Python and Machine Learning: How to use algorithms to create yara rules with a malware zoo for hunting

PassTheSalt 2018

Who’s who

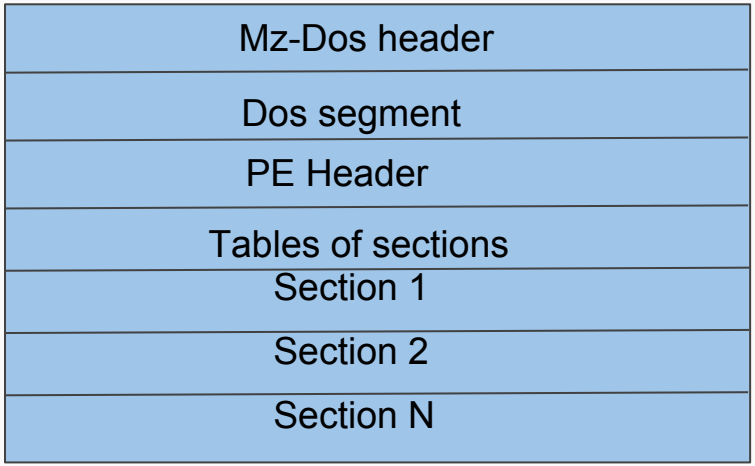
- Sebastien Larinier (ceo of SCTIF)
- @sebdraven, slarinier@gmail.com
- DFIR, malware analysis
- Honeynet project chapter France and co organizer of Botconf
- Francomisp, Yeti and some open source stuffs here: https://github.com/sebdraven

Topics

- Malwares PE format
- Clustering
- Yara and Hunting

PE format: definitions and generalities

Pe format


Mz-Dos header
Dos segment
PE Header
Tables of sections
Section 1
Section 2
Section N

Pe format

```
typedef struct _IMAGE_DOS_HEADER
{
    WORD e_magic;                   //magic number
    WORD e_cblp;                    //Bytes on last page of file
    WORD e_cp;              //Pages on file
    WORD e_crlc;            //relocations
    WORD e_cparhdr;         // Size of header in paragraphs
    WORD e_minalloc;        // Min extra paragraphs needed
    WORD e_maxalloc;        // Max extra paragraphs needed
    WORD e_ss;              // Max extra paragraphs needed
    WORD e_sp;              // Initial SP value
    WORD e_csum;                    // Checksum
    WORD e_ip;              // Initial IP value
    WORD e_cs;              // Initial (relative) CS value
    WORD e_lfarlc;                  // File add of relocation table
    WORD e_ovno;                    // Overlay number
    WORD e_res[4];                  // Reserved words
    WORD e_oemid;                   // OEM identifier
    WORD e_oeminfo;         // OEM information
    WORD e_res2[10];        // Reserved words
    LONG e_lfanew;                  // File addr of new exe header
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

Pe Format

- PE header

```
typedef struct _IMAGE_NT_HEADERS {
  DWORD                 Signature;
  IMAGE_FILE_HEADER     FileHeader;
  IMAGE_OPTIONAL_HEADER OptionalHeader;
} IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;
```

- PE\0\0

Pe format

- PE header

```
typedef struct _IMAGE_FILE_HEADER {
  WORD  Machine;
  WORD  NumberOfSections;
  DWORD TimeDateStamp;
  DWORD PointerToSymbolTable;
  DWORD NumberOfSymbols;
  WORD  SizeOfOptionalHeader;
  WORD  Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

Pe format

- PE header

```
Typedef struct _IMAGE_OPTIONAL_HEADER {
  WORD            Magic;
  BYTE            MajorLinkerVersion;
  BYTE            MinorLinkerVersion;
  DWORD           SizeOfCode;
  DWORD           SizeOfInitializedData;
  DWORD           SizeOfUninitializedData;
  DWORD           AddressOfEntryPoint;
  DWORD           BaseOfCode;
  DWORD           BaseOfData;
  DWORD           ImageBase;
  DWORD           SectionAlignment;
  DWORD           FileAlignment;
```

Pe format

```
WORD        MajorOperatingSystemVersion;
WORD        MinorOperatingSystemVersion;
WORD        MajorImageVersion;
WORD        MinorImageVersion;
WORD        MajorSubsystemVersion;
WORD        MinorSubsystemVersion;
DWORD       Win32VersionValue;
DWORD       SizeOfImage;
DWORD       SizeOfHeaders;
DWORD       CheckSum;
WORD        Subsystem;
WORD        DllCharacteristics;
```

Pe format

```
DWORD            SizeOfStackReserve;
  DWORD            SizeOfStackCommit;
  DWORD            SizeOfHeapReserve;
  DWORD            SizeOfHeapCommit;
  DWORD            LoaderFlags;
  DWORD            NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

Pe Format

- Pe Header

```
typedef struct _IMAGE_DATA_DIRECTORY {

DWORD VirtualAddress;

DWORD Size;

} IMAGE_DATA_DIRECTORY,*PIMAGE_DATA_DIRECTORY;
```

Pe format

Position	Name	Description
0	IMAGE_DIRECTORY_ENTRY_EXPORT	Export table
1	IMAGE_DIRECTORY_ENTRY_IMPORT	Import table
2	IMAGE_DIRECTORY_ENTRY_RESOURCE	Ressources table
3	IMAGE_DIRECTORY_ENTRY_EXCEPTION	Exceptions table
4	IMAGE_DIRECTORY_ENTRY_SECURITY	Certificats table
5	IMAGE_DIRECTORY_ENTRY_BASERELOC	Relocalisations table
6	IMAGE_DIRECTORY_ENTRY_DEBUG	debug
7	IMAGE_DIRECTORY_ENTRY_COPYRIGHT / IMAGE_DIRECTORY_ENTRY_ARCHITECTURE	copyright
8	IMAGE_DIRECTORY_ENTRY_GLOBALPTR	Global pointer
9	IMAGE_DIRECTORY_ENTRY_TLS	Threads table
10	IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG	Configuration table
11	IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT	Bound import
12	IMAGE_DIRECTORY_ENTRY_IAT	...
13	IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT	...
14	IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR	...
15	-	vide

Pe format

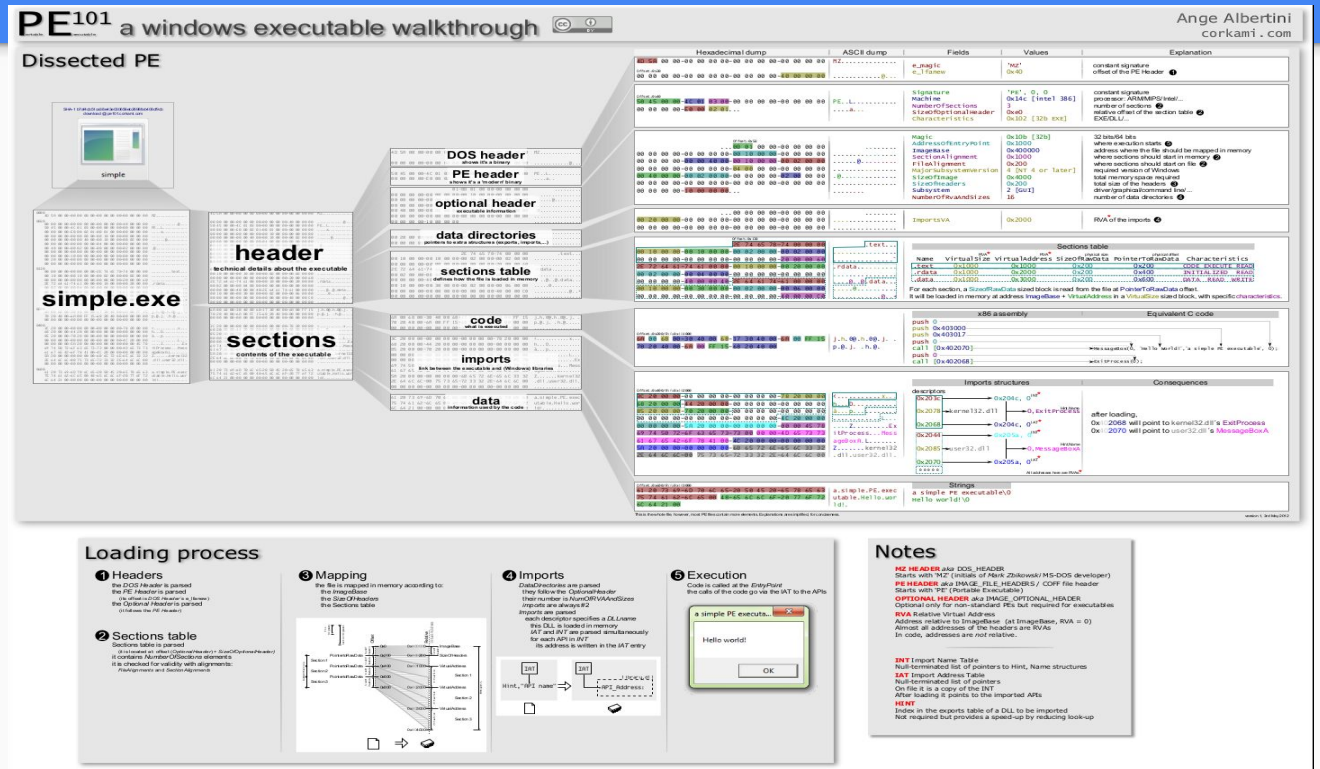
- Table of sections

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Pe Format

- Imports table
  - Functions used by the binary from external lib
- Exports table
  - Functions shared by the binary (basically DLLs)
- Ressources Tables
  - icons, strings, langage using

Pe Format


PE101 a windows executable walkthrough
Ange Albertini
corkami.com
Dissected PE
simple.exe
header
technical details about the executable
sections
contents of the executable
DOS header
PE header
optional header
data directories
sections table
code
imports
data
Hexadecimal dump
ASCII dump
Fields
Values
Explanation
Sections table
x86 assembly
Equivalent C code
Imports structures
Consequences
Strings
Loading process
1 Headers
2 Sections table
3 Mapping
4 Imports
5 Execution
Notes
MZ HEADER aka DOS_HEADER
PE HEADER aka IMAGE_FILE_HEADERS / COFF file header
OPTIONAL HEADER aka IMAGE_OPTIONAL_HEADER
RVA Relative Virtual Address
INT Import Name Table
IAT Import Address Table
HINT

Few words about machine learning algorithms

Clustering vs Classification

- Clustering: Automatic grouping of similar objects into sets.
- Classification: Identifying the category an object belongs to

So clustering is applied on dataset unlabeled and classification on dataset labeled

Algorithms

So usually, for classification (on labeled dataset) we use supervised algorithms.

And for clustering (on unlabeled dataset) we use unsupervised algorithms.

Vector of features

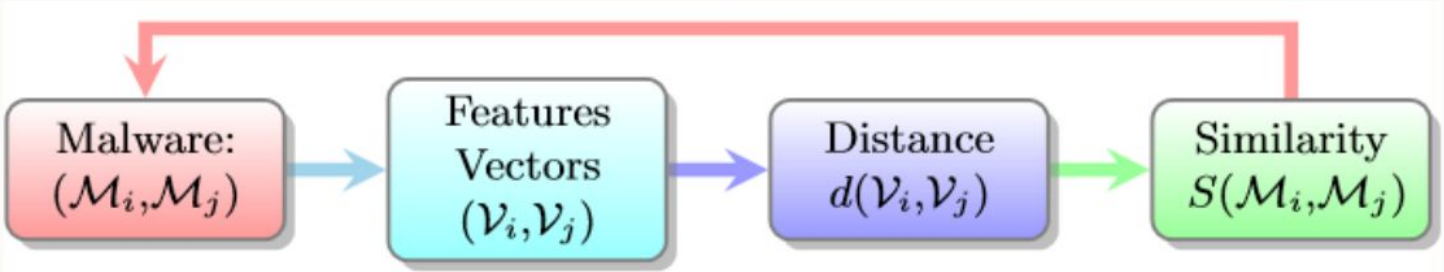
When using machine learning algorithms, the inputs are vectors of features. A vector of features describes the characteristics of an object.

Differents features of a malware can be:

- The size
- The imports table
- The number of sections
- The entropy of the file
- The entropy by section
- ...

Similarities and distance

Two objects with a small distance between them are similar.



For this presentation

Here we mix the previous concepts: we use unsupervised algorithms on a labeled dataset.

Because we want to have clusters with similar malwares to generate yara rules on each cluster.

Malwares and Clustering

Why clustering malware ?

- Create signatures to catch a complete family
- Minimize false positives
- Hunting campaigns

Fuzzy hashing

- Fuzzy hashing: ssdeep

The idea is to create a signature for each file you want to compare and then, make a comparison based on edition distance between those signatures.

Examples on PlugX

5b183e99ed532cb143c55ded6a46f5b0545c575f6b54b25afbb04b0bbf40170c

302b1196dbe7d90effe72de832e0696016ea69a7a6915217081f013c1db7bc37

Fuzzy hashing

Ssdeep format:

chunksize:chunk:double_chunk
6144:z4lRkAehaKuqT+FXErGKhVPBD3A29ELNLnqpC+:zkWAehJuqTvbV5OLIpD

6144:z4lRkAehaKuqT+FXErGKhVPBD3A2LLOfY2axu/oI4R6ZDIB:zkWAehJuqTvbV5D2dsRHB

75 % of matching between the two signatures

Limitations

Signature A with chunk size A’ and Signature B witch chunk size B’

If B’ != A’:

Then

match(A,B) is 0

So if the code of two malwares is the same and one has a section with garbage data, the result of the matching is 0

PeHash

The concept of PeHash is to use the characteristics of PE format to make a clustering.

The clustering is possible because the hash depends on an approximation of Kolmogorov complexity.

But it’s impossible to compute a distance between two files. It’s equal or not.

ImpHash and ImpFuzzy

The idea of this clustering is that if two malwares have the same import table or near, they are from the same family because they use the same functions (network,system, I/O…)

Imphash = md5(Import Table of PE)

ImpFuzzy = ssdeep(Import Table of PE)

The major disadvantage is if two malwares have the same symbols but not in the same order, their imphash are different and their Impfuzzy match at 40 %

Polichombr, Machoc and r2graphity

Each technic disass the binary, generate the graph flow and compare them.

Polochombr and Machoc process a fuzzy hash on the graph flow instructions.

R2graphity is used to print the differents graphs.

The major disadvantage is the scalability. Indeed, comparing the graph flow signatures together has a complexity of n2 (n = number of signatures)

Our Strategy

For the scalability we decided to use two algorithms:

- Dbscan
- K-means

And the following dataset :

https://github.com/ytisf/theZoo

We decided to check those unsupervised algorithms on a dataset labeled to verify some hypothesis

Our strategy

- Construct the best vector of features on this dataset
- Generate yara rule by cluster
- Generalize the system

Algorithms Dbscan - Kmeans

K Means Algorithm

The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields.

The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster. The means are commonly called the cluster “centroids”; note that they are not, in general, points from X, although they live in the same space. The K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum of squared criterion:

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_j - \mu_i||^2)$$

K Means Algorithm

First step:

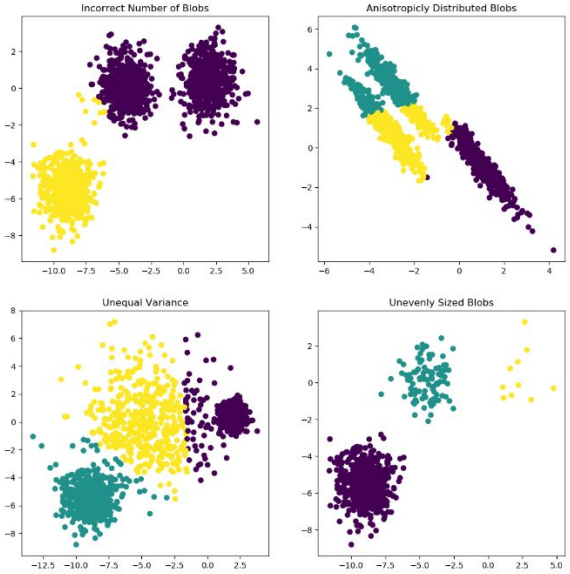
- We choose the number of cluster
- We choose the initial centroids in three ways:
  - With k-mean ++ init, the algorithm chooses k centroids in processing an index called inertia in a loop and choose the better value
  - Randomly, the algorithm chooses k centroids in the matrix
  - Nparray, the user chooses the k centroids

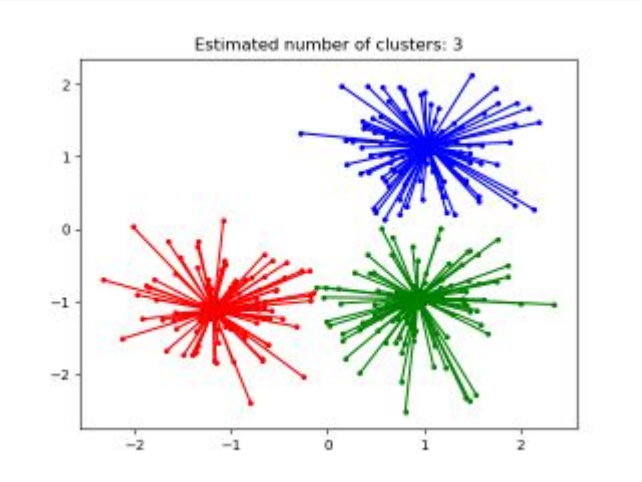
K Means Algorithm

Second step:

- The algorithm calculates distance between k- centroids and all vectors in the matrix and constructs k clusters minimizing inertia with k centroids and the nearest vectors with this k centroid

K-Means Example

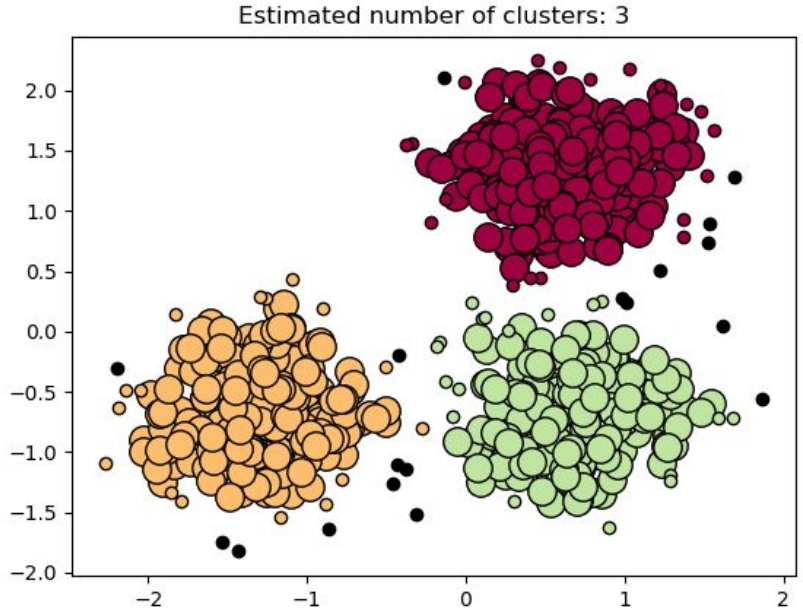


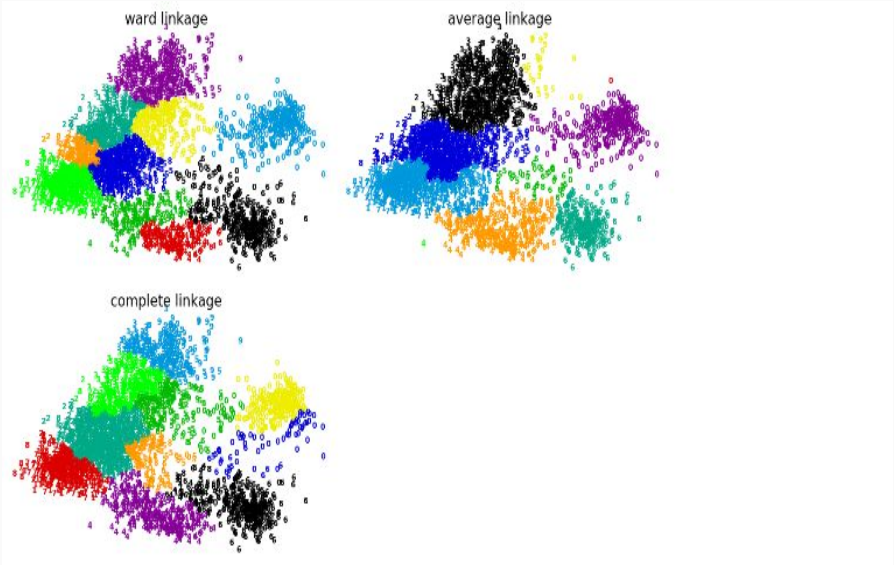


DBScan Algorithm

The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to k-means which assumes that clusters are convex shaped. The central component to the DBSCAN is the concept of core samples, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples). There are two parameters to the algorithm, min_samples and eps, which define formally what we mean when we say dense. Higher min_samples or lower eps indicate higher density necessary to form a cluster.

Dbscan: Example





PE and Featuring

How to transform a PE into an array to make a vector of features

First step is to extract data in json files.

Vector of features is an array in numpy library

Malwares and featuring

Interesting informations for a malware are:

1. Sections: name,size, entropy, characteristics
2. Imports: number of modules, number of symbols, functionalities
3. Exports: number of modules, number of symbols, functionalities
4. Size of file

First Vector of features

So we make a first feature vector:

[size of file, number of sections, median of entropy, number of imports, number of exports]

Results with the first vector with K-means

The first results are interesting but are not totally efficient.

If you check the norm of vectors, the size of file is the feature which erase all other values

Results with the first vector with DBScan

It’s worse than K-Means because the vector is depending on the size of file and this make the density bad. We have to normalize the vector.

The key of success

Here, the vector of feature depends on the size of file. If v is a vector of feature,

norm(v) ~ size of file

So, we normalize the vector of features.

Second vector of features

Now we normalize the vector of features:

[size of file / max(size of all files), number of sections/ max(number of sections of all files), median of entropy /max(median of entropy of all files), number of imports / max(number of imports of all files), number of exports / max(number of exports of all files)]

Results with second vectors with K-Means

The classification is better than the first one, whereas we just normalized the values.

Now we add check with DBscan algorithm with the first and second vectors

Results with the second vector with DBScan

We have a good classification by families and by versions of families

Yara rules Generation

Yaragenerator

- https://github.com/Xen0ph0n/YaraGenerator
- Generate automatically yara rules based on an intersection of strings

Using ours results of clustering malware

On the EquationGroup Cluster we have a rule matching this family.

But if we try with the Regin family, it doesn’t work because the tool doesn’t find an intersection based on strings.

Results

On VT hunting, we have found 39 new Equation_Group malwares with this yara rule during the six month later.

We don’t have false positive

Conclusions

We have seen machine learning is not magic, a work of featuring must be done including the of the dataset.

Here, our dataset is very heterogeneous with a big cluster of EquationGroup, and others clusters with few malwares

The machine learning is useful to make a first filter to clusterize a big dataset because the algorithms have been thought to be scalable contrary to algorithms which compare signatures. (ssdeep,impfuzzy,machoc…)