Malware Triage based on Static Features and Public APT Reports

Giuseppe Laurenza[1], Leonardo Aniello[1], Riccardo Lazzeretti[1], and Roberto Baldoni[1 2]

[1] Research Center of Cyber Intelligence and Information Security (CIS), IT Dept. of Computer and System Sciences "Antonio Ruberti", Sapienza Università di Roma, IT
{laurenza, aniello, baldoni, lazzeretti}@dis.uniroma1.it
[2] CINI Cybersecurity National Laboratory, Italy

Abstract. Understanding the behavior of malware requires a semi-automatic approach including complex software tools and human analysts in the loop. However, the huge number of malicious samples developed daily calls for some *prioritization mechanism* to carefully select the samples that really deserve to be further examined by analysts. This avoids computational resources be overloaded and human analysts saturated. In this paper we introduce a *malware triage* stage where samples are quickly and automatically examined to promptly decide whether they should be immediately dispatched to human analysts or to other specific automatic analysis queues, rather than following the common and slow analysis pipeline. Such triage stage is encapsulated into an architecture for semi-automatic malware analysis presented in a previous work. In this paper we propose an approach for sample prioritization, and its realization within such architecture. Our analysis in the paper focuses on malware developed by Advanced Persistent Threats (APTs). We build our knowledge base, used in the triage, on known APTs obtained from publicly available reports. To make the triage as fast as possible, only static malware features are considered, which can be extracted with negligible delay, without the necessity of executing the malware samples, and we use them to train a random forest classifier. The classifier has been tuned to maximize its precision, so that analysts and other components of the architecture are mostly likely to receive only malware correctly identified as being similar to known APT, and do not waste important resources on false positives. A preliminary analysis shows high precision and accuracy, as desired.

Keywords: Malware Analysis, Advanced Persistent Threats, Static Analysis, Malware Triage

1 Introduction

Cyber threats keep evolving relentlessly in response to the corresponding progress of security defenses, resulting in an impressive number of new malware that are

being discovered daily, in the order of millions [6]. To cope with this enormous volume of samples there is the necessity of a malware knowledge base, to be kept updated over time, and to be used as the main source of information to realize novel and powerful countermeasures to existing and new malware. Some malware are part of sophisticated and target-oriented cyber attacks, which often leverage customized malware to remotely control the victims and use them for accessing valuable information inside an enterprise or institutional network target. According to NIST Glossary of Key Information Security Terms [3], such "*adversary that possesses sophisticated levels of expertise and significant resources which allow it to create opportunities to achieve its objectives by using multiple attack vectors (e.g., cyber, physical and deception)*" is known as Advance Persistent Threats (APT). APTs typically target Critical Infrastructures (CIs) as well as important organizations, stealthily intruding them and persisting there over time spans of months, with the goal of exfiltrating information, undermining or impeding critical aspects of a mission, program, or organization; or positioning itself to carry out these objectives in the future. Therefore, among the large amount of collected malware, those belonging to some APT should be considered as the most dangerous. In addition, the sooner an APT malware is identified, the smaller is the loss it can cause. Within this scenario, it is important to define an efficient workflow for APT malware analysis, aimed first at quickly identifying malware that could belong to APTs and increase their priority in successive analysis (i.e., *APT malware triage*), and then determine whether these suspicious samples are really related to APTs (i.e., *APT malware detection*). This early identification can be embedded in the malware analysis architecture recently presented in [15], which provides semi-automatic malware analysis, and supports a flow of analysis, continuous over time, from the collection of new samples to the feeding and consequent growth of the malware knowledge base. Such an architecture includes totally automated stages, in order to keep up with today's pace of new malware, and also manual stages, where human analysts have to reverse engineer and study in details the samples that have not been completely understood through automatic means. Although the architecture is framed in a scenario tailored for CIs, its employment can be naturally extended to any situation where a malware knowledge base is desired. Within the architecture, a rank is produced for each sample as the intermediate output of some automatic analyses, based on current malware knowledge base, representing to what extent such sample resembles something that is already known and included in that knowledge base. Such rank determines whether the sample should be further analyzed by a human analyst. This can be seen as a specific instance of sample prioritization, where samples follow different paths within a complex analysis workflow depending on priority scores they get assigned during the first stages. To this end, in this paper we introduce a *malware triage* stage, where samples are timely analyzed to understand as soon as possible whether they likely belong to some known APT campaign and should be dispatched, with highest priority, to human analysts or other components of the architecture for further analysis.

[3] http://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7298r2.pdf

Such prioritization is mainly aimed at giving precedence to what we deem to be more important to analyze. In fact understanding whether we are being threaten by an APT is much more urgent than dissecting an unknown variant of some adware. It is to note that the objective of this triage stage is not APT malware detection, which is instead pursued at a later stage by human analysts and specialized architecture components, rather the final goal of the triage is spotting with the highest precision samples that seem to be related to known APTs. The addition of this triage stage does not call for any relevant change in the architecture, rather it can be easily realized using already envisaged components. A prompt priority assignment is thus fundamental when designing a malware triage stage, which translates to employing analysis techniques very efficient in terms of time performance. There is hence the necessity of a triage stage that, in order, (i) has low computational complexity to timely analyze a great number of samples, (ii) has high precision, i.e. does not prioritize non-APT malware to not overload human analysts and/or complex components with urgent but not necessary analyses, and (iii) has high accuracy, i.e. a high number of samples are correctly identified. For the first purpose, we adopt an approach based only on static analysis methods. Although it is known that static features are not as effective as dynamic features for malware analysis [7], we choose anyway to only rely on static features because for the triage stage we deem prioritization speed more important than accuracy, and we leave more accurate analyses to successive stages along the analysis pipeline. We leverage on publicly available reports on APTs, which include MD5s of related malware. We collect these malware from public sources, such as VirusTotal [4], then the content of sample binaries are examined to extract *static features* to produce required feature vectors. No sample execution is hence needed, and no expensive virtualized or emulated environment needs to be setup and activated. These static features are then used in machine learning tools, where APT name represents the class label, to identify whether the sample is similar to some known APT malware. A part of the samples is used in the training phase and the other samples are used in the validation of the classifier, together with non-APT samples. For classification we train a random forest classifier, because it guarantees high efficiency and low complexity. The classifier has been tuned to provide the precision and accuracy constraints. Results are encouraging, as they suggest this approach can be easily realized within the architecture proposed in [15], and effective in identifying samples similar to malware realized by known APTs with a precision of 100% and accuracy up to 96%. The contributions of this paper are (i) the definition of a novel policy for malware triage, based on the similarity to malware developed by known APTs, (ii) the design of the malware triage stage within the architecture proposed in [15], (iii) a prototype implementation of such architecture, and (iv) an experimental evaluation regarding the performance of the proposed malware triage, using a number of public reports about APTs as dataset. The rest of the paper is structured as follows. § 2 reports on the state of the art in the field of malware analysis architectures and malware triage. § 3 describes

[4] https://www.virustotal.com/

the reference architecture. The malware triage approach is detailed in § 4, while prototype implementation and experimental evaluations about triage accuracy are reported in § 5. Conclusions are drawn and future works presented in § 6.

2 Related Work

The analysis of *Advanced Persistent Threats* is an important topic of research within the cyber-security area: many researchers focus on the *avoidance* and/or *detection* of this type of attacks. In [24] and [16] methodologies are shown to detect the presence of these advanced intruders through *anomaly detection* systems that use network traffic features. In [23] a framework is proposed to leverage dynamic analysis to find evidences of APT presence. Other researchers concentrate their effort in the hardening of organizations [5, 11, 26, 27]. They propose procedures to raise security levels through the implementation of various precautions based on the analysis of previous attacks.

On the contrary, our work is not oriented to develop a monitor to detect suspicious activity or to improve the robustness of organization's defenses. Rather, we aim to develop a triage approach to support expert analysts in their work, trough a prioritization of *interesting* threats. Several works in literature try to face the problem of malware triage by using the same basic principle: finding similarities among malware to identify variants of samples already analyzed in the past, so that they are not analyzed in details and thus do not waste resources such as human analysts.

One famous work in this field is Bitshred [10]: it is a framework for fast large-scale malware triage using features hashing. The main idea is to reduce the size of the feature vector to speed up the machine learning analysis.

VILO [14] is another tool for malware classification: it is based on nearest neighbor algorithm with weighted opcode mnemonic permutation features and it aims to be a *rapid learner of malware families*. It is well suited for malware analysis because it makes minimal assumptions about class structures and thus it can adapt itself to the continuous changes of this world.

An interesting triage approach is the one use by SigMal [13]: using signal processing to measure the similarity among malware. This approach permits to define more noise-resistant *signatures* to quickly classify malware.

All these works propose a triage approach based on the idea of performing deep analysis only on malware that are not similar to known classes (like new malware families), instead our approach prioritizes malware that are related to already known malicious samples in order to find novel samples possibly developed by APTs. We base our system on static features extracted by static analysis. While it is quite unreliable for malware detection [19], in our application static analysis represent a lightweight and efficient tool for classification of detected malware among APTs campaigns. Structural properties [28] would add important knowledge to the classifier, however we discarded them because of their high complexity

3 Architecture

Malware triage is a pre-processing phase, aiming at proritize APT malware analysis in the architecture presented in [15]. In this section we summarize a description of the given malware analysis framework, showing how sample analysis flow is arranged through a staged view of the architecture. For a detailed description of the building blocks composing the architecture, and interactions of the framework within multiple organizations and Critical Infrastructures (CIs), we remind to the original paper [15].

Sample analysis is organized in a series of stages, from sample retrieval to information sharing, shown in Fig. 1.

In the *Loading Stage*, malware samples are gathered from known datasets, malware crawlers, honeypots and other distinct sources. Also APT reports are collected and related malware are retrieved. In the *Input Stage*, samples collected are stored together with a set of metadata characterizing the samples themselves, including the APT they belong to, if any, and their source.

Samples collected are then analyzed in the *Analysis Stage*. *Analysis Tools* are used to examine sample content and analysis in order to extract significative features representing the samples. Machine Learning *Classifiers* are in charge of assigning samples to predefined learned classes on the base of features values. *Clustering tools* group samples according to their similarity, with the goal of isolating specific groups of malware and link unknown samples to them. *Correlators* try to match samples with information about cyber threats retrieved from external sources.

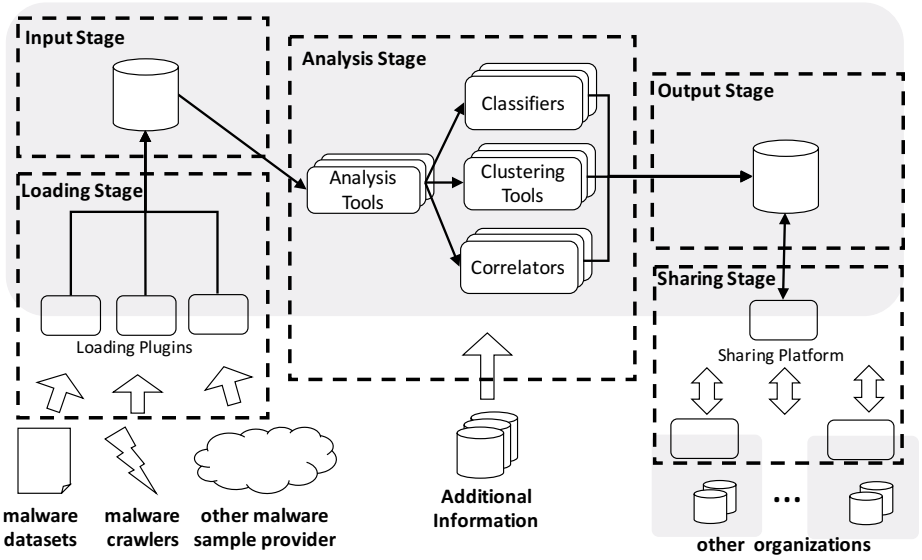


Fig. 1. Staged view of the architecture of the malware analysis framework.

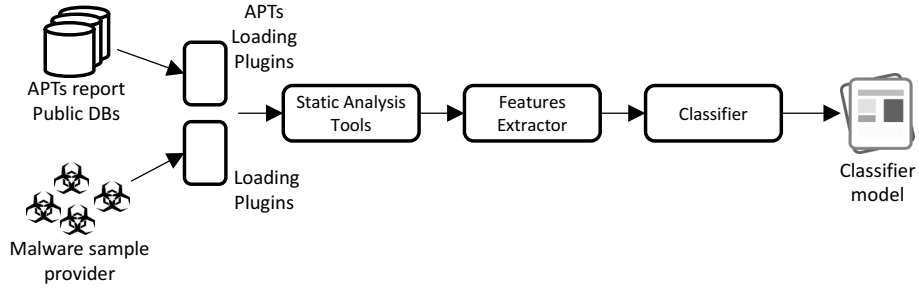
Results obtained by these tools are pushed to the *Output Stage* and eventually made available to other organizations in the *Sharing Stage*.

We underline that both input and output stage share the same storage space, hence the output of the analysis also enrich the information associated to samples. Samples can pass through the analysis stage several times, in the case new tools are added to the architecture, some tool is updated, or samples deserve special attention by analysts. As shown in § 4, the triage approach aims to promptly analyze malware samples to associate them a rank indicating whether samples can be related to some known APT and hence they deserve further investigation.

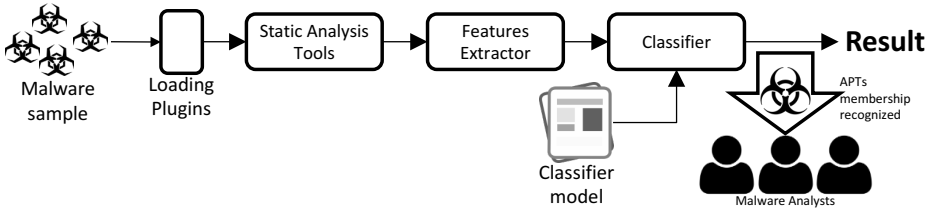
4 Triage Approach

We propose an approach for malware triage based on the identification of samples similar to malware known to be developed by APTs. From now on, we say that these samples are *related to known APTs*. The basic idea is to generate a dataset by collecting public APT reports (such as [17], [20] and [25]) and retrieving the binaries of the malware referenced in these reports. Each malware is assigned to an APT based on what is written in the reports. Static features are extracted from these binaries and used to train a classifier to be used for the triage.

Fig. 2 highlights the components of the architecture (see § 3 and [15]) that are involved in the malware triage process. The flow starts with the *APT loading plugin*, which continuously crawls different public sources in order to obtain re-



(a) Training Phase



(b) Analysis Phase

Fig. 2. Data Classification Flow.

ports about APTs and feeds the system with the information contained in them. For the training phase, the *Features Extraction* component periodically analyses all new static analysis reports of malware related to known APTs to produce the feature vectors required to train the *Classifier*. For the analysis phase, novel samples have to first pass through the malware triage stage. This includes a static analysis phase (performed in the *Sample Analysis Section* of the architecture) aimed at producing the feature vectors (done by *Features Extraction* component) to feed to the trained *Classifier*. If a sample is classified as related to known APT, then it is directed to an alternative path within the analysis chain, e.g., it is submitted to some human analyst for manual examination. We first present the phase of malware retrieval based on public APT reports (§ 4.1), then describe what static features are considered (§ 4.2), and finally detail how the classifier is trained and then used for the actual triage (§ 4.3).

4.1 APT Malware Loading

This component crawls external, publicly-available sources to collect reports related to malicious campaigns, activities and software that have been associated to APTs. These reports are produced by security firms and contain different *Indicator Of Compromises* [IOCs] related to specific APTs, including domains, IPs and MD5s of malware. The loading plugin parses them and add these information into the Knowledge Base. When a malware is added through its MD5, the architecture searches for the corresponding binary file in order to store and analyze it. Unfortunately, many of these malware are not available on public sources, so it is not possible to collect all of them.

4.2 Feature Extraction

To obtain a prompt triage process, we base the classification on static features only. Indeed, they take shorter time to be extracted compared to dynamic features, which instead require sample execution in some controlled environment (e.g., a sandbox). Table 1 reports all the classes of features that are extracted. In this work, considered samples are PE files. There are seven feature classes.

Table 1. Features Classes

Class	Count
Optional Header	30
MS-DOS Header	17
File Header	7
Obfuscated String Statistics	3
Imports	158
Function lengths	50
Directories	65

Optional Header features. These features are extracted from the optional header of the PE. It contains information about the logical layout of the PE file, such as the address of the entry point, the alignment of sections and the sizes of part of the file in memory.
MS-DOS Header features. Features related to the execution of the file, including the number of bytes in the last page of the file, the number of pages or the starting address of the *Relocation Table*.
File Header features. These features are extracted from the file header, which is a structure containing information about the physical layout and the properties of the PE file in general, like number of sections, timestamp and the CPU platform which the PE is intended for.
Obfuscated String features. FireEye Labs Obfuscated String Solver (FLOSS) is a tool to automatically de-obfuscate strings from files through static analysis. The result of this tools is used to compute some statistics, like how many entry-points or relocations are present in the file, that compose the features of this class.
Imports features. Functions can be imported from other executables or from DLLs. We are interested in the import of a specific set of known DLLs and APIs, and use their occurrences as feature. We also use three counters representing the total number of imported APIs, the total number of imported DLLs, and the total number of exported functions.
Function Lengths features. FLOSS also provides measurements of function lengths. This class contains different counters to store that information. Due to the huge number of different functions, we use *bucketing* to reduce the number of possible features.
Directories features. PE header includes an array containing all the *DATA DIRECTORY* structures, so, similarly to what we do for imports, we check the occurrence in the file of some particular directory names. We use their size as features, similarly to function lengths features.

4.3 Classification

We firstly tried to setup the classifier using a class for each known APT, representing the malware collected on the base of APT reports, and an additional class to represent all the samples that have not been created by any known APT. If a sample were assigned to the latter class, then it would be considered not related to any known APT. Otherwise, the classifier would establish the most likely APT which developed malware similar to that sample. The problem of this approach lies in training the classifier on such additional class. Indeed, the overwhelming majority of samples belongs to that class, including most of malicious samples, as a really tiny percentage of malware have been actually created by some APT. This translates to an excessive heterogeneity among samples of that class, and an extreme imbalance among classes in the training set, which makes this approach infeasible. Hence, we give up such additional class and only use classes representing known APTs for training. Given $C = \{c_i\}$ be the set of classes, with $N = |C|$ being the number of classes, equal to the number of

actually used APTs, we train the classifier on N classes (one for each APT in the dataset). In the analysis flow we label the input samples in $N + 1$ classes, where the additional class is composed by all the outlier samples, that can be non-APT malware samples as well as samples of unknown APTs. Being the classifier trained on N samples, the idea is that all the samples too distant from all the centroids belong to such $(N + 1)$-th class. We use random forest [4] as learning method for the classification, as it turned out to be really effective in several works related to malware analysis [9, 12, 22], also because of its ensemble approach. Moreover random forest permits to classify samples by using different types of features (numbers, binary and non-numeric labels). A random forest operates by generating a set of decision trees at training time, and using them as classifier by picking the most frequent chosen class among them. Let $T = \{t_j\}$ be the set of decision trees of the random forest. $N_T = |T|$ is the number of trees. In order to determine whether a sample is related to a known APTs or classified as non-APT, we rely on the confidence score of the classification: if this score exceeds a threshold, then the sample is considered as related to the relative APT, otherwise it is not. As one of the main goals is minimizing how many irrelevant samples are delivered to human analysts or keep scarce resources busy, we are interested in using only those APTs where the classifier can perform with higher precision. We train the classifier with malware of all known APTs, and use a K-fold cross validation to obtain accuracy results. We then remove those APTs where both precision and recall are below a given threshold, and use only remaining APTs to train the actual model. We also have to remove the APTs where available malware are less than K. In the experimental evaluation, we show triage accuracy results for two distinct thresholds.

Classification Confidence Computation The class assigned by a decision tree depends on the leaf node where the decision path terminates. Each leaf node corresponds to a subset of its training samples, which can belong to distinct classes, and the output class of the leaf node is the most frequent one among them. For a decision tree t_j, let $l_j = \{l_{j,k}\}$ be the set of its leaf nodes, with $N_j = |l_j|$. Let $N_{j,k}$ be the number of training samples of leaf node $l_{j,k}$. We define $class_{i,j,k}$ as the number of training samples of $l_{j,k}$ that belong to class c_i.

Intuitively, the diversity of classes among the training samples of a leaf node reflects how much the decision tree is confident about its output, when this output is determined by that leaf node. Thus, as confidence score for the single decision tree, we use the percentage of training samples that belong to the same class output by the leaf node. We then assign a confidence score to the classification of the whole random forest by averaging the confidence scores of all its decision trees. In a similar way, we can assign to a classified sample a confidence score for each class, to represent to what extent that sample relates to each class. We assign a confidence vector $confidence_{j,k}$ to each leaf node $l_{j,k}$, where the i-th entry represents the confidence for class c_i, defined as follows

$$confidence_{j,k}[i] = \frac{class_{i,j,k}}{\sum_{m=1}^{N} class_{m,j,k}} \quad i = 1 \dots N \tag{1}$$

For each sample to analyze, we setup the classifier to output a confidence value for each class, which represents the likelihood that the sample resembles malware created by the APT corresponding to that class. Given a sample s to classify with the random forest, we introduce the function $decision_j(s)$ which determines the leaf node of t_j where the decision path for s ends. Let $l_{j,k}$ be such leaf node, then $decision_j(s) = k$. We define the confidence vector $confidence(s)$ assigned to a sample s classified with the random forest as follows, where the i-th entry represents the confidence for class c_i

$$confidence(s)[i] = \frac{1}{N_T} \sum_{j=1}^{N_T} confidence_{j,decision_j(s)}[i] \quad i = 1 \dots N \tag{2}$$

Confidence Threshold Computation Malware developed by a same APT can be very different among each other. For example, they may relate to different phases of an attack (e.g., the payload for intruding target system, and the remote access trojan to enable external control), or they may have been used for attacks to distinct victims. Furthermore, we empirically observe that collected malware are distributed really unevenly among known APTs. This implies that confidence scores obtained for distinct classes cannot be fairly compared. Thus, rather than using a unique confidence threshold to discriminate whether a sample can be considered as related to a known APT, we compute a different threshold for each APT.

We first compute the confidence vector for each sample of the training set TS by using *leave-one-out* cross validation: for each training sample $s \in TS$, we use all the other training samples to train the random forest and then classify s to identify the leaf nodes to use to compute $confidence(s)$. Let $class(s)$ be a function that returns i if the class of training sample s is c_i. Let $TS_i = \{s \in TS : class(s) = i\}$ be the subset of the training set containing all and only the samples of class c_i. We then calculate the threshold vector as follows

$$threshold[i] = \frac{\sum_{s \in TS_i} confidence(s)[i]}{|TS_i|} - \Delta \quad i = 1 \dots N \tag{3}$$

For each class, rather than directly using the average of its confidence scores as threshold, we decrease it by a *tolerance band* Δ in order to avoid having too many false negatives. During the actual triage in production, a test sample s is classified by the random forest and assigned a confidence vector $confidence(s)$, which is compared to the threshold vector to check whether the following condition holds

$$\exists i \quad confidence(s)[i] > threshold[i] \quad i = 1 \dots N \tag{4}$$

In positive case, s is considered related to known APTs and dispatched accordingly, together with its confidence vector which may guide the subsequent analyses, as it suggests to what extent s resembles malware developed by each of the APTs used for training the random forest.

5 Experimental Evaluation

In this section we present details about the prototype and the preliminary results achieved. As explained in previous sections, we design our system in order to require the minimum amount of time to produce an evaluation for the triage: we use static analysis that is the faster type of analysis, due to the fact that it does not require sample execution. Moreover, we use a classifier based on Random Forest, which requires a shorter period of time for the classification with respect to other algorithms.

5.1 Prototype Implementation

We implement a prototype of the architecture presented in [15], by developing custom tools and adapting and extending open-source products. The description of prototype implementation is organized according to the same layered view presented in § 3.

Visual Analytics Layer. For this layer we extend *CRITs* [18], an open-source malware and threat repository developed by MITRE. It offers two important characteristics: a well organized knowledge base and an extendible service platform, both accessible trough a web interface. To integrate CRITs into our architecture, we have to develop a set of services to enable the communication with the other components, and to modify some interfaces to show additional information.

Analysis Layer. For the analysis layer we adapt different open-source analysis tools, both for static and dynamic analysis. For example, we extend PEFrame [3] with functions from Flare-Floss [8] in order to have more information at disposal. The modified version of PEFrame is also the source for the Features Extractor Component described in § 4 (developed in R language [21]), which in turn feeds the random forest classifier (implemented in R as well). The details of feature extractor and classifier are reported in § 4.

Storage Layer. For the storage layer we use a MongoDB [2] cluster.

Loading Plugins. We also develop various plugins to gather required data from public sources. We adapt some open-source crawlers and develop some other by ourselves. The APT loader plugin is based on the IOC Parser [1], modified to collect APT reports from some public sources, extract data if interest and insert them into the storage.

5.2 Triage Evaluation

To validate the effectiveness of our approach we perform some preliminary experiments, using datasets prepared by including samples retrieved on the base

of the MD5s found in the APT reports crawled by loading plugins. Unfortunately, many referenced malware are not available in public sources, thus some APTs have not enough samples to be properly represented. Furthermore, distinct APTs have very different number of samples, which leads to a highly unbalanced datasets, thus we choose to include only the most distinguishing APTs, basing our decision on the average precision and recall that the default random forest classifier would obtain.

Dataset We collect 403 different reports about APTs, containing overall references to 9453 MD5. From public sources we manage to collect only 5377 binaries. The resulting dataset contains 5685 sample belonging to 199 different APTs. We discard all the APTs with less than 20 samples to avoid classes not sufficiently represented, which leads to a dataset with 4783 samples and 47 APTs. We also collect from public sources 2000 additional malware that are not known to be developed by any APT.

Training Phase We build two datasets by using distinct thresholds for precision and recall (see § 4.3): dataset D1 with threshold 0,95 and dataset D2 with threshold 0,90. Table 2 shows details about these datasets. For each dataset, we trained three diverse classifiers by using distinct confidence thresholds Δ: 5%, 10% and 15%.

	APTs	Samples	Mean Class Size
D1	7	1308	187
D2	15	2521	168

Table 2. Dataset Composition

Test Phase For the test we choose to use a K-fold cross validation with k equals to 10, a common value in literature for this kind of tests. For each execution, we generate the model with $k-1$ folds and test it with both the remaining fold and all the collected malware not developed by APTs (2000 samples). We consider the triage as a binary classification, and measure its quality by using *Accuracy*, *Precision*, *Recall* and *F1*. If a sample is classified as related to known APTs we say it is *positive*, otherwise *negative*.As explained in § 4.3, the most important measure for the triage is the Precision (i.e., minimize false positives), due to the fact that human analysts are a limited resources and we have to reduce as much as possible their workload by striving to deliver them only samples that are highly confidently related to known APTs. Tables 3 and 4 show the results of the classification test for both datasets, which highlight that obtained false positives are indeed very low: we are able to reduce false positives to zero for D1 and to less than 70 for D2. It is to note that these numbers are computed over 20000 actual tests, in fact each of the 2000 negative samples is tested for each of the 10 folds.

Tables 5 to 10 display the confusion matrices obtained by considering the classifier as an $N+1$ classifier, for both the datasets and the same three Δ

Table 3. Binary Confusion Matrix D1 [APTs / non-APTs]

	Δ	5%		10%		15%	
	Triage	Pos	Neg	Pos	Neg	Pos	Neg
Ground	Pos	1209	99	1232	76	1250	58
Truth	Neg	0	20000	0	20000	0	20000

Table 4. Binary Confusion Matrix D2 [APTs / non-APTs]

	Δ	5%		10%		15%	
	Triage	Pos	Neg	Pos	Neg	Pos	Neg
Ground	Pos	2125	396	2197	324	2251	270
Truth	Neg	2	19998	18	19982	53	19947

considered before. Results are coherent with the previous ones: with D1 we always achieve zero false positives, while with D2 we incorrectly label less than 0,002% as related to known APTs.

Table 5. Confusion Matrix D1 [$\Delta = 5\%$]

	A	B	C	D	E	F	G	NA
A	23	0	0	0	0	0	0	2
B	0	36	0	0	0	0	0	8
C	0	0	275	0	0	0	0	28
D	0	0	0	414	0	0	0	22
E	0	0	0	0	16	0	0	6
F	0	0	0	0	0	313	0	28
G	0	0	0	0	0	0	132	5
NA	0	0	0	0	0	0	0	20000

Table 11 reports quality metrics for all the tests, and shows that our approach is really promising as it scores high levels of accuracy and precisions.

6 Conclusion and Future Works

Among the huge amount of malware produced daily, those developed by Advanced Persistent Threats (APTs) are highly relevant, as they are part of massive and dangerous campaigns that can exfiltrate information and undermine or impede critical operations of a target. This paper introduces an automatic malware triage process to drastically reduce the number of malware to be examined by human analysts. The triage process is based on a classifier which evaluates to what extent an incoming malicious sample could have been developed by a

Table 6. Confusion Matrix D1 [$\Delta = 10\%$]

	A	B	C	D	E	F	G	NA
A	23	0	0	0	0	0	0	2
B	0	39	0	0	0	0	0	5
C	0	0	277	0	0	0	0	26
D	0	0	0	419	0	0	0	17
E	0	0	0	0	17	0	0	5
F	0	0	0	0	0	325	0	16
G	0	0	0	0	0	0	132	5
NA	0	0	0	0	0	0	0	20000

Table 7. Confusion Matrix D1 [$\Delta = 15\%$]

	A	B	C	D	E	F	G	NA
A	24	0	0	0	0	0	0	1
B	0	40	0	0	0	0	0	4
C	0	0	283	0	0	0	0	20
D	0	0	0	424	0	0	0	12
E	0	0	0	0	19	0	0	3
F	0	0	0	0	0	328	0	13
G	0	0	0	0	0	0	132	5
NA	0	0	0	0	0	0	0	20000

Table 8. Confusion Matrix D2 [$\Delta = 5\%$]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	NA
A	74	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27
B	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	5
C	0	0	33	0	0	0	0	0	0	0	0	0	0	0	0	11
D	0	0	0	66	0	0	0	0	0	0	0	0	0	0	0	38
E	0	0	0	0	122	0	0	0	0	0	0	0	0	0	0	28
F	0	0	0	0	0	476	0	0	0	0	0	0	0	0	0	79
G	0	0	0	0	0	0	39	0	0	0	0	0	0	0	0	26
H	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	9
I	0	0	0	0	0	0	0	0	272	0	0	0	0	0	0	31
J	0	0	0	0	0	1	0	0	0	407	0	0	0	0	0	28
K	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	13
L	0	0	0	0	0	0	0	0	0	0	0	299	0	0	0	42
M	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	17
N	0	0	0	0	0	0	0	0	0	0	0	0	0	27	0	8
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	138	34
NA	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	19998

Table 9. Confusion Matrix D2 [$\Delta = 10\%$]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	NA
A	76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
B	0	22	0	0	0	0	0	0	0	0	0	0	0	0	0	3
C	0	0	34	0	0	0	0	0	0	0	0	0	0	0	0	10
D	0	0	0	69	0	0	0	0	0	0	0	0	0	0	0	35
E	0	0	0	0	125	0	0	0	0	0	0	0	0	0	0	25
F	0	0	0	0	0	494	0	0	0	0	0	0	0	0	0	61
G	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0	18
H	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	9
I	0	0	0	0	0	0	0	0	275	0	0	0	0	0	0	28
J	0	0	0	0	0	1	0	0	0	414	0	0	0	0	0	21
K	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	10
L	0	0	0	0	0	0	0	0	0	0	0	309	0	0	0	32
M	0	0	0	0	0	0	0	0	0	0	0	0	128	0	0	9
N	0	0	0	0	0	0	0	0	0	0	0	0	0	29	0	6
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	140	32
NA	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	19982

Table 10. Confusion Matrix D2 [$\Delta = 15\%$]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	NA
A	78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23
B	0	22	0	0	0	0	0	0	0	0	0	0	0	0	0	3
C	0	0	38	0	0	0	0	0	0	0	0	0	0	0	0	6
D	0	0	0	73	0	0	0	0	0	0	0	0	0	0	0	31
E	0	0	0	0	128	0	0	0	0	0	0	0	0	0	0	22
F	0	0	0	0	0	501	0	0	0	0	0	0	0	0	0	54
G	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	15
H	0	0	0	0	0	0	0	23	0	0	0	0	0	0	0	8
I	0	0	0	0	0	0	0	0	277	0	0	0	0	0	0	26
J	0	0	0	0	0	1	0	0	0	419	0	0	0	0	0	16
K	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	7
L	0	0	0	0	0	0	0	0	0	0	0	324	0	0	0	17
M	0	0	0	0	0	0	0	0	0	0	0	0	131	0	0	6
N	0	0	0	0	0	0	0	0	0	0	0	0	0	29	0	6
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	142	30
NA	0	0	0	0	0	53	0	0	0	0	0	0	0	0	0	19947

Table 11. Quality measures

Dataset		D1				D2			
Measures		Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
Δ	5%	0.995	1.000	0.886	0.938	0.982	1.000	0.764	0.859
	10%	0.996	1.000	0.910	0.952	0.985	0.998	0.807	0.888
	15%	0.997	1.000	0.938	0.968	0.986	0.994	0.850	0.913

known APT, hence relieving analysts from the burden of analyzing these malware. The classifier is trained with static features obtained by static analysis of available malware known to be developed by APTs, as attested by public reports. Although static features alone are not sufficient to completely exclude relations with APTs, they allow to perform a quick triage and recognize malware that deserve higher attention, with minimal risk of wasting analysts time. In fact the experimental evaluation has shown encouraging results: malware realized by known APTs have been identified with a precision of 100% and an accuracy up to 96%.

At the time of this writing, we are testing our approach in the *real world*, i.e., we are analyzing large malware datasets. As future work, we want to study more effective functions for the evaluation of the threshold (see § 4), in order to improve the overall accuracy of the system. Moreover, we plan to include an additional prioritization step for the samples that result nearer to the chosen threshold: as this situation indicates a higher degree of uncertainty about these sample, they can be sent to a second classifier trained with dynamic features of malware known to be developed by APTs.

Acknowledgments

This present work has been partially supported by a grant of the Italian Presidency of Ministry Council, and by CINI Cybersecurity National Laboratory within the project FilieraSicura: Securing the Supply Chain of Domestic Critical Infrastructures from Cyber Attacks (www.filierasicura.it) funded by CISCO Systems Inc. and Leonardo SpA.

References

1. IOC parser. https://github.com/armbues/ioc_parser/, accessed: 2017-03-17
2. MongoDB. https://www.mongodb.com/, accessed: 2017-03-17
3. PEFrame. https://github.com/guelfoweb/peframe/, accessed: 2017-03-17
4. Breiman, L.: Random forests. Machine learning 45(1), 5–32 (2001)
5. Chen, P., Desmet, L., Huygens, C.: A study on advanced persistent threats. In: IFIP International Conference on Communications and Multimedia Security. pp. 63–72. Springer (2014)
6. CNN: Nearly 1 million new malware threats released every day (2014), http://money.cnn.com/2015/04/14/technology/security/cyber- attack-hacks-security/
7. Damodaran, A., Di Troia, F., Visaggio, C.A., Austin, T.H., Stamp, M.: A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques pp. 1–12 (2015)
8. Fireeye: FireEye Labs Obfuscated String Solver. https://github.com/fireeye/flare-floss/, accessed: 2017-03-17
9. Islam, R., Tian, R., Batten, L.M., Versteeg, S.: Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications 36(2), 646–656 (2013)

10. Jang, J., Brumley, D., Venkataraman, S.: Bitshred: Fast, scalable malware triage. Cylab, Carnegie Mellon University, Pittsburgh, PA, Technical Report CMU-Cylab-10 22 (2010)
11. Jeun, I., Lee, Y., Won, D.: A practical study on advanced persistent threats. In: Computer applications for security, control and system engineering, pp. 144–152. Springer (2012)
12. Khodamoradi, P., Fazlali, M., Mardukhi, F., Nosrati, M.: Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms. In: Computer Architecture and Digital Systems (CADS), 2015 18th CSI International Symposium on. pp. 1–6. IEEE (2015)
13. Kirat, D., Nataraj, L., Vigna, G., Manjunath, B.: Sigmal: A static signal processing based malware triage. In: Proceedings of the 29th Annual Computer Security Applications Conference. pp. 89–98. ACM (2013)
14. Lakhotia, A., Walenstein, A., Miles, C., Singh, A.: Vilo: a rapid learning nearest-neighbor classifier for malware triage. Journal of Computer Virology and Hacking Techniques 9(3), 109–123 (2013)
15. Laurenza, G., Ucci, D., Aniello, L., Baldoni, R.: An architecture for semi-automatic collaborative malware analysis for CIs. In: Dependable Systems and Networks Workshop, 2016 46th Annual IEEE/IFIP International Conference on. pp. 137–142. IEEE (2016)
16. Marchetti, M., Pierazzi, F., Colajanni, M., Guido, A.: Analysis of high volumes of network traffic for advanced persistent threat detection. Computer Networks (2016)
17. Micro, T.: Ixeshe: An apt campaign. Trend Micro Incorporated Research Paper (2012)
18. MITRE: CRITS:Collaborative Research Into Threats. https://crits.github.io/, accessed: 2017-03-17
19. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual. pp. 421–430. IEEE (2007)
20. O'Gorman, G., McDonald, G.: The elderwood project. Symantec Whitepaper (2012)
21. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing
22. Santos, I., Devesa, J., Brezo, F., Nieves, J., Bringas, P.G.: Opem: A static-dynamic approach for machine-learning-based malware detection. In: International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions. pp. 271–280. Springer (2013)
23. Su, Y., Lib, M., Tang, C., Shen, R.: A framework of apt detection based on dynamic analysis (2016)
24. Tankard, C.: Advanced persistent threats and how to monitor and deter them. Network security 2011(8), 16–19 (2011)
25. Villeneuve, N., Bennett, J.T., Moran, N., Haq, T., Scott, M., Geers, K.: Operation" Ke3chang: Targeted Attacks Against Ministries of Foreign Affairs (2013)
26. Virvilis, N., Gritzalis, D., Apostolopoulos, T.: Trusted computing vs. advanced persistent threats: Can a defender win this game? In: Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC). pp. 396–403. IEEE (2013)

27. Vukalović, J., Delija, D.: Advanced persistent threats-detection and defense. In: Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on. pp. 1324–1330. IEEE (2015)
28. Wicherski, G.: pehash: A novel approach to fast malware clustering. LEET 9, 8 (2009)