Malware Triage for Early Identification of Advanced Persistent Threat Activities

GIUSEPPE LAURENZA, RICCARDO LAZZERETTI, and LUCA MAZZOTTI,
Sapienza University of Rome, Italy

In the past decade, a new class of cyber-threats, known as "Advanced Persistent Threat" (APT), has emerged and has been used by different organizations to perform dangerous and effective attacks against financial and politic entities, critical infrastructures, and so on. To identify APT related malware early, a semi-automatic approach for malware samples analysis is needed. Recently, a *malware triage* step for a semi-automatic malware analysis architecture has been introduced. This step identifies incoming APT samples early, among all the malware delivered per day in the cyber-space, to immediately dispatch them to deeper analysis. In the article, the authors have built the knowledge base on known APTs obtained from publicly available reports. For efficiency reasons, they rely on static malware features, extracted with negligible delay, and use machine learning techniques for the identification. Unfortunately, the proposed solution has the disadvantage of requiring a long training time and needs to be completely retrained each time new APT samples or even a new APT class are discovered. In this article, we move from multi-class classification to a group of one-class classifiers, which significantly decreases runtime and allows higher modularity, while still guaranteeing precision and accuracy over 90%.

CCS Concepts: • **Social and professional topics** → *Malware/spyware crime*;

Additional Key Words and Phrases: Malware analysis, advanced persistent threats, isolation forest

ACM Reference format:
Giuseppe Laurenza, Riccardo Lazzeretti, and Luca Mazzotti. 2020. Malware Triage for Early Identification of Advanced Persistent Threat Activities. *Digit. Threat.: Res. Pract.* 1, 3, Article 16 (August 2020), 17 pages.
https://doi.org/10.1145/3386581

1 INTRODUCTION

Since the origin of the Internet, cyber-attacks have evolved in parallel with computer development, changing ways and means of execution. From the first viruses and worms to the modern botnets and rootkits, from the first singular and non-organized attacks to the advanced and well-crafted persistent ones, all these kinds of vectors mutate their behavior by updating their "source" in line with the new software and hardware technologies. During the past two decades, the number of delivered malware has exponentially increased: According to a

This work is partially supported by La Sapienza University of Rome Bando Ricerca 2017 within the project n. RM11715C7878B045 and by the Consorzio Interuniversitario Nazionale Informatica (CINI) National Laboratory of Cyber Security.

Authors' addresses: G. Laurenza, R. Lazzeretti, and L. Mazzotti, Research Center of Cyber Intelligence and Information Security (CIS), Dept. of Computer, Control, and Management Engineering Antonio Ruberti, Sapienza University of Rome, Via Ariosto 25, Rome, Italy; email: {laurenza, lazzeretti}@diag.uniroma1.it, mazzotti.luca.93@gmail.com.

2576-5337/2020/08-ART16
https://doi.org/10.1145/3386581

survey conducted by Panda Security [21], just in 2015, about 230,000 malware samples were delivered per day with an increase of 40% with respect to 2014, assailing the cybersecurity community. A recent analysis from McAfee [2] showed that in the third quarter of 2018 more than 60 million new malware samples were discovered. In the mid-2000s, indeed, the black hat community evolved from adolescent hackers to organized crime networks, fueling highly profitable identity theft schemes with massive loads of personal data harvested from corporate and government networks. In recent times, in fact, a new powerful and dangerous threat is on the rise, identified in the community as "Advanced Persistent Threat" (APT). According to the NIST Glossary of Key Information Security Terms,[1] APT is "*an 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).*" Hence the APT name identifies the main peculiarities of the threat:

Advanced. Criminal minds behind attacks utilize the full spectrum of computer intrusion technologies and techniques. While an individual attacker may not be classed as particularly "advanced" (e.g., single-stage malware component found on the black market), their operators typically access and develop more advanced tools as required.

Persistent. Criminal operators give priority to a specific task rather than opportunistically seeking immediate financial gain. The attack is indeed conducted through continuous monitoring and interaction to achieve the defined objectives. A "low-and-slow" approach is usually more successful.

Threat. The attack has a malicious nature. Malevolent attackers have a specific objective and are skilled, motivated, organized, and, of course, well funded.

Such advanced attacks are strongly targeted to overcome all the *general* defenses that the target can apply. According to the FireEye studies [4], based on the huge amount of APT operations they analyzed, APTs principally target big companies, critical infrastructure, and institutions to gain financial secrets, intellectual property, national secrets, and private personal information or damage critical infrastructures by interrupting or decreasing their functionality. Despite different targets and origin, it has been demonstrated that APTs perform attacks in a standard way that can be represented as an intrusion kill chain [7]. The attack starts with a *Reconnaissance* phase in which the APT deeply studies the victim's infrastructure, followed by the *initial exploitation* phase, where those malicious organizations actually "enter" inside the victim's systems. Given the persistent appellation that characterizes APT, in the next phase APTs *establish persistence* in the host, which provides attackers the possibility to install backdoors or other tools for following attack stages. *Lateral movement* is actually accomplished to escalate privileges and hence to be able to elevate the capability in the system. Finally, in the *exfiltration* phase the real aim of the attack is achieved. It is hence important to identify any malware used by APTs in any of these phases so that other malware used in previous steps can be found and analyzed and defenses can be improved against future activities. In this article, we refer to the malware developed by Advanced Persistent Threats as APT-related malware or APT-malware for the sake of simplifying the text.

1.1 Contribution

According to Reference [7], defenders must implement countermeasures faster than adversaries evolve. It is evident that there is a need for a prioritization mechanism to promptly identify samples that deserve to be further analyzed by security analysts. Taking inspiration from the medical field, in this article we propose a triage-based approach. Doctors are a limited resource, and thus there is the need to prioritize patients that require urgent care. Similarly, the number of security analysts is very low, and thus we need to prioritize malware to avoid wasting time analyzing suspicious software that is not dangerous. Malware developed by APTs are the most dangerous, because they apply a high level of sophistication, and they target important victims. However, the

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

proposed approach has the scope of only highlighting this class of malware. The full analysis is pursued at a later stage by human analysts and specialized architecture components.

We privilege precision metric on accuracy metric; in fact, precision is necessary, because the malware triage should not prioritize non-APT malware (*false positive*) so as to not overload human analysts and/or complex components with urgent but not necessary analyses. However, also, high accuracy is desired to correctly identify samples really belonging to some known APT classes (*true positive*). We believe that this triage step is an important first step in a defense strategy, intending to automate the identification process and reduce the analysis burden for human analysts. In fact, we propose our approach as a module of a complex security architecture, like the one presented in Reference [13]. Our module tries to detect and identify the most malware developed by APTs as possible, prioritizing them, and leaving the remaining ones to be analyzed by other components. In such a way, we save the time of analysts, being confident that the architecture is able to later identify the few undetected APT-related malware with other deeper but slower tools, thus improving the general security of the environment.

A security architecture including our proposed tool can hence be helpful to any critical infrastructure or company that can be target of an APT attack. We believe that adding it to the malware analysis pipelines that are already in place in industries can heavily improve the workflow.

It is possible to directly use a classifier to identify APT-malware. It should output the APT class the malware sample belongs to, or a "non-APT label" in the case where the sample is not associated to any class. Unfortunately, such a solution presents a huge problem. In fact, this so-called *negative class* has a cardinality of several orders of magnitude bigger than the set of APT samples, and there is a large variety among its samples. Thus, the result in terms of accuracy and precision are very poor, and for this reason in Reference [11] the authors proposed a *Random Forest*-based triage (*RFT*) approach. They trained their model on a knowledge base built upon a collection of ATPs' related reports publicly released by cyber-security firms. This solution works well and achieves great accuracy and precision but has the disadvantage of requiring a long training time and needs to be re-trained each time new APT samples or even a new APT class are discovered.

To overcome such problems, in this article we present several key improvements w.r.t. existing literature:

- We propose a novel modular and lightweight malware triage framework, based on *One-class classification*, which permits us to train each classifier only on samples related to the relative class. This permits us to add new APT classifiers and modify a single classifier when new samples are discovered without affecting the whole triage framework;
- We design the malware triage framework on the *Isolation Forest* learning concept. Each Isolation Forest is trained on the samples of a specific APT;
- Our training set is composed of more than 2,000 samples belonging to 15 different APTs: *APT28, APT29, APT30, Carbanak, Desert Falcon, Hurricane Panda, Lazarus Group, Mirage, Patchwork, Sandwork, Shiqiang, Transparent Tribe, Violin Panda, Volatile Cedar*, and *Winnti Group*;
- We introduce features dimensionality reduction through *Linear Discriminant Analysis* to decrease the computational time needed in the training phase, avoid the curse of dimensionality problem, and increase the overall precision and accuracy for each class;
- We compare our work with the results obtained by *RFT* and show that our approach allows fast, precise, and also accurate identification of APT malware through an experimental evaluation performed on a dataset composed by samples obtained by public APT reports. We also test its performance on a non-APT dataset.

1.2 Outline

The remaining part of the article is divided into the following sections: Section 2 shows the state of the art of the analysis of malware and Advanced Persistent Threats; Section 3 introduces the main tools used in this article; Section 4 explains in details the methodology behind our system; Section 5 presents experimental evaluations of our solution; and, finally, Section 6 sums up the results of our work and shows some directions for future activities.

2 RELATED WORKS

The awareness around APT has been increasing over the past few years, becoming an important research topic within the cybersecurity area. Important works have been done to deploy APT detection as well as avoidance frameworks to identify compromised hosts. In Reference [13], the authors propose the design of an architecture composed by various tangled phases that starts from the uninterrupted collection of malware and reports from different sources, continues through different analysis components that work on single elements (like static analysis tools) and groups of elements (like classification algorithms), and at the end stores all the information in a knowledge base that can be easily linked to others to share different information. Inside such an architecture, several malware analysis tools can be used. The number of works carried out to identify and classify malware samples, even if not focused on APT, is really huge, and hence we here present a small subset of them, more related to our work. In Section 2.1, we focus on malware analysis through machine learning techniques, while in Section 2.2, we present works related to APT malware.

2.1 Malware Identification

An important malware analysis technique is the BitShred framework [8], which extracts information from the sample and, using feature hashing, creates a probabilistic data structure to large-scale correlate samples.

Another fast and precise malware analysis framework is SigMal [9], a framework improving the state of the art of the previous systems based on the concept of malware similarity by leveraging signal processing techniques to extract noise-resistant signatures from the samples. In Reference [16], malware triage has been deployed by machine learning classification using Artificial Neural Network. In particular, the dataset used in the classification involves 3,131 samples spread over 24 different unique malware classes, and the overall detection and classification accuracy is about 96%. Each binary executable is represented as a greyscale image through GIST descriptors [20]; then neural networks are used to derive and extract similar patterns among the various samples. In Reference [1], Ahmadi et al. propose a learning-based system using different malware characteristics to assign malware samples to their corresponding families. Because of accurate and fast classification, they propose to extract static features both from the binary hex view and the assembly one to exploit complementary information by these two representations without relying on other information derivable from dynamic analysis of malware. Kong et al. [10] propose a framework for automated malware classification based on the function call graph of the malware. First, they disassemble the single sample building the relative function call graph. Then, discriminant malware distance is computed on each pair of samples: They perform pairwise graph matching between the attributed function call graphs of two malware instances to measure their structural similarity. In Reference [19], Nari and Ghorbani introduce a framework for automated classification of malware samples based on their network behavior. IP addresses, port numbers, protocols, and dependencies between network activities are abstracted for further analysis.

2.2 APT Detection

Many works about APT detection focus on the identification of indicators of suspicious activities, like anomaly detection systems. In Reference [5], authors propose an anomaly detection system for APTs built around several security recorded logs. In particular, this white-list-based approach keeps track of system events, their dependencies, and occurrences, learning the normal system behavior and reporting all *strange* actions. Another interesting work is the framework proposed by Marchetti et al. [17] that aims to detect, among all the hosts inside the company, the ones infected by APT to be further analyzed in the future. In line with the idea of the previous works, Ussath et al. [26] develop a Security Investigation Framework (SIF) that supports the analysis and the tracing of multi-stage APTs. In particular, it leverages different information sources to give a comprehensive overview of the different stages of an attack.

In the previous malware triage work [11], the authors aim to recognize malware developed by APTs, but, contrary to the other described works, they focus on the sample analysis instead of the malware activity observed during the infection. The work achieves high precision and very high accuracy in detecting the ownership of this kind of malware but suffers the problem highlighted in Section 1.

3 TOOLS

In this section, we introduce the main tools used in this article. In particular, Section 3.1 presents PEFrame, a static malware analysis tool we use for feature extraction. We then outline Random Forest and Isolation Forest in Sections 3.2 and 3.3, respectively. Section 3.4 presents principles of feature reduction.

3.1 PEFrame

PEFrame[2] is an open-source tool that performs static analysis on Portable Executable malware and generic suspicious file. The Portable Executable format contains the information necessary for the Windows OS loader to manage the wrapped executable code. It consists of the MS-DOS stub, the PE file header, and the sections and can provide an enormous amount of features, containing relevant information for a malware analyst.

Thanks to PEFrame, we successfully extract more than 4,000 features. We then leave classifiers and Linear Discrimination Analysis the handling of this huge number (see Section 3.4 for more details). These features can be roughly grouped into eight categories as follows:

Optional Header (30 features). Every file has an optional header that provides information to the loader. This header is optional in the sense that some files (specifically, object files) do not have it. For image files, this header is required. An object file can have an optional header, but generally this header has no function in an object file except to increase its size. Features are extracted from the optional header of the PE and contain 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 (17 features). The MS-DOS executable-file header is composed of four distinct parts: a collection of header information (such as the signature word, the file size, etc.), a reserved section, a pointer to a Windows header (if one exists), and a stub program. MS-DOS uses the stub program to display a message if Windows has not been loaded when the user attempts to run a program. In this contest, we are interested in 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 (18 features). The Windows executable-file header contains information that the loader requires for segmented executable files. This includes the linker version number, data specified by the linker, data specified by the resource compiler, tables of segment data, and tables of resource data. Moreover, the features related to this class highlight information about timestamp and the CPU platform that the PE is intended for.

Obfuscated String Statistics (3 features). Binaries contain program messages stored as strings that can be useful to understand their behavior. Classical tools like *String* extract byte sequences that can be readable strings to find these messages. Malware authors encode strings in their program to avoid extraction, in fact, even simple schemes can defeat this kind of tool and complicate static and dynamic analysis. In addition to PEFrame, we use functionalities of the FireEye Labs Obfuscated String Solver (*FLOSS*[3]). It is an open-source tool that automatically detects, extracts, and decodes obfuscated strings, such as malicious domains,

[2]https://github.com/guelfoweb/peframe.
[3]https://github.com/fireeye/flare-floss.

IP addresses, suspicious file paths, and so on, from Windows Portable Executable files availing of advanced static analysis techniques. We leverage this tool to compute some statistics, such as how many entry-points or relocations are present in the file.

Mutex (7 features). Mutex are objects commonly used to avoid simultaneous access to a resource, like a variable. If different software checks for the same mutex, then they can be linked. Our features are Boolean values that map the use of particular mutex identified in the training data.

Packer (64 features). Packers are software that compress binaries, keeping them executable. Similarly to the mutex related features, our features highlight if some particular packers are recognized.

Imported API (3917 features). Each software imports functions from common libraries or external files. The combination of imported functions can show similar behavior. We store this information as a vector with a Boolean value for each imported function, using the list of functions present in the training set as a taxonomy.

Buckets (98 features). Similar size in functions and directories can be a proof of similarity in the file structure and thus in the behavior. We observed that, usually, function size values range from 0 to 1,822 bytes, while directory size values range from 0 to 2,638 kbytes. To track these properties, both of them are subdivided into 49 buckets. Each bucket represents a range and counts the elements whose size is in the range. To choose the different ranges we observe the distribution of sizes and lengths in the training set, trying to form buckets that can better characterize the various classes.

3.2 Random Forest Classifier

A Random Forest [3] is a supervised classification tool that aggregates the results provided by a set of *Decision Trees* through a *Bootstrap aggregated* (a.k.a. *Bagging*) technique.

In general, a *Decision Tree* [6] is a decision support tool that uses a treelike graph as model of decisions. In machine learning science, *Decision tree learning* concept uses decision trees as a predictive model to go from observations about an object to a conclusion about the objects' target value, represented in the leaves of the tree.

Although single decision trees can be effective classifiers, increased accuracy often can be achieved by combining the results of a collection of decision trees. As highlighted by the name, a forest is generated by randomly ensembling different decision trees. The ensemble method for Random Forest is focused on *feature bagging* concept, which has the advantage of significantly decreasing the correlation between each decision tree and thus increasing, on average, its predictive accuracy. Feature bagging works by randomly selecting a subset of the feature dimensions at each split in the growth of the individual decision trees. Although this might sound counterproductive, since it is often desired to include as many features as possible, it has the purposes of deliberately avoiding on average very strong predictive features that lead to similar splits in trees, thereby increasing correlation. If a particular feature is strong in predicting the response value, then it will be selected for many trees.

To classify a new object from an input vector, the algorithm analyzes the input with all the trees composing the forest. Each tree outputs a class label, and the forest gives as result the class having the highest number of votes.

3.3 Isolation Forest Classifier

In classification problems, there is the possibility to rely on *one-class classification*, where samples used in training phase belong to the same class, and in the classification phase, we use the classifier to distinguish between correct samples, i.e., samples probably related to the class, and outliers, i.e., samples not belonging to the class used for training.

Isolation Forest [15] is a one-class classifier where the data structure is built on the same conceptual principle of Random Forests. In particular, an Isolation Forest is still an ensemble of bootstrapped decision trees where distinguishing features are selected by the protocol, but, instead of profiling normal points, it shows all possible anomalies that are *isolated* with respect to the models just created. As stated by Liu et al. [15], anomalies are data patterns that have different data characteristics from normal instances. An isolation forest is composed by several trees similar to binary search trees, and anomalies are identified when an anomaly score, based on the average path length in visiting the isolation trees, exceeds a given threshold. Many existing model-based approaches to anomaly detection construct a profile of normal instances and hence isolate instances that do not conform to the normal profile.

Technically speaking, the Isolation Forest algorithm isolates observations by randomly selecting a feature and then randomly selects a split value between the minimum and maximum values assumed by the selected feature. Hence, the algorithm first constructs the separation by creating isolation trees or random decision trees; then, when they collectively produce shorter path lengths for some particular points, they are highly likely to be anomalies. Based on the average path length derived in the testing phase, it calculates the *anomaly score* for each sample. In the various online tool libraries, it is possible to set a *tolerance* threshold that discriminates the given sample as isolated or not according to the model: The higher it is, the more tolerant the Isolation Forest is with anomalies. The implementation that we used is based on heights of trees as a metric for decisions, as proposed in Reference [15].

3.4 Principles of Features Reduction

When dealing with high-dimensional features space, it is actually infeasible to think that each feature has the same "importance." In particular, both *redundant* and *irrelevant* features exist. We have performed an initial *feature selection* basing it on the importance value given by Random Forest algorithm to the features described in Section 3.1 as a reference, discovering that only 264 of them influence our data, less than the 300 ones used by *RFT*.

Moreover, feature reduction techniques can be hence used to reduce the dataset dimensionality to facilitate the training process, decrease training and classification computational time, reduce the variance among features, and avoid *curse of dimensionality*, i.e., with the increasing of the dimensionality, space volume increases so fast that training data become sparse.

In this article, for features reduction purposes, we rely on the *Linear Discriminant Analysis* (LDA) technique, because of its simplicity. LDA is a generalization of Fisher's linear discriminant [18], a method used in statistics, pattern recognition and machine learning to project the original features space in a smaller one through a linear combination of features. In practice, given the original dimension M of the features, it is possible to reduce the dimension to a chosen L by projecting into the linear subspace H_L maximizing inter-class variance after projection.

4 METHODOLOGY

In this section, we define the methodologies that lead us to define the proposed framework with high performances. In particular, in Section 4.1, we explain the feature extraction process, in Sections 4.2 and 4.3, we respectively discuss the *RFT* multi-class methodology proposed by Laurenza et al. [11] and the one-class classification approach through which we implement the malware triage.

4.1 Features Extraction

To compare the approaches, we rely on the data provided by dAPTaset [12]. It is a publicly available database[4] with different information about APTs activities, including various *Indicators of Compromise* like malware hashes, network domains, and IP addresses. dAPTaset contains more than 9,000 unique malware hashes. This number is nowhere near the total amount of malware in the world. However, known APTs are a limited number, and they are usually developed with a small group of suspicious software. In fact, they usually tailor their malware to the specific victims, without spreading their attacks in the wild. Moreover, we focus our work on windows executable files, which are a subset of all the malware included in dAPTaset. We have searched such samples in

[4]http://github.com/GiuseppeLaurenza/dAPTaset.

Table 1. Malware Count per APT Class

Class	**Count**
APT28	68
APT29	205
APT30	101
Carbanak	105
Desert Falcon	45
Hurricane Panda	315
Lazarus Group	58
Mirage	54
Patchwork	559
Sandwork	44
Shiqiang	31
Transparent Tribe	267
Violin Panda	23
Volatile Cedar	35
Winnti Group	176
Total	2086

available public sources, and, unfortunately, we came up with only 2,086 samples related to 15 distinct groups (we have discarded APT groups where we retrieved fewer than 10 samples). Table 1 shows the distributions of these malware. Our approach should be validated also with a larger dataset, and we hope that in the future it will be possible to access to security firms exclusive binaries.

After having retrieved the malware samples, we process them with the PEFrame tool to extract features from each binary. As highlighted in Section 3.1, for each analyzed executable, the extraction tool provides us seven distinct features classes with more than 4,000 characteristics.

4.2 Multi-class Classification

RFT [11] creates a multi-class model to deal with APT classification. The authors explain that, when training a multi-class classifier, all the output classes need to be also in the training set. Thus, when dealing with APT-malware, this set must contain a class for each APT and also a class for all the malware that are not APT-related. However, the group composed of the elements of the latter class is not homogeneous, making hard to train a classifier on it. Moreover, the huge difference in the cardinality of this class with the respect to APT classes leads to an excessive imbalance in the training set. For this reason, in the training phase, authors only consider classes of known APTs. To discriminate a sample as not related to any APT, they set a threshold for each class in the Knowledge Base (KB): If the score of a given sample analyzed is lower than all the relative APT thresholds, then it is assumed to belong to the non-APT class.

Laurenza et al. [11] tested *RFT* with 10-fold cross-validation, a common value in literature. For each execution, they generate the model with $k - 1$ folds and test it with both the remaining fold and all the collected malware not developed by APTs.

4.3 One-class Classification

In the multi-class classification, each time an APT sample is going to enrich the Knowledge Base, there is the need to re-train the whole *RFT* model, to update classification parameters and APT thresholds, wasting time and resources. In this article, we propose to leverage on an Isolation Forest for every single APT class. This classification algorithm requires in the training set only the single class it should identify. In such a way, we can create a model that identifies all those elements belonging to the related APT while considering the other samples as anomalies. All the one-class classifiers form an ensemble classifier [14] where it is sufficient that one of them recognizes a sample to be associated to the relative APT to raise the alarm in the triage phase. Whenever a new sample is identified and associated with a known APT, it is going to enforce the knowledge base, and it is sufficient to re-train the related APT classifier in the case it would exhibit a *concept drift* [25], i.e., its statistical properties drifts from the ones at the time of training. The other one-class classifiers are not impacted by the concept drift and are hence kept unchanged.

We implemented our classifiers on the open-source Scikit library,[5] which provides different machine learning tools in Python programming language. Linear discriminant analysis has been used to reduce the feature dimensionality from 264 to $N - 1$ features, with N the number of APTs. After having split the dataset in training set and validation set through 10-folder cross-validation, we have trained one Isolation Forest for each class by using only malware of the training set developed by the specific corresponding APT. Isolation Forest implementation requires two main arguments to tune in the fitting phase: *contamination* expresses how much the classifier has to be tolerant in creating the model for detecting outliers; *number of estimators* is an attribute totally related to the tree-nature of the Isolation Forest, and it defines how many trees it has to use in the ensemble methods. To tune the ⟨*contamination*, *number of estimators*⟩ tuple for achieving the best result for each Isolation Forest classifier, we performed cross-validation over the APT training set.

To train our Isolation Forest classifiers, we decided to provide a non-zero value to contamination. This means that some of the training samples can be wrongly considered as outliers but increases precision in the classification phase. Such a decision is mainly motivated by the following reasons: (i) APT reports are a known unreliable source of labeled data [24], and hence we cannot be sure that any identified sample is really related to the specific APT. (ii) Malware samples have high feature variability, which can compromise the performance of the classifiers if we do not allow them to discard atypical samples in the training phase, and (iii) APT can leverage on masking (a non-APT sample is detected) to induce misclassification errors [23]. This is possible if an APT is going to use many different malware samples not really necessary for its activity but only to confuse analysts and ML tools during training. This would result in many false alarms in the classification test. Introducing contamination does not completely solve the problem but makes the system more robust. On the other side, APTs usually perform swamping (an APT sample is classified as an outlier) to hide their presence behind the samples. However our Isolation Forests are trained only on samples belonging to APTs according to public reports, and hence swamping is not affecting the training. We only miss not recognized samples and similar samples during classifications. In the choice of the correct parameters, we mainly rely on the following: (i) *precision*, due to the malware triage nature of the proposed framework; a high number of false positives is undesirable, since it would increase the workload of human analysts, and (ii) *accuracy*, expressing how much the APT classifier is accurate in determining the right APT class of an APT sample. The choice of the contamination-number of estimators parameters for the Isolation Forest is hence totally based on the precision-accuracy tradeoff. Because of the triage nature of the framework, we have been more oriented to tune this choice toward precision measure. In fact, in our view, in a triage application, stating with high precision that an APT sample belongs to a given APT with high certainty, even if some APT samples are not identified, is more important than recognizing all the APT samples, but also raising false alarms related to non-APT sample that are assigned to some class.

4.4 Result Validation

As previously explained, the triage system should prioritize APT-related malware that are the most dangerous ones. Samples associated with some APT are immediately dispatched to human analysts, which have the tasks of validating the classifier results and performing in-depth sample analysis. We note that our triage prefers precision

[5] http://scikit-learn.org/.

to accuracy. Hence false negatives can occur. In this case, APT samples are not recognized. However this does not mean that they are excluded by further analysis, but they are not prioritized. Indeed, a high classification score can indicate that a sample deserves further inspection, even if not classified in an APT class. Scores can be a base of a further prioritization; malware with a higher score has more probabilities to be related to an APT, and thus it can deserve more attention than others.

The output of the triage is the label predicted by the frameworks, but, as *collateral* results, it gives also a score for each of the possible labels. These scores can be very helpful for human analysts, because by knowing in advance which group is using the sample, analysts can easily retrieve information about the APT activities. Moreover, these scores can show interesting relationships that analysts or other tools can successively investigate. For example, they may indicate that an APT is trying to imitate another one or even that previously separated groups are part of a single APT.

5 ANALYSIS

In this section, we compare the proposed triage scheme with *RFT*. For a fair comparison, we have implemented and tested both the solutions. We have built our dataset by following the methodology used in Reference [11]. We have trained a Random Forest classifier on the 15 APT classes in Table 1 and then created a new dataset by selecting only the classes that are recognized with Precision and Recall over 95%. This approach, already adopted in Reference [11], gives us two sets of samples, one with malware belonging to 15 APT classes and the other one with malware of 6 APT classes, the latter having higher performances. These classes are APT28, APT30, Carbanak, Hurricane Panda, Patchwork, and Transparent Tribe. The same 2 sets have been used to train and test the Isolation Forest classifiers. Following the same approach of the previous work, we have computed the thresholds for *RFT* : We take the average *confidence* for each class, and we decrease them by a factor Δ that we set as 5%, 10%, and 15%.

Due to the huge difference in size between the positive and negative classes, we use the weighted version of *Precision* and *Recall*. *F1-Score* is computed with these values instead of the classical ones. The weighted formulas are as follows:

$$WeightedPrecision = \frac{1}{\sum_{l \in L} TP_l} \sum_{l \in L} TP_l * Precision_l$$

$$WeightedRecall = \frac{1}{\sum_{l \in L} TP_l} \sum_{l \in L} TP_l * Recall_l$$

- L is the set of labels;
- TP_l is the set of samples that have the true label l;
- $Precision_l | Recall_l$ compute the precision or recall for samples that have the label l.

In Section 5.1, we evaluate the performances of the two solutions in discriminating whether a sample *belongs* or *not* to some APTs and their performances in the identification of the correct APT class, showing the achieved results for each classifier. Finally, in Section 5.2, we compare the execution time required by both our works. A draft of the code with the used dataset can be found on GitHub.[6]

5.1 APT-Triage

To generate a sound and uniform test, we rely on the Scikit's *Stratified 10-Fold Validation*, which splits data into train/test sets, guaranteeing the same percentage of samples for each class. The train set is used to train an Isolation Forest for each APT class in the dataset, thus creating 6 and 15 Isolation Forests for the two identified

[6]https://github.com/GiuseppeLaurenza/I_F_Identifier.

Table 2. APT-Triage Confusion Matrix with Six APTs

One-class classifier		*Predicted*	
		APT	non-APT
Real	APT	1173	242
	non-APT	69	90141

***RFT* (Δ = 5%)**		*Predicted*	
		APT	non-APT
Real	APT	1227	164
	non-APT	12	8618

***RFT* (Δ = 10%)**		*Predicted*	
		APT	non-APT
Real	APT	1261	130
	non-APT	25	8605

***RFT* (Δ = 15%)**		*Predicted*	
		APT	non-APT
Real	APT	1272	119
	non-APT	100	8530

Table 3. APT-Triage Quality Measures with Six APTs

	Accuracy	**F1**	**Precision**	**Recall**
One-class classifier	0.9966	0.9966	0.9965	0.9966
***RFT* (Δ = 5%)**	0.9824	0.9825	0.9826	0.9824
***RFT* (Δ = 10%)**	0.9845	0.9845	0.9845	0.9845
***RFT* (Δ = 15%)**	0.9781	0.9781	0.9780	0.9781

scenarios. We built a group of classifiers for each scenario, because we related the number of features produced by the LDA algorithm to the number of classes. In this way, each scenario has its own number of features for the generation of the forest, thus requesting two separate training phases. A sample is recognized to belong to some APT if at least one Isolation Forest recognizes it in its class. Otherwise, a malware is classified as *non-APT* if no classifier recognizes it. In *RFT*, we consider a sample as *non-APT* if all the scores given by the framework are lower than the relative thresholds. We performed a preliminary test of *RFT* solution with a set of more than 800 binaries not belonging to Advanced Persistent Threats. Then, we validated our proposed solution based on Isolation Forests on a larger set of non-APT samples containing 9,000 binaries. While APT samples are used in both training and classification in our 10-folder cross-validation (hence each sample is used only once in classification), non-APT samples have been used only to evaluate all the 10 trained models. Thus, in the various confusion matrices, the obtained results are the sum of the 10 confusion matrices, obtained each one with the corresponding *testing folder* and the entire non-APT set of malware.

Tables 2 and 3 show details about the result of the first experiment in the 6 APTs scenario, while Tables 4 and 5 in the 15 APTs one. Our results are slightly better than the ones obtained by *RFT*, in fact, we correctly detect most of the malware developed by APTs, achieving a precision of over 99%. Even if our solution detects a bit less APT malware than *RFT*, we strongly reduced the number of false-positive cases, cutting more than half of them in percentage with respect to the other solution. As described in the previous section, a triage approach must focus on lowering the false detection rate to not waste analysts' effort. Hence, this small loss in accuracy can be

Table 4. APT-Triage Confusion Matrix with 15 APTs

One-class classifier		*Predicted*	
		APT	non-APT
Real	APT	1756	330
	non-APT	685	89525

***RFT* (Δ = 5%)**		*Predicted*	
		APT	non-APT
Real	APT	1759	327
	non-APT	45	8585

***RFT* (Δ = 10%)**		*Predicted*	
		APT	non-APT
Real	APT	1803	283
	non-APT	60	8570

***RFT* (Δ = 15%)**		*Predicted*	
		APT	non-APT
Real	APT	1831	255
	non-APT	77	8553

Table 5. APT-Triage Quality Measures with 15 APTs

	Accuracy	**F1**	**Precision**	**Recall**
One-class classifier	0.9890	0.9895	0.9901	0.9890
***RFT* (Δ = 5%)**	0.9653	0.9654	0.9656	0.9653
***RFT* (Δ = 10%)**	0.9680	0.9680	0.9680	0.9680
***RFT* (Δ = 15%)**	0.9690	0.9689	0.9688	0.9690

well accepted if it implies a heavy reduction of *false alarms* and, thus, of time loss. However, the results of these tests show improvements in all the measures, confirming the goodness of our solution.

Tables 6 and 8 show the results of the second experiment in the 6 APTs scenario, and Table 7 presents the detailed confusion matrix for this scenario. Tables 9 and 10 contain results of the second experiment in the 15 APTs scenario. Again, in this experiment, we achieve results comparable with *RFT*. The small number of APTs misclassified as other ones in the second experiment is related to the structure of our framework. In fact, using different classifiers, it is possible that more than one classifier labels the same sample as belonging to the relative APT. As introduced in Section 4.4, our framework can provide a *score vector*, thus this kind of error can be probably reduced by evaluating it before taking a decision. Indeed, this number is small enough to be negligible, in fact, the average metrics are only less than 0.002% lesser than the ones obtained by the other solution. Moreover, the assignment of a sample to more classes can highlight some relations between APTs that an analyst can deepen.

Summarizing the result of both tests, we always achieve results comparable to the ones obtained by Laurenza et al., even better in most of the cases, and thus we can conclude that we have improved the results of *RFT*.

5.2 Execution Time

We here compare the runtime of training and evaluation phases for both *RFT* and Isolation Forests. In our tests, we have measured the time of 10 executions and computed the average time.

Table 6. APT-Identification Confusion Matrix with Six APTs

One-class classifier		*Predicted*	
		Correct APT	Other APTs
Real	Correct APT	1171	2
	Other APTs	2	5865
***RFT* (Δ = 5%)**		*Predicted*	
		Correct APT	Other APTs
Real	Correct APT	1224	3
	Other APTs	3	6132
***RFT* (Δ = 10%)**		*Predicted*	
		Correct APT	Other APTs
Real	Correct APT	1258	3
	Other APTs	3	6302
***RFT* (Δ = 15%)**		*Predicted*	
		Correct APT	Other APTs
Real	Correct APT	1269	3
	Other APTs	3	6357

Table 7. Detailed APT-Identification Confusion Matrix of Our Isolation Forest-based Triage with Six APTs

	Predicted						
		APT28	**APT30**	**Carbanak**	**Hurricane Panda**	**Patchwork**	**Transparent Tribe**
Real	**APT28**	44	0	0	0	0	0
	APT30	0	81	0	0	0	0
	Carbanak	0	0	97	0	0	0
	Hurricane Panda	0	0	0	293	0	2
	Patchwork	0	0	0	0	408	0
	Transparent Tribe	0	0	0	0	0	248

Table 8. APT-Identification Quality Measures with Six APTs

	Precision	**Recall**	**Accuracy**	**F1**
One-class classifier	0.9994	0.9994	0.9994	0.9994
***RFT* (Δ = 5%)**	0.9992	0.9992	0.9992	0.9992
***RFT* (Δ = 10%)**	0.9992	0.9992	0.9992	0.9992
***RFT* (Δ = 15%)**	0.9992	0.9992	0.9992	0.9992

Tables 11 and 12, respectively, show execution time in seconds with 6 and 15 APTs. Tables also include the training time required by the LDA algorithm to generate the LDA transformation, based on the training set. On the other side, LDA runtime for feature reduction in classification is not provided, because it is negligible. We underline that the configuration of the LDA transformation is performed only at the first training and repeated when we need re-training of all the Isolation Forests (because precision and accuracy significantly decreased after the introduction of several new samples) or when we need to add another APT. In both training and prediction, our solution is faster than the previous one, especially in the training phase. In fact, training of a set of Isolation

Table 9. APT-Identification Confusion Matrix with 15 APTs

One-class classifier		*Predicted*	
		Correct APT	Other APTs
Real	Correct APT	1731	56
	Other APTs	25	25018
***RFT* (Δ = 5%)**		*Predicted*	
		Correct APT	Other APTs
Real	Correct APT	1753	6
	Other APTs	6	24620
***RFT* (Δ = 10%)**		*Predicted*	
		Correct APT	Other APTs
Real	Correct APT	1797	6
	Other APTs	6	25236
***RFT* (Δ = 15%)**		*Predicted*	
		Correct APT	Other APTs
Real	Correct APT	1822	9
	Other APTs	9	25625

Table 10. APT-Identification Quality Measures with 15 APTs

	Precision	**Recall**	**Accuracy**	**F1**
One-class classifier	0.9970	0.9970	0.9970	0.9970
***RFT* (Δ = 5%)**	0.9995	0.9995	0.9995	0.9995
***RFT* (Δ = 10%)**	0.9996	0.9996	0.9996	0.9996
***RFT* (Δ = 15%)**	0.9993	0.9993	0.9993	0.9993

Table 11. Execution Time with Six APTs in Seconds

	One-class classifier	***RFT***
Feature Reduction (LDA)	0.1729 ± 0.0004	0.0000 ± 0.0000
Training Phase	1.6948 ± 0.3495	1031.9726 ± 9.1574
Prediction Phase	3.7564 ± 0.2253	7.0076 ± 0.0012

Forests is orders of magnitude lower than the training of *RFT* and computing the various thresholds. Moreover, we note that the training time of our solution is the sum of the time required to build each single Isolation Forest; thus, when there is the need to re-train only a single class, for example, after collecting new samples developed by a particular APT, the time is only a fraction of that total time. Our solution requires an average of 0.3 s to build an Isolation Forest related to a particular APT.

6 CONCLUSION

Among the huge amount of malware daily produced, those developed by 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.

Table 12. Execution Time with 15 APTs in Seconds

	One-class classifier	***RFT***
Feature Reduction (LDA)	0.2129 ± 0.0005	0.0000 ± 0.0000
Training Phase	1.6179 ± 0.0111	1600.7085 ± 25.1686
Prediction Phase	4.7240 ± 0.1566	8.4778 ± 0.3841

This work improves the previous automatic malware triage process, reducing needed computational resources, reducing the time required for the training phase, and providing higher flexibility. These improvements are achieved through the change of the classification approach, moving from a single multi-class classifier to a set of one-class classifiers. Both algorithms were used with features obtained by static analysis on 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 prioritize the analysis of malware that surely deserve higher attention, with minimal risk of wasting analysts' time. In fact, the experimental evaluation has shown encouraging results: Malware developed by known APTs have been detected with precision and accuracy over 90%.

As future work, first, we want to enlarge the Knowledge Base. In this work, we have realized classifiers relative to 15 distinct classes, but the actual number of APTs is higher. Thus, we are interested in enriching our Knowledge Base by crawling other public APT reports to be able to detect, among the huge number of malware delivered per day, the ones more relevant for further analysis.

As outlined in Section 4.3, a re-training is necessary each time the discovery of new malware would cause a concept drift in the model. Tracking concept drift [25] is hence critical to the successful application of the proposed solution and must be included in our future works.

Another interesting future development regards the improvements of the classification through the revise of the *features reduction* phase and the classification algorithm. For the former, we want to observe the impact of variation of the number of projections or applying other techniques such as *Principal Component Analysis.* Given the modularity of our proposal, we also want to test other algorithms based on different principles respect to Isolation Forests, also examining the possibility to rely on different approaches (not necessarily based on machine learning tools) for different APTs. Also, different metrics can be considered in our approach. For example, an interesting approach to be investigated is the one proposed in Reference [27] for the identification of malicious emails, which presents a new scoring system that is based on the combination of various scores applied on different subgroups of features. The modularity of our approach is a strong advantage of our proposal, in fact, we should use different classification algorithms and metrics for different APTs, choosing each time the one that performs better or combining the output of more single classifiers. Modularity also permits to introduce classifiers for other problems of interest, such as the recognition of important malware families, even if not necessarily connected to APTs.

Another aspect that we want to study is the use of different types of features: In our works, we focus on characteristics extracted from the headers of the file and its structure, but the static analysis also includes *code analysis* that we slightly use only in the form of function size observation. *Code reuse* and *code similarities* are techniques that we can exploit to add new information to our knowledge base. Similarly, both our proposal and previous work rely on static features to provide a fast response, but many obfuscation techniques can hide characteristics useful for APT identification. To overcome these *protections* made by adversaries, it is always possible to use dynamic analysis tools that can observe the behavior of the malware. Even if these tools require a higher amount of time respect to static analysis, we can use dynamic analysis in a second step after the first triage one to validate results of the first phase or take decision on samples that have not been identified in the first step but whose classification score is close to the threshold. Moreover, we plan to investigate also the field of *semantic features*. Works like Reference [22] show how it is possible to use semantic features to determining malware similarity and the temporal ordering of malware, generating also malware lineages. We believe that these information can heavily help to detect and identifying malware developed by APTs, with the addition of highlighting the evolution of the tools used by attackers to perform their activities.

Finally, as described in Section 4.4, a collateral result of our framework is a *scoring vector* showing the probability of a binary to be developed by each APT. Such score vectors can be exploited in many different ways. First, they can be used in a multi-level prioritization system where samples not classified as belonging to some APT are scheduled to analysts according to their scores, because a higher score highlights some correlation with other malware developed by that particular APT. Moreover, they provide much other useful information, beyond our triage analysis. These scores might highlight relationships among different groups, and investigating them can then lead to interesting discoveries, such as, for example, a group trying to imitate another one, more APTs collaborating against a common target, or also an APT incorrectly divided in more groups. It is of our interest to explore their utility in future applications, beyond malware triage.

REFERENCES

[1] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. 2016. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the Conference on Data and Application Security and Privacy (CODASPY'16)*. ACM, 183–194.

[2] Alex Bassett, Christiaan Beek, Niamh Minihane, Eric Peterson, Raj Samani, Craig Schmugar, ReseAnne Sims, Dan Sommer, and Bing Sun. 2018. *McAfee Labs Threats Report: December 2018*. Technical Report. McAfee. Retrieved from https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2018.pdf.

[3] Leo Breiman. 2001. Random forests. In *Machine Learning*. Springer, Berlin, 5–32.

[4] FireEyeLabs. 2014. *Advanced Threat Report 2013*. Technical Report. FireEyeLabs. Retrieved from https://www2.fireeye.com/advanced-threat-report-2013.html.

[5] Ivo Friedberg, Florian Skopik, Giuseppe Settanni, and Roman Fiedler. 2015. Combating advanced persistent threats: From network event correlation to incident detection. In *Computers & Security*. Elsevier, 35–57.

[6] Kamil A. Grajski, Leo Breiman, Gonzalo Viana Di Prisco, and Walter J. Freeman. 1986. Classification of EEG spatial patterns with a tree-structured methodology: CART. *IEEE Trans. Biomed. Eng.* 12 (1986), 1076–1086.

[7] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. 2011. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. In *Leading Issues in Information Warfare & Security Research*. 80.

[8] Jiyong Jang, David Brumley, and Shobha Venkataraman. 2010. Bitshred: Fast, scalable malware triage. Cylab, Technical Report CMU-Cylab-10, Carnegie Mellon University, Pittsburgh, PA.

[9] Dhilung Kirat, Lakshmanan Nataraj, Giovanni Vigna, and B. S. Manjunath. 2013. Sigmal: A static signal processing based malware triage. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC'13)*. ACM, 89–98.

[10] Deguang Kong and Guanhua Yan. 2013. Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining, ACM SIGKDD*. ACM, 1357–1365.

[11] Giuseppe Laurenza, Leonardo Aniello, Riccardo Lazzeretti, and Roberto Baldoni. 2017. Malware triage based on static features and public APT reports. In *Proceedings of the International Conference on Cyber Security Cryptography and Machine Learning (CSCML'17)*. Springer, 288–305.

[12] Giuseppe Laurenza and Riccardo Lazzeretti. 2019. dAPTaset: A comprehensive mapping of APT-related data. In *Computer Security*. Springer, 217–225.

[13] Giuseppe Laurenza, Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2016. An architecture for semi-automatic collaborative malware analysis for cis. In *Proceedings of the International Conference on Dependable Systems and Networks Workshop (DSN'16)*. IEEE, 137–142.

[14] Charles LeDoux and Arun Lakhotia. 2015. Malware and machine learning. In *Intelligent Methods for Cyber Warfare*. Springer, 1–42.

[15] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *Proceedings of the International Conference on Data Mining (ICDM'08)*. IEEE, 413–422.

[16] Aziz Makandar and Anita Patrot. 2015. Malware analysis and classification using artificial neural network. In *Proceedings of the International Conference on Trends in Automation, Communications and Computing Technology (I-TACT'15)*. IEEE, 1–6.

[17] Mirco Marchetti, Fabio Pierazzi, Michele Colajanni, and Alessandro Guido. 2016. Analysis of high volumes of network traffic for advanced persistent threat detection. In *Computer Networks. Elsevier*, 127–141.

[18] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. 1999. Fisher discriminant analysis with kernels. In *Proceedings of the Neural Networks for Signal Processing, Signal Processing Society Workshop*. IEEE, 41–48.

[19] Saeed Nari and Ali A Ghorbani. 2013. Automated malware classification based on network behavior. In *Proceedings of the International Conference on Computing, Networking and Communications (ICNC'13)*. IEEE, 642–647.

[20] Aude Oliva. 2005. Gist of the scene. In *Neurobiology of Attention*. Elsevier, 251–256.

[21] PandaLabs. 2015. *Annual Report 2015*. Technical Report. PandaLabs. Retrieved from https://www.pandasecurity.com/mediacenter/src/uploads/2014/07/Pandalabs-2015-anual-EN.pdf.

[22] Avi Pfeffer, Catherine Call, John Chamberlain, Lee Kellogg, Jacob Ouellette, Terry Patten, Greg Zacharias, Arun Lakhotia, Suresh Golconda, John Bay, et al. 2012. Malware analysis and attribution using genetic information. In *Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software*. IEEE, 39–45.

[23] Robert Serfling and Shanshan Wang. 2014. General foundations for studying masking and swamping robustness of outlier identifiers. In *Statistical Methodology*. 79–90.

[24] Deana Shick and Kyle OMeara. 2016. Unique Approach to Threat Analysis Mapping: A Malware Centric Methodology for Better Understanding the Adversary Landscape. Technical Report.

[25] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia. 2012. Tracking concept drift in malware families. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*. 81–92.

[26] Martin Ussath, Feng Cheng, and Christoph Meinel. 2015. Concept for a security investigation framework. In *Proceedings of the International Conference on New Technologies, Mobility and Security (NTMS'15)*. IEEE, 1–5.

[27] Ian Welch, Xiaoying Gao, Peter Komisarczuk, et al. 2012. A novel scoring model to detect potential malicious web pages. In *Proceedings of the International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 254–263.

Received May 2019; revised February 2020; accepted March 2020