On the Use of Data Mining Techniques for the Clustering of URLs Extracted from Network-based Malware Traces

Anthony Verez*
*Télécom SudParis, Institut Mines-Télécom, 91000 Evry, France

Tuesday 18th February, 2014

Abstract: We present the use of data mining techniques to classify malware communication traces. Each trace corresponds to a sequence of URLs that are contacted by sandboxed malware.
We create clusters of URLs using a k-means coarse-grained algorithm followed by a DBSCAN fine-grained one.
A platform was developed to launch and study clustering experiments with different clustering parameters using a command-line interface or a web application. For this study, our dataset consists of 3 millions URLs reduced to 1,2 million unique GET URLs after statistical analysis. We found common patterns in these URLs resulting in clusters that can be used to produce high quality signatures. We can also associate to clustering quality measures to the signatures to assess their relative accuracy.

1. Introduction

In a few decades, the Internet has become a central marketplace where anyone with a credit card can do business. This extraordinary evolution came with many information security problems. One of the main threat is malware. Malware is often distributed on the web using URLs. It uses the web to transfer commands and information. In this section, we define important terms of this paper and then explain the goals and hypotheses of our study.

1.1. Vocabulary

A *URL*, Uniform Resource Locator, is the identifier of a resource in the World Wide Web.

> Familiar examples include an electronic document, an image, a source of information with a consistent purpose (e.g., "today's weather report for Los Angeles"), a service (e.g., an HTTP-to-SMS gateway), and a collection of other resources. [1]

As shown in Figure 1, a URL consists of several components: a scheme, a domain name or an IP address, a port, a path, query string and a fragment identifier. The query string is itself divided into pairs of keys, also called attributes, and values and fragments. Each component of the URL has, at least in theory, a specific purpose:

- The domain name or the IP address gives the location of the destination server of

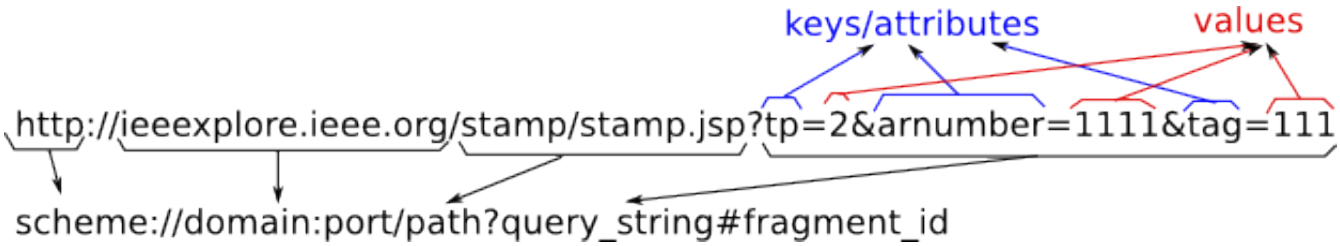

Figure 1. URL syntax

a request. The domain name is case insensitive.

- The port is the destination port of the web server running on this server. When no port is specified, standard ports are implied. HTTP uses TCP port 80 and HTTPS TCP port 443 as standard ports.
- The path is used to specify and find the resource requested. Although the path supposed to be case sensitive, some web servers such as Microsoft IIS treat the path as case insensitive.
- The query string contains data passed to the application running on the web server, such as a website. It can be for example the search query of a search engine.
- The fragment identifier specifies a position within the document. It can be used for example to create a table of contents with links.

In our dataset, we collected only URLs using the HTTP scheme on the standard port 80 without any fragment identifier.

In this work, malware were collected by a third party. By *malware*, we designate here a malicious binary for a Windows environment. This dataset contains different types of malware [2]: dropper, trojans, worms, virus, ransomware, etc.

By *malicious URL* or URL request by a malware, we designate an URL that appeared in the HTTP flow capture when the binary sample was executed in a sandbox (unrelated traffic was filtered out). Malware can request URLs serving malicious goals, such as downloading other malware, but we discovered also URLs of legitimate websites without a malicious purpose such as Google Analytics.

Clustering, also known as cluster analysis, is defined in [3]:

> Clustering analysis is a generic name for a variety of mathematical methods, numbering in hundreds, that can be used to find out which objects in a set are similar.

In this work, we describe clustering algorithms in the data mining and machine learning fields which create groups, called clusters, of URLs sharing common patterns.

We decided to call *experiment* in this paper a set of parameters for the algorithms we use. Creating an experiment means that we took a dataset, chose parameters for clustering (including distance functions), typing, visualization and quality evaluation on the platform we developed and then launched the computation task.

1.2. Goals

Our work aims at discovering families of malware by grouping similar URLs and producing high quality signatures. To do so we rely on data mining and machine learning algorithms for clustering. These signatures could then be exploited for filtering or detection in a mixed traffic environment, such as an operator network.

1.3. Assumptions

To decide what could be interesting data and properties to study malware similarities, we had to make some assumptions.

Security vendors classify malware samples they collect into families. In a family, samples have similarities with which security vendors try to base their signatures or anomaly definitions. For example the Microsoft Windows Malicious Software Removal Tool protects Windows users against 175 malware families at the time of this writing [4]. Although Microsoft receives thousands of malware samples a day, they usually share traits with samples of an existing family. Such binaries are called variants. This classification allows for tracking malware authorship, correlating information (writing an indicator of compromise for e.g.) and identifying new threats.

We made the choice to focus our study on communication initiated or received by malware. A typical case where communication is needed is for botnets [5] where the C&C server send tasks to zombies and gather information about the targets.

This paper examines URL patterns for HTTP based URL paths and queries, we do not consider domain names. We prove that some malware share commons patterns for URL paths and queries requested by malware samples. The discovery of these patterns would be a hint at malware families or code reuse between malware communication modules.

By taking this approach, we deliberately do not consider URL polymorphism such as a htaccess file used to rewrite URL paths to random ones. Such an approach is lightweighted in the future malicious traffic detection step. This type of techniques has been discovered in the the recent versions of the RedKit exploit kit [6]. While such methods are becoming increasingly common for exploit kits and domain names with domain generation algorithms (DGA) including web malware [7]. To the best of our knowledge, they are not popular for paths and queries of URLs requested by malware.

Our choice for the HTTP protocol rather than, for example, IRC or P2P-based communications is due to the prevalence of HTTP based botnets nowadays [8]. This protocol is difficult to filter, especially using a whitelist rather than blacklist. A malicious behavior can hide behind legitimate HTTP traffic generated by the victim and legitimate websites such as Twitter can be used as C&C [9].

We focus on paths and queries of URLs because we think that the studies of paths and queries on one hand and domain names on the other hand are distinct since they do not share the same semantic, the control needed by the attacker is not the same. Paths and queries usually depend on the victim by including information about the victim's system whereas the FQDN part of an URL is often used by the infrastructure obfuscation in the case of a botnet where a DGA is implemented [5].

2. Data Mining and Machine learning

In this section, we give a quick introduction to data mining and machine learning. We look into the main concepts and applications.

2.1. Definitions and examples

Data mining means analyzing data to found hidden information in it.

Machine learning consists of having computers learning from data, not entirely on code. For several decades, machine learning has revolutionized many fields and made new technologies such as speech recognition, better web search, recommendations, self-driving cars possible. Machine learning even helped us understand the human genome. Machine learning is now part of everybody's life whether we see it or not. Machine learning

techniques are also powerful to analyze trends and patterns from big data. Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed" [10]. Tom M. Mitchell came with a more formal definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [11]. For example, suppose you have an email program watching which emails you do or do not mark as spam. Based on that, the program learns how to better filter out spam. T consists of classifying emails as spam or not spam. E is watching you label emails as spam or not spam. P is the ratio of emails the algorithm correctly classifies as spam or not spam.

2.2. Brief history

In 1952, Arthur Samuel developed the first game-playing program thanks to machine learning. It was designed for checkers and was smart enough to compete against world champions. Machine learning was also popular in the 1960s. In the 1970s, AI was dominated by expert systems at the expense of machine learning. But in the mid 1980s, the creation and implementation of decision trees allowed for the comeback of machine leaning. During the same period, a lot of improvements were made on neural networks. The two approaches had huge implications not only in the financial sector for loan approval, fraud detection or portfolio management but also in the industry. In the 1990s, the fast evolution of the Internet allowed sharing huge volumes of data. New algorithms were developed to make sense of it such as Support Vector Machine (SVM). In the beginning of the 21st century, logistic regression came back to light and allowed the development of large scale machine learning systems. A lot of other algorithms were also developed, machine learning gave birth to a big amount of algorithms that have specific properties. They have been developed since the 1950s and have made dramatic progresses since then so that they are used in diverse fields and play majors role in AI. To give an example, almost every internet company uses machine learning one way or another. With the big data phenomenon machine learning has become mainstream in the sense that a lot of computer scientists, not especially specialized in AI, implement them daily.

The term "Data Mining" appeared in 1989. The same year, a new Special Interest Group (SIG) of the Association for Computing Machinery (ACM) hosted its first international conference [12].

2.3. Machine learning for information security

Because the information security world relies on data, machine learning seems to be an effective tool for security analysts. Historically, the first security system to massively rely on machine learning for production systems is spam filtering for emails. Naïve Bayes classification algorithm was the first popular machine learning algorithm used for spam detection. Variants and other algorithms have improved detection ever since [13].

Internet companies have, sometimes from their very beginning, tried to analyze data [14]. They have made great efforts on scaling machine learning algorithms.

Literature also studies user anomaly detection to detect attackers using a system [15].

Literature is abundant about data mining techniques for network anomaly detection but sometimes has limited success [16].

2.4. Supervised and unsupervised learning

Machine learning algorithms are mainly divided into two distinct learning methods: supervised and unsupervised [17].

Supervised learning algorithms create a function based on training data. Training data are pairs of input entities, often represented as vectors, and labels or values describing the expected output. When the output function is continuous the algorithm class is called regression, when the function is discrete the algorithm class is classification. The job of a supervised learning algorithm is to learn from training data so that it can afterwards make a prediction for any other similar data. Popular classification algorithms include neural networks (NN), support vector machines (SVMs) or Naïve Bayes. Linear regression and logistic regression are examples of well known regression algorithms.

Unsupervised learning algorithms have different goals. They do not require the labeled training data to generate training data. They usually take as input not only a dataset but also parameter settings. Two main classes of unsupervised learning algorithms exist: clustering and decomposition. Clustering techniques try to create clusters which are groups of data points sharing common traits. When a dataset or points properties are represented by a vector, we may want to reduce the number of dimensions used to filter out irrelevant data for relevance evaluation, performances and visualization. Dimensions are combined into fewer ones while keeping most of the information. Principal component analysis (PCA) and single value decomposition (SVD) are common examples of dimension reduction algorithms.

2.5. Clustering algorithms

Several clustering algorithms have been proposed [18]. They are to be used depending on what we consider a cluster for our data and what is the best approach to discover them. Clustering algorithms can be classified depending on their cluster models. We introduce main families of cluster models.

Hierarchical clustering creates clusters using their distance value. They can be top-down or bottom-up. Bottom-up (agglomerative) methods consider each object as a cluster and then create bigger clusters by merging them until we have a single hierarchical tree with all clusters in it. Top-bottom (divisive) methods consider one cluster including all objects and define a method for splitting a cluster. It splits clusters recursively until each object is in a singleton cluster. The precision of the cluster, therefore divisions, are based on distances. This family is also called connectivity models. Results can be represented as dendrograms. CURE [19] and BIRCH [20] are hierarchical clustering algorithms.

Centroid models define "the average across all the points in the cluster" [21]. When using the k-means [22] algorithm, the number of desired clusters k (which is also the number of centroids) is specified. This algorithm finds the k centroids and then assigns each point to the nearest centroid. Differences between variants are the methods for selecting these k centroids. The default algorithm selects them randomly and is generally run several times. We keep the best attempt to minimize the squared distances from the cluster.

Distribution models use probability distributions and a cluster include objects that most likely belong to the same distribution. This model fits well sampled random data but can have problems with noise.

In density models a cluster is an area that have a higher density than average in the dataset. In this model we generally have a noise concept so that isolated objects in sparse areas are considered as noise. DBSCAN [23] and OPTICS [24] are examples of density-based algorithms. DBSCAN takes as input k, the number of minimum objects in a cluster,

and ϵ, the radius of the neighborhood of an object.

Table 1 gives a summary of popular clustering algorithms. Other models such as subspace models, group models or graph-based models are not discussed in this paper.

2.6. Features and distance function

Features are attributes on objects that we want to consider for our learning algorithms. We may have one or several features. If we want to study price of housing in an area for example, features can be the number of rooms and the surface area. One of the main problem of machine learning is the selection of relevant features for a dataset. Based on these features a distance matrix, also called similarity matrix, is often defined. It indicates how similar or distant are objects of our dataset with respect to each other. This requires a distance function which depends on the nature of our features.

- Binary features of an object have True or False as values. Russell and Rao Index, Hamming distance or Jaccard distance are good distances for binary features.
- Quantitative features have number values. Some distance functions that can considered are Euclidean distance, Manhattan distance, Chebyshev distance, Minkowski distance or Canberra distance.
- Nominal features are features that cannot be represented by numbers. A sentence for example cannot be represented by a quantitative value without loosing information. Distance functions have been created to deal with nominal features depending on their types, strings for example.

2.7. Evaluation and quality

We saw that we have several algorithms, settings and distance functions. We need to know how to choose between them to have the best result. We also need to evaluate hypotheses we make for a given dataset. The No free lunch theorem of Wolpert [25] states that without prior assumptions about the nature of the learning problem, no machine learning algorithm is better or worse to any other (or even to random guessing). We want to make good choices for model selection and model assessment. Model selection describes the performance of different hypotheses and algorithms for a problem. Model assessment is the results of the final algorithm and parameters on a new dataset.

Statistics are used to analyze features and results. Usually a loss function is defined. For regression it can be for example squared error loss. The goal is to minimize it. Several problems may appear related to our hypotheses and training data. To illustrate them, we consider a regression problem as an example. We define h a linear hypothesis such that the sum-squared error over the training data $< x_i, y_i >_{i=1...m}$ is $\sum_{i=1}^{m}(y_i - h(x_i))^2$ The bias, also called systemic error, is $h(x) - f(x)$. It can be seen as the average error of $h(x)$, a high bias means low sensitivity. The variance represents how different $h(x)$ is depending on the training data, a high variance means low precision. An error rate can be decomposed as follows: Expected prediction error = Variance + Bias^2 + Noise^2. Overfitting occurs when hypotheses are too close to training data and do not provide a good generalization so that the result differs a lot with real data. High variance results in overfitting, results may be better with more training examples and by trying a smaller set of features. Underfitting is the result of high bias, a solution could be to try a larger set of features.

For unsupervised clustering results evaluation, we have two approaches: internal evaluation and external evaluation. External evaluation requires labeled data known as ground truth whereas internal evaluation does not. It is based on internal properties of produced

Table 1. Computational complexity of Clustering Algorithms

Algorithm	Complexity	Setting parameters	Notes
K-means	$O(NKd)$	k: number of clusters seed: random seed to initialize centroids	
Hierarchical Clustering	$O(N^2)$	cut function	Cut needs to be defined
BIRCH	$O(N)$	Distance threshold Distance function Maximum node entries	Identifies only convex or spherical clusters of uniform size
CURE	$O(N^2log(N))$	k: number of clusters	
DBSCAN	$O(Nlog(N))$	k: minimum number of point in a cluster ϵ: radius around each object	Can provide clusters of different sizes, noise filtering

clusters. Internal evaluation usually consider a good clustering when a cluster has high similarity between its members and low similarity with other clusters. To use these indexes we have to make the claim that some sort of structure exists in the dataset. Furthermore, these indexes are tied to the properties of the clustering model used. Without a ground truth, it is difficult to prove that we have discovered all or sometimes even some hidden patterns in data. However, it is often impossible to have a ground truth for a machine learning problem. The best evaluation method depends on what is a good cluster with respect to what we consider as a cluster for a given dataset.

Some internal evaluation methods:

- The Davies-Buildin index [27] computes for a cluster the mean distance between member points and the centroid.
- The Dunn index [28] can be used to identify dense and well-separated clusters. The Dunn index is the quotient of the minimal inter-cluster distance divide by the maximum intra-cluster distance.
- When ordering the similarity matrix by objects that belong to the same cluster, we can visually see if objects in the same cluster are close to each others, see Figure 2 and Figure 3.

Some external evaluation methods:

- The Rand index [29] describes a measure of the percentage of correct decisions made by the clustering algorithms. It takes as input the number of true positives, true negatives, false positives and false negatives.
- The Jaccard index [30] gives the similarity between two datasets. The Jaccard index is the number of unique elements that are in both datasets divided the total number of unique elements.
- Adjusted mutual information [31] can be used with the ground-truth and the resulting clustering to measure their mutual dependence.

3. Related work

Rafique et al. created FIRMA [32] which takes a network capture as input and generates family clusters of malware and network signatures. Contrary to our algorithm, FIRMA needs full benign traffic captures, which are more difficult to obtain due to privacy issues. They base their analysis not only on URLs or even HTTP but extract features for HTTP, SMTP, IRC, UDP and TCP protocols. Regarding URLs, they only consider keys in the query string with the Jaccard index

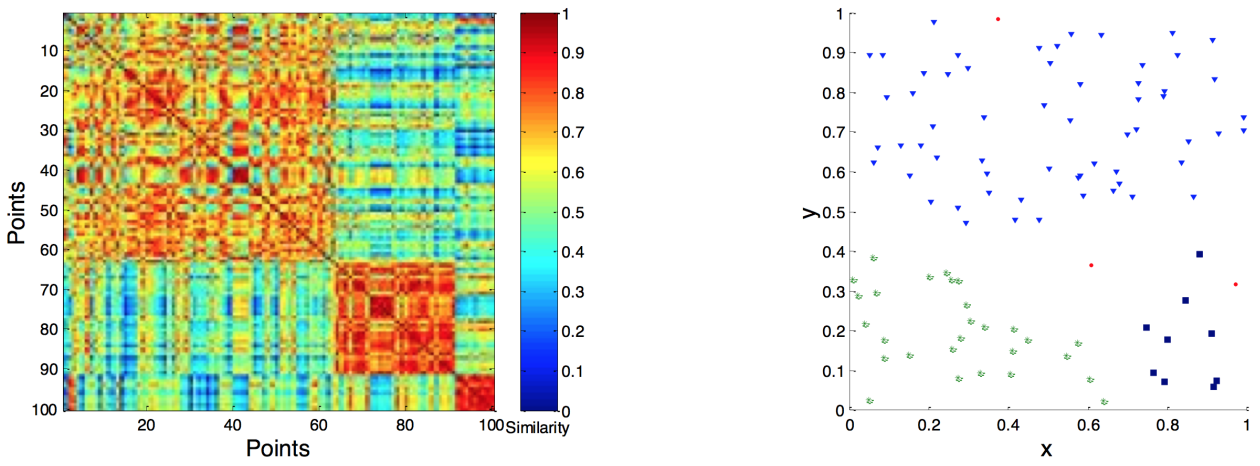

Figure 2. Ordered similarity matrix when DBSCAN is used on random data [26]

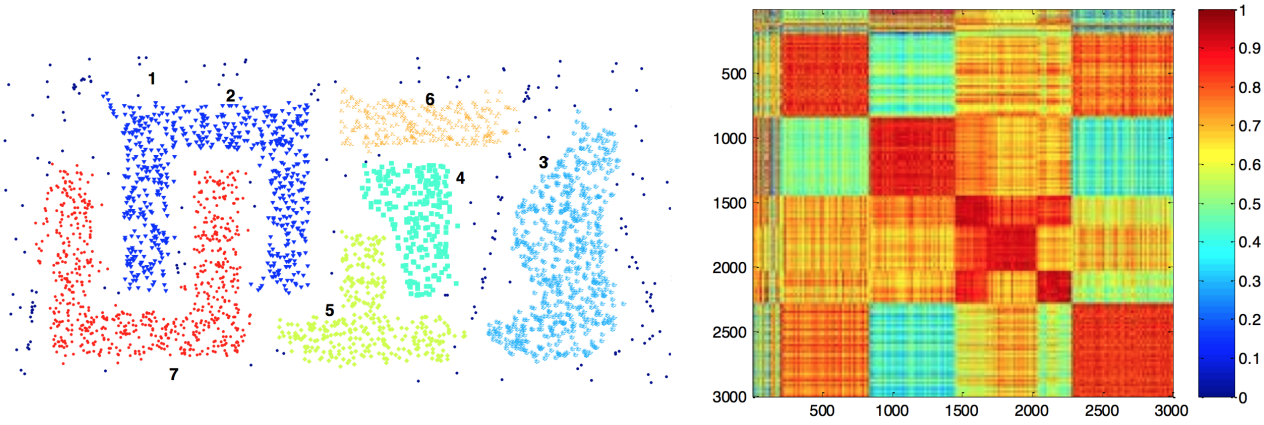

Figure 3. Ordered similarity matrix when DBSCAN fits well the dataset [26]

(see Section 2.6). Although they have good results with the malicia dataset [33], their signatures are valid only for a short time. They perform two steps of clustering, the first is based on protocols based features extracted from network captures, then they generate signatures which are merged later. They use a second clustering algorithm on the resulting signatures. To apply these signatures IDS have to support many protocols and have good performances for a lot of fields. Some signature generations are not automated. Due to the nature of their dataset, they have access to the true malware families, the ground truth, to evaluate their work. Therefore they use F-measure for performance evaluation on their dataset. A ground truth is not easy to find and for malware family the ground truth often depends on AV vendors research.

In [34] Le et al. study with PhishDef phishing URLs that are used to spread malware or steal sensitive information, so before malware are executed on a victim's computer. They observed a lot of obfuscation but on the domain name only and they also have to consider noise. Their work uses supervised online clustering and aims at client-side deployment. They prove that adding external features (such

as WHOIS or Team Cymru information) to lexical features (in our work we only rely on the latter) does not provide more accuracy for classification and give more than 95% overall accuracy on their datasets. Their features include the domain name of the URL which we do not select because we do not believe that they have significant correlation with malware families. However finding malware families is not the goal of Le. et al. in their work.

Jacob et al. [35] conducted their analysis on botnets communication only on host-based features using system calls.

In [36] Perdisci et al. analyze HTTP traces of malware, not just URL paths. They take features by separating malware from each other and consider for example the number of GET and POST requests by malware. They consider data sent by POST request, collecting those in real world data can be a privacy issue. This can split malware families into different clusters when malware authors add features and does not take noise into account. Because they use hierarchical clustering and three clustering steps, they need to define three cut functions and thresholds. Noise may affect their algorithm.

To the best of our knowledge, no similar study includes a typing step before clustering which is one of our contributions and allows to consider most of the dynamic values such as MD5, timestamps, etc. to be considered as token and not consider their length. As far as we know we are also the first to consider only paths and to use DBSCAN to create clusters of them.

4. Our dataset

Our original dataset consists of over 3 million URLs. The collection was done by a third party security vendor on malware binaries for Windows environments. They include all type of malware: rootkits, droppers, trojans, etc. The dataset is proprietary. Figure 4 illustrates the collection process. Malware was executed in a sandbox where network communications were captured and noise such as legitimate software updates were filtered out. This process is not a contribution of our work and therefore is not detailed here. Once we have all the network captures, we can extract URLs. We generate a dataset file with one path per line. After removing duplicates, we have a dataset of 2 million URLs. After removing domain names and selecting only GET requests, we end up with a dataset of 1.2 million URLs.

5. Preliminary statistical analysis of the dataset

A statistical analysis is often performed to know what features to select for a dataset. It also gives an insight about the dataset so that we can know what kind of shapes the clustering algorithm has to find. This information is essential to select the best clustering algorithm for this dataset.

In current literature, the common features of an URL can be divided between the domain name, the path and the query string of the URL. For the path, the most popular features are:

- Length of the path
- Number of subpaths
- Lexical features, usually stored as a “collection of words” which is a simple list of keywords such as “images”, “img”, “admin”, “paris”, etc.
- File extension type. E.g.: html, jpeg, png, exe, etc.

For the query string (key-value pairs), the following features can be used:

- Length of the query string
- Number of keys
- Lexical features, usually stored as a “collection of words” which is a simple list of keywords such as “id”, “name”, “page”, etc.
- Value data types such as real, integer, MD5, SHA1, etc.

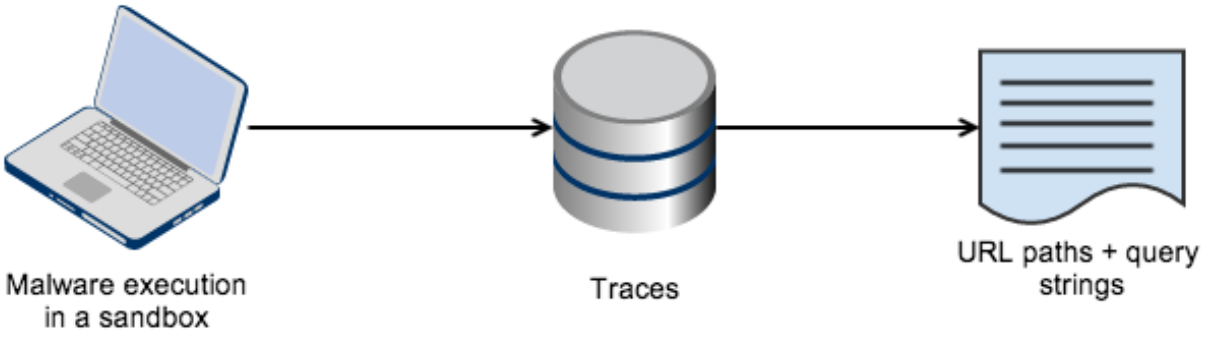

Figure 4. Malware collection, credits to Everaldo Coelho and YellowIcon, License LGPL

5.1. Verbs

Table 2. HTTP/1.1 verbs

Name	Description
GET	Retrieve information. In practice, GET is most used verb. GET requests have no browser-supplied payloads.
POST	Submit information to the server. A payload is usually attached which is not visible in the URL
HEAD	Identical to GET but return only the HTTP headers. Rarely used.
OPTIONS	Return supported methods for a given URL.
PUT	Upload files to the server. Not implemented by browsers which often use POST instead.
DELETE	Delete a specified resource. Almost never used.
TRACE	Return information about all the proxy hops encountered in processing a request, like a "traceroute". Often disabled by server administrators for security reason.
CONNECT	Establish non-HTTP connections using HTTP proxies.

In HTTP/1.1 provides several request types called verbs. Table 2 gives an overview of HTTP/1.1 verbs [37].

As shown in Figure 5, our initial dataset consisted of approximately 2/3 of GET and 1/3 of POST requests. Data in POST requests is not in the URL but in HTTP headers. Analyzing HTTP headers rather than just URL paths and query strings poses more privacy issues because POST data is more likely to include credentials for non-encrypted communication for example. For these reasons, we decided to select only GET requests for our study. From now on in this paper we analyze only GET requests.

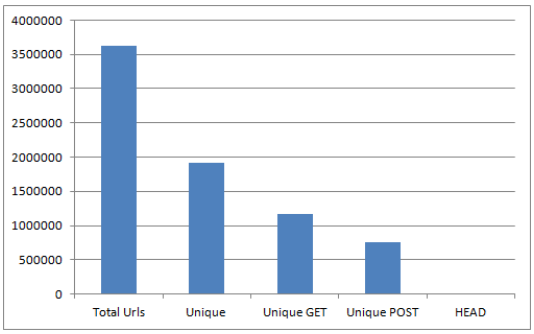

Figure 5. HTTP request by verb for our full initial dataset

5.2. Length

One of the easiest way to discriminate URLs is to look at their lengthwise distribution.

Some studies [36] have used domain name's length as a feature for coarse-grained clustering of URLs requested by malware. In Figure 6 is the lengthwise distribution of

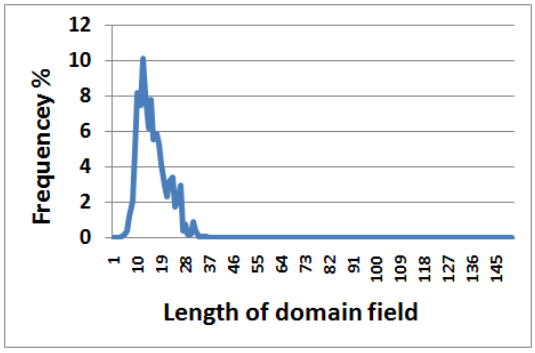

Figure 6. Frequency distribution - Length of domain name of the URL in the dataset. It can be observed that the graph has a long tail distribution.

domain names in our dataset. We can observe that most URLs have domain names not exceeding 50 characters. Malicious domain names tend to be longer than most legitimate domain names [38].

As shown in Figure 7 malware use obfuscation techniques. They are efficient to evade URL blacklists or pattern signatures. We designate by query fields a unique pair of key/value in the query string. We see that after 100 characters in the domain name, very few URLs have a query string.

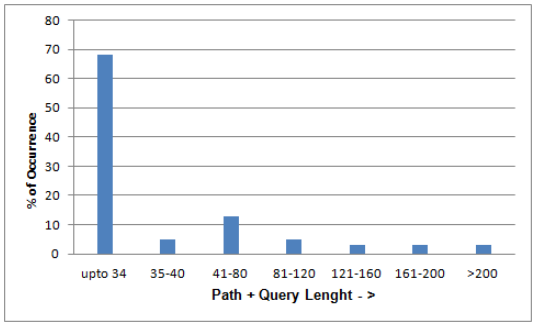

Figure 8. Frequency distribution of the sum of the path and query string lengths

We have plotted the frequency distribution of the sum of the path and query string lengths for all URLs in the database in Figure 8. As we can see, for almost all URLs this sum is less than 100 characters. For 65% URLs, this sum does not exceed 32 characters. A MD5 hash has a length of 32 characters and none of the URLs having the sum of their path and query string length up to 32 characters had a MD5. A SHA1 hash has a length of 40 characters and a base64 at least 76 characters. So, at least 65% URLs do not include any hash or serialization using these algorithms. 15% of the URLs have a length sum between 40 and 80 characters. We saw that URLs whose length sum is greater than 120 characters usually include a HTTP redirection or a base64 value.

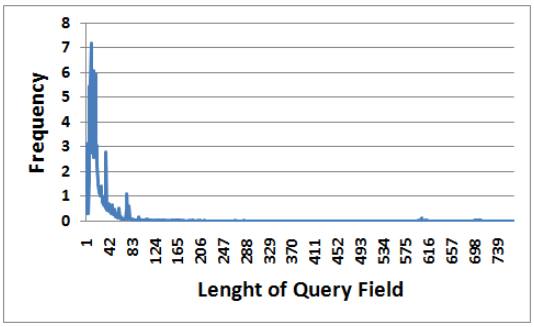

Figure 9. Frequency distribution - Length of query strings of the URLs in the dataset.

We decided to look inside the length of query strings in Figure 9. We can see that the graph has a long tail distribution. Most of the query strings have short to medium query string length. Such kind of distribution raises the possibility of dividing dataset on the basis of length. In the long tail we mostly have URLs that include MD5, SHA1, URLs or base64-encoded data in their query string.

5.3. Subpaths and query fields

The distribution of path length with respect to the number of subpaths in our dataset is given in Figure 10. The number of subpaths does not seem related to the path length since both short path length and long path length URLs tend to have up to 10 subpaths. Thus

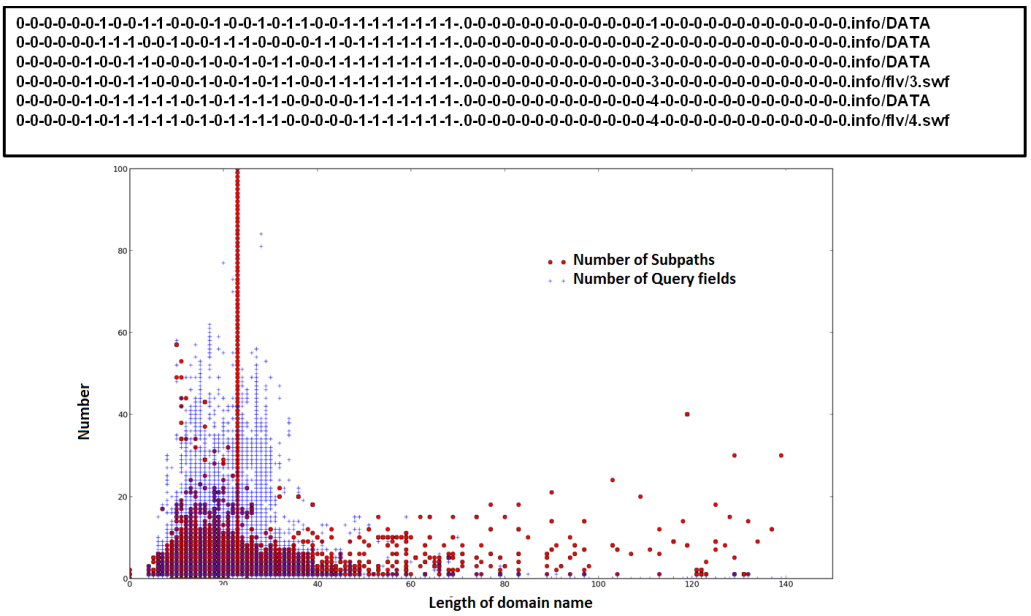

Figure 7. Domain name obfuscation example in the dataset

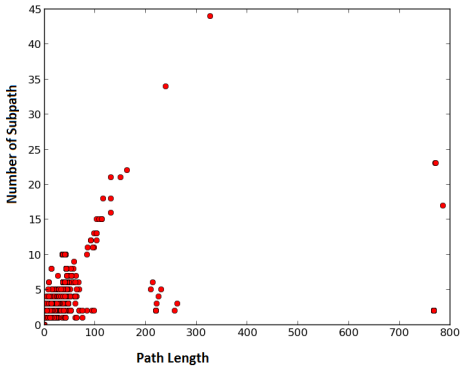

Figure 10. Path length vs number of subpaths

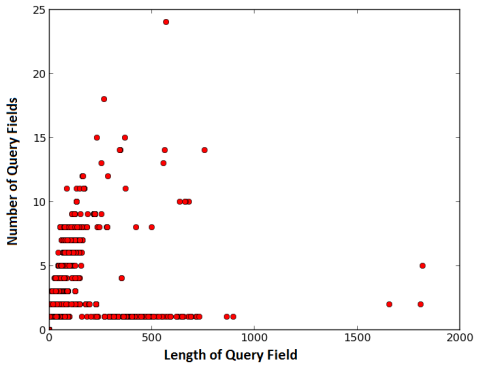

Figure 11. Length of the query string vs number of query fields

we can infer that the number of subpaths and the path length are independent features.

Figure 11 shows the distribution of the query string length with respect to the number of query fields (unique pairs of key/value in the query string) in our dataset. While most query strings have less than 400 characters, we do not have a strong relation between the query string length and the number of query fields. The maximum length of a URL depends on the client and server combination but is usually around 2,000 characters.

5.4. File extensions

We plotted in Figure 12 the distribution of extension types for URLs having the sum of their path length and query string length less than 40 characters. While we would not have a lot of different values if we chose to use file extensions as a feature, we have an interesting

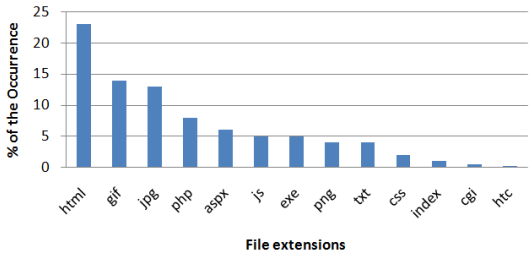

Figure 12. Distribution of extension types for URLs having the sum of their path length and query string length less than 40 characters

distribution since we have only a few outliers and many significant percentages for different extensions. The file extension in a URL is not always associated with the corresponding file format because of obfuscation and filtering evasion techniques.

5.5. Bag of words

Figure 13. A pictorial representation of the bag of words of keys in query strings of our dataset. Most occurring keys are larger than least occurring ones. Some of the most occurring words are from benign URLs used by malware in our dataset.

A *bag of words* is a set of unique words associated with their occurrence in a given text. It can be seen as a way to store word frequency. We represented in Figure 13 the bag of words of keys in query strings of our dataset.

Because the query strings is often accessible by a key-value dictionary for web developers, the key should be in most cases easily recognizable and understandable. This way, we can filter out keys associated with non-malicious popular websites:

- Several URLs contain terms such as "id", "ver" or "version". These keys occur frequently in both benign and malicious traffic.
- Keys containing the "utm" prefix such as "utmhid", "utmwv" or "utmac" are associated with the popular Google analytics service.

We can also see in Figure 13 the keys "SAIRND", "browser" and "locale". One of the URL in our dataset containing these keys is:

```
uci.secure-softwaremanager.com/
?SAIRND=300125&icname=GPLCPLite43
&icversion=15623
&tid=4/1/2011%204:49:10%20AM&os=6.1
&locale=fr-FR&browser=FIREFOX.4.0
```

This URL is generated by Troj/Mdrop-DGS. The keys "browser" and Locale are too generic and can be found in various URLs. However, we can easily identity SAIRND as one of the markers of Troj/Mdrop-DGS.

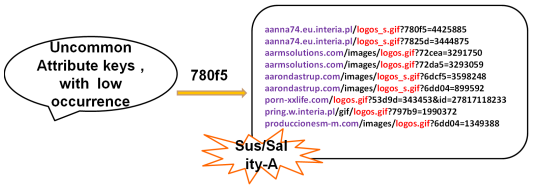

Figure 14. By tracking key "780f5" in our dataset we found traces of Sus/Sality-A.

We also looked into some keys having very

low occurrences in the bag of the words but were suspicious in nature. As we saw earlier, in a URL a key should be easily recognizable and understandable. Therefore, keys not fulfilling these criteria raise suspicion. For example, by tracking key "780f5" in our dataset we found traces of Sus/Sality-A [39], see Figure 14.

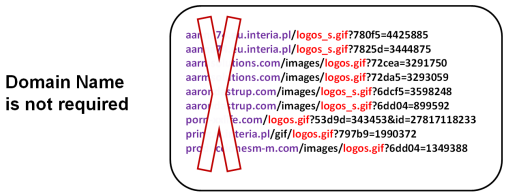

Figure 15. Domain name is not a good criteria for clustering a malware.

Figure 15 also proves our hypothesis that the domain name is not a good criteria for clustering since a malware can easily change its host but its path and query strings signature remains similar. Here, "logos*.gif should be part of the signature of the malware.

This preliminary analysis on the URLs motivates us to try to classify URLs based on some patterns exhibited in path and query string.

6. Methodology and implementation

We propose a novel architecture to solve our problem (Figure 16). Because our dataset of 1.3 million URLs is very large, we decided to perform two successive clustering steps instead of one. We first have a coarse-grained clustering and we perform a fine-grained clustering on coarse-grained clusters. Between the two clustering steps, we replace values in query strings by their type because of polymorphism, values specific to a victim or binary in order to have clusters for which we can easily create signatures. After the final clustering step, we have fine-grained clusters with typed values. We can then visualize our clusters in two or three dimensions and evaluate their quality. Finally we generate a signature for each fine-grained cluster.

We integrated almost all our tools in a web interface where with a few clicks we can set parameters for the different steps and analyze results.

In this section we introduce our implementation and algorithms.

6.1. Coarse-grained clustering

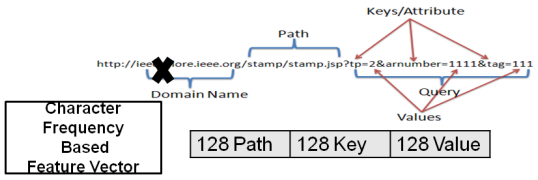

Figure 17. Features used for k-means clustering are based on letters frequency. We separate the path, keys and values.

The objective reason of this clustering step is to improve performance by creating coarse-grained clusters which should be smaller input files for the fine-grained clustering algorithm. So we need to find features and a clustering algorithm that are fast to compute, create clusters as equal in size as possible. This step should not deteriorate results too much.

To meet these requirements, we initially tried statistical features on our URLs because these features are often used in the literature [36]. They included: number of subpath fields, total length of subpaths, number of query fields, length of query string, etc.

We selected k-means [22] as clustering algorithm for this step because of its low complexity and because we can specify the number of clusters we want, see Table 1. Furthermore, we needed an unsupervised clustering algorithm such as k-means because our goal is to completely automate malware families discovery.

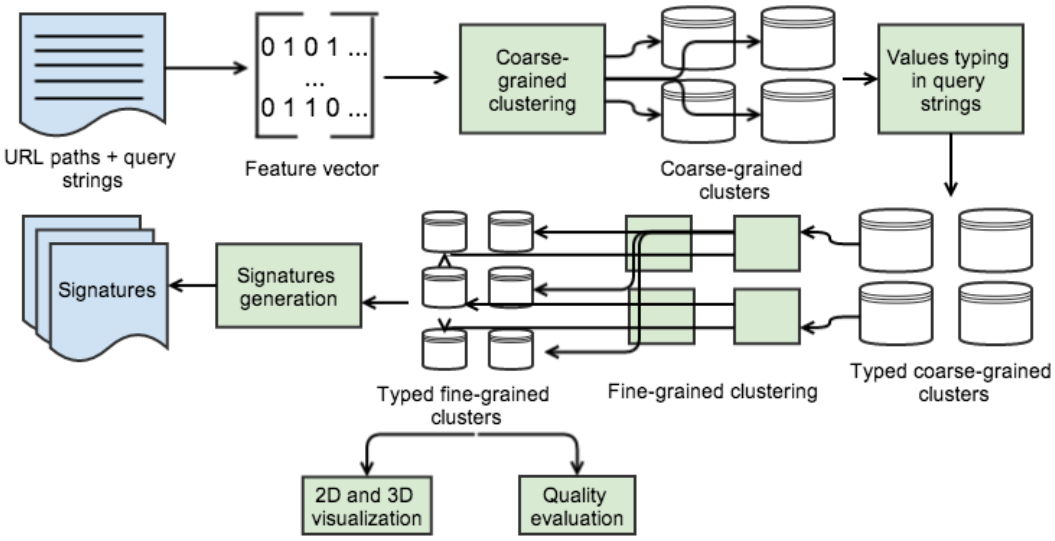

Figure 16. Proposed architecture with two steps clustering and typing after the first clustering algorithm

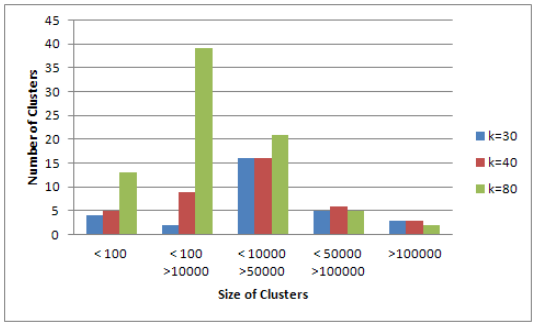

Figure 18. K-means clusters distribution

However our dataset tends to include a lot of short URLs. Because of this, nearly 70% of our URLs were in the same cluster. Having such a majority of URLs in one cluster does not satisfy our requirements so we decided to try other features. We also saw URLs in the same cluster while not having a single common character but only the same length and number of subpaths.

Because of this phenomenon, we needed features that are more string-based. We selected the frequency of a character in a URL. The HTTP standard specifies that characters in a URL have their ASCII value lower than 127. Therefore, we divided our feature vectors into three parts : path, keys and values. Each one of this part is a vector of 128 dimensions (value 0 is valid). The feature vector is the concatenation of these three vectors so that its size is 384.

For example in the path */docs/latest/widgets*, the occurrence frequency of the character: o is 1, e is 2, / is 3 and z is 0.

Figure 18 gives the size of coarse-grained clusters with respect to the value of k which is the number of clusters. We see that after $k = 30$ we add very small clusters which is not desirable because it does not tend to an equally distribution of size of the clusters. Having too many very small clusters would also require us to add a merging step in our architecture. Empirically, we verified that the clusters are not worse with $k = 30$ than with higher values. Because of these reasons, we chose $k = 30$ for the k-means algorithm. We implemented features extraction for coarse-grained clustering and k-means with python

using the mlpy library [40].

6.2. Typing

After the coarse-grained clustering step, we have a typing step. We replace the values in query strings by their types by parsing them. Table 3 shows that we discovered 13 important types in the dataset.

Because of its different subtypes, base64 is particularly hard to parse. Perfect URL parsing is impossible because browsers have specific support for some special characters. URL encoding is also hard to implement in a regular expression [37]. Table 4 provides an example of our typing for a few URLs. As we can see, duplicates are generated by parsing and need to be removed.

We decided to type values in query strings because more than 70% of them in our dataset match with a type. Because changing the type of a value demands far more effort than changing just the value, we believe that this step allows our algorithm to be more efficient against polymorphism. We saw in our dataset that values often depend on the environment of the victim's computer, for example their MAC address, a unique victim identifier, etc. With typing we take this into account. We also think that signature generation is easier to implement in a final step with this typing step. Finally, this step reduces our clusters. This improves performance of the computation-intensive fine-grained clustering step. We implemented typing using sed and regular expressions. In the following steps, we assume that values in query strings are typed.

6.3. Fine-grained clustering

The fine-grained clustering is the step that takes most computation time. That is why we inserted it after the coarse-grained clustering and the typing steps.

The features are the path, keys and value types of URLs. Using these features we defined a distance function.

The initial distance function is the mean of three distance functions : one for the path, one for keys and one for value types.

- The path distance is based on the longest common substring (LCS). The longest common substring of two paths is the longest substring that is in both paths. For example the longest common substring of "images/a.gif" and "img/b.gif" is ".gif". The path distance between two paths is the longest common substring of these paths divided by the length of the longest of these paths.
- The key distance is based on the Jaccard distance. The Jaccard distance of two sets A and B is defined as $d_J(A,B) = 1 - J(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}$. Here we consider the set of keys of an URL. For example, assume we have URL A and URL B sharing two common keys. URL A has three keys and URL B has two keys in total. $J(A,B) = 1 - \frac{2}{3} = \frac{1}{3}$. If URL A and URL B had no common keys, their distance would have been 1.
- The value type distance is the same as the key distance but using value types instead of keys.

This initial distance gives the same weight to keys and value types but we observed that keys and value types are tightly linked. So, we developed another distance that considers key/value types pairs instead of separating these two components. This distance is the combination of two distances:

- The path distance is the same as in the previous distance function.
- The keyByVal distance considers key/value types pairs. Using Jaccard distance, we select pairs that share common value types. We compute the longest common substring on the keys of these selected pairs.

Table 3. Types in values of query strings

Name	Example	Description
MD5	4f863423326e85d44aae147d2d86e1c0	Cryptographic hash function consisting of 32 hexadecimal figures.
SHA1	97d07314f735998585bb8e2d6b5acb5ac7956690	Cryptographic hash function consisting of 40 hexadecimal figures.
Base64	dG90bw==	binary-to-text encoding schemes to represent binary data using ASCII or UTF-8 string formats.
URL redirection	http://example.com	Can be used for phishing attacks or obfuscation using URLs similar to legitimate websites.
Float	0.5	
Integer	42	
Boolean	true	Can have true of false value
Resolution	800x600	Number of distinct pixels in each dimension that can be displayed.
Hexadecimal number	ff4eb	Representation of a number in base 16.
MAC address	0a:00:27:00:00:01	Identifier of a network card.
File path	C:\test.txt	Location of a file
Timestamp	2008-02-14	Identifies when a certain event occurred.
Country code	FR	Identifies a country
No type	utv42	Any value not matching a previous type

Table 4. Typing example for a few URLs

Before typing		After typing		After removing duplicates
/1.exe?t=0,0378992		/1.exe?t=Float		
/1.exe?t=0,1008417		/1.exe?t=Float		/1.exe?t=Float
/1.exe?t=0,1019098		/1.exe?t=Float		
/cool.gif?t=0,361706		/cool.gif?t=Float		
/cool.gif?t=0,3632929		/cool.gif?t=Float		
/cool.gif?t=0,3652765		/cool.gif?t=Float		/cool.gif?t=Float
/cool.gif?t=0,3674585		/cool.gif?t=Float		
/cool.gif?t=0,3681452		/cool.gif?t=Float		
/cool.gif?t=0,3702814		/cool.gif?t=Float		
/cs.gif?t=0,1790583	$\Longrightarrow$	/cs.gif?t=Float	$\Longrightarrow$	
/cs.gif?t=0,1791803		/cs.gif?t=Float		/cs.gif?t=Float
/cs.gif?t=0,1792719		/cs.gif?t=Float		
/ftse2.exe?t=0,73164		/ftse2.exe?t=Float		/ftse2.exe?t=Float
/ftse2.exe?t=0,7345697		/ftse2.exe?t=Float		
/gggg.exe?t=0,1857721		/gggg.exe?t=Float		
/gggg.exe?t=0,1858026		/gggg.exe?t=Float		/gggg.exe?t=Float
/gggg.exe?t=0,1859552		/gggg.exe?t=Float		
/gggg.exe?t=0,1861994		/gggg.exe?t=Float		

When we have only paths in the two URLs, we use only the path distance. If with have only one URL with a query strings, we use only the path distance with a discriminating

factor. If both URLs have a query string, we take the mean of the two distances.

To understand why this distance is better than the previous one, consider the following example.
A = "index.php?a=integer&b=no_type&c=md5"
B = "index.php?a=no_type&b=integer&c=sha1"
C = "index.php?a=integer&b=integer&c=md5"
Our intuition shows that B should be closer to C than to A, so we should have $D(A, B) > D(B, C)$. This is false for the old distance function but true for the new distance function.

None of these distances is biased by the length of URLs. Also instead of considering keys and value types as strings, we consider them as sets belonging to an URL. Parameters order in the query strings are irrelevant.

For the clustering algorithm, we also adopted an unsupervised clustering algorithm to completely automate this clustering step. We wanted an algorithm that identifies noise because we have a lot of URLs that are very unique and are not representative of any malware family. Noise can also be used for evaluation of our clustering and the quality of our dataset. We also looked for low complexity because our distance already takes a lot of time and resources to compute. Finally, we needed an algorithm that does not take the explicit number of clusters as input because we do not know how many malware families are in our dataset. DBSCAN requires parameters related the distance between two points of the same cluster and the minimum number of objects to create clusters. While these parameters have an influence on the number of clusters, the latter is a result of the clustering process and not an input parameter. We found that DBSCAN (see section 2.5) fits with our requirements.

We implemented our distances using dynamic programming. Our distances and our DBSCAN implementations use MATLAB [41]. We also used parallelism so that we dedicate one core per fine-grained clustering of a coarse-grained cluster. Therefore, we can have up to 30 cores used. Despite these efforts, our global clustering algorithm still takes more than 6 hours to compute clusters for a dataset of 100,000 URLs on a server with more than 30 cores. It takes two weeks to have final clusters for the full 1.3 million URLs dataset. We believe that our distances are to blame for that because LCS takes time to compute and our key sets are very large.

6.4. Visualization

We implemented visualization tools to have a look at the shapes of our clusters. We wanted to confirm that a density-based clustering algorithm fits well with our dataset. To have visualizations of our clusters, we first need to compute coordinates of our URLs based on the whole distance matrix, then we need interfaces to see and navigate through the results in two and three dimensions where each point represents a URL. One goal of this visualization is also to give hints at how to tweak our clustering algorithms to have better results.

6.4.1. Dimension reduction. Dimension reduction is a machine learning and statistics problem. It consists in reducing the number of random variables considered [42]. Because we have a distance matrix, we used classical multidimensional scaling (MDS) [43] which takes a dissimilarities matrix as input and gives a coordinates matrix that minimizes a loss function as output. This algorithm tries to minimize information loss. One of its advantages is that it also reports how efficient the process is because we have variances (called eigenvalues) associated with the different axes. We keep axes that provide the most information. With that we can also compute the variance percentage of each axis and get an idea of how correct is our dimension

reduction. This is essential to know if we should use two or three dimensions.

In our dataset, we often have around 30% of the variance on three axis. This is too low to get a correct visualization. We only have a few clusters with more than 70% of the variance on three axis. So, when using visualization we need take into account that the representation is often highly biased by the dimension reduction.

We implemented classical MDS with MATLAB.

6.4.2. Interfaces. We first used matplotlib [44] (see Figure 19) to visualize our clusters. But one problem is that we did not see the labels of our points giving the associated URLs. Without the URLs, we cannot use visualization to tweak our clustering algorithms nor navigate through the results. Because of the big number of points, we could not either display all labels at the same time. We did not find any tool to do that directly, we decided to create our own visualization interfaces.

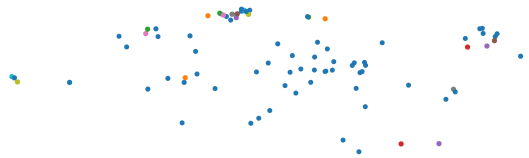

Figure 20. 2D visualization of fine-grained clusters of a coarse-grained cluster with 46% information variance on the x axis and 33% information variance on the y axis. One color is associated with each fine-grained cluster.

With the help of the D3.js javascript library [45], we created using mainly javascript a web page where we can select which cluster we want to visualize. It gives a two dimension representation (see Figure 20), when the mouse is on a point we have the URL associated. We have zoom and navigation features.

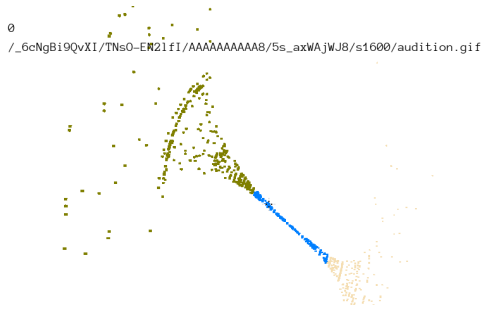

Figure 21. 3D visualization of coarse-grained clusters of a extract of the dataset. One color is associated with each fine-grained cluster.

Because the third axis also often contains a lot of information, we wanted to create an interface for 3D visualization. We first tried some libraries to use WebGL in the browser but the performances were not good enough because of the huge number of points we have (up to 1.3 millions). The only solution we found was to call on Urho3D game engine [46]. This impressive game engine has a built-in feature to optimize performances for duplicate objects. We used this feature by considering all points in the same dataset as duplicates and create our own scene. This tool runs on a client and not on a webpage using Linux or MacOS X. It can be ported with a few lines of code to Windows, Android or IOS. Figure 21 is a screenshot of this tool. 100,000 points are rendered at 30 frames per second. This tool is implemented in C++.

We think these interfaces are big contributions of this work since we did not find any similar and generic tools for this task.

6.5. Quality

Once we have our clusters, we need to evaluate them. Our goal here is to be able to label them with a quality value. In this way we could have a certain level of confidence our malware families and also compare our results for different experiments. We imple-

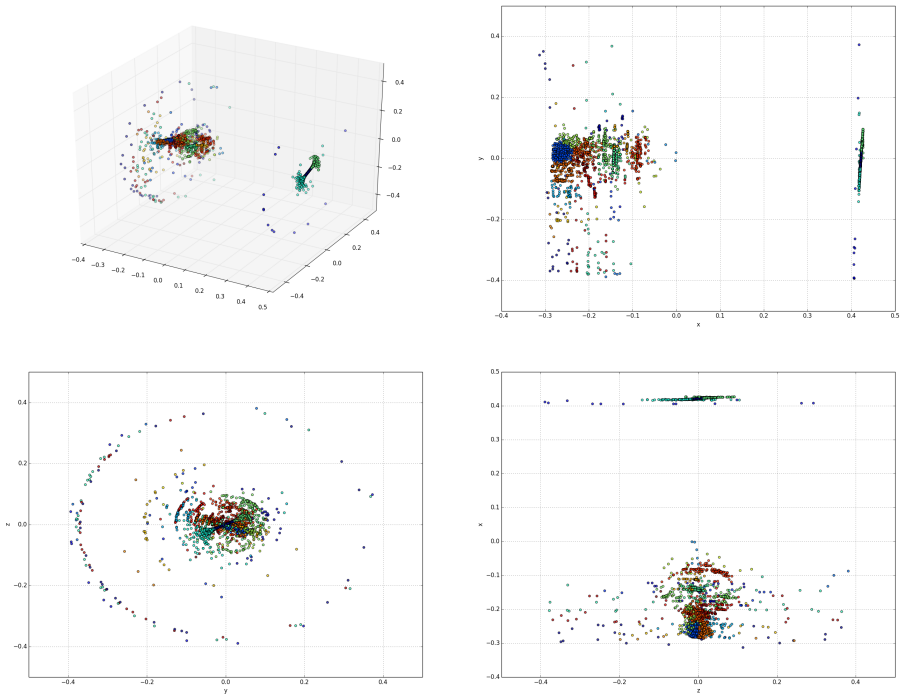

Figure 19. 3D visualization of coarse-grained clusters using projection. One color is associated with each fine-grained cluster.

mented two quality metrics with MATLAB.

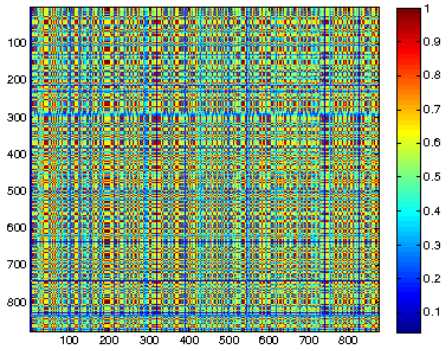

Figure 22. Similarity matrix of a coarse-grained cluster before DBSCAN clustering.

As we saw in section 2.7, a similarity matrix S can be defined as $S = 1 - D$ where D is a distance matrix. Therefore, if $S(r, c)$ is close to 1, URLs r and c are very similar. Visualizing the density of a similarity matrix gives a hint on the efficiency of a clustering algorithm. A good clustering process for a dataset creates areas with similar densities by grouping URLs. In Figure 22 we show the density of a similarity matrix of a given coarse-grained cluster before our fine-grained clustering. We can see that the matrix is not organized, we don't see uniform areas. On the contrary after clustering, Figure 23 shows an organized matrix with uniform areas where we can find two families introduced in section 5.5. Therefore, our fine-grained clustering algorithm gives good results.

We also saw in section 2.7 the Dunn Index.

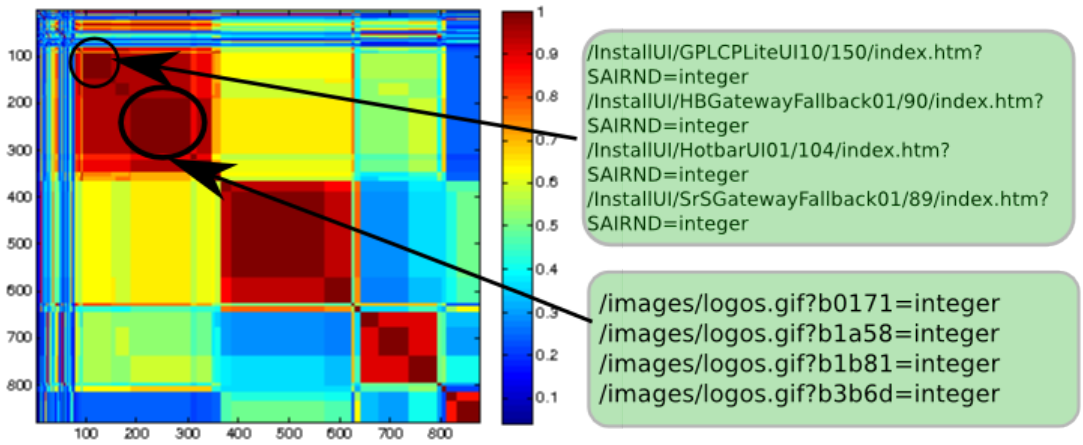

Figure 23. Similarity matrix of a coarse-grained cluster after DBSCAN clustering.

We use it in our work to give a quality value on our coarse-grained clusters. Thanks to this index we saw that the quality of our results differ a lot depending on the size of our dataset.

6.6. Signatures

A signature is generated for each fine-grained cluster. Our signatures generation relies on generalized suffix trees [47]. We had to consider typed values as symbols and not as strings. Because we have unordered key/values pairs due to the sets we consider for our fine-grained clustering, we also used positive lookahead [48] in regular expressions for the signatures generation.

6.7. Web interface

All the tools and algorithms we developed are integrated in a web application, see Figure 24. This single page application is based on Ember.js [49] and Node.js [50]. It aims at providing a centralized interface where all current and past experiences are displayed and can be analyzed. A few clicks are enough to launch a new experiment. Finally, it also avoids the pain of connecting to the server using SSH and learning the command-line interfaces of both Linux and our tools.

7. Future work

While giving excellent results with a randomly-chosen 5,000 extract of URLs and satisfactory results for 100,000 URLs. We still need to improve the performance for the entire dataset. We are currently trying to find appropriate clustering parameters for each extract. We also think our model may be overfitting. Another problem is performance of our fine-grained clustering step.

7.1. Signatures

We would like to deploy our signatures on live traffic to evaluate them. It would be interesting to collect the number of times a signature matched to try to correlate this with the clustering quality associated with this signature.

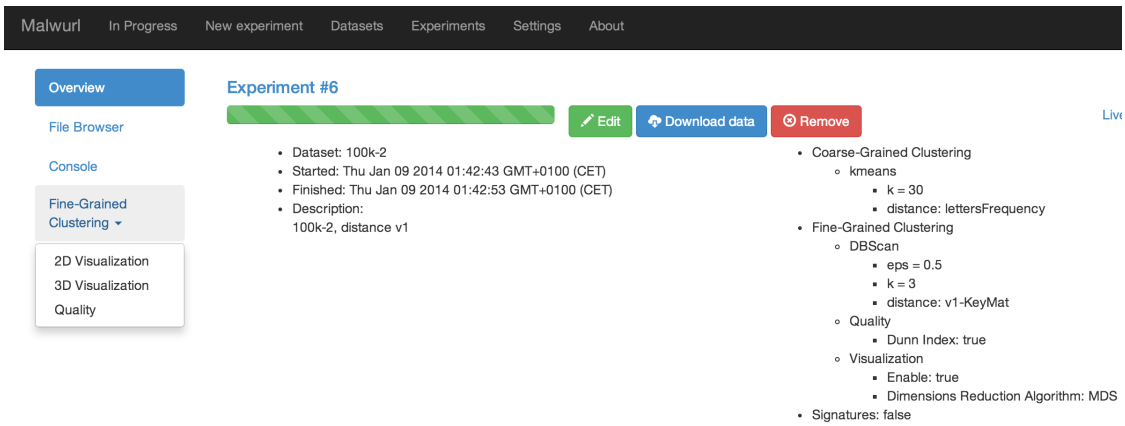

Figure 24. Screenshot of the web interface to launch experiments and analyze results

7.2. Incremental datasets and online mode

Once we have signatures, they can be used to filter out unused clusters. To circumvent our performance problem on DBSCAN, an online mode can be considered. Instead of working with an old dataset, live traffic would be used. This approach would maybe allow pruning unmatched rules and crowded clusters. We would only keep the best reference distance matrix with respect to the size we want it to have.

The workflow for a new URL would then be: try to match a signature, if a signature matches we do not perform clustering. If no signature matches we only compute the missing distances in DBSCAN to perform the fine-grained clustering step.

A similar approach may be studied if incremental datasets are used.

7.3. Coarse-grained clustering

Our coarse-grained clustering gives results that are not optimal. We often have similar looking URLs in different clusters which is very bad for the whole clustering architecture. We see that a significant number of URLs in noise after DBSCAN have similarities with other URLs in other coarse-grained cluster. We believe that if these URLs were in the same coarse-grained cluster, a new fine-grained cluster with these URLs instead of having noise. Because of that, we have detected a lot of noise, sometimes up to three quarters of a coarse-grained cluster. We also observed that there is no relation between the percentage of URLs in noise and the Dunn index.

To resolve these issues, other features should be tried. For example we could move the typing step before the coarse-grained clustering step and have features based on types present in value types of query strings. Because these features seem similar to the distance function of the fine-grained clustering step, we should have closer URLs for this distance in the same coarse-grained cluster.

Another solution may be to try canopy clustering [51] instead of k-means. Canopy clustering is an unsupervised algorithm specifically designed as a step before another clustering algorithm. It is usually performed before K-means or Hierarchical clustering. It speeds up the clustering in the case of large datasets. For these datasets a single step with the main algorithm may be impractical due to its size.

The algorithm is as follows. Start with a list of points and two distance thresholds $T1 >$

$T2$.

1) Select a random point from this list to create a canopy center.
2) Compute its distance to all other points in the list.
3) Insert all the points which fall within the distance threshold of $T1$ into this canopy.
4) Remove from the main list all the points which fall within the threshold of $T2$. These points, already in a canopy, cannot be a canopy center or create new canopies.
5) Repeat from step 1 to 4 until the main list is empty.

Figure 25 provides an illustration where canopy clustering is used in two dimensions with three canopies.

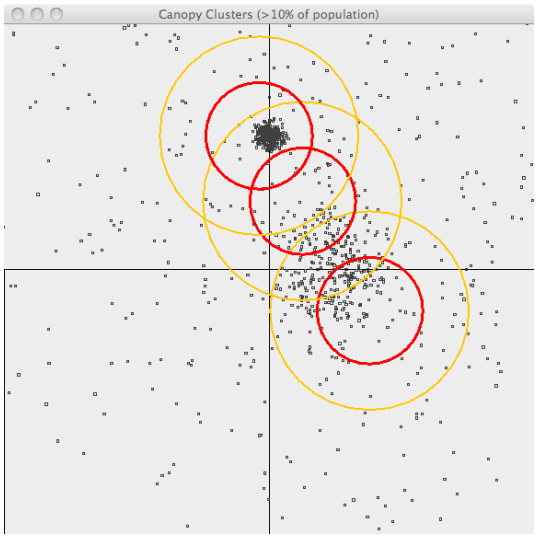

Figure 25. Canopy clustering with three canopies [52]

7.4. Typing

Improvements can be proposed for typing. We detected another type, the volume serial number which is present in 4.5% of URLs in our entire dataset. This is a sequence of hexadecimal figures separated by dashes that matches the regular expression *[0-9a-fA-F]{8}-([0-9a-fA-F]{4}-){3}[0-9a-fA-F]{12}*. This serial is generated when the disk is formated based on the current computer's time. We believe that volume serial numbers are used to uniquely identify a victim even when the victim's IP changes. It has for example been used to generate unique victim identifiers for the "Red October" attack [53]. Currently, volume serial numbers are replaced by no type.

We empirically observed around 30% of URLs having types also in their path or keys. It would be interesting to study typing in these fields as well.

Finer types can be defined using length of values replaced by a type for types such as no type, hexa, integer or float.

8. Conclusion

Detection systems usually differentiate signature-based detection and network behavior anomaly detection. Our work takes advantage of the two methods by clustering URLs depending not on signatures but on heuristics and giving fine-grained clusters that can be used to produce signatures as output. While there is still work to improve our system, we have been able to create clusters of malware families using clustering algorithms on URLs. Our contributions include studying clustering using only paths and query strings of URLs. Our novel clustering architecture use a typing step before clustering which improve performance and results. As far as we know we are the first to use DBSCAN, rather than for example hierarchical clustering, for URLs clustering. We created innovative generic tools for machine learning such as a centralized web platform to monitor and execute machine learning experiments. We developed visualization tools for points navigation that scale with huge datasets.

Acknowledgment

The author would like to thank Joaquin Garcia-Alfaro and Hervé Debar for tutoring this project. This work would have not be possible without the collaboration of Orange Labs and Nizar Kheir. Dingqi Yang greatly helped on the data mining ideas and algorithms. This work is based on previous work from Ashish Gupta and Sergi Martinez-Bea. The author is grateful to Gregory Blanc for his ideas and Emmanuel Chaboud for his precious help with Urho3D.

References

[1] T. Berners-Lee, R. Fielding, and L. Masinter, "Rfc 3986: Uniform resource identifier (uri): Generic syntax," *The Internet Society*, 2005.

[2] "Glossary, Microsoft Malware Protection Center," http://www.microsoft.com/security/portal/mmpc/shared/glossary.aspx, accessed: 2014-01-21.

[3] C. Romesburg, *Cluster analysis for researchers*. Lulu. com, 2004.

[4] "The Microsoft Windows Malicious Software Removal Tool," http://www.microsoft.com/en-gb/security/pc-security/malware-families.aspx, accessed: 2014-01-10.

[5] N. Hachem, Y. Ben Mustapha, G. G. Granadillo, and H. Debar, "Botnets: lifecycle and taxonomy," in *Network and Information Systems Security (SAR-SSI), 2011 Conference on*. IEEE, 2011, pp. 1–8.

[6] "The Resurrection of RedKit," http://www.kahusecurity.com/2014/the-resurrection-of-redkit/, published: 2014-01-07, Accessed: 2014-01-10.

[7] M. Janus, "runforestrun, gootkit and random domain name generation," https://www.securelist.com/en/blog/208193713/RunForestRun_gootkit_and_random_domain_name_generation, published: 2012-08-01, Accessed: 2014-01-10.

[8] Z. Bu, P. Bueno, R. Kashyap, and A. Wosotowsky, "The new era of botnets," *White paper from McAfee*, 2010.

[9] E. J. Kartaltepe, J. A. Morales, S. Xu, and R. Sandhu, "Social network-based botnet command-and-control: emerging threats and countermeasures," in *Applied Cryptography and Network Security*. Springer, 2010, pp. 511–528.

[10] P. Simon, *Too Big to Ignore: The Business Case for Big Data*. Wiley. com, 2013.

[11] M. Learning, "Tom mitchell," *ISBN: 0-07-042807-7, Publisher: McGraw Hill*, 1997.

[12] "Proceedings, international conferences on knowledge discovery and data mining," ACM.

[13] T. S. Guzella and W. M. Caminhas, "A review of machine learning approaches to spam filtering," *Expert Systems with Applications*, vol. 36, no. 7, pp. 10 206–10 222, 2009.

[14] "Dat Mining research papers from Google," http://research.google.com/pubs/DataMining.html, accessed: 2014-01-14.

[15] T. Lane and C. E. Brodley, "An application of machine learning to anomaly detection," in *Proceedings of the 20th National Information Systems Security Conference*, vol. 377. Baltimore, USA, 1997.

[16] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 305–316.

[17] T. O. Ayodele, "Machine learning overview," *New Advances in Machine Learning, Y. Zhang, Ed*, vol. 41, pp. 19–48.

[18] R. Xu, D. Wunsch *et al.*, "Survey of clustering algorithms," *Neural Networks, IEEE Transactions on*, vol. 16, no. 3, pp. 645–678, 2005.

[19] S. Guha, R. Rastogi, and K. Shim, "Cure: an efficient clustering algorithm for large databases," in *ACM SIGMOD Record*, vol. 27, no. 2. ACM, 1998, pp. 73–84.

[20] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *ACM SIGMOD Record*, vol. 25, no. 2. ACM, 1996, pp. 103–114.

[21] A. Rajaraman and J. D. Ullman, "Mining of massive datasets."

[22] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[23] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *KDD*, vol. 96, 1996, pp. 226–231.

[24] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: ordering points to identify the clustering structure," *ACM SIGMOD Record*, vol. 28, no. 2, pp. 49–60, 1999.

[25] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural computation*, vol. 8, no. 7, pp. 1341–1390, 1996.

[26] M. Riedewald, "CS 6220: Data Mining Techniques class - Clustering," http://www.ccs.neu.edu/home/mirek/classes/2012-S-CS6220/Slides/Lecture4-Clustering.pdf, accessed: 2014-01-16.

[27] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 2, pp. 224–227, 1979.

[28] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," 1973.

[29] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

[30] M. Levandowsky and D. Winter, "Distance between sets," *Nature*, vol. 234, no. 5323, pp. 34–35, 1971.

[31] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: is a correction for chance necessary?" in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 1073–1080.

[32] M. Z. Rafique and J. Caballero, "Firma: Malware clustering and network signature generation with mixed network behaviors."

[33] "The malicia project," http://www.malicia-project.com, accessed: 2014-01-17.

[34] A. Le, A. Markopoulou, and M. Faloutsos, "Phishdef: Url names say it all," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 191–195.

[35] G. Jacob, R. Hund, C. Kruegel, and T. Holz, "Jackstraws: Picking command and control connections from bot traffic." in *USENIX Security Symposium*, vol. 2011, 2011.

[36] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces." in *NSDI*, 2010, pp. 391–404.

[37] M. Zalewski, *The Tangled Web: A Guide to Securing Modern Web Applications*. No Starch Press, 2012.

[38] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in *Proceedings of the 2007 ACM workshop on Recurring malcode*, ser. WORM '07. New York, NY, USA: ACM, 2007, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/1314389.1314391

[39] Sophos, "Sus/sality-a," http://www.sophos.com/en-us/threat-center/threat-analyses/suspicious-behavior-and-files/Sus~Sality-A/detailed-analysis.aspx, accessed: 2014-01-21.

[40] D. Albanese, R. Visintainer, S. Merler, S. Riccadonna, G. Jurman, and C. Furlanello, "mlpy: machine learning python," *arXiv preprint arXiv:1202.6548*, 2012.

[41] M. U. Guide, "The mathworks," *Inc., Natick, MA*, vol. 5, 1998.

[42] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[43] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[44] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, pp. 90–95, 2007.

[45] M. Bostock, "d3. js data-driven documents," http://d3js.org/, 2011, accessed: 2014-01-23.

[46] L. rni, "Urho3d," https://github.com/urho3d/Urho3D, accessed: 2014-01-23.

[47] M. Apel, C. Bockermann, and M. Meier, "Measuring similarity of malware behavior," in *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*. IEEE, 2009, pp. 891–898.

[48] T. Stubblebine, *Regular Expression Pocket Reference: Regular Expressions for Perl, Ruby, PHP, Python, C, Java and. NET*. " O'Reilly Media, Inc.", 2007.

[49] "ember.js," http://emberjs.com/, accessed: 2014-01-23.

[50] "node.js," http://nodejs.org/, accessed: 2014-01-23.

[51] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2000, pp. 169–178.

[52] "Canopy Clustering - Apache Mahout - Apache Software Foundation," https://mahout.apache.org/users/clustering/canopy-clustering.html, accessed: 2014-01-22.

[53] Kapersky, "Red October. Detailed Malware Description 3. Second Stage of Attack," https://www.securelist.com/en/analysis/204792264/Red_October_Detailed_Malware_Description_3_Second_Stage_of_Attack, accessed: 2014-01-23.