An automated behaviour-based malware analysis method based on free open source software.

Sebastian Rubio-Ayala[1]

[1]Universitat Oberta de Catalunya
[*]srubioay@uoc.edu

Abstract: Most of the currently proposed solutions for automated malware behaviour-analysis in the literature are fully or partly based on commercial software or in obsolete software. An automated behaviour-based method of analysis based on free open source software as alternative to the existing ones is proposed here. This method will help to determine if a software sample is malware that can allow in a later step to generate static fingerprints for IPS/IDS and Antivirus software. The results of some experiments based on the proposed model are commented.

Keywords: behaviour analysis, malware, FOSS, Cuckoo, Weka, machine learning

1. Introduction

Typically the antivirus programs and anti-malware suites identify the malware software by the use of static patterns. Before they can identify the malware software a previous analysis must have been done to identify it and get static fingerprints. By the use of behavioral detection techniques it can be observed how the program executes and try to identify malware by looking for suspicious behaviours.

To create a malware analysis laboratory is a very important task, mainly for security consultant enterprises and medium and large enterprises with its own IT security department.

There are several software tools that help to automate the creation of sandboxes and obtain behaviour data of a software. This data can be analysed and used to generate signatures that could be used later in antivirus or IPS/IDS software or anti-malware suits using static analysis.

A behaviour-based analysis system based on free open source software (FOSS) tools will be proposed in order to create models based in machine learning (ML) techniques to determine if a software sample is malware or not in base of its behaviour patterns. Some experiments based in the proposed model were done and are commented in this paper.

2. Background

There are currently many studies that discuss automated malware analysis in different ways [1]. Most of them use a combination of static analysis and dynamic analysis. Static methods have the problem of obfuscation. Saed et al. [2] propose a novel technique that extracts the semantics of binary code in term of both data and control flow. It was called semantic flow graphs(SFG) [2] and can be used to detect malware in cases where refactoring and obfuscation make difficult to analyse the code. Even of that static analysis could help to analyse some files it could not determine malware code in highly obfuscated or deeply packed binaries.

Dynamic analysis methods have a high accuracy, but they produce higher overhead and cannot cover all malicious behaviours a malware could have, as it could not test all the possible cases and combinations of events [3]. In these cases a static analysis can help to get a more robust detection system.

Malware needs one or more sequential processes to achieve its purpose. This sequential processes is defined as behaviour chain [3]. The creation of behaviour chains models [3] in conjunction with other models like machine learning could be used to detect and classify malware with a high accuracy.

Sanjay et al. [4] proposes a method using the WEKA data mining tool [5] to detect malware based on machine learning. Zahra et al. [6] proposes a malware detection system based on API calls, their arguments and return values. Aziz et al. [7] proposes a malware detection system based in the usage of the file system, memory, network and Windows registry. Ding et al. [8] proposes a malware detection system based on the analysis of static system calls. Naoto et al. [9] proposes a malware classification using APIs in Initial Behaviour. Saja et al. [1] proposes a malware detection system based in the Windows registry monitoring.

Dynamic analysis is done by the use of sandboxing techniques. The software tool MAAR [6] provides a system model to monitor, analyse, model and monitor Portable Executables (PEs) files used in 32-bit and 64-bit versions of Windows. Currently there are free software tools that could be used to automate the dynamic analysis of malware [10][11]. Some evasion techniques and countermeasures are also used by malware to avoid it detection in sandboxes environments [12].

Most of the proposed solutions for automated malware behaviour-analysis in the literature are fully or partly based on commercial software or in obsolete software.

3. System design

3.1. Analysed Software

In order to automatize the Sandbox creation task and the behaviour data extraction process for a later analysis the next software was analysed.

3.1.1. Cuckoosandbox:

Cuckoo sand box is a malware analysis system and it is currently under active development. Cuckoo is completely written in Python and currently it only fully support Python 2.7. It supports the next OSes:

- Host system: GNU/Linux (Debian or Ubuntu preferably), also tested to run in Windows 7 and Mac OS X.
- Sandboxed OS: Windows (Windows XP and Windows 7 64bits recommended), Mac OS X, GNU/Linux, and Android.

It allows to:

- Analyse many different malicious files (executables, document exploits, Java applets) as well as malicious websites, in Windows, OS X, Linux, and Android virtualized environments. Trace API calls and general behaviour of the file.
- Dump and analyse network traffic, even when encrypted.
- Perform advanced memory analysis of the infected virtualized system with integrated support for Volatility.

It can retrieve the following type of results:

- Traces of calls performed by all processes spawned by the malware.
- Files being created, deleted and downloaded by the malware during its execution.
- Memory dumps of the malware processes.
- Network traffic trace in PCAP format.

– Screenshots taken during the execution of the malware.
– Full memory dumps of the machines.

It can be used to analyse:

– Generic Windows executables
– DLL files
– PDF documents
– Microsoft Office documents
– URLs and HTML files
– PHP scripts
– CPL files
– Visual Basic (VB) scripts
– ZIP files
– Java JAR
– Python files
– Other type of files.

Yara and Pydeep are optional plugins. In order to use the Django-based Web Interface, MongoDB is required

3.1.2. Yara:

Yara is a tool for data analysis that allows to apply predefined rules to search patterns. It is currently under active development. It Allows to create descriptions of malware families based on textual or binary patterns.
It is BSD 3 licensed and runs on Windows, Linux and Mac OS X. It can work also as a cuckoo plugin.

3.1.3. Malheur:

Malheur is a tool for the automatic analysis of malware behavior. It is currently not under active development. It works with popular malware sandboxes (e.g. CWSandbox, Anubis, Norman Sandbox, Joebox) Can be used for:

– Extraction of prototypes
– Clustering of behaviour
– Classification of behaviour
– Incremental analysis

There is a modified version of Cuckoo that supports Malheur.

3.1.4. Other sandboxing software:

There are others malware analysis solutions in the market. Most of them are offered as sample analysis service and some of them are not currently under active development. E.g. Buster Sandbox Analyzer, Zero Wine Malware Analysis Tool, CWSandbox, Anubis, Norman Sandbox, Joebox, VMRay, ThreatAnalyzer, Norman Sandbox, GFI Sandbox, ValidEDGE.

4. Proposed method

The proposed method consist in the use of the FOSS Tools indicated in the Table 1 to execute an automated analysis of software samples. Once the results are obtained they are processed to extract relevant data. After that, the resulting data is analysed to determine if the sample could be considered as malware in base of it registered behaviour.

Table 1 List of main FOSS Tools used

Name	Description
Cuckoosandbox	Malware analysis system
VirtualBox	x86/AMD64/Intel64 virtualization product
WEKA	Data mining and machine learning tool

Cuckosandbox was selected due it is a free open licensed project under active development that can be used to automatically extract dynamic behaviour data from software executed under a controlled environment. Oracle VirtualBox was selected as virtualization environment as it is a free open software project (GPL licensed) widely used and supported with cuckoo sandbox. Cuckoosandbox allows to submit software to be analysed. The results of the analysis are stored in JavaScript Object Notation (JSON) format. In order to analyse a wide amount of data obtained from the cuckoo software the Weka software was selected. Weka is

GPL licenced and contains a collection of visualization tools and algorithms for data analysis and predictive modelling. It has an user graphical interface as well as a command line one. Weka is written in Java and allows to integrate its components as java classes. Weka has support to import JSON files, but the cuckoo JSON file is complex, in same cases the resulting file can reach a big size (over 1Gb) and it is not directly importable by Weka. A custom python program is used in order to process the cuckoo results. It extracts relevant data from the JSON files and export them in Attribute Relationship File Format (ARFF) format, which is the Weka native format. In this format a list of attributes considered relevant for malware analysis or detection will be provided to Weka. The liac-arff [13], bigjson[14], python libraries and custom functions are used to import and export the data. Python provides a flexible and powerful environment to create rules to extract the relevant data from the JSON files.

To determine the independent variables that causes the malware behaviour could be a difficult task and Weka with the obtained data from the custom Python program could be used to facilitate this by the analysis of the data and study of the models to classify the software.

Table 2 Used FOSS Licence

Name	Version	Licence
Cuckoosandbox	2.0	GPLv3
VirtualBox	5.1.16	GPLv2
WEKA	3.8.1	GPLv3

Listing 1. *Example of an ARFF file*

```
% ARFF example file
@RELATION normal

@ATTRIBUTE file_created NUMERIC
@ATTRIBUTE regkey_written
    NUMERIC
@ATTRIBUTE dll_loaded NUMERIC
@ATTRIBUTE connects_hosts
    NUMERIC
@ATTRIBUTE regkey_opened
    NUMERIC
@ATTRIBUTE command_line NUMERIC
@ATTRIBUTE file_written NUMERIC
@ATTRIBUTE regkey_deleted
    NUMERIC
@ATTRIBUTE class {malware,
    normal}

@DATA
1,1,1,0,1,1,1,0,normal
1,1,1,0,1,1,1,0,malware
1,1,1,0,1,1,1,0,normal
1,1,1,0,1,0,1,0,normal
0,0,0,0,0,0,0,0,normal
1,0,1,0,1,0,1,0,normal
1,1,1,0,1,1,1,1,malware
1,1,1,0,1,1,1,0,normal
1,0,1,0,1,0,1,0,malware
%
%
%
```

4.1. System setup

Cuckoosandbox is used in conjunction with VirtualBox to analyse software samples.
Before start working with couckoosandbox a system set up is needed.
Cuckoosandobox's documentation [10] explains how to setup the system to work in conjunction with VirtualBox and how to submit analysis requests. Cuckoosandbox also allow to work in conjunction with other virtualization tools or even with real systems used as sandboxes.
A special configuration of the sandbox system is required for the selected setup based in virtualbox. The steps to configure the guest system to work properly with the cuckoo agent are described in the cuckoosandbox documentation. The samples under analysis can be submitted to cuckoosandbox by the use of the submit.py Python script included in cuckoo *utils* folder.

Listing 2. *E.g. Submit all .exe files in the current folder to cuckoo*

```
for f in *.exe; do
 ~/cuckoo/utils/submit.py "$f";
done
```

Cuckoo sandbox also have a web interface and an API to help to automatize the submit task. Once the analysis of a sample is finished a report is generated by cuckoosandbox.

Cuckoo generates JSON files as output.

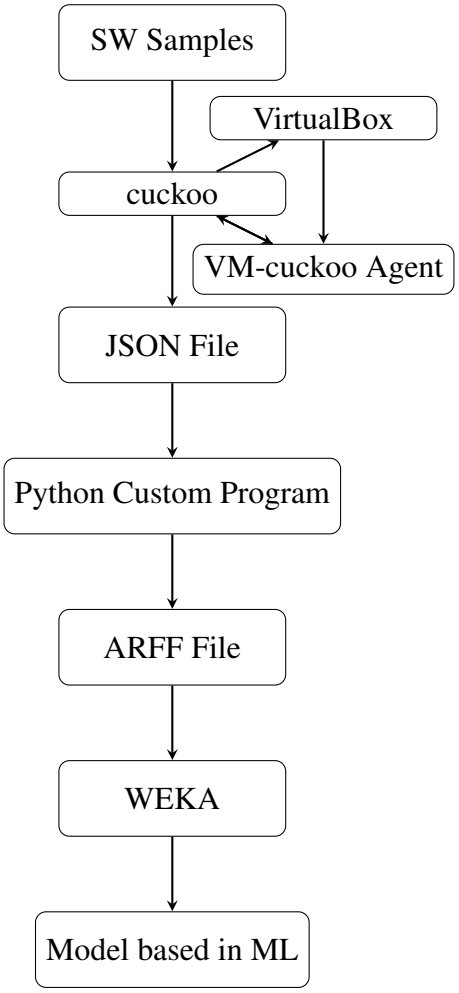
Listing 3. *Structure of a typical cuckoo JSON result file*

```
{  info{...}
   signatures{...}
   target{...}
   virustotal{...}
   network{...}
   behavior{
      generic{...}
      apistats{...}
      processes{...}
      processstree{...}
      summary{...}}
   debug{...}
   strings{...}
   metadata{...}}
```

Listing 4. *Example of python function to extract information about the accessed registry keys from the cuckoo JSON file*

```
def analizeReg(json_data, D,
    regList,firstIndex):
 try:
  for i in json_data["behavior"
      ]["summary"]["regkey_read"
      ]:
   idx = firstIndex
    for k in regList:
     if i == k:
      D[idx]=1
     idx += 1
 except Exception:
  raise
 return D
```

The JSON report is adapted by a python script to ARFF format. The Python script produces an output in Attribute-Relation File Format (ARFF), that can be processed by the WEKA software package.



5. Experiment results

In order to test the proposed method a series of experiments were executed and their results are exposed here.

5.1. Test equipment

The system used for testing was an Intel i5 M480 CPU PC, 4GB of RAM and Ubuntu 16.04.2 LTS GNU/Linux as Operating System. A Windows 7 professional SP1 (64 bits) virtual machine (VM) was used by cuckoo to

execute the binary files with the samples under evaluation. The virtual machine (VM) was properly configured to work with cuckoo Agent.

Samples of software were obtained to be analysed:

- 997 samples of Windows PE Malware and 38152 samples of CryptoRansom malware from the VirusShare [15] website with malware samples.
- 54 software samples from the CNET.com [16] website with free software download.
- 549 software samples from the Freewarefiles.com [17] website with free software download.

Cuckoosandbox, Virtualbox and Weka were installed as indicated in the system setup section.

5.2. Data extraction

For these experiments, instances of ransomware malware, other types of malware and normal software were used. Malware classified samples could include ransomware malware samples as it is a specific type of malware. We could use this. All the samples under analysis run in the virtualized environment.

The samples were placed in a folder and submitted to cuckoo for testing with the next results:

- Some of the files in the ransomware dataset were not Windows PE files and the analysis was not performed.
- A report in JavaScript Object Notation (JSON) format was generated. After a analysis of 3016 files, 19 files were over 250MB.
- Due the limited resources of the used equipment the JSON files with size over 250MB were discarted and only 3016 files were analysed with cuckoo sandbox.

The dataset used as input for the machine learning scheme is composed by a set of instances. Each instance is an independent example of the concept to be learned. Preparing the input for a data mining investigation consumes the bulk of the effort invested in the entire data mining process [18].

5.3. First data extraction

The resulting data from the cuckoo analysis was processed to obtain the next attributes:

- file created NUMERIC
- regkey written NUMERIC
- dll loaded NUMERIC
- connects hosts NUMERIC
- regkey opened NUMERIC
- command line NUMERIC
- file written NUMERIC
- regkey deleted NUMERIC
- file exists NUMERIC
- mutex NUMERIC
- file failed NUMERIC
- resolves host NUMERIC
- guid NUMERIC
- file read NUMERIC
- regkey read NUMERIC
- directory enumerated NUMERIC

This attributes were selected as they describe a set of actions registered by cuckoo and that could be interesting to evaluate it relation with the malware classification. All of them are set as NUMERIC type and normalized as 0 or 1. The software under test was classified as {ransomware, malware, normal}. The software obtained from the VirusShare PE malware samples was classified as malware. The software obtained from the VirusShare ransomware malware samples was classified as ransomware. The software obtained from other sources was classified as malware or normal depending if

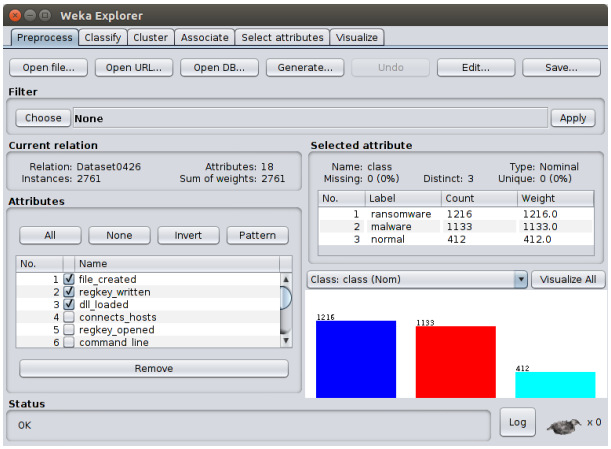


Fig. 1. *Weka Explorer - Preprocess*

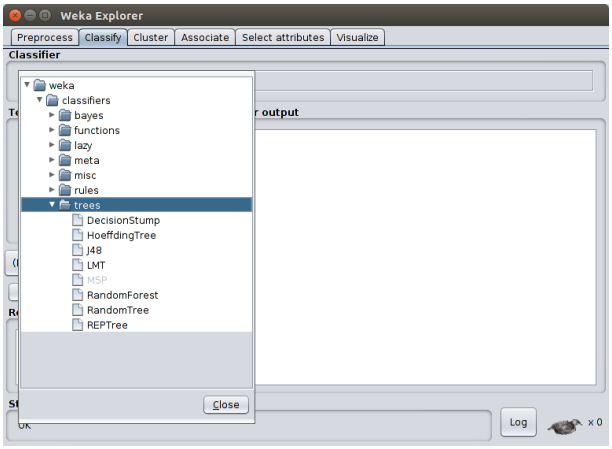


Fig. 2. *Weka Explorer - Classifier*

there was a positive value in the Virustotal database. This data was extracted from the JSON file.

Listing 5. *Software classification function*

```
def virustotalSum(json_data,D):
  try:
    if json_data["virustotal"]["
       summary"]["positives"] ==
       0:
     D.append('normal')
    else:
     D.append('malware')
  except Exception:
    raise
  return
```

There are usually lots of possible attributes which could be directly extracted from the cuckoo result data and also others derived from association rules and others data processing techniques. Weka was used for investigating this data and evaluate how well each attribute can be predicted from the others. It also allows to find association rules and clustering. With the help of Weka explorer (fig.1) it is possible to do a pre-analysis of the data. Data cleaning is a time-consuming task, mainly with large dataset, but absolutely necessary.

5.3.1. First dataset analysis results: To analyse the dataset the cross-validation Weka option was used. A value of 10-fold generates 10 cross-validated trees to estimate "out-of-sample" error. This help to ensure that the results were representative. A decision tree based in the C4.5 learner algorithm is implemented in Weka as J4.8 algorithm. C4.5 is a widely used algorithm for machine learning. J4.8 will be used to evaluate the dataset.

The results after executing the J4.8 classifier to the dataset is shown in Fig.3.

5.4. Second data extraction

In Salehi et al.[6] it was considered that the use of relevant API calls was a relevant indicator to characterize malware software. Only a subset of 126 API calls of advapi32.dll, kernel32.dll, ntdll.dl1, user32.dell, wininet.dll and ws2_32.dll were considered in that case.

In this case a list with all the dlls used by software samples was extracted and the resulting data was analysed in Weka. 213 API Calls were extracted and used as numeric attributes with a value of 1 if the API call was present in the behaviour section of the JSON file or 0 otherwise.The generated ARFF file was processed with Weka an analysed with the J4.8 classifier algorithm. Weka Classifier Tree Visualizer allows to see the decision tree generated by the algorithm and also other relevant graphical representations and statistical data.

```
=== Run information ===
Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     Dataset0426
Instances:    2761
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===
J48 pruned tree
------------------
Number of Leaves  :  71
Size of the tree :  141

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1958               70.9163 %
Incorrectly Classified Instances       803               29.0837 %
Kappa statistic                          0.5099
K&B Relative Info Score             108837.9181 %
K&B Information Score                 1586.5704 bits      0.5746 bits/instance
Class complexity | order 0            4025.2765 bits      1.4579 bits/instance
Class complexity | scheme            56466.8785 bits     20.4516 bits/instance
Complexity improvement     (Sf)     -52441.602  bits    -18.9937 bits/instance
Mean absolute error                      0.2715
Root mean squared error                  0.3801
Relative absolute error                 66.1843 %
Root relative squared error             83.9232 %
Total Number of Instances             2761

=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC   ROC Area PRC Area Class
0.752   0.177   0.770     0.752  0.761     0.577 0.824    0.801    ransomware
0.835   0.277   0.677     0.835  0.748     0.549 0.815    0.713    malware
0.238   0.034   0.554     0.238  0.333     0.297 0.728    0.369    normal
---------------------------------------------------------------------------------
0.709   0.197   0.700     0.709  0.692     0.524 0.806    0.701    Weighted Avg

=== Confusion Matrix ===
   a   b   c   <-- classified as
 914 277  25 |   a = ransomware
 133 946  54 |   b = malware
 140 174  98 |   c = normal
```

Fig. 3. *Dataset0426 J48 Result*

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     normal
Instances:    2967
Attributes:   214
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------
Number of Leaves  :  90
Size of the tree :  179

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2416               81.4291 %
Incorrectly Classified Instances       551               18.5709 %
Kappa statistic                          0.6926
Mean absolute error                      0.1679
Root mean squared error                  0.3142
Relative absolute error                 41.6287 %
Root relative squared error             69.9712 %
Total Number of Instances             2967

=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
0,835    0,054    0,935      0,835    0,882      0,789    0,916     0,889     ransomware
0,876    0,199    0,730      0,876    0,797      0,660    0,873     0,790     malware
0,573    0,040    0,698      0,573    0,629      0,580    0,849     0,558     normal
---------------------------------------------------------------------------------------
0,814    0,107    0,824      0,814    0,814      0,711    0,890     0,805     Weighted Avg

=== Confusion Matrix ===

    a    b    c   <-- classified as
 1189  215   20 |    a = ransomware
   58  991   82 |    b = malware
   25  151  236 |    c = normal
```

Fig. 4. *Dataset0428 J48 Result*

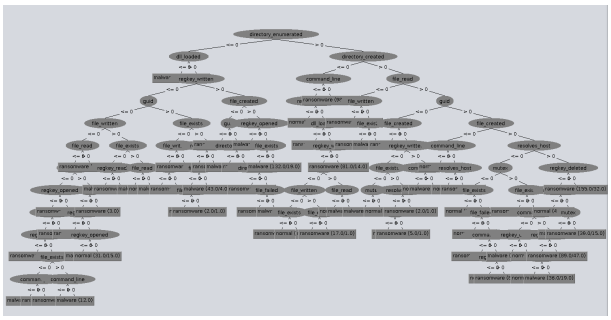


Fig. 5. *Weka Classifier Tree Visualizer*

Listing 6. *API Call extraction from the JSON file*

```
def exportData(inFile,apiList):
 try:
  with open(inFile,'r') as i:
   json_data=json.load(i)
  try:
   dict1=json_data["behavior"][
      "apistats"]
   for i in dict1.keys():
    dict2=json_data["behavior"
       ]["apistats"][i]
    listKeys=dict2.keys()
    for key in listKeys:
     if key in apiList:
      pass
     else:
      apiList.append(key)
   except:
    raise
  except:
   raise:
  return
```

6. Conclusion

A new method to obtain and analyse behaviour-based malware data based in FOSS Tools was described and some examples of its use were exposed. This method can be used to create new models based in machine learning to detect and classify software samples automatically. Static fingerprints could then be generated to allow IDS/IPS systems, anti-malware suites and Antivirus programs detect then new malware classified samples.

7. References

[1] Alqurashi, S., Batarfi, O. (2016). Automated behavioral malware analysis system. Advances in Intelligent Systems and Computing, 448, 12431248.

[2] Alrabaee, S., Wang, L., Debbabi, M. (2016). BinGold: Towards robust binary analysis by extracting the semantics of binary code as semantic flow graphs (SFGs). Digital Investigation, 18, S11S22.

[3] Wang, Z., Li, C., Yuan, Z., Guan, Y., Xue, Y. (2016). DroidChain: A novel Android malware detection method based on behavior chains. Pervasive and Mobile Computing, 32, 314.

[4] Sahay, S. K., Sharma, A. (2016). Grouping the Executables to Detect Malwares with High Accuracy. Physics Procedia, 78(December 2015), 667674.

[5] Weka 3 - Data Mining (accessed March 9, 2017). `http://www.cs.waikato.ac.nz/ml/weka/`

[6] Salehi, Z., Sami, A., Ghiasi, M. (2017). MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values. Engineering Applications of Artificial Intelligence, 59, 93102.

[7] Mohaisen, A., Alrawi, O., Mohaisen, M. (2015). AMAL: High-fidelity, behavior-based automated malware analysis and classification. Computers and Security, 52, 251266.

[8] Yuxin, D., Xuebing, Y., Di, Z., Li, D., Zhanchao, A. (2011). Feature representation and selection in malicious code detection methods based on static system calls. Computers and Security, 30(67), 514524.

[9] Kawaguchi, N., Omote, K. (2015). Malware function classification using apis in initial behavior. In Proceedings - 2015 10th Asia Joint Conference on Information Security, AsiaJCIS 2015 (pp. 138144). Institute of Electrical and Electronics Engineers Inc.

[10] Cuckoo Sandbox (accessed March 9, 2017). http://www.cuckoosandbox.org

[11] VirtualBox (accessed March 9, 2017). http://www.virtualbox.org

[12] Kirat, D., Vigna, G., Kruegel, C. (2014). BareCloud: Bare-metal Analysis-based Evasive Malware Detection. 23rd USENIX Security Symposium (USENIX Security 14), 287301. Retrieved from http://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kirat

[13] liac-arff python module documentation(accessed April 28, 2017)https://pythonhosted.org/liac-arff

[14] bigjson python module documentation(accessed April 28, 2017)https://github.com/henu/bigjson

[15] VisurShare (accessed April 17, 2017). https://virusshare.com/

[16] CNET.com (accessed April 17, 2017). https://descargar.cnet.com/windows/

[17] Freewarefiles.com (accessed April 17, 2017). https://http://www.freewarefiles.com/

[18] Witten, I. H., Frank, E., & Hall, M. A. (2011). Data mining: practical machine learning tools and techniques . Burlington, MA : Morgan Kaufmann Publishers/Elsevier .