

# Malware Classifier From Network Capture

---

## Introduction

"Malware Classifier is a simple free software project done during an [university workshop of 4 hours](#). The objective of the 4 hours workshop was to introduce network forensic and simple techniques to classify malware network capture (from their execution in a virtual machine). So the software was kept very simple while using and learning existing tools ([networkx](#), [redis](#) and [Gephi](#))."

We get interested in the different ways of classifying the malware and we decided to focus more on this subject.

First we will focus on the manipulations and installations needed to start the study. We use a Virtual Machine running the OS Ubuntu 14.04 amd64 bit. The computer we use is only 4Go memory and 2Go of them is allocated for the VM.

## 1) Requirements

*Use `pip install -r REQUIREMENTS` or `apt-get install`*

### 1.1) Python 2.7

```
coralie@coralie-VirtualBox:~/Projet_Dulanoys$ sudo apt-get install python
```

```
python est déjà la plus récente version disponible.
```

```
coralie@coralie-VirtualBox:~$ python -V
Python 2.7.6
```

### 1.2) networkx and redis modules

"NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks."

Installation of networkx :

```
coralie@coralie-VirtualBox:~/Projet_Dulanoys$ sudo pip install networkx
Downloading/unpacking networkx
  Downloading networkx-1.9.1-py2.py3-none-any.whl (1.2MB): 1.2MB downloaded
Requirement already satisfied (use --upgrade to upgrade): decorator>=3.4.0 in /usr/lib/python2.7/dist-packages (from networkx)
Installing collected packages: networkx
Successfully installed networkx
Cleaning up...
```

We use `networkx_reference` in order to understand how `networkx` works. (eg. <https://networkx.github.io/download.html>)

Simple use of `networkx` :

```
import redis
import networkx as nx

#test redis
r=redis.StrictRedis(host='localhost',port=6379,db=0)
r.set('foo','bar')
print(r.get('foo'))

#test networkx
G=nx.Graph()
G.add_edge(1,2)
G.add_edge(2,3,weight=0.9)
print(G.adj)
```

Result :

```
{1: {2: {}}, 2: {1: {}, 3: {'weight': 0.9}}, 3: {2: {'weight': 0.9}}}
```

This line can be read as follow :

- node number 1 is adjacent to number 2,
- node number 2 is adjacent to number 1,
- node number 2 is adjacent to number 3, with a weight of 0.9 on the edge
- node number 3 is adjacent to number 2, with a weight of 0.9 on the edge

### 1.3) tshark (part of Wireshark)

```
coralie@coralie-VirtualBox:~/Projet_Dulanoy$ sudo apt-get install tshark
```

```
tshark est déjà la plus récente version disponible.
```

```
coralie@coralie-VirtualBox:~/Projet_Dulanoy$ tshark -v
TShark 1.10.6 (v1.10.6 from master-1.10)
```

## 1.4) a Redis server

redis in /usr/local/lib/python2.7/dist-packages

```
coralie@coralie-VirtualBox:~/Bureau/Malwares/redis-2.8.19/src$ ./redis-server
[3419] 17 Mar 14:47:14.135 # Warning: no config file specified, using the default
t config. In order to specify a config file use ./redis-server /path/to/redis.co
nf
[3419] 17 Mar 14:47:14.139 # You requested maxclients of 10000 requiring at leas
t 10032 max file descriptors.
[3419] 17 Mar 14:47:14.139 # Redis can't set maximum open files to 10032 because
of OS error: Operation not permitted.
[3419] 17 Mar 14:47:14.139 # Current maximum open files is 1024. maxclients has
been reduced to 4064 to compensate for low ulimit. If you need higher maxclients
increase 'ulimit -n'.
```

In order to see what is happening in the database of redis, we start a new redis client :

```
coralie@coralie-VirtualBox:~/Bureau/Malwares/redis-2.8.19/src$ ./redis-cli
127.0.0.1:6379> █
```

## 2) How to use the Malware Classifier

"You'll need of a set of network packet captures. In the workshop, we use a dataset with more than 5000 pcap files generated from the execution of malware in virtual machines."

Here we are working with the archive named pcap\_2012-09-16.zip

```
...
0580c82f6f90b75fcf81fd3ac779ae84.pcap
05a0f4f7a72f04bda62e3a6c92970f6e.pcap
05b4a945e5f1f7675c19b74748fd30d1.pcap
05b57374486ce8a5ce33d3b7d6c9ba48.pcap
05bbddc8edac3615754f93139cf11674.pcap
...

coralie@coralie-VirtualBox:~/Projet_Dulanoy$ ls
pcap_2012-09-16.zip test.py
coralie@coralie-VirtualBox:~/Projet_Dulanoy$ unzip pcap_2012-09-16.zip
Archive:  pcap_2012-09-16.zip
[pcap_2012-09-16.zip] pcap/00357a39128552887471e1bc094929de.pcap password:
  inflating: pcap/00357a39128552887471e1bc094929de.pcap
  inflating: pcap/0048a43bdc064e85bdaad7f6c29ff6b4.pcap
  inflating: pcap/004df4bdb9dcce53a33515aaa666d084.pcap
  inflating: pcap/00560cc7492a3f791a51896c6a538ab2.pcap
  inflating: pcap/006f919d07094f253f1b6fad93ad95d6.pcap
```

password = "infected"

"The filename includes the MD5 malware executed in the virtual machine."

Once unzip, we notice that the extended content include an "index.txt" file :

```
coralie@coralie-VirtualBox:~/Bureau/Projet_Dulanoy$ cat index.txt | head -20
This is an index file that provides detections, in order, of each
contained file. The scanner used for a given detection is also listed.

Each detection is of the form:
[Sample Name]: [Detection] ([Scanner])

For Example:
6aabbccddeeff0011223344556677884.EX$: Trojan.Win32 (Sunbelt)

Where no detection is present, "No Detection" is given without vendor.

Indexed samples shall follow:
-----

5db92fa00ecfa812f13cdfc9f0a5ecfc: Trojan.Win32.Generic!BT (sunbelt)
7b350dfd3ba1f82e89d25bdf5a7cc4e2: Trojan.Win32.Agent.njc (v) (sunbelt)
c48a376645b4b5285f301a5f251f8880: Trojan.Win32.Jorik.Mokes.cai (sunbelt)
```

As the index file explains, different kind of malware are present in the archive. We only want the pcap files in the folder pcap so we put the index file out of the pcap folder.

The pcap files included in the folder pcap are named with their md5. We want to extract the first part of the name, which is the md5 value of the file (without the extension).

We have a program which extract the md5 part of the name of the file : `extract_md5_from_pcap.py`

<pre>#need to catch the MD5 filename of the malware (from the pcap filename) #action : print the md5 part of the pcap files # use : pcap\$ ls -1   parallel -gnu " python ../extract_md5_from_pcap.py -f {1}"  import argparse import sys  argParser=argparse.ArgumentParser(description='Pcap classifier') argParser.add_argument('-f',action='append',help='Filename') args=argParser.parse_args()  if args.f is not None:      md5 = args.f[0].split(".")[0]     print(md5)  else:     argParser.print_help()</pre>	<pre>00357a39128552887471e1bc094929de 0048a43bdc064e85bdaad7f6c29ff6b4 004df4bdb9dce53a33515aaa666d084 00560cc7492a3f791a51896c6a538ab2 006f919d07094f253f1b6fad93ad95d6 00917c4ee6c669860c2558c174d8fe0e 0091eb6930b2f3a405fb5c3077bee28e 009c41d46680283348b8fcab26064b5e 00a5380865d99ec7c6338e7cf930f3f1 00aede4e86394b25bf62af07734260d7 00b0b60d23a49998bfff0971f73ae4f2 00e43c91d2f8d7d136c7e99a69f9b743 010a7b6626f7f2a67a2de5877b9d2be4 010fb2018309ca09d338a680b174ab24 012e7b1e80dfbcdd2d4c7168492940a0</pre>
--	--

We can now use redis to store these md5.

We check that the set which is going to be insert in redis doesn't exist :

```
127.0.0.1:6379> SMEMBERS md5_list
(empty list or set)
```

We have a program which insert the md5 part of the name of the files in redis : `import_md5_redis.py`

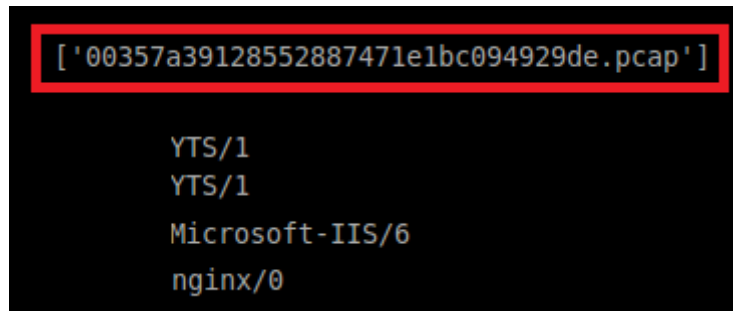
<pre>#action : insert the md5 part of the pcap files in a redis database # use : pcap\$ ls -1   parallel -gnu " python ../import_md5_redis.py -f {1}"  import argparse import sys import redis  argParser=argparse.ArgumentParser(description='Pcap classifier') argParser.add_argument('-f',action='append',help='Filename') args=argParser.parse_args()  r=redis.StrictRedis(host='localhost',port=6379,db=0)  if args.f is not None:      md5 = args.f[0].split(".")[0]     print(md5)     r.sadd('md5_list',md5)  else:     argParser.print_help()</pre>	<p>We check that the md5 have been correctly import in redis :</p> <pre>127.0.0.1:6379&gt; SMEMBERS md5_list 1) "00aede4e86394b25bf62af07734260d7" 2) "009c41d46680283348b8fcab26064b5e" 3) "00e43c91d2f8d7d136c7e99a69f9b743" 4) "00b0b60d23a49998bfff0971f73ae4f2" 5) "012e7b1e80dfbcdd2d4c7168492940a0" 6) "0048a43bdc064e85bdaad7f6c29ff6b4" 7) "00560cc7492a3f791a51896c6a538ab2" 8) "0091eb6930b2f3a405fb5c3077bee28e" 9) "010a7b6626f7f2a67a2de5877b9d2be4" 10) "006f919d07094f253f1b6fad93ad95d6" 11) "010fb2018309ca09d338a680b174ab24" 12) "00357a39128552887471e1bc094929de" 13) "00917c4ee6c669860c2558c174d8fe0e" 14) "004df4bdb9dce53a33515aaa666d084" 15) "00a5380865d99ec7c6338e7cf930f3f1"</pre>
--	---

We will try to classify them :

"If you want to classify malware communications based on the Server HTTP headers of the (potential) C&C communication."

```
cd capture
ls -1 . | parallel --gnu "cat {1} | tshark -E header=yes -E separator=, -Tfields -e http.server -r {1} | python ./bin/import.py -f {1} "
```

-E : field print option



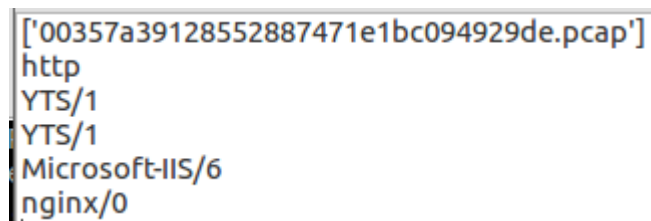
A terminal window with a black background. The first line shows a file path enclosed in red square brackets: ['00357a39128552887471e1bc094929de.pcap']. Below this, several lines of text are displayed, representing HTTP headers: YTS/1, YTS/1, Microsoft-IIS/6, and nginx/0.

For the first pcap :

```
$ ls | head -1 | parallel --gnu "cat {1} | tshark -E header=yes -E separator=, -Tfields -e http.server -r {1} | python ../import_sans_redis.py -f {1} " > ../pcap1.txt
```

We clean the file with sed :

```
$ sed -i "/^[ \t]*$/d" ../pcap1.txt
```



A terminal window showing the output of the sed command. The first line is the file path ['00357a39128552887471e1bc094929de.pcap']. Below it, the HTTP headers are listed: http, YTS/1, YTS/1, Microsoft-IIS/6, and nginx/0.

We simplify in one line :

```
coralie@coralie-VirtualBox:~/Bureau/Projet_Dulanoy/pcap$ ls | head -1 | parallel --gnu "cat {1} | tshark -E header=yes -E separator=, -Tfields -e http.server -r {1} | python ../import_sans_redis.py -f {1} > ../pcap0.txt " && sed -i "/^[ \t]*$/d" ../pcap0.txt
```

```
import_sans_redis.py
Fichier  Édition  Rechercher  Options  Aide
#need to catch the MD5 filename of the malware (from the pcap filename)
import argparse
import sys

argParser=argparse.ArgumentParser(description='Pcap classifier')
argParser.add_argument('-f',action='append',help='Filename')

args=argParser.parse_args()

if args.f is not None:
    filename=args.f
    #print(filename)
    print(args.f[0].split(".")[0])

    for line in sys.stdin:
        print(line.split(".")[0])
else:
    argParser.print_help()
```

Result :

```
pcap0.txt
Fichier  Édition  Rechercher  Options  Aide
00357a39128552887471e1bc094929de
http
YTS/1
YTS/1
Microsoft-IIS/6
nginx/0
```

Now, we try to import the result in redis :

pour chaque pcap on affiche son n° ,nom et lance le tshark

-> sudo pip install pyshark -> sudo pip install logbook -> sudo pip install trollius -> sudo pip install py -  
> sudo pip install lxml.objectify

Let's have a look on the 00357a39128552887471e1bc094929de.pcap :

```
coralie@coralie-VirtualBox:~/Bureau/Projet_Dulanoy/pcap$ sudo wireshark 00357a39128552887471e1bc094929de.pcap
```

179	47.462771	172.16.2.8	216.115.107.2	TCP	40	1050	>	80	[RST, ACK]	Seq=266	Ack=14702	Win=0	Len=0
180	47.462879	172.16.2.8	211.108.61.17	TCP	40	1051	>	80	[RST, ACK]	Seq=100	Ack=330	Win=0	Len=0
181	47.463196	172.16.2.8	121.78.127.50	TCP	40	1052	>	80	[RST, ACK]	Seq=101	Ack=112067	Win=0	Len=0

► Frame 1: 48 bytes on wire (384 bits), 48 bytes captured (384 bits)

This pcap file contained 181 packets. We use the wireshark filter to have information on http.headers :

Filter:		http.server		▼		Expression...		Clear		Apply		Enregistrer	
No.	Time	Source	Destination	Protocol	Length	Info							
6	0.522877	216.115.10	172.16.2.81	HTTP	893	HTTP/1.1 302 Found (text/html)							
25	1.821452	216.115.10	172.16.2.81	HTTP	974	HTTP/1.1 404 Not Found (text/html)							
31	2.345848	211.108.61	172.16.2.81	HTTP	369	HTTP/1.1 200 OK (text/plain)							
176	17.326930	121.78.127	172.16.2.81	HTTP	716	HTTP/1.1 200 OK (image/gif)							

Only 4 packets mentionned some http server header. This is what we discover thanks to the tshark filter command.

degager ligne vide et afficher nb lignes. dans redis on stock un ens de http server par md5.

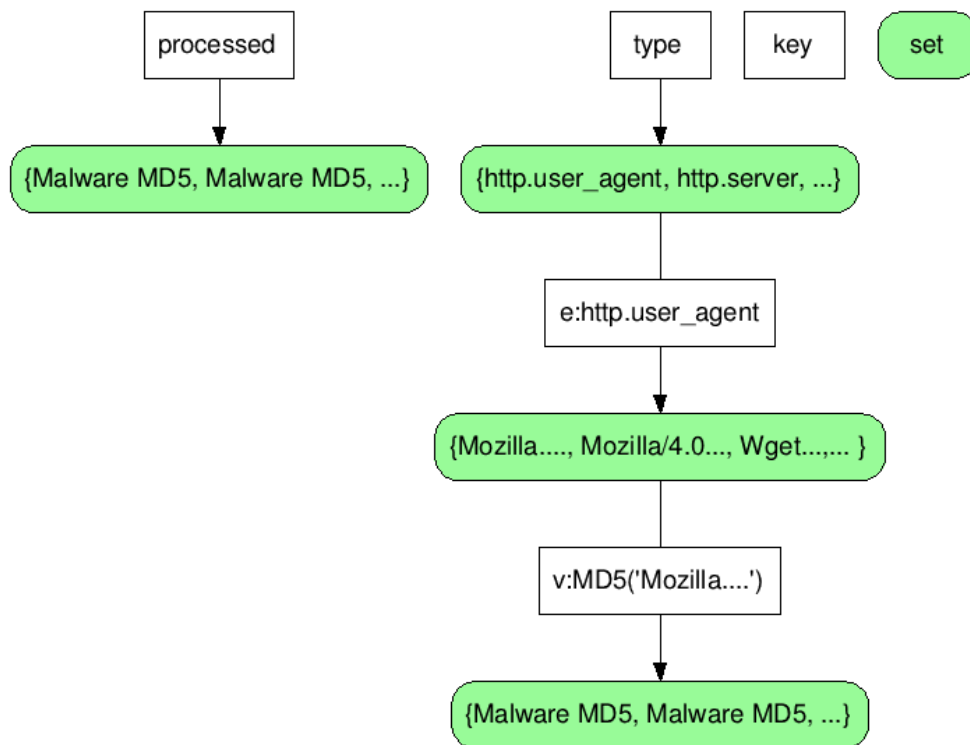
"You can add additional attributes like any fields from the dissectors available within tshark (tshark -G fields). You can add additional fields in the command above. This will update the redis data structure. Then when you have enough attributes, you can dump a graph out of the relationships between the attributes and the malware packet captures."

```
python ./bin/graph.py
```

"graph.py generates a GEXF file that you can import in gephi."

## Redis data structure





## Notes for the student

---

Check the git log and the commits, these include the steps performed during the workshop especially regarding the improvement of the Python scripts.

Generate graph relationship from Redis database : bin/graph.py

```

import redis
import networkx as nx
import hashlib
r = redis.StrictRedis(host='localhost', port=6379, db=0)
g = nx.Graph()
for malware in r.smembers('processed'):
    g.add_node(malware)
for fieldtype in r.smembers('type'):
    g.add_node(fieldtype)
for v in r.smembers('e:'+fieldtype.decode('utf-8')):
    g.add_node(v)
ehash = hashlib.md5()
ehash.update(v)
  
```

```
ehhex = ehash.hexdigest()
for m in r.smembers('v:'+ehhex):
    print (m)
    g.add_edge(v,m)
nx.write_gexf(g,"graph.gexf")
```

Workshop - 5th step added : import.py :

```
import argparse
import sys
import redis
import hashlib

argParser = argparse.ArgumentParser(description='Malware classifier')
argParser.add_argument('-f', action='append', help='Filename')
args = argParser.parse_args()
r = redis.StrictRedis(host='localhost', port=6379, db=0)
if args.f is not None:
    md5 = args.f[0].split(".")[0]
    r.sadd('processed', md5)
    lnumber = 0
    fields = None
    for line in sys.stdin:
        if lnumber == 0:
            fields = line.rstrip().split(",")
            for field in fields:
                r.sadd('type', field)
        else:
            elements = line.rstrip().split(",")

            i = 0
            for element in elements:
                try:
                    r.sadd('e:'+fields[i], element)
                    #
                    ehash = hashlib.md5()
                    ehash.update(element.encode('utf-8'))
                    ehhex = ehash.hexdigest()
                    if element is not "":
                        r.sadd('v:'+ehhex, md5)
                except IndexError:
                    print("Empty fields")
                i = i+1
            lnumber = lnumber + 1
    else:
        argParser.print_help()
```

Overview of the Redis data structures : redis-datastruct.dot :

```
digraph g{
  a -> b
  a [label="processed",shape=box,fillcolor="white",style="filled"];
  b [label="{Malware MD5, Malware MD5,
...}",shape=box,fillcolor="palegreen",style="filled,rounded"];

  c -> d
  c [label="type",shape=box,fillcolor="white",style="filled"];
  d [label="{http.user_agent, http.server,
...}",shape=box,fillcolor="palegreen",style="filled,rounded"];
  d -> g [dir=none];
  g [label="e:http.user_agent", shape=box];
  g -> h
  h [label="{Mozilla..., Mozilla/4.0..., Wget...,...
...}",shape=box,fillcolor="palegreen",style="filled,rounded"];
  h -> e [dir=none];
  e -> f
  e [label="v:MD5('Mozilla...')",shape=box,fillcolor="white",style="filled"];
  f [label="{Malware MD5, Malware MD5,
...}",shape=box,fillcolor="palegreen",style="filled,rounded"];

  z [label="key", shape=box];
  y [label="set", shape=box, fillcolor="palegreen",style="filled,rounded"];

}
```