agent to
                        turn httpagentparser off
  -x EXCLUDE, --exclude EXCLUDE
                        exclude from extracted data with a json (string or
                        filename) in the form { "field" : "pattern" }
  -f OUTPUT_FORMAT, --output-format OUTPUT_FORMAT
                        decide if output is in a specified formater amongst :
                        csv, json
  -lf LOG_FORMAT, --log-format LOG_FORMAT
                        log format amongst apache_log_combined, lighttpd
  -lp LOG_PATTERN, --log-pattern LOG_PATTERN
                        add a custom named regexp for parsing log lines
  -lpn LOG_PATTERN_NAME, --log-pattern-name LOG_PATTERN_NAME
                        the name with witch you want to register the pattern
  -dp DATE_PATTERN, --date-pattern DATE_PATTERN
                        add a custom date format, usefull if and only if using
                        a custom log_pattern and date pattern differs from
                        apache.
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        output file
```

CHAPTER 1

A commented jumbo command line example

The following command line:

```
./speed_shoot -g data/GeoIP.dat -lf lighttpd -x '{ "datetime" : "^01/May", "uri" : "(.
↪*munin|.*(png|jpg))$"}' -d rejected -d match -i '{ "_country" : "(DE|GB)"  }' *log␣
↪yahi/test/biggersample.log
```

does:

- locate geoIP *g* file in data/GeoIP.dat;
- set log format *lf* to lighttpd;
- **exclude (*x*) any match of either**
  - an uri containing munin or ending by jpg or png
  - May the first;
- **include (*i*) all match containing**
  - any IP which has been geoloclaized,
  - any non authentified user;
- will diagnose (*d*) (thus print on stderr) any lines that would not match

the log format regexp or any lines rejected by *-x* and *-i*

for all the given log files.

CHAPTER 2

Using a config file

Well, not impressive:

```
./speed_shoot -c config.json
```

If any option is specified in the config file it will override those setted in the command line.

Here is a sample of a config file:

```
{
    "exclude" : {
        "uri"  : ".*munin.*",
        "referer" : ".*(munin|php).*"
    },
    "include" : { "datetime" : "^04" },
    "silent" : "False",
    "files" : [ "yahi/test/biggersample.log" ]
}
```

CHAPTER 3

Easter eggs or bad idea

The following options *-x -i -c* can either take a string or a filename, which makes debugging of badly formatted json a pain.

CHAPTER 4

How it works?

4.1 Notch: setting up a context

notch is all about setting up a context::

```
>>> context=notch(
     'yahi/test/biggersample.log',
     'yahi/test/biggersample.log',
     include="yahi/test/include.json",
     silent=True,
     exclude='{ "_country" : "US"}',
     output_format="csv"
)
```

Would I have been smart, it would have been called «aim». Since you tell your target, and the parameters of your parsing (log_format...).

4.2 Command line arguments vs notch arguments

- first command lines arguments are parsed, and setup;
- then the arguments of *notch* are parsed.

Warning: notch arguments always override arguments given in the command line.

4.3 Context methods & attributes

4.3.1 attribute: data_filter

Stores the filter used to filter the data. If nothing specified it will use include and exclude.

4.3.2 method: output

Given output_file / output_format write a Mapping in the specified file with the sepcified format.

> **Warning:** Output will close output_file once it has written in it. Thus, reusing it another time will cause an exception. you should notch once for every time you shoot if you *context.output* for writing to a file.

4.4 Shoot

4.4.1 Logic

Given one or more context, you now can shoot your request to the context given back by notch.

Note:

for each lines of each input file

- use a regexp to transform the parsed line in a dict
- add to record datetime string in **_datetime** key

if geoIP in context.skill add *_country* to the record bades on *ip*

if user_agent in context.skill add *_dist_name*, *_browser_version*, *_browser_version* on *agent*

if not filtered out by context.data_filter add actual transformed record to the previous one

---

It is basically a way to **GROUP BY** like in mysql. As my dict supports addition we have the following logic for each line (given you request an aggregation on country and useragent and you are at the 31st line:

```
>>> { '_country' : { 'BE' : 10, 'FR' : 20
... },  'user_agent' : { 'mozilla' : 13, 'unknown' : 17  } } + {
... 'country' : { 'BE' : 1}, 'user_agent' : { 'safari': 1 } }
{ '_country' : { 'BE' : 11, 'FR' : 20 },
'user_agent' : { 'mozilla' : 13, 'unknown' : 17,'safari': 1 } }
},
```

4.5 How lines of your log are transformed

First since we use named capture in our log regexps, we directly transform a log in a dict. You can give the name you want for your capture except for 3 special things:

- *datetime* is required, because logs **always** have a datetime associated with each record;

- *agent* is required if you want to use **httpagentparser**;
- *ip* is required if you want to use **geoIP**

4.5.1 Datetime

Once *datetime* is captured since datetime objects are easier to use than strings *datetime* value is transformed in *_datetime* with the date_pattern.

4.5.2 GeoIP

Once *ip* is catpured given *geo_ip* is enabled *_country* will be set with the 2 letters of the ISO code of the country.

4.5.3 HttpUserAgentParser

Once agent is captured, it will be transformed -if *user_agent* is enabled- into

- *_dist_name*: the OS;
- *_browser_name*: the name of the web browser;
- *_browser_version*: the version of the browser.

CHAPTER 5

Notch and shoot by the example

For this exercice I do have a preference for *bpython*, since it has the ctrl+S shortcut. Thus, you can save any «experiments» in a file.

It is pretty much a querying language in disguise.

Initially I did not planned to use it in a console or as a standalone module so the API is not satisfying.

5.1 Notch: choose your input

So let's take an example::

```
>>> context=notch(
     'yahi/test/biggersample.log' ,'another_log',
     include="yahi/test/include.json",
     exclude='{ "ip" : "^(192\.168|10\.)"}',
     output_format="csv"
)
# include.json contains : { "_country"  : "GB","user" : "-" }
```

Here you parse two files, you want:

- only GB hits,
- non authed users,
- to filter out private IP,
- and you may want to use a CSV formater as an output format.

(Since no output file is set, output is redirected to stdout (errors are directed on stderr)).

5.2 Shoot: choose and aggregate your data

Shoot has 2 inputs:

- a context (setup by notch);
- an extractor;

An extractor is a function extracting and transforming datas, and since I love short circuits, that may contain some on the fly filtering :)

5.2.1 Total hits in a log matching the conditions from notch

Example::

```
>>> from archery import Hankyu as _dict
>>> shoot(
... context,
... lambda data: _dict({ 'total_lines' : 1 })
... )
```

5.2.2 Gross total hits in business hours and off business hour

Business hour being each weekday from monday to friday, between 8 am and 5 pm.

Example::

```
>>> from archery import Hankyu as _dict
>>> shoot(
... context,
... lambda data: _dict({ (
...         8 >= data["_datetime"].hour >= 17 and
...         data["_datetime"].weekday() < 5
...     ) and "business_hour" or "other_hour" :  1 })
... )
```

Hankyu is a dict supporting addition.

5.2.3 Grouping hits per country code

Example::

```
>>> from archery import Hankyu as _dict
>>> shoot(
... context,
... lambda data: _dict({ data["_country"]: 1 })
... )
```

ToxicSet is a set that maps add to union.

5.2.4 Distinct IP

Example::

```
>>> from archery import Hankyu as _dict
>>> from yahi import ToxicSet
>>> shoot(
... context,
... lambda data: _dict(distinct_ip = ToxicSet({ data["ip"]}))
... )
```

ToxicSet is a set that maps add to union.

5.2.5 Hits per day

example::

```
>>> date_formater= lambda dt :"%s-%s-%s" % ( dt.year, dt.month, dt.day)
>>> from archery import Hankyu as _dict
>>> shoot(
... context,
... lambda data: _dict({
...     date_formater(data["_datetime"]) : 1
... }))
```

5.2.6 Parallelizing request

You can now parallize all your requests by adding one key in the aggregator dict.

Just beware of the memory consumption.

5.3 Custom filtering

Sometimes regexp are not enough, imagine you have a function for checking if a user belongs to the employees, and you want to check all the workhaolic in your company reaching an authentified realm out of the working hours:

```
>>> context.data_filter= lambda data: (
...     is_employee(data["user"]) and not working_hours(data["_datetime"])
... )
>>> shoot( context, _dict(workaholicness = _dict({data["user"] : 1})))
```

Warning: data_filter will override any include/exclude rules given in notch

CHAPTER 6

A complete example of a perverted use: parsing auth.log

Imagine you are a sysadmin and your boss want a graph of all the request you do, and you don't like using excel

```
#!/usr/bin/env python
from archery.bow import Hankyu as _dict
from yahi import notch, shoot, ToxicSet
from datetime import datetime
from datetime import date
import locale

import dateutil
import re
import pylab as plt
from collections import OrderedDict
import numpy as np

locale.setlocale(locale.LC_ALL,"C")

def ordered_top(a_dict, rank=10):
    res=OrderedDict({"other": 0})
    for i,(k,v) in enumerate(
            sorted(a_dict.items(),
                key=lambda (k,v): (v,k),
                reverse=True)
            ):
        if i < rank:
            res[k]=v
        else:
            res["other"]+=v
    return res


######################## Setting UP ###################################
# parsing command line & default settings. Return a not fully qualified object
context=notch(
```

(continues on next page)

(continued from previous page)

```
    off="user_agent",
    log_format="custom",
    output_format="json",
    date_pattern="%b %d %H:%M:%S",
    log_pattern="""^(?P<datetime>[^\ ]+\s{1,2}\d{1,2}\s\d{2,2}:\d{2,2}:\d{2,2})\s
    (?P<nawak>[^:]+):\s
    Invalid\ user\ (?P<user>.*?)\s
    from\ (?P<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})$""")
# log sample
#May 20 12:14:15 lupin sshd[36291]: Invalid user dave from 69.60.114.57

date_formater= lambda dt :"%s-%s-%s" % ( dt.year, dt.month, dt.day)
res= shoot(
        context,
        lambda data: _dict({
            "black_list" : ToxicSet([ data["ip"] ]),
            "by_country" : _dict({ data["_country"] : 1 }),
            "date_s" : _dict({ date_formater(data["_datetime"]) : 1 }),
            "by_ip" : _dict({ data["ip"] : 1 }),
            "date" : _dict({ date(2012,
                data["_datetime"].month,
                data["_datetime"].day)
                : 1 }),
            "by_user" : _dict({ data["user"] : 1 }),
            "total" : 1
        })
    )

# Let's go draw some plot
def labeled_bar(ax, _dict):
    pos=np.arange(len(_dict)) + .5
    ax.set_xticks(pos, _dict.keys())
    rects=ax.bar(pos,_dict.values(),label=_dict.keys(),align='center')
    for i,rect in enumerate(rects):
        height = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., height-100,
           '%.1f\n%s'%(height,_dict.keys()[i]),
           ha='center', va='bottom',color='white', fontsize=8)
    return rects

fig=plt.figure(221,figsize=(18,9))
fig.suptitle(
    "%d SSH unauthorized access from %d sources" % (
        res["total"], len(res["black_list"])),
    fontsize=16,
)
ax=fig.add_subplot(221)
by_country=ordered_top(res["by_country"],5)
ax.set_title("Top 5 country by sources")
ax.pie(by_country.values(),
    labels=map(lambda (k,v):"%s (%d)"%(k,v),by_country.items()),
    shadow=True
)

ax=fig.add_subplot(222)
ax.set_title(
    "Top 10 tested users (amongst %d trials)" % len( res["by_user"])
```

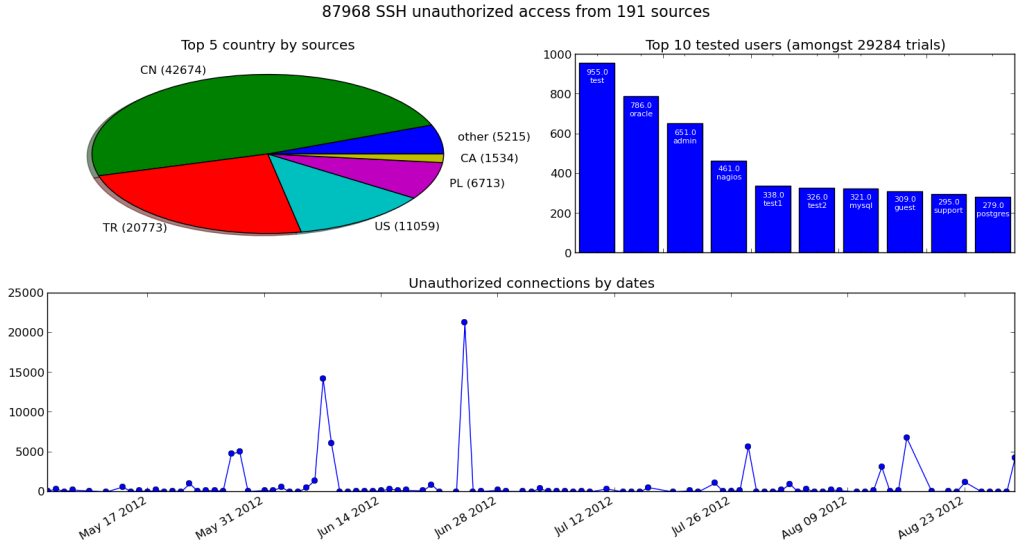
(continues on next page)

(continued from previous page)

```
)
by_user=ordered_top(res["by_user"])
del(by_user['other'])
labeled_bar(ax,by_user)

ax=fig.add_subplot(212)
ax.set_title("Unauthorized connections by dates")
res["date"]=OrderedDict([
    (k,v) for k,v in sorted( res["date"].items()) ]
)
ax.plot_date(plt.date2num(np.array( res["date"].keys())),
    res["date"].values(), linestyle="-")

fig.autofmt_xdate()
plt.savefig("attack.png")
```



CHAPTER 7

Plotting the result with flask

Here is a sample of the website I run with the data that results from yahi on an intel atom on my DSL (for fun): http://wwwstat.julbox.fr/

The sources are here: https://github.com/jul/yahi/tree/master/www

The work is still in development, since I had harder time writing the doc than the code.

- source: https://github.com/jul/yahi
- doc: http://yahi.readthedocs.org/
- ticketting: https://github.com/jul/yahi/issues

CHAPTER 8

Versatile log parser (providing default extractors for apache/lighttpd/varnish)

8.1 Command line usage

Simplest usage is:

```
speed_shoot -g /usr/local/data/geoIP /var/www/apache/access*log
```

it will return a json in the form:

```
{
    "by_date": {
        "2012-5-3": 11
    },
    "total_line": 11,
    "ip_by_url": {
        "/favicon.ico": {
            "192.168.0.254": 2,
            "192.168.0.35": 2
        },
        "/": {
            "74.125.18.162": 1,
            "192.168.0.254": 1,
            "192.168.0.35": 5
        }
    },
    "by_status": {
        "200": 7,
        "404": 4
    },
    "by_dist": {
        "unknown": 11
    },
```

(continues on next page)

(continued from previous page)


```
    "bytes_by_ip": {
        "74.125.18.162": 151,
        "192.168.0.254": 489,
        "192.168.0.35": 1093
    },
    "by_url": {
        "/favicon.ico": 4,
        "/": 7
    },
    "by_os": {
        "unknown": 11
    },
    "week_browser": {
        "3": {
            "unknown": 11
        }
    },
    "by_referer": {
        "-": 11
    },
    "by_browser": {
        "unknown": 11
    },
    "by_ip": {
        "74.125.18.162": 1,
        "192.168.0.254": 3,
        "192.168.0.35": 7
    },
    "by_agent": {
        "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/12.0,
↪gzip(gfe) (via translate.google.com)": 1,
        "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/12.0
↪": 10
    },
    "by_hour": {
        "9": 3,
        "10": 4,
        "11": 1,
        "12": 3
    },
    "by_country": {
        "": 10,
        "US": 1
    }
}
```


If you use:

```
speed_shoot -f csv -g /usr/local/data/geoIP /var/www/apache/access*log
```

Your result is:


```
by_date,2012-5-3,11
total_line,11
ip_by_url,/favicon.ico,192.168.0.254,2
ip_by_url,/favicon.ico,192.168.0.35,2
ip_by_url,/,74.125.18.162,1
```


(continues on next page)

(continued from previous page)

```
ip_by_url,/,192.168.0.254,1
ip_by_url,/,192.168.0.35,5
by_status,200,7
by_status,404,4
by_dist,unknown,11
bytes_by_ip,74.125.18.162,151
bytes_by_ip,192.168.0.254,489
bytes_by_ip,192.168.0.35,1093
by_url,/favicon.ico,4
by_url,/,7
by_os,unknown,11
week_browser,3,unknown,11
by_referer,-,11
by_browser,unknown,11
by_ip,74.125.18.162,1
by_ip,192.168.0.254,3
by_ip,192.168.0.35,7
by_agent,"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/12.
↪0,gzip(gfe) (via translate.google.com)",1
by_agent,Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/12.0,
↪10
by_hour,9,3
by_hour,10,4
by_hour,11,1
by_hour,12,3
by_country,,10
by_country,US,1
```

Well I guess, it does not work because you first need to fetch geoIP data file:

```
mkdir data
wget -O- "http://www.maxmind.com/download/geoip/database/GeoLiteCountry/GeoIP.dat.gz"␣
↪| zcat > data/GeoIP.dat
```

Of course, this is the geoLite database, I don't include the data in the package since geoIP must be updated often to stay accurate.

Default path for geoIP is data/GeoIP.dat

8.2 Use as a script

speed shoot is in fact a template of how to use yahi as a module:

```
#!/usr/bin/env python
from archery.bow import Hankyu as _dict
from yahi import notch, shoot
from datetime import datetime


context=notch()
date_formater= lambda dt :"%s-%s-%s" % ( dt.year, dt.month, dt.day)
context.output(
    shoot(
        context,
        lambda data : _dict({
```

(continues on next page)

(continued from previous page)

```
            'by_country': _dict({data['_country']: 1}),
            'by_date': _dict({date_formater(data['_datetime']): 1 }),
            'by_hour': _dict({data['_datetime'].hour: 1 }),
            'by_os': _dict({data['_os_name']: 1 }),
            'by_dist': _dict({data['_dist_name']: 1 }),
            'by_browser': _dict({data['_browser_name']: 1 }),
            'by_ip': _dict({data['ip']: 1 }),
            'by_status': _dict({data['status']: 1 }),
            'by_url': _dict({data['uri']: 1}),
            'by_agent': _dict({data['agent']: 1}),
            'by_referer': _dict({data['referer']: 1}),
            'ip_by_url': _dict({data['uri']: _dict( {data['ip']: 1 })}),
            'bytes_by_ip': _dict({data['ip']: int(data['bytes'])}),
            'week_browser' : _dict({data['_datetime'].weekday():
                _dict({data["_browser_name"] :1 })}),
            'total_line' : 1,
        }),
    ),
)
```

CHAPTER 9

Installation

easy as:

```
pip install yahi
```

or:

```
easy_install yahi
```

CHAPTER 10

Recommanded usage

- for basic log aggregation, I do recommand using command line;
- for one shot metrics I recommend an interactive console (bpython or ipython);
- for specific metrics or elaborate filters I recommand using the API.

CHAPTER 11

CHANGELOG

11.1 0.1.3

Adding varnish incomplete regexp for log parsing (I miss 2 fields)

CHAPTER 12

What's new?

12.1 0.1.1

- bad url for the demo

12.2 0.1.0

- it is NEW, seen on TV, and is guaranteed to make you tenfolds more desirable.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search