Companion to HPSR Security Briefing Podcast Episode 14

Security Briefing Episode 14, June 2014

HP Security Research

Table of contents

Episode 14

Thank you for subscribing to Episode 14 of the *HP Security Research SecurityBriefing*. In this briefing we discuss file geometry visualization and clustering experiments on malicious files using the R language.

Malware files clustering based on file geometry and visualization using R language

The daily explosive growth of malware samples over the last decade or so presented many antivirus companies with a problem - how to efficiently and accurately process and label such a large number of incoming files. With the advent of cloud-based services, Big Data and increasing computational power, many put their hopes in machine-learning classification algorithms. In this paper we attempt to describe a hands-on approach and show the basic principles used in the classification of files. Intuitively it pays to note that variants of a malware family are generally derived from a similar codebase and exhibit similar file structures. This notion, when applied through freely accessible tools and visualization techniques used in R language, helps us to analyze and label a set of incoming files. It also provides a basis for further research and experiments in malware identification and processing. The following paper presents a hands-on approach, coupled with numerous examples of malware visualization and clustering using R language and other freely available tools.

Background – PE files

The vast majority of executable files in Windows are in Portable Executable file format (PE file format).
A PE file format has a fairly regular structure. This requirement is imposed by an OS loader and the OS execution framework. The OS loader requirements might be viewed as a multi-dimensional gatekeeper or filter which only accepts objects conforming to its prescribed patterns. This standardization creates an opportunity for us to examine possible similarities amongst files based on their "geometrical" properties, that is, the properties which ensure the file's structure conforms to the OS loader requirements.

The PE file structure is very well documented in numerous publications, beginning with the official Microsoft specification[1], but there are numerous popular articles uncovering the ins and outs of the specifications (see the Further reading section at the end of this report for details).

The file is organized as a flat stream of data. At the beginning of the file there are a number of structures called headers. The first structure is called an MS-DOS header, which is a tribute to the old days of DOS. This is a stub which gracefully exits a file if it is run in an unsupported PE file mode - the real mode. It is worth noting that normally, PE files are structured and compiled to run in the protected mode of the OS, where each section of memory has its own descriptor governing its access by running processes. The MS-DOS header is followed by a PE file signature - a sequence of 4 bytes which marks the beginning of the PE file structures. The PE file signature is followed by the PE file header and by an optional header. Following this are the section headers and actual sections bodies. The file is concluded by various miscellaneous regions of information such as relocation, the symbol table, line numbers, and string table data. The headers contain many pieces of unique information which can be used in the identification of the file.

[1] Microsoft PE and COFF Specification http://msdn.microsoft.com/en-us/library/gg463119.aspx

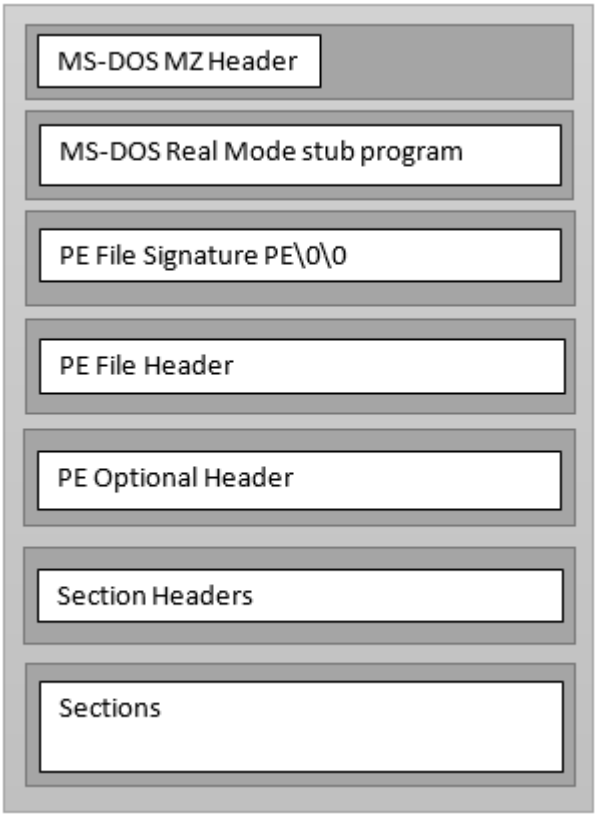

Figure 1 Cursory PE file format layout

Research in the field

There have been a number of studies into this area and extensive research done in the classification of malware samples based on various machine learning algorithms and selected features of files.[2][3] Most of the algorithms require training and a test set of samples. Such algorithms, when properly tuned, are fully automated and efficient. However, they are prone to failures when targeted by malware writers. This paper explores an interactive approach which involves PE file attribute visualization. It also examines an unsupervised learning clustering technique which removes training sets and brings human interaction to a process of classification, hence creating greater flexibility and intelligence in the analysis and classification of malware.

Our study - Malicious file clustering and visualization

Would any characteristics from a given set of files be suitable for a clustering algorithm? If so, this information could later be used to distinguish new or previously unseen malicious files under examination. The following initial set of file attributes was selected for consideration:

- Entry point address
- Number of sections
- Code size
- Image size
- The virtual and raw sizes of the first, second and third sections.

[2] Selecting Features to Classify Malware
http://2012.infosecsouthwest.com/files/speaker_materials/ISSW2012_Selecting_Features_to_Classify_Malware.pdf

[3] Malware Detection Using Perceptrons and Support Vector Machines,
http://thor.info.uaic.ro/~ciortuz/PAPERS/ALL/athens.malware.pdf

It was decided that it would be optimal to create a visual representation of the resulting attributes in a graph that allowed mutual aggregation of this data in a two dimensional space. A parallel coordinates graph appeared to work well in this case. Using the R language and its framework was found to be good for the visualization. R is a language and environment for statistical computing and graphics and as such, it provides a wide variety of statistical techniques such as linear and nonlinear modelling, time-series analysis, classification, clustering and more. The R language also encompasses powerful graphical visualization capabilities and is highly extensible.[4]

Method

A number of different visualizing and clustering techniques were applied to several sets of known malware family files, and their efficacy for meaningfully grouping these files was assessed. These analysis methods were also applied to a set of clean files for comparison.

A note about the tools: The intention of this study was not only to be able to cluster malicious files using file geometry and visualization - but to share research in such a way that other researchers could replicate these experiments. As such software tools that were open source or available under the GNU general public license were selected for this purpose.

Case studies – The shapes of Gamarue and Ursnif

Gamarue – Parse and plot example

Before we can apply visualization and clustering, we need to be able to parse a PE file and extract its attributes to a readable file format. There are a number of products available on the market which can be used to parse a PE structure and extract PE file attributes to a readable format. One of them is PeStudio[5], which produces an XML file containing attributes in batch command line mode. This is convenient since the XML file can be parsed by any number of products for visualization, such as our choice, R language. (3) As an example let's look at a number of executables detected by various security products as Gamarue. Gamarue is a moderately widespread malware family that spreads via removable drives and allows attackers to remotely control victim's machines.

Running PeStudio on our set of files yields a set of XML files, each containing attributes for a file (Figure2).

```
cluster_sets\gamarue>for %i in (*) do c:\Cluster\PeStudio828\PeStudioPrompt.exe -file:%i -xml:xml\%i
```

Figure 2 Batch mode of PeStudioPrompt.exe

For a quick glimpse at the attributes we can use XML Explorer. It provides a convenient way of quickly navigating through an XML file and viewing its attributes.

[4] The R Project for Statistical Computing - http://www.r-project.org/

[5] PeStudio - http://www.winitor.com/

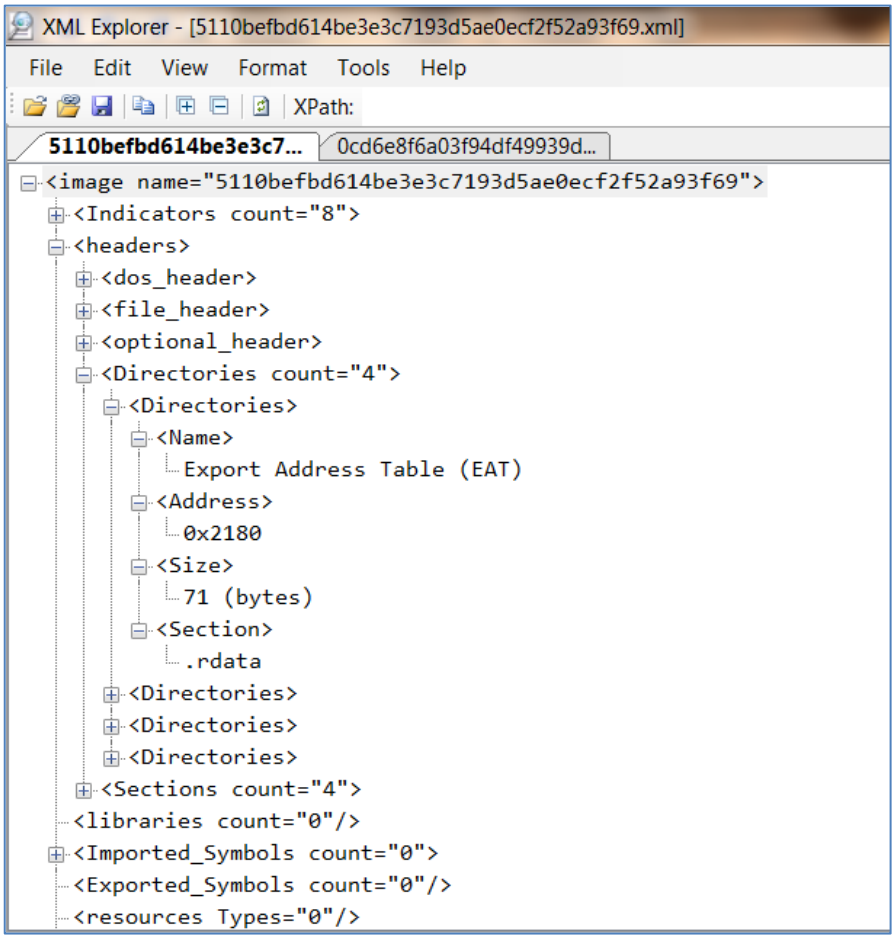

Figure 3 XML Explorer view of the XML file data

Using R language, we can read the XML files that contain the file attributes to a data frame. We then use these data sets to produce a parallel graph. A parallel coordinate graph is a chart that has a number of vertical vertices each representing a single variable. Such visualization allows us to gauge the similarity of files based on the selection of the attributes chosen for each vertex. The initial set of attributes was selected because they could be closely coupled with code properties such as entry point addresses, numbers of sections, code size, image size, and the virtual and raw sizes of first, second and third sections. These attributes are normally available in most PE files and would not require additional complex processing, such as unpacking or behavioral analysis of files, or analyzing the streams of data in file sections. Processing a set of PE files which are detected as Gamarue worm by at least one AV product with the aforementioned selection of attributes yields the following parallel graph.

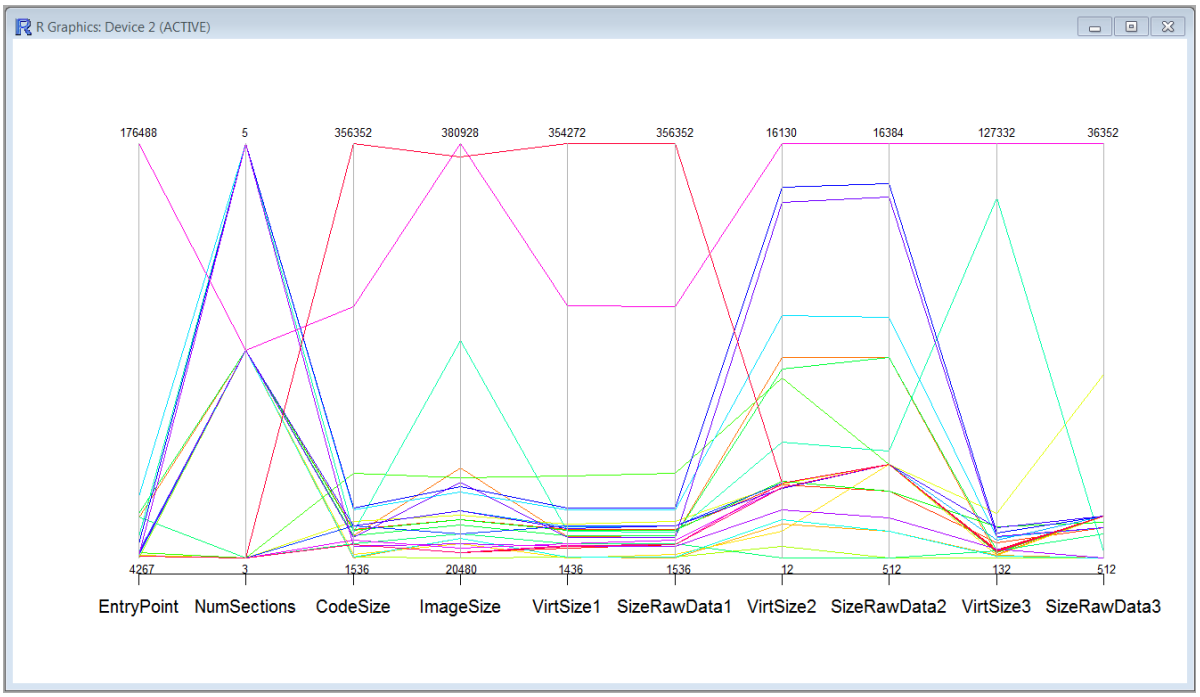

Figure 4 Parallel graph based on a number of files attributes - Gamarue worm family

Each line in this graph represents a file. Each vertex represents an attribute of a file that we decided to use for the parallel graph visualization. While the graph gives a sense of files clustering based on selected attributes it makes it difficult to visually trace each individual line - especially when the number of files, and hence lines, increases. In this case it is helpful to employ an interactive type of parallel graph where a group of lines can be highlighted manually. For instance, interactively selecting a group of lines with a similar CodeSize shows how these lines fare together in comparison to other attributes.

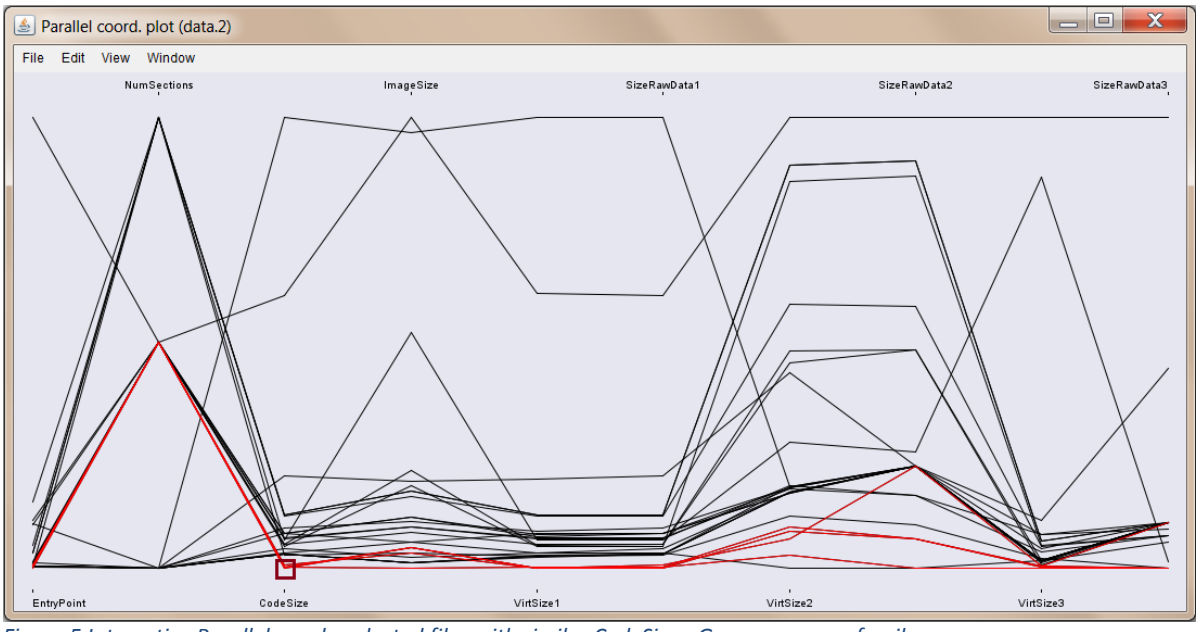

Figure 5 Interactive Parallel graph, selected files with similar CodeSize - Gamarue worm family

It shows that the CodeSize selection criteria create a fairly closed coupled cluster, which also includes files with similar EntryPoint values, VirtualSize1, SizeOfRawData1, VirtualSize3. But there are small discrepancies in VirtualSize2, SizeOfRawData2, SizeOfRawData3. Alternatively when selecting files with a similar Virtual Size value of the 3rd section (VirtSize3) the cluster is somewhat dispersed but does include a larger number of files.

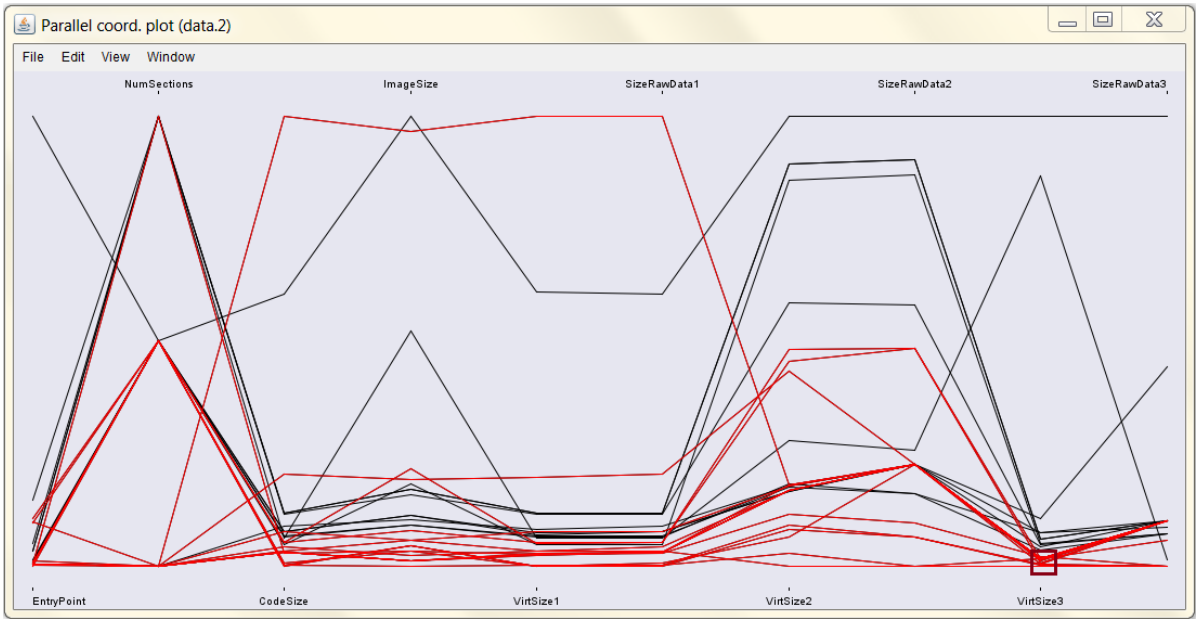

Figure 6 Interactive Parallel graph, selected files with similar Virtual Size of the third section value - Gamarue worm family

It has to be noted when comparing these results, even though the selected files used in this study come from a single malware family (Gamarue), they show great diversity when trying to cluster them based solely on the files' geometry.

It also became apparent during this experiment that the selection and order of the file attributes used for the construction of a parallel graph are very important for the quality of the visualization, and that they might be different for each individual malware family. The example provided and the easy availability of the tools used allows for a quick set up and ability to experiment with any given set of files. Provided the output of a file parser can be rendered in XML or any other readable format that can be processed by R (normally flat files in ASCII) a great number of attributes can be examined and used for such visualization.

Ursnif – Objects clustering example

Another way of looking at data visually is to apply objects clustering. This is an unsupervised learning technique that looks at an attribute space and tries to identify groups or patterns within the attributes.

Let's examine malware based on the Ursnif family. Ursnif is a widespread trojan that steals sensitive information. The trojan can affect 32-bit as well 64-bit Windows platforms and normally carries components (such as DLLs) in its resource section (which it uses accordingly). This feature of its behavior should provide some interesting file geometry. Looking at a set of 32 files detected as Ursnif by various antivirus products, we can produce an interactive parallel graph (as described above) that shows possible clustering based on only a few properties.

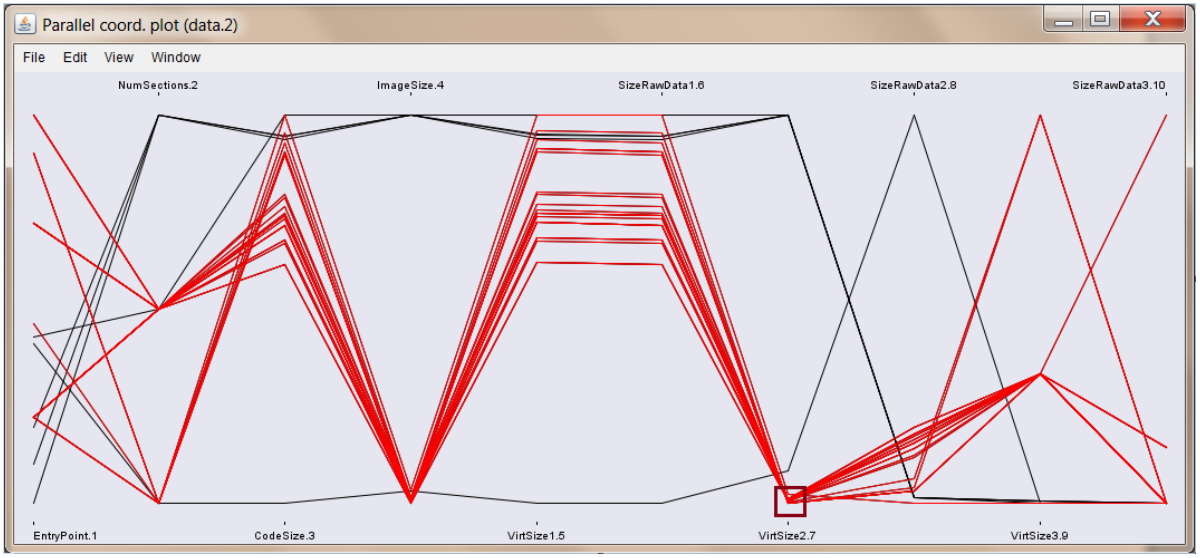

Figure 7 Interactive parallel graph of Ursnif family - highlighted files are grouped by VirtualSize of section 2

The basic principal of clustering is to group objects so that their attribute similarity would be high within a group and low across multiple groups. There are a number of algorithms available within the R langauge based on various clustering methodologies such as partitioning, hierarchical, density-based and grid-based approaches. Each of these methodologies has a number of distinct features and could be advantageous for each particular case of a data frame.

Let's consider K-Means clustering as one of the most popular and simple unsupervised learning partitioning methods. To understand how the K-Means clustering algorithm works, for the sake of simplicity, imagine a sand box which has a handful of marbles thrown into it. How do we find clusters which would fairly well cover the marbles? First we would need to define how many clusters we anticipate. The K-Mean algorithm requires this information as a parameter. Then we randomly choose the locations of cluster centers in the sandbox. We compute the distance from each marble to a corresponding center. Then we calculate a new center to which the squared error sum of distances from each belonging to the previous center marbles is smaller. Once the new centers are computed, we select a new set of marbles belonging to each new center and continue computing the next centers based on the new marble selections. We then continue until the newly computed centers are exactly the same as the old ones. At this point we could say that we have found all the centers for the clusters.

As previously mentioned, one of the distinct characteristics of the K-mean algorithm is that it requires a predicted number of clusters as an argument. This is where the parallel graph visualization described above might come in handy. For instance, looking at a set of files from the Ursnif family, the parallel graph gives us a sense of how many clusters we can expect within a set. We might also possibly judge the quality of the PE file attributes we decided to use for clustering. Looking at Figure 7 we can anticipate that there are possibly 2 to 3 clusters in the Ursnif pool of files when grouping is based on the selected file attributes.

Applying K-mean clustering with the Ursnif data frame we have:

ClusteringResults <- kmeans(ursnif_files_attributes, 3)

where *3* is the proposed number of groups. Within the *ClusteringResults* we could see the following available components:

cluster, centers, totss, withinss, tot.withinss, betweenss, size, iter, ifault

Below is a brief description of what each of them mean in our particular case:

- ***cluster*** - is a vector of integers with the size of a total number of clustered objects where each integer indicates a cluster number to which each object is allocated. In our example the above clustering produced the following vector:

  > ClusteringResults$cluster
  [1] 2 3 3 3 3 3 3 3 1 3 3 2 3 3 3 3 2 3 3 2 3 3 3 3 3 3 3

This means that the overall number of elements is 27; the first element belongs to cluster 2, the second to cluster 3, and the 9th to cluster 1. Cluster 1 has only one element. Cluster 2 has 4 elements and so forth.

- ***centers*** – is a matrix of group centers. In the above example the group centers are:

	entry_point	num_sections	size_of_code	size_of_image	virtual_size	size_raw_data	virtual_size2	size_raw_data2	virtual_size3	size_raw_data3
1	5359.000	3.000000	2048.00	102400.00	1641.00	2048.00	90112.000	66048.000	20.000	512
2	5045.750	4.750000	64256.00	1134592.00	63989.50	64256.00	1049859.250	1536.000	65.250	512
3	5496.455	3.727273	52410.18	79499.64	52388.55	52410.18	9767.227	8845.864	4655.455	768

> |

Figure 8 Centers for each of the attribute space within clusters

The centers are computed according to the method's clustering algorithm.

- ***totss*** – total sum of squares

> ClusteringResults$totss

[1] 7.472679e+12

- ***withinss*** – vector identifying a within cluster sum of squares, one data point per cluster

> ClusteringResults$withinss

[1] 0 31174032 5696360435

Note: *0* indicates that the first cluster consists of only one element.

Using a cluster sum of squares, we could better identify a number of possible clusters by computing its sum for each number of predicted clusters and plotting the data on a graph.

```
wss <- (nrow(data_cluster)-1)*sum(apply(data_cluster,2,var))
for (i in 2:5) wss[i] <- sum(kmeans(data_cluster, centers=i)$withinss)
plot(1:5, wss, type="b", xlab="Number of Clusters", ylab="Within groups sum of squares")
```

In this case we are looking for a distinct line bend which would point out a number of best possible clusters in a data frame.

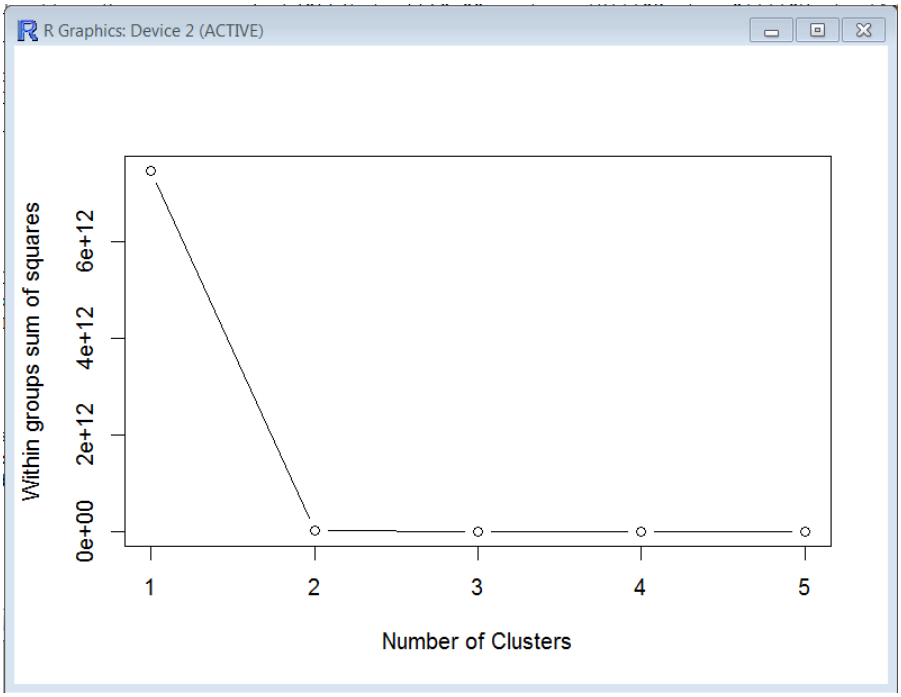

Figure 9 Sum of squared errors screen plot

It looks like 2 or at most 3 is the number of clusters we need to consider.

- *tot.withinss* - Total within cluster sum of squares
- *betweenss* - The between clusters sum of squares
- *size* - The number of points in each cluster, for instance, in our case that would be 3 clusters with 1,4 and 22 elements respectively

> ClusterResults$size

[1] 1 4 22

- *iter* - The number of outer iterations
- *fault* – Integer indicative of possible algorithm problems

To visualize the clusters we can use a clusplot() function, available in R language, which draws a two dimensional clustering plot on the current graphic device.

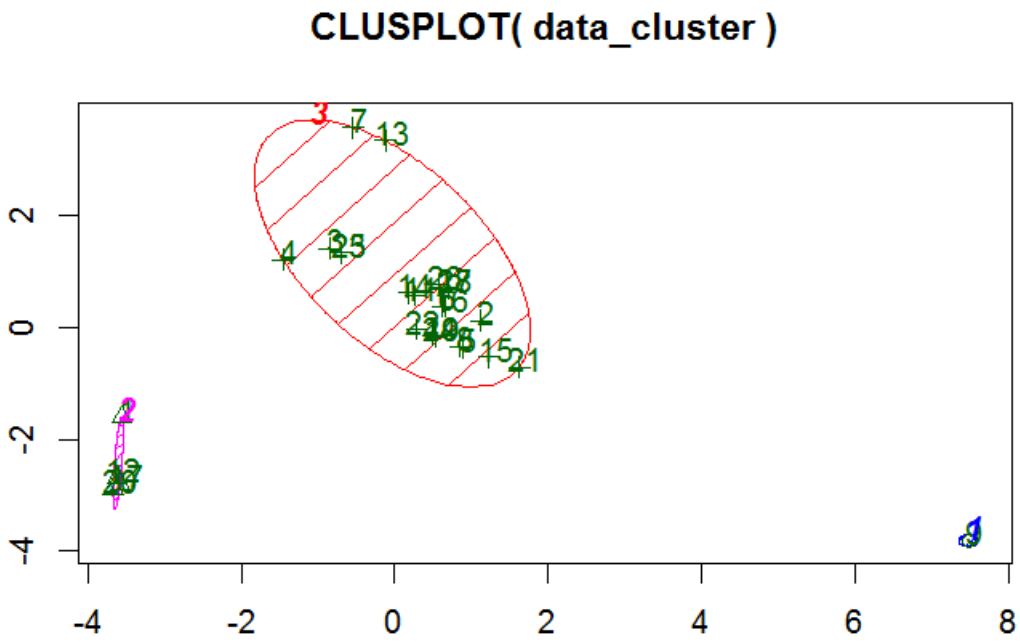

Figure 10 Cluster Plot based on attributes of files of the Ursnif family

As can be seen from the example above, file attributes, even in a limited fashion, might provide a solid basis for clustering. As we noted earlier, file geometry is very susceptible to various file packers, packages and self-extracting archives. All of this has to be taken into account when preparing a set of files for processing. Let's increase the number of attributes and see how it affects the grouping.

Adding only four distinct attributes such as SizeOfInitializedData, CheckSum, SizeOfStackReserve and SizeOfHeapReserve, immediately shows that there are more possible clusters available when looking at the sum of squared screen plot. The distinct bend extends itself towards four possible cluster centers.

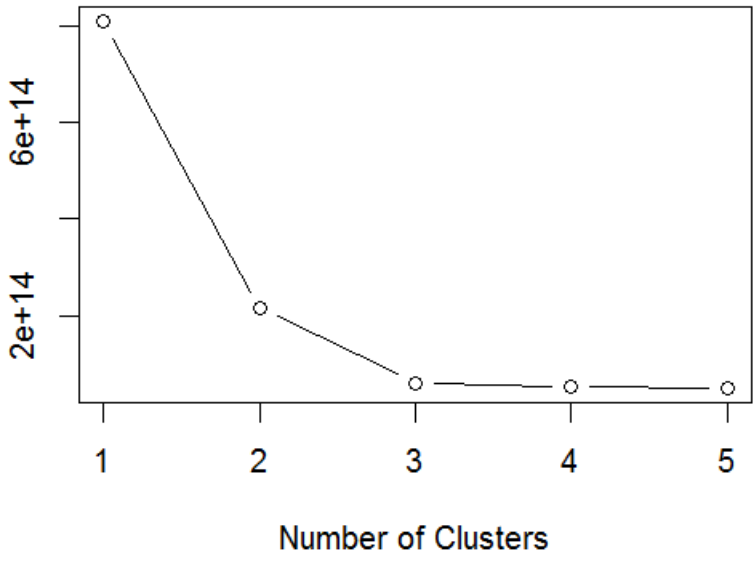

Figure 11 Predicted number of clusters

Running the K-means partitioning algorithm on four possible cluster centers yields us a higher granularity of groups and allows us to look deeper into the samples' malware family division. For instance, cluster 3 (red shaded oval area) on Figure 10 in the current partitioning is now split into two subgroups, see Figure 12. These results show that the number and the quality of attributes play a very important role in quality of clustering outcomes and should be carefully considered - perhaps individually - for each family.

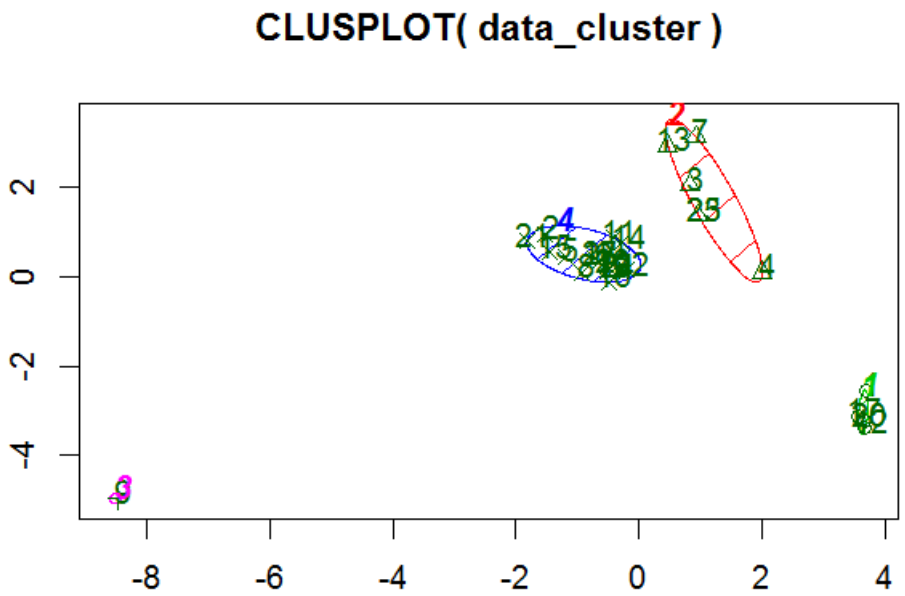

Figure 12 Clustering plot based on the extended set of attributes - Ursnif malware family

Two malware families file comparison example

Let's consider K-mean clustering of a set of files based on our example two malware families - Ursnif and Gamarue. A sum of squared errors plot shows a distinct bend at around 10 possible clusters which can be considered within the presented set of files.

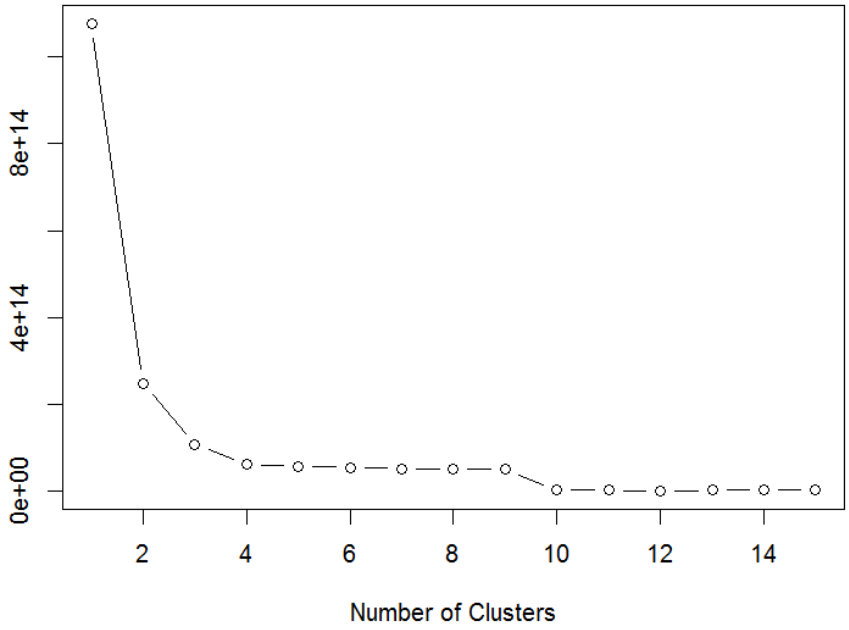

Figure 13 Predicted number of Gamarue and Ursnif clusters based on squared sum of errors

Running the clustering algorithm with the 10 clusters as an argument shows good separation between the clusters. See Figure 14.

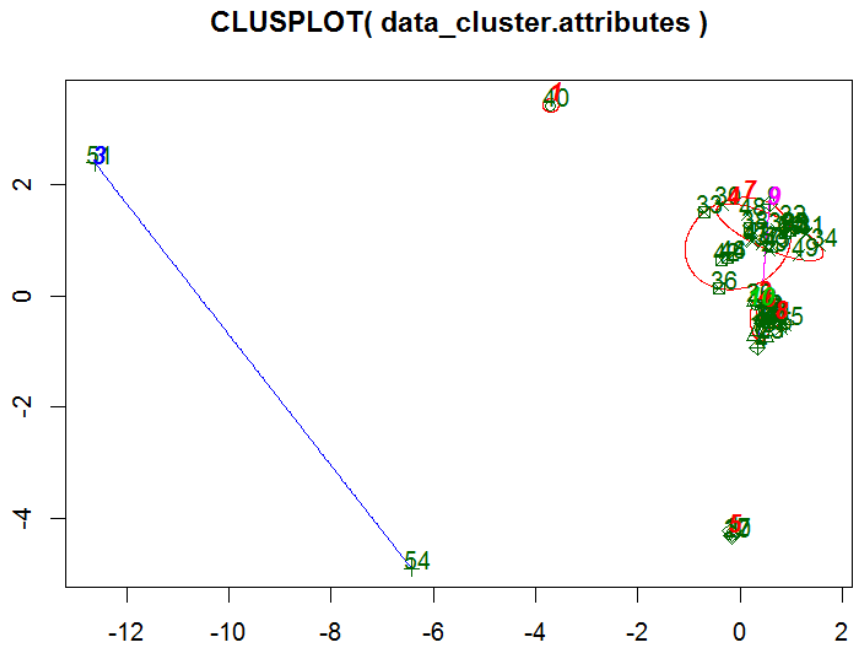

Figure 14 Clustering plot of two malware families - Gamarue and Ursnif

The cluster sizes show that there are 10 clusters with the object numbers of 1, 13, 2, 13, 4, 2, 11, 4, 2 and 2.

> ClusterResults$size
[1] 1 13 2 13 4 2 11 4 2 2

Let's take a look at how the different malware families fell within the created clusters. First we create a table where the names of the analyzed malware families are associated with each file represented by its attributes. Then we apply this information to the clusters created earlier.

> table(data_cluster$Name, clus_results$cluster)

Table 1 - Gamarue and Ursnif clusters

	1	2	3	4	5	6	7	8	9	10
Gamarue	1	0	2	13	0	0	11	0	0	0
Ursnif	0	13	0	0	4	2	0	4	2	2

As we can see from the table above, we have 10 clusters where most of the Gamarue family files position well within clusters 4 and 7. Most of the Ursnif files are located within clusters 2, 5 and 8 and are well separated from the Gamarue family files. The larger number of clusters occupied by the Ursnif family probably means that the set of files that the malware is represented by is greatly affected by its geometry and might contain various variants and sub-groups of the same family. Gamarue, on the other hand, is tighter coupled by the represented set and clusters well.

These results once again highlight the problem of attribute selection for the set of files. It shows that most of the time it would be insufficient to rely on just the geometric properties of a file and to achieve decent clustering one would have to dig deeper into the malware characteristics - perhaps relying on a dictionary of imported or exported APIs, behavioral properties, an entropy of file sections or more.

Clean file comparison example

Now, let's throw a couple of clean files from the Windows system into the mix. Running a squared sum of errors plot shows us a distinct bend around 12 to 13 clusters.

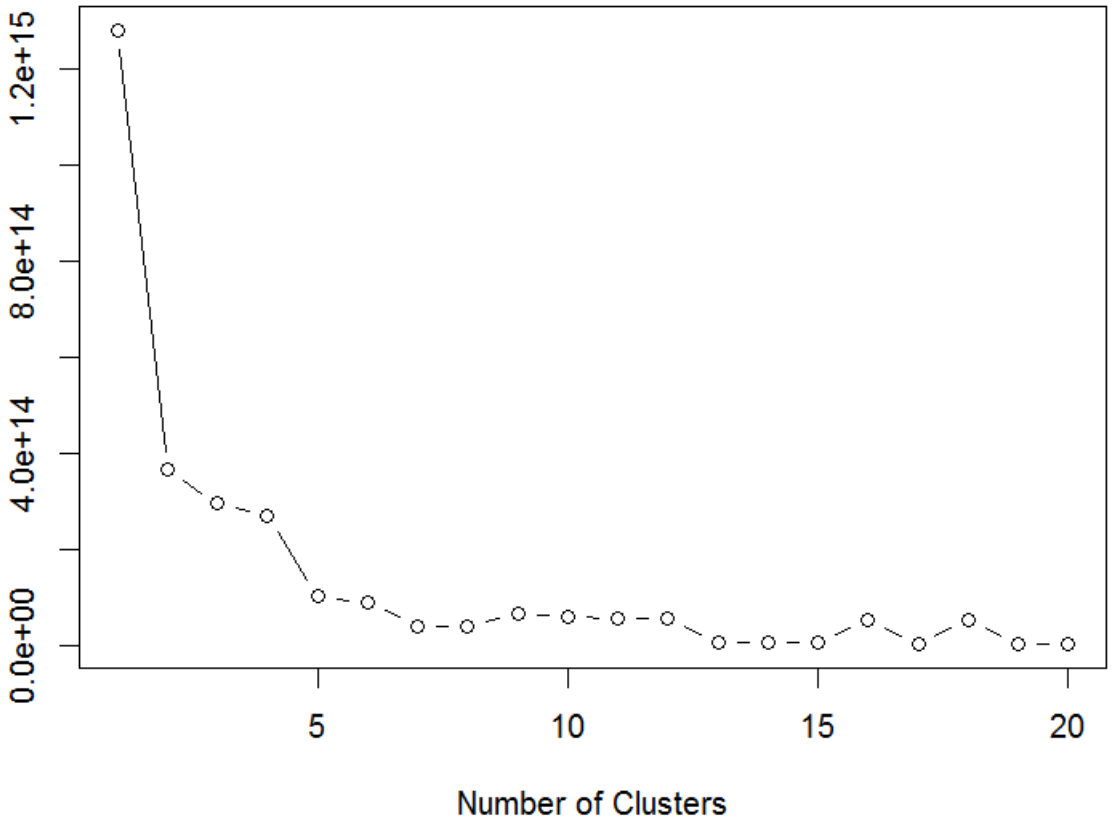

Figure 15 Squared sum of errors plot for Ursnif, Gamarue and a clean set of files

Running the K-mean clustering algorithm on 12 centers gives us a quite a dense group distribution.

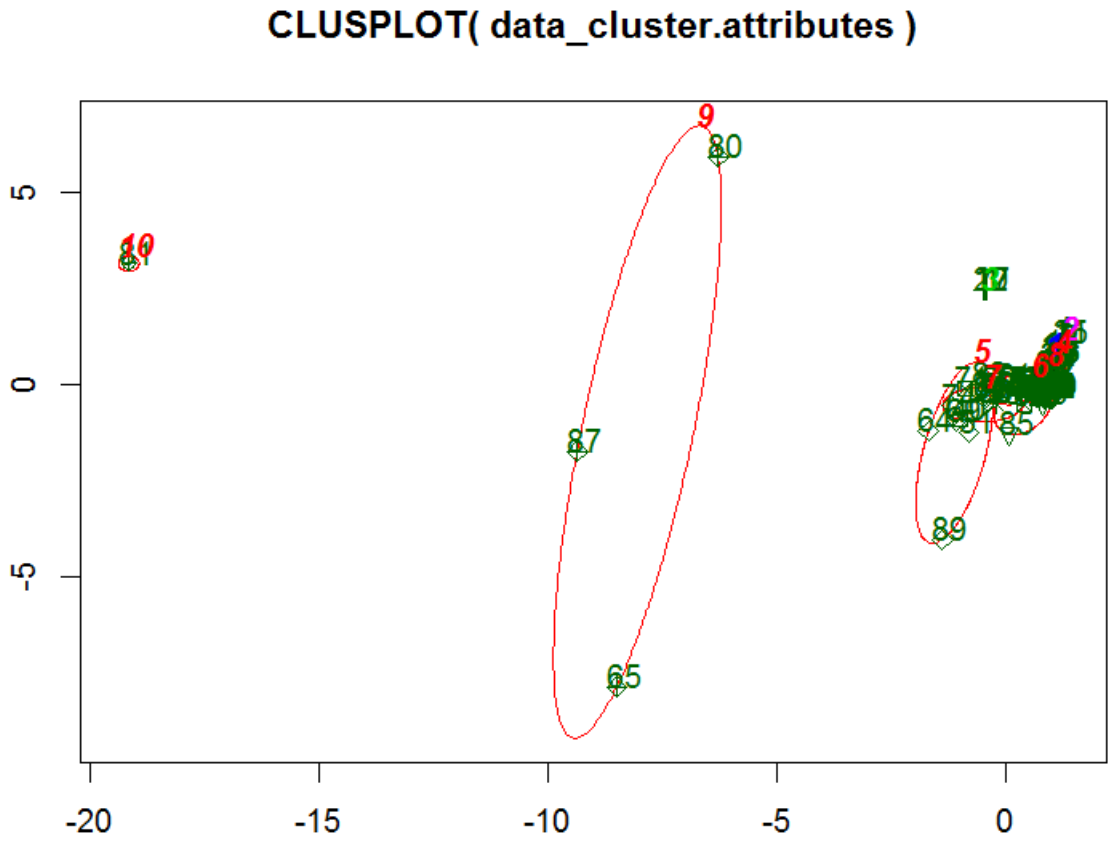

Figure 16 Cluster plot for a mix of Ursnif, Gamarue and clean files

Looking at the specifics of the file distribution within generated clusters we see that Gamarue is quite well separated from the Ursnif family. However, the clean set - while separate from Ursnif - overlaps quite significantly with Gamarue.

Table 2 - Gamarue, Ursnif and clean file clusters

	1	2	3	4	5	6	7	8	9	10	11	12
clean	9	0	0	2	1	1	0	13	0	0	4	8
Gamarue	0	0	0	0	0	0	0	24	0	1	0	2
Ursnif	2	4	4	0	0	0	13	0	4	0	0	0

Conclusion

What we managed to show during this study is that an interactive approach to malware file analysis, with the use of visualization techniques and tools freely available online, could potentially aid in file attribute selections for machine learning algorithms and automated file processing. We showed that the K-means unsupervised learning clustering algorithm could be used for file grouping and labeling provided that the file attribute space is carefully considered, perhaps individually for each family. We found that the selected attributes were sufficient to separate two malware families into clusters but fell short when we brought in a clean set of files. The clean set of files overlapped significantly with the Gamarue malware family while stood clear of Ursnif clusters - this is partially explained by the fact that Ursnif file geometry is quite specific and is not commonly found amongst clean files. It also shows that when comparing parallel graphs of file attributes of Gamarue and Ursnif (see Figure 5 and Figure 7) Gamarue attributes are less tightly coupled for the selected set and cover a larger space of values which are also found amongst clean files. This shows that file geometry alone might not be sufficient for accurate grouping and that other sets of attributes need to be explored and considered. Such attributes could be the results of static and behavioral analysis of code, section entropies, imported and exported APIs and combinations of the above. However, the methodology provided here might allow for the selection, testing and assessment of possible attributes for the clustering of malware families.

Further reading

Microsoft PE and COFF Specification
Peering Inside the PE: A Tour of the Win32 Portable Executable File Format
An In-Depth Look into the Win32 Portable Executable File Format

Learn more at
hp.com/go/hpsr