COMPUTERS & SECURITY 81 (2019) 123–147

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Survey of machine learning techniques for malware analysis

Daniele Ucci[a,*], *Leonardo Aniello*[b], *Roberto Baldoni*[a]

[a] *Research Center of Cyber Intelligence and Information Security, "La Sapienza" University of Rome, Italy*
[b] *Cyber Security Research Group, University of Southampton, United Kingdom*

ARTICLE INFO

Article history:
Received 1 February 2018
Revised 30 October 2018
Accepted 9 November 2018
Available online 24 November 2018

Keywords:
Portable executable
Malware analysis
Machine learning
Benchmark
Malware analysis economics

ABSTRACT

Coping with malware is getting more and more challenging, given their relentless growth in complexity and volume. One of the most common approaches in literature is using machine learning techniques, to automatically learn models and patterns behind such complexity, and to develop technologies to keep pace with malware evolution. This survey aims at providing an overview on the way machine learning has been used so far in the context of malware analysis in Windows environments, i.e. for the analysis of Portable Executables. We systematize surveyed papers according to their objectives (i.e., the *expected output*), what information about malware they specifically use (i.e., the *features*), and what machine learning techniques they employ (i.e., what *algorithm* is used to process the input and produce the output). We also outline a number of issues and challenges, including those concerning the used *datasets*, and identify the main current topical trends and how to possibly advance them. In particular, we introduce the novel concept of *malware analysis economics*, regarding the study of existing trade-offs among key metrics, such as analysis accuracy and economical costs.

1. Introduction

Despite the significant improvement of cyber security mechanisms and their continuous evolution, malware are still among the most effective threats in the cyber space. Malware analysis applies techniques from several different fields, such as program analysis and network analysis, for the study of malicious samples to develop a deeper understanding on several aspects, including their behaviour and how they evolve over time. Within the unceasing arms race between malware developers and analysts, each advance in security technology is usually promptly followed by a corresponding evasion. Part of the effectiveness of novel defensive measures depends on what properties they leverage on. For example, a detection rule based on the MD5 hash of a known malware can be easily eluded by applying standard techniques like obfuscation, or more advanced approaches such as *polymorphism* or *metamorphism*. For a comprehensive review of these techniques, refer to Ye et al. (2017). These methods change the binary of the malware, and thus its hash, but leave its behaviour unmodified. On the other side, developing detection rules that capture the semantics of a malicious sample is much more difficult to circumvent, because malware developers should apply more complex modifications. A major goal of malware analysis is to capture additional properties to be used to improve security measures and make evasion as hard as possible. Machine learning is a natural choice to support such a process of knowledge extraction. Indeed, many works in literature have taken this direction, with a variety of approaches, objectives and results.

* Corresponding author.
E-mail addresses: ucci@diag.uniroma1.it (D. Ucci), l.aniello@soton.ac.uk (L. Aniello), baldoni@diag.uniroma1.it (R. Baldoni).
https://doi.org/10.1016/j.cose.2018.11.001

This survey aims at reviewing and systematising existing literature where machine learning is used to support malware analysis of Windows executables, i.e. Portable Executables (PEs). The intended audience of this survey includes any security analysts, i.e. security-minded reverse engineer or software developer, who may benefit from applying machine learning to automate part of malware analysis operations and make the workload more tractable. Although mobile malware represents an ever growing threat, Windows largely remains the preferred target (AV-TEST, 2017) among all the existing platforms. Malware analysis techniques for PEs are slightly different from those for Android apps because there are significant dissimilarities on how operating system and applications work. As a matter of fact, literature papers on malware analysis commonly point out what specific platform they target, so we specifically focus on works that consider the analysis of PEs. 64 recent papers have been selected on the basis of their bibliographic significance, reviewed and systematised according to a taxonomy with three fundamental dimensions: (i) the specific *objective* of the analysis, (ii) what types of *features* extracted from PEs they consider and (iii) what machine learning *algorithms* they use. We distinguish three main objectives: *malware detection, malware similarity analysis* and *malware category detection*. PE features have been grouped in eight types: *byte sequences, APIs/system calls, opcodes, network, file system, CPU registers, PE file characteristics* and *strings*. Machine learning algorithms have been categorized depending on whether the learning is *supervised, unsupervised* or *semi-supervised*. The characterisation of surveyed papers according to such taxonomy allows to spot research directions that have not been investigated yet, such as the impact of particular combination of features on analysis accuracy. The analysis of such a large literature leads to single out three main issues to address. The first concerns overcoming modern anti-analysis techniques such as encryption. The second regards the inaccuracy of malware behaviour modelling due to the choice of what operations of the sample are considered for the analysis. The third is about the obsolescence and unavailability of the datasets used in the evaluation, which affect the significance of obtained results and their reproducibility. In this respect, we propose a few guidelines to prepare suitable benchmarks for malware analysis through machine learning. We also identify a number of topical trends that we consider worth to be investigated more in detail, such as malware attribution and triage. Furthermore, we introduce the novel concept of *malware analysis economics*, regarding the existing trade-offs between analysis accuracy, time and cost, which should be taken into account when designing a malware analysis environment.

The novel contributions of this work are

- the definition of a taxonomy to synthesise the state of the art on machine learning for malware analysis of PEs;
- a detailed comparative analysis of existing literature on that topic, structured according to the proposed taxonomy, which highlights possible new research directions;
- the determination of present main issues and challenges on that subject, and the proposal of high-level directions to investigate to overcome them;
- the identification of a number of topical trends on machine learning for malware analysis of PEs, with general guidelines on how to advance them;
- the definition of the novel concept of malware analysis economics.

The rest of the paper is structured as follows. Related work are described in Section 2. Section 3 presents the taxonomy we propose to organise reviewed malware analysis approaches based on machine learning, which are then characterised according to such a taxonomy in Section 4. From this characterisation, current issues and challenges are pointed out in Section 5. Section 6 highlights topical trends and how to advance them. Malware analysis economics is introduced in Section 7. Finally, conclusions and future works are presented in Section 8.

2. Related work

Other academic works have already addressed the problem of surveying contributions on the usage of machine learning techniques for malware analysis. The survey written by Shabtai et al. (2009) is the first one on this topic. It specifically deals with how classifiers are used on static features to detect malware. As most of the other surveys mentioned in this section, the main difference with our work is that our scope is wider as we target other objectives besides malware detection, such as similarities analysis and category detection. Furthermore, a novel contribution we provide is the idea of malware economics, which is not mentioned by any related work. Also in Sahu et al. (2014), the authors provide a comparative study on papers using pattern matching to detect malware, by reporting their advantages, disadvantages and problems. Souri and Hosseini (2018) proposes a taxonomy of malware detection approaches based on machine learning. In addition to consider detection only, their work differs from ours because they do not investigate what features are taken into account. LeDoux and Lakhotia (2015) describe how machine learning is used for malware analysis, whose end goal is defined there as "automatically detect malware as soon as possible, remove it, and repair any damage it has done".

Bazrafshan et al. (2013) focus on malware detection and identify three main methods for detecting malicious software, i.e. based on signatures, behaviours and heuristics, the latter using also machine learning techniques. They also identify what classes of features are used by reviewed heuristics for malware detection, i.e. API calls, control flow graphs, n-grams, opcodes and hybrid features. In addition to going beyond malware detection, we propose a larger number of feature types, which reflects the wider breadth of our research.

Basu (2016) examines different works relying on data mining and machine learning techniques for the detection of malware. They identify five types of features: API call graph, byte sequence, PE header and sections, opcode sequence frequency and kernel, i.e. system calls. In our survey we establish more feature types, such as strings, file system and CPU registers. They also compare surveyed papers by used features, used dataset and mining method.

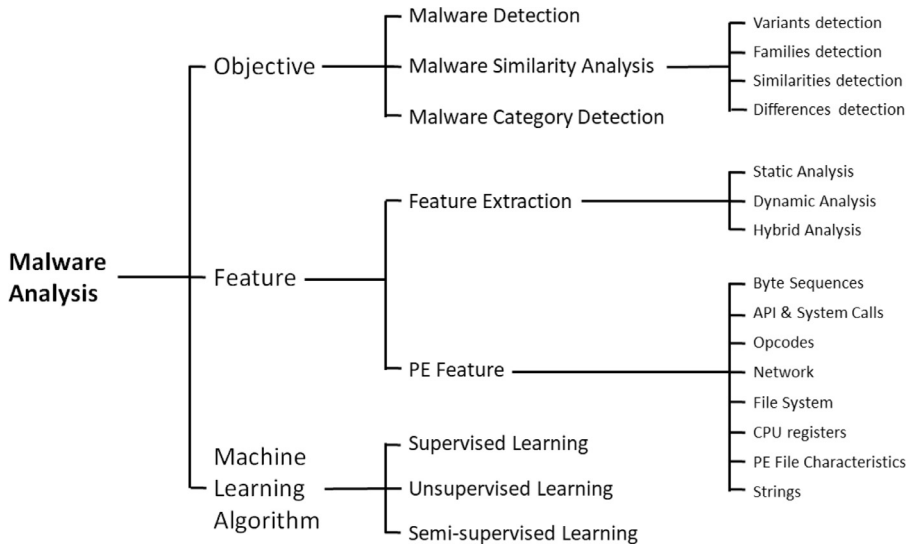

Fig. 1 – Taxonomy of machine learning techniques for malware analysis.

Ye et al. (2017) examine different aspects of malware detection processes, focusing on feature extraction/selection and classification/clustering algorithms. Also in this case, our survey looks at a larger range of papers by also including many works on similarity analysis and category detection. They also highlight a number of issues, mainly dealing with machine learning aspects (i.e. incremental learning, active learning and adversarial learning). We instead look at current issues and limitations from a distinct angle, indeed coming to a different set of identified problems that complement theirs. Furthermore, they outline several trends on malware development, while we rather report on trends about machine learning for malware analysis, again complementing their contributions.

Barriga and Yoo (2017) briefly survey literature on malware detection and malware evasion techniques, to discuss how machine learning can be used by malware to bypass current detection mechanisms. Our survey focuses instead on how machine learning can support malware analysis, even when evasion techniques are used. Gardiner and Nagaraja (2016) concentrate their survey on the detection of command and control centres through machine learning.

3. Taxonomy of machine learning techniques for malware analysis

This section introduces the taxonomy on how machine learning is used for malware analysis in the reviewed papers. We identify three major dimensions along which surveyed works can be conveniently organised. The first one characterises the final *objective* of the analysis, e.g. malware detection. The second dimension describes the *features* that the analysis is based on in terms of how they are extracted, e.g. through dynamic analysis, and what features are considered, e.g. CPU registers. Finally, the third dimension defines what type of *machine learning algorithm* is used for the analysis, e.g. supervised learning.

Fig. 1 shows a graphical representation of the taxonomy. The rest of this section is structured according to the taxonomy. Section 3.1 describes in details the *objective* dimension, *features* are pointed out in Section 3.2 and machine learning algorithms are reported in Section 3.3.

3.1. *Malware analysis objectives*

Malware analysis, in general, demands for strong detection capabilities to find matches with the knowledge developed by investigating past samples. Anyway, the final goal of searching for those matches differs. For example, a malware analyst may be specifically interested in determining whether new suspicious samples are malicious or not, while another may be rather inspecting new malware looking for what family they likely belong to. This section details the analysis goals of the surveyed papers, organized in three main objectives: malware detection (Section 3.1.1), malware similarity analysis (Section 3.1.2) and malware category detection (Section 3.1.3).

3.1.1. *Malware detection*

The most common objective in the context of malware analysis is detecting whether a given sample is malicious. This objective is also the most important because knowing in advance that a sample is dangerous allows to block it before it becomes harmful. Indeed, the majority of reviewed works has this as main goal (Ahmadi et al., 2015; Ahmed et al., 2009; Anderson et al., 2011; 2012; Bai et al., 2014; Chau et al., 2010; Chen et al., 2015; Elhadi et al., 2015; Eskandari et al., 2013; Feng et al., 2015; Firdausi et al., 2010; Ghiasi et al., 2015; Kolter and Maloof, 2006; Kruczkowski and Szynkiewicz, 2014; Kwon et al., 2015; Mao et al., 2015; Raff and Nicholas, 2017; Santos et al., 2013b; 2011; Saxe and Berlin, 2015; Schultz et al., 2001; Tamersoy et al., 2014; Uppal et al., 2014; Vadrevu et al., 2013; Wüchner et al., 2015; Yonts, 2012). Depending on what machine learning technique is used, the generated output can be provided with a confidence value that can be used by analysts to understand if a sample needs further inspection.

3.1.2. *Malware similarity analysis*

Another relevant objective is spotting similarities among malware, for example to understand how novel samples differ from previous, known ones. We find four slightly different versions of this objective: *variants detection*, *families detection*, *similarities detection* and *differences detection*.

Variants detection. Developing variants is one of the most effective and cheapest strategies for an attacker to evade detection mechanisms, while reusing as much as possible already available codes and resources. Recognizing that a sample is actually a variant of a known malware prevents such strategy to succeed, and paves the way to understand how malware evolve over time through the development of new variants. Also this objective has been deeply studied in literature, and several reviewed papers target the detection of variants. Given a malicious sample m, variants detection consists in selecting from the available knowledge base the samples that are variants of m (Gharacheh et al., 2015; Ghiasi et al., 2015; Khodamoradi et al., 2015; Liang et al., 2016; Upchurch and Zhou, 2015; Vadrevu and Perdisci, 2016). Considering the huge number of malicious samples received daily from major security firms, recognising variants of already known malware is crucial to reduce the workload for human analysts.

Families detection. Given a malicious sample m, families detection consists in selecting from the available knowledge base the families that m likely belongs to Lee and Mody (2006), Huang et al. (2009), Park et al. (2010), Ye et al. (2010), Dahl et al. (2013), Hu et al. (2013), Islam et al. (2013), Kong and Yan (2013), Nari and Ghorbani (2013), Ahmadi et al. (2015), Kawaguchi and Omote (2015), Lin et al. (2015), Mohaisen et al. (2015), Pai et al. (2015) and Raff and Nicholas (2017). In this way, it is possible to associate unknown samples to already known families and, by consequence, provide an added-value information for further analyses.

Similarities detection. Analysts can be interested in identifying the specific similarities and differences of the binaries to analyse with respect to those already analysed. Similarities detection consists in discovering what parts and aspects of a sample are similar to something that has been already examined in the past. It enables to focus on what is really new, and hence to discard the rest as it does not deserve further investigation (Bailey et al., 2007; Bayer et al., 2009; Egele et al., 2014; Palahan et al., 2013; Rieck et al., 2011).

Differences detection . As a complement, also identifying what is different from everything else already observed in the past results worthwhile. As a matter of fact, differences can guide towards discovering novel aspects that should be analysed more in depth (Bayer et al., 2009; Lindorfer et al., 2011; Palahan et al., 2013; Polino et al., 2015; Rieck et al., 2011; Santos et al., 2013a).

3.1.3. *Malware category detection*

Malware can be categorized according to their prominent behaviours and objectives. They can be interested in spying on users' activities and stealing their sensitive information (i.e., *spyware*), encrypting documents and asking for a ransom (i.e., *ransomware*), or gaining remote control of an infected machine (i.e., *remote access toolkits*). Using these categories is a coarse-grained yet significant way of describing malicious samples (Attaluri et al., 2009; Chen et al., 2012; Comar et al., 2013; Kwon et al., 2015; Sexton et al., 2015; Wong and Stamp, 2006). Although cyber security firms have not still agreed upon a standardized taxonomy of malware categories, effectively recognising the categories of a sample can add valuable information for the analysis.

3.2. *Malware analysis features*

This section deals with the features of samples that are considered for the analysis. How features are extracted from executables is reported in Section 3.2.1, while Section 3.2.2 details which specific features are taken into account.

3.2.1. *Feature extraction*

The information extraction process is performed through either static or dynamic analysis, or a combination of both, while examination and correlation are carried out by using machine learning techniques. Approaches based on static analysis look at the content of samples without requiring their execution, while dynamic analysis works by running samples to examine their behaviour. Several techniques can be used for dynamic malware analysis. *Debuggers* are used for instruction level analysis. *Simulators* model and show a behaviour similar to the environment expected by the malware, while *emulators* replicate the behaviour of a system with higher accuracy but require more resources. Sandboxes are virtualised operating systems providing an isolated and reliable environment where to detonate malware. Refer to Ye et al. (2017) for a more detailed description of these techniques. Execution traces are commonly used to extract features when dynamic analysis is employed. Reviewed articles generate execution traces by using either sandboxes (Anderson et al., 2011; Bayer et al., 2009; Firdausi et al., 2010; Graziano et al., 2015; Kawaguchi and Omote, 2015; Lee and Mody, 2006; Lin et al., 2015; Lindorfer et al., 2011; Mao et al., 2015; Palahan et al., 2013; Park and Jun, 2009; Rieck et al., 2011) or emulators (Asquith, 2015; Liang et al., 2016). Also program analysis tools and techniques can be useful in the feature extraction process by providing, for example, disassembly code and control- and data-flow graphs. An accurate disassembly code is important for obtaining correct *Byte sequences* and *Opcodes* features (Section 3.2.2), while control- and data-flow graphs can be employed in the extraction of *APIs and system calls* (Section 3.2.2). For an extensive dissertation on dynamic analyses, refer to Egele et al. (2012).

Among reviewed works, the majority relies on dynamic analyses (Anderson et al., 2011; Bailey et al., 2007; Bayer et al., 2009; Comar et al., 2013; Dahl et al., 2013; Elhadi et al., 2015; Firdausi et al., 2010; Ghiasi et al., 2015; Kawaguchi and Omote, 2015; Kruczkowski and Szynkiewicz, 2014; Lee and Mody, 2006; Liang et al., 2016; Lin et al., 2015; Lindorfer et al., 2011; Mohaisen et al., 2015; Nari and Ghorbani, 2013; Palahan et al., 2013; Park et al., 2010; Rieck et al., 2011; Uppal et al., 2014; Wüchner et al., 2015), while the others use, in equal proportions, either static analyses alone (Ahmadi et al., 2015; Attaluri

et al., 2009; Bai et al., 2014; Caliskan-Islam et al., 2015; Chen et al., 2015; 2012; Feng et al., 2015; Gharacheh et al., 2015; Hu et al., 2013; Khodamoradi et al., 2015; Kolter and Maloof, 2006; Kong and Yan, 2013; Pai et al., 2015; Santos et al., 2013a; 2011; Schultz et al., 2001; Sexton et al., 2015; Siddiqui et al., 2009; Srakaew et al., 2015; Tamersoy et al., 2014; Upchurch and Zhou, 2015; Vadrevu et al., 2013; Wong and Stamp, 2006; Yonts, 2012) or a combination of static and dynamic techniques (Anderson et al., 2012; Egele et al., 2014; Eskandari et al., 2013; Graziano et al., 2015; Islam et al., 2013; Jang et al., 2011; Polino et al., 2015; Santos et al., 2013b; Vadrevu and Perdisci, 2016). Depending on the specific features, extraction processes can be performed by applying either static, dynamic, or hybrid analysis.

3.2.2. *Portable executable features*

This section provides an overview on what features are used by reviewed papers to achieve the objectives outlined in section 3.1. In many cases, surveyed works only refer to macro-classes without mentioning the specific features they employed. As an example, when n-grams are used, only a minority of works mention the size of *n*.

Byte sequences. A binary can be characterised by computing features on its byte-level content. Analysing the specific sequences of bytes in a PE is a widely employed static technique. A few works use chunks of bytes of specific sizes (Raff and Nicholas, 2017; Schultz et al., 2001; Srakaew et al., 2015), while many others rely on n-grams (Ahmadi et al., 2015; Anderson et al., 2011; 2012; Chen et al., 2015; Dahl et al., 2013; Feng et al., 2015; Jang et al., 2011; Kolter and Maloof, 2006; Lin et al., 2015; Rieck et al., 2011; Sexton et al., 2015; Srakaew et al., 2015; Upchurch and Zhou, 2015; Uppal et al., 2014; Wüchner et al., 2015).

An n-gram is a sequence of *n* bytes, and features correspond to the different combination of these *n* bytes, namely each feature represents how many times a specific combination of *n* bytes occurs in the binary. The majority of works that specified the size of used n-grams relies on sequences no longer than 3 (i.e. trigrams) (Ahmadi et al., 2015; Anderson et al., 2011; 2012; Dahl et al., 2013; Islam et al., 2013; Lin et al., 2015; Sexton et al., 2015; Srakaew et al., 2015). Indeed, the number of features to consider grows exponentially with *n*.

Opcodes. Opcodes identify the machine-level operations executed by a PE, and can be extracted through static analyses by examining the assembly code (Ahmadi et al., 2015; Anderson et al., 2011; 2012; Attaluri et al., 2009; Gharacheh et al., 2015; Hu et al., 2013; Khodamoradi et al., 2015; Kong and Yan, 2013; Pai et al., 2015; Santos et al., 2013a; 2013b; Sexton et al., 2015; Srakaew et al., 2015; Wong and Stamp, 2006; Ye et al., 2010). Opcode frequency is one of the most commonly used feature. It measures the number of times each specific opcode appears within the assembly or is executed by a PE (Khodamoradi et al., 2015; Ye et al., 2010). Others (Anderson et al., 2012; Khodamoradi et al., 2015) count opcode occurrences by aggregating them by operation type, e.g., mathematical instructions, memory access instructions. Similarly to n-grams, also sequences of opcodes are used as features (Gharacheh et al., 2015; Khodamoradi et al., 2015; Srakaew et al., 2015; Ye et al., 2010).

API and system calls. Similarly to opcodes, APIs and system calls enable the analysis of samples' behaviour, but at a higher level. They can be either extracted statically or dynamically by analysing the disassembly code (to get the list of all calls that can be potentially executed) or the execution traces (for the list of calls actually invoked). While APIs allow to characterise what actions are executed by a sample (Ahmadi et al., 2015; Ahmed et al., 2009; Bai et al., 2014; Egele et al., 2014; Islam et al., 2013; Kawaguchi and Omote, 2015; Kong and Yan, 2013; Liang et al., 2016), looking at system call invocations provides a view on the interaction of the PE with the operating system (Anderson et al., 2012; Asquith, 2015; Bayer et al., 2009; Dahl et al., 2013; Egele et al., 2014; Elhadi et al., 2015; Lee and Mody, 2006; Mao et al., 2015; Palahan et al., 2013; Park et al., 2010; Rieck et al., 2011; Santos et al., 2013b; Uppal et al., 2014). Data extracted by observing APIs and system calls can be really large, and many works carry out additional processing to reduce feature space by using convenient data structures. One of the most popular data structures to represent PE behaviour and extract program structure is the control flow graph. This data structure allows compilers to produce an optimized version of the program itself and model control flow relationships (Allen, 1970). Several works employ control flow graphs and their extensions for sample analysis, in combination with other feature classes (Anderson et al., 2012; Eskandari et al., 2013; Graziano et al., 2015; Polino et al., 2015; Wüchner et al., 2015).

Network activity. A huge number of key information can be obtained by observing how the PE interacts with the network. Contacted addresses and generated traffic can unveil valuable aspects, e.g. regarding the communication with a command and control centre. Relevant features include statistics on used protocols, TCP/UDP ports, HTTP requests, DNS-level interactions. Many surveyed works require dynamic analysis to extract this kind of information (Bailey et al., 2007; Bayer et al., 2009; Graziano et al., 2015; Kwon et al., 2015; Lee and Mody, 2006; Liang et al., 2016; Lindorfer et al., 2011; Mohaisen et al., 2015; Nari and Ghorbani, 2013). Other papers extract network-related inputs by monitoring the network and analysing incoming and outgoing traffic (Comar et al., 2013; Kruczkowski and Szynkiewicz, 2014; Vadrevu and Perdisci, 2016). A complementary approach consists in analysing download patterns of network users in a monitored network (Vadrevu et al., 2013). It does not require sample execution and focuses on network features related to the download of a sample, such as the website from which the file has been downloaded.

File system. What file operations are executed by samples is fundamental to grasp evidence about the interaction with the environment and possibly detect attempts to gain persistence. Features of interest mainly concern how many files are read or modified, what types of files and in what directories, and which files appear in not-infected/infected machines (Bailey et al., 2007; Chau et al., 2010; Graziano et al., 2015; Kong and Yan, 2013; Lee and Mody, 2006; Lin et al., 2015; Mao et al., 2015;

Mohaisen et al., 2015). Sandboxes and memory analysis toolkits include modules for monitoring interactions with the file system, usually modelled by counting the number of files created/deleted/modified by the PE. In Mohaisen et al. (2015), the size of these files is considered as well, while Lin et al. leverage the number of created hidden files (Lin et al., 2015).

A particularly relevant type of file system features are those extracted from the *Windows Registry*. The registry is one of the main sources of information for a PE about the environment, and also represents a fundamental tool to hook into the operating system, for example to gain persistence. Discovering what keys are queried, created, deleted and modified can shed light on many significant characteristics of a sample (Lee and Mody, 2006; Lin et al., 2015; Mao et al., 2015; Mohaisen et al., 2015). Usually, works relying on file system inputs monitor also the Windows Registry.

CPU registers. The way CPU registers are used can also be a valuable indication, including whether any hidden register is used, and what values are stored in the registers, especially in the FLAGS register (Ahmadi et al., 2015; Egele et al., 2014; Ghiasi et al., 2015; Kong and Yan, 2013).

PE file characteristics. A static analysis of a PE can provide a large set of valuable information such as sections, imports, symbols, used compilers (Asquith, 2015; Bai et al., 2014; Kirat et al., 2013; Lee and Mody, 2006; Saxe and Berlin, 2015; Yonts, 2012).

Strings. A PE can be statically inspected to explicitly look for the strings it contains, such as code fragments, author signatures, file names, system resource information (Ahmadi et al., 2015; Islam et al., 2013; Saxe and Berlin, 2015; Schultz et al., 2001).

3.3. *Malware analysis algorithms*

This ection reports what machine learning algorithms are used in surveyed works by organising them on the basis of whether the learning is supervised (Section 3.3.1), unsupervised (Section 3.3.2) or semi-supervised (Section 3.3.3).

3.3.1. *Supervised learning*

Supervised learning is the task of gaining knowledge by providing statistical models with correct instance examples, during a preliminary phase called training. The supervised algorithms used by reviewed papers are *rule-based classifier* (Ahmed et al., 2009; Feng et al., 2015; Ghiasi et al., 2015; Liang et al., 2016; Lindorfer et al., 2011; Schultz et al., 2001; Sexton et al., 2015; Tian et al., 2008), *Bayes classifier* (Kawaguchi and Omote, 2015; Santos et al., 2013a; 2013b; Uppal et al., 2014; Wüchner et al., 2015), *Naïve Bayes* (Firdausi et al., 2010; Kawaguchi and Omote, 2015; Kolter and Maloof, 2006; Schultz et al., 2001; Sexton et al., 2015; Uppal et al., 2014; Wüchner et al., 2015), *Bayesian Network* (Eskandari et al., 2013; Santos et al., 2013a; 2013b), *Support Vector Machine* (SVM) (Ahmadi et al., 2015; Ahmed et al., 2009; Anderson et al., 2011; Chen et al., 2012; Comar et al., 2013; Feng et al., 2015; Firdausi et al., 2010; Islam et al., 2013; Kawaguchi and Omote, 2015; Kolter and Maloof, 2006; Kong and Yan, 2013; Kruczkowski and Szynkiewicz, 2014; Lin et al., 2015; Mohaisen et al., 2015; Santos et al., 2013a; 2013b; Sexton et al., 2015; Uppal et al., 2014; Wüchner et al., 2015), *Multiple Kernel Learning* (Anderson et al., 2012), *Prototype-based Classification* (Rieck et al., 2011), *Decision Tree* (Ahmed et al., 2009; Bai et al., 2014; Firdausi et al., 2010; Islam et al., 2013; Kawaguchi and Omote, 2015; Khodamoradi et al., 2015; Kolter and Maloof, 2006; Mohaisen et al., 2015; Nari and Ghorbani, 2013; Santos et al., 2013a; 2013b; Srakaew et al., 2015; Uppal et al., 2014), *Random Forest* (Ahmadi et al., 2015; Comar et al., 2013; Islam et al., 2013; Kawaguchi and Omote, 2015; Khodamoradi et al., 2015; Kwon et al., 2015; Mao et al., 2015; Siddiqui et al., 2009; Uppal et al., 2014; Wüchner et al., 2015), *Gradient Boosting Decision Tree* (Chen et al., 2012; Sexton et al., 2015), *Logistic Model Tree* (Dahl et al., 2013; Graziano et al., 2015; Palahan et al., 2013; Sexton et al., 2015), *k-Nearest Neighbors* (k-NN) (Ahmed et al., 2009; Firdausi et al., 2010; Islam et al., 2013; Kawaguchi and Omote, 2015; Kong and Yan, 2013; Lee and Mody, 2006; Mohaisen et al., 2015; Raff and Nicholas, 2017; Santos et al., 2013a), *Artificial Neural Network* (Dahl et al., 2013; Saxe and Berlin, 2015), *Multilayer Perceptron Neural Network* (Firdausi et al., 2010).

3.3.2. *Unsupervised learning*

Unsupervised approaches do not rely on any training phase and learn directly from unlabeled data. Reviewed papers use these unsupervised learning algorithms: *Clustering with locality sensitive hashing* (Bayer et al., 2009; Tamersoy et al., 2014; Upchurch and Zhou, 2015), *Clustering with Distance and Similarity Metrics* (using either Euclidean (Mohaisen et al., 2015; Rieck et al., 2011) or Hamming distances (Mohaisen et al., 2015), or cosine (Mohaisen et al., 2015) or Jaccard similarities (Mohaisen et al., 2015; Polino et al., 2015)), *Expectation Maximization* (Pai et al., 2015), *k-Means Clustering* (Huang et al., 2009; Pai et al., 2015), *k-Medoids* (Ye et al., 2010), *Density-based Spatial Clustering of Applications with Noise* (Vadrevu and Perdisci, 2016), *Hierarchical Clustering* (Jang et al., 2011; Mohaisen et al., 2015), *Prototype-based Clustering* (Rieck et al., 2011), *Self-Organizing Maps* (Chen et al., 2012).

3.3.3. *Semi-supervised learning*

Semi-supervised learning combines both labeled and unlabeled data for feeding statistical models to acquire knowledge. *Learning with Local and Global Consistency* is used in Santos et al. (2011) while *Belief Propagation* in Chau et al. (2010), Tamersoy et al. (2014) and Chen et al. (2015).

4. Characterization of surveyed papers

In this section we characterize each reviewed paper on the basis of analysis objective, used machine learning algorithm and features. Several details are also reported on the dataset used for the evaluation, including whether it is publicly available (*Public* column), where samples have been collected from (*Source* column) and whether the specific set of samples considered for the experiment is available (*Available* column). Indeed, many works declare they do not use all the executables in the dataset but they do not specify what samples they choose, which prevents to reproduce their results. The *Label* column states how samples have been labelled. Finally, *Benign,*

Malicious and *Total* columns report benign executables count, malware count and their sum, respectively.

Malware detection. Table 1 lists all the reviewed works having malware detection as objective. Most used features are byte sequences and API/system call invocations, derived by executing the samples. Most of the works use more than one algorithm to find out the one guaranteeing more accurate results.

Malware similarity analysis. A table is provided for each version of this objective (Section 3.1.2). Tables 2 and 3 describe the works dealing with variants detection and families detection, respectively. For both, APIs and system calls are largely used, as well as malware interactions with the environment, i.e. memory, file system, and CPU registers. Tables 4 and 5 report the papers on similarities and differences detection, respectively. All the analysed papers but (Santos et al., 2013a) rely on APIs and system calls collection. Works on differences detection, in general, do not take into account the interactions with the hosting system, while those on similarities detection do.

Malware category detection. These articles focus on the identification of specific threats and, thus, on particular features such as byte sequences, opcodes, function lengths and network activity. Table 6 reports the works whose objective is the detection of malware category.

By reasoning on what algorithms and features have been used and what have not for specific objectives, the provided characterisation allows to easily identify gaps in the literature and, thus, possible research directions to investigate. For instance, all works on *differences detection* (see Table 5) but (Santos et al., 2013a), rely on dynamically extracted APIs and system calls for building their machine learning models. Novel approaches can be explored by taking into account other features that capture malware interactions with the environment (e.g., memory, file system, CPU registers and Windows Registry).

5. Issues and challenges

Based on the characterization detailed in Section 4, this section identifies the main issues and challenges of surveyed papers. In the specific, the main problems regard the usage of anti-analysis techniques by malware (Section 5.1), what operation set to consider (Section 5.2) and used dataset (Section 5.3).

5.1. *Anti-analysis techniques*

Malware developers want to avoid their samples to be analysed, so they devise and refine several *anti-analysis techniques* that are effective in hindering the reverse engineering of executables. Indeed many surveyed works claim that the solution they propose does not work or loses in accuracy when samples using such techniques are considered (Section 4).

Static analysis (Section 3.2.1) is commonly prevented by rendering sample binary and resources unreadable through *obfuscation, packing* or *encryption*. Anyway, at runtime, code and any other concealed data has to be either deobfuscated, unpacked or decrypted to enable the correct execution of the payload. This implies that such a kind of anti-analysis techniques can be overcome by using dynamic analysis (Section 3.2.1) to make the sample unveil hidden information and load them in memory, where they can then be extracted by creating a dump. Refer to Ye et al. (2017) for a detailed disquisition on how obfuscation, packing and encryption are used by malware developers.

More advanced anti-analysis techniques exist to keep malware internals secret even when dynamic analysis is used. One approach, commonly referred to as *environmental awareness*, consists in the malware trying to detect whether it is being executed in a controlled setting where an analyst is trying to dissect it, for example by using a virtual machine or by running the sample in debug mode. If any cue is found of possibly being under analysis, then the malware does not execute its malicious payload. Miramirkhani et al. (2017) show that a malware can easily understand if it is running into an artificial environment. Other approaches rely on *timing-based evasion*, i.e. they only show their malicious behaviour at predetermined dates and times. Other malware instead require or wait for some *user interaction* to start their intended activity, in order to make any kind of automatic analysis infeasible.

Identifying and overcoming these anti-analysis techniques is an important direction to investigate to improve the effectiveness of malware analysis. Recent academic and not-academic literature are aligned on this aspect. Karpin and Dorfman (2017) highlight the need to address very current problems such as discovering where malware configuration files are stored and whether standard or custom obfuscation/packing/encryption algorithms are employed. Deobfuscation (Blazytko et al., 2017; Kotov and Wojnowicz, 2018) and other operations aimed at supporting binary reverse engineering, such as function similarity identification (Liao et al., 2018), are still very active research directions. Symbolic execution techniques (Baldoni et al., 2018) are promising means to understand what execution paths trigger the launch of the malicious payload.

5.2. *Operation set*

Opcodes, instructions, APIs and system calls (hereinafter, we refer to them in general as *operations*) are the most used and powerful features employed for malware analysis (Section 4), as they allow to directly and accurately model sample behaviour. Normally, to reduce complexity and required computational power, only a subset of all the available operations is considered. This choice of ignoring part of the operations at disposal can reduce the accuracy of malware behaviour model, which in turn reflects on the reliability of analysis outcomes. This issue has been raised explicitly in some surveyed papers, including Anderson et al. (2012), Gharacheh et al. (2015) and Khodamoradi et al. (2015) while others are anyway affected although they do not mention it, such as Ghiasi et al. (2015), Liang et al. (2016) and Huang et al. (2009).

On one hand, this challenge can be addressed by either improving or using different machine learning techniques to achieve a more effective feature selection. On the other hand, program analysis advances can be leveraged to enhance the accuracy of disassemblers and decompilers, indeed these tools are known to be error-prone (Guilfanov, 2008; Rosseau

Table 1 – Characterization of surveyed papers having malware detection as objective.

Paper	Algorithms	Features	Limitations	Dataset samples						
				Public	Source	Available	Labeling	Benign	Malicious	Total
Schultz et al. (2001)	Rule-based classifier, Naïve Bayes	Strings and byte sequences	Proposed solutions are not effective against encrypted executables and the dataset used in the evaluations is small.	-	FTP sites		Automated	1,001	3,265	4,266
Kolter and Maloof (2006)	Decision Tree, Naïve Bayes, SVM	Byte sequences	Payload classification fails in presence of binary obfuscation and the dataset used in the evaluations is very small.	✗	Internet, VX Heavens, and MITRE	✗	Automated	1,971	1,651	3,622
Ahmed et al. (2009)	Decision Tree, Naïve Bayes, SVM	APIs/System calls	The dataset used in the experimental evaluations is very small.		Legitimate apps and VX Heavens	✗	-	100	416	516
Chau et al. (2010)	Belief propagation	File system	Rare and new files cannot be accurately classified as benign or malicious.		Symantec's Norton Community Watch	✗	-	?	?	$903 \cdot 10^6$
Firdausi et al. (2010)	Decision Tree, Naïve Bayes, SVM, k-NN, Multilayer Perceptron Neural Network	APIs/System calls, file system, and Windows Registry	The dataset used in the experimental evaluations is very small.		Windows XP SP2	✗	-	250	220	470
Anderson et al. (2011)	SVM	Byte sequences and APIs/system calls	The dataset used in the evaluations is small.	-	-	✗	-	615	1,615	2,230
Santos et al. (2011)	Learning with Local and Global Consistency	Byte sequences	Proposed approach is not effective against packed malware and requires manual labeling of a portion of the small dataset. In particular, the dataset used in the experimental evaluations is small.		Own machines and VX Heavens	✗	-	1,000	1,000	2,000
Anderson et al. (2012)	Multiple Kernel Learning	Byte sequences, opcodes, and APIs/system calls	Instruction categorization is not optimal.	✗	Offensive Computing	✗	-	776	21,716	22,492
Yonts (2012)	Rule-based classifier	PE file characteristics	Only a subset of all the potential low level attributes is considered.	✗	SANS Institute	✗	-	65,000	$25 \cdot 10^5$	$25.65 \cdot 10^5$

(*continued on next page*)

Table 1 (*continued*)

Paper	Algorithms	Features	Limitations	Dataset samples						
				Public	Source	Available	Labeling	Benign	Malicious	Total
Eskandari et al. (2013)	Bayesian Network	APIs/System calls	Ignore specific instructions. Evasion/obfuscation techniques and samples requiring user interactions reduce the effectiveness of the proposed approach. The dataset is small.	✗	Research Laboratory at Shiraz University	✗	-	1,000	2,000	3,000
Santos et al. (2013b)	Bayesian Network, Decision Tree, *k*-NN, SVM	Opcodes, APIs/system calls, and raised exceptions	Packed malware, evasion techniques, and samples requiring user interactions reduce the accuracy of the proposed solution. The dataset is small.		Own machines and VX Heavens	✗	-	1,000	1,000	2,000
Vadrevu et al. (2013)	Random Forest	PE file characteristics and network	Requires a huge number of samples labeled either as malicious or benign.	✗	Georgia Institute of Technology	✗	Automated	170,780	15,182	185,962
Bai et al. (2014)	Decision Tree, Random Forest	PE file characteristics	Assume that samples are not packed and malware authors can properly modify PE header to remain undetected.		Windows and Program Files folders and VXHeavens	✗	Automated	8,592	10,521	19,113
Kruczkowski and Szynkiewicz (2014)	SVM	Network	The dataset used in the experimental evaluations is small.	✗	N6 Platform	✗	-	?	?	1,015
Tamersoy et al. (2014)	Clustering with locality sensitive hashing	File system	Rare and new files cannot be accurately classified as benign or malicious.		Symantec's Norton Community Watch	✗	-	1,663,506	47,956	4,970,865
Uppal et al. (2014)	Decision Tree, Random Forest, Naïve Bayes, SVM	Byte sequences and APIs/system calls	The dataset used in the experimental evaluations is very small.		Legitimate apps and VX Heavens	✗	-	150	120	270
Chen et al. (2015)	Belief propagation	File system	Rare and new files cannot be accurately classified as benign or malicious.	✗	Comodo Cloud Security Center	✗	-	19,142	2,883	69,165
Elhadi et al. (2015)	Malicious graph matching	APIs/System calls	The dataset used in the experimental evaluations is extremely small.		VX Heavens		-	10	75	85

(*continued on next page*)

Table 1 (*continued*)

Paper	Algorithms	Features	Limitations	Dataset samples						
				Public	Source	Available	Labeling	Benign	Malicious	Total
Feng et al. (2015)	Rule-based classifier, SVM	Byte sequences	Only specific malware classes are considered for the approach evaluation.	✗	Windows system files and own AV platform	✗	-	100,000	135,064	235,064
Ghiasi et al. (2015)	Rule-based classifier	APIs/System calls and CPU registers	APIs/System calls categorization could be not optimal and the dataset size is small.		Windows XP system and Program Files folders and private repository	✗	-	390	850	1,240
Kwon et al. (2015)	Random Forest	Network	Not able to detect bots with rootkit capabilities.		Symantec's Worldwide Intelligence Network Environment	✗	-	?	?	$24*10^6$
Mao et al. (2015)	Random Forest	APIs/System calls, file system, and Windows Registry	Evasion techniques and samples requiring user interactions reduce the accuracy of the proposed approach. The dataset is small.		Windows XP SP3 and VX Heavens	✗	-	534	7,257	7,791
Saxe and Berlin (2015)	Neural Networks	Strings and PE file characteristics	Label assigned to training set may be inaccurate and the accuracy of the proposed approach decreases substantially when samples are obfuscated.	✗	Legitimate apps and own malware database	✗	Automated	81,910	350,016	431,926
Srakaew et al. (2015)	Decision Tree	Byte sequences and opcodes	Obfuscation techniques reduce detection accuracy.	✗	Legitimate files and apps and CWSandbox	✗	-	600	3,851	69,165
Wüchner et al. (2015)	Naïve Bayes, Random Forest, SVM	Byte sequences, APIs/system calls, file system, and Windows Registry	Obfuscation techniques applied by the authors may not reflect the ones of real-world samples. The dataset is small.	✗	Legitimate app downloads and Malicia	✗	-	513	6,994	7,507
Raff and Nicholas (2017)	k-NN with Lempel-Ziv Jaccard distance	Byte sequences	Obfuscation techniques reduce detection accuracy.	✗	Industry partner	✗	-	240,000	237,349	477,349

Table 2 – Characterization of surveyed papers having malware variants selection as objective. [1]Instead of using machine learning techniques, Gharacheh et al. rely on Hidden Markov Models to detect variants of the same malicious sample Gharacheh et al. (2015).

Paper	Algorithms	Features	Limitations	Dataset samples						
				Public	Source	Available	Labeling	Benign	Malicious	Total
Gharacheh et al. (2015)	-[1]	Opcodes	Opcode sequence is not optimal and the dataset size is very small.		Cygwin and VX Heavens	✗	-	?	?	740
Ghiasi et al. (2015)	Rule-based classifier	APIs/System calls and CPU registers	APIs/System calls categorization could be not optimal and the dataset size is small.		Windows XP system and Program Files folders and private repository	✗	-	390	850	1,240
Khodamoradi et al. (2015)	Decision Tree, Random Forest	Opcodes	Opcode sequence is not optimal and the dataset size is very small.		Windows XP system and Program Files folders and self-generated metamorphic malware	✗	-	550	280	830
Upchurch and Zhou (2015)	Clustering with locality sensitive hashing	Byte sequences	The dataset size is extremely small.	✗	Sampled from security incidents		Manual	0	85	85
Liang et al. (2016)	Rule-based classifier	APIs/System calls, file system, Windows Registry, and network	Monitored API/system call set could be not optimal and the dataset size is small.	✗	Anubis website	✗	-	0	330,248	330,248
Vadrevu and Perdisci (2016)	DBSCAN clustering	APIs/System calls, PE file characteristics, and network	Evasion techniques and samples requiring user interactions reduce the accuracy of the proposed approach.	✗	Security company and large Research Institute	✗	-	0	1,651,906	1,651,906

Table 3 – Characterization of surveyed papers having malware families selection as objective. [2]Asquith describes aggregation overlay graphs for storing PE metadata, without further discussing any machine learning technique that could be applied on top of these new data structures.

Paper	Algorithms	Features	Limitations	Dataset samples						
				Public	Source	Available	Labeling	Benign	Malicious	Total
Huang et al. (2009)	*k*-Means-like algorithm	Byte sequences	Instruction sequence categorization could be not optimal and the dataset size is small.	✗	Kingsoft Corporation	✗	-	0	2,029	2,029
Park et al. (2010)	Malicious graph matching	APIs/System calls	Approach vulnerable to APIs/system calls injection and the dataset used in the experimental evaluations is very small.	✗	Legitimate apps and Anubis Sandbox	✗	Automated	80	300	380
Ye et al. (2010)	*k*-Medoids variants	Opcodes	Instruction categorization could be not optimal.	✗	Kingsoft Corporation	✗	-	0	11,713	11,713
Dahl et al. (2013)	Logistic Regression, Neural Networks	Byte sequences and APIs/system calls	The authors obtain a high two-class error rate.	✗	Microsoft	✗	Mostly manual	817,485	1,843,359	3,760,844
Hu et al. (2013)	Prototype-based clustering	Opcodes	Obfuscation techniques reduce the effectiveness of their prototype for malware family selection.	✗	-	✗	Manual and automated	0	137,055	137,055
Islam et al. (2013)	Decision Tree, *k*-NN, Random Forest, SVM	Strings, byte sequences and APIs/system calls	The proposed approach is less effective novel samples. The dataset is small.	✗	CA Labs	✗	-	51	2,398	2,939
Kong and Yan (2013)	SVM, *k*-NN	Opcodes, memory, file system, and CPU registers	Significant differences in samples belonging to the same family reduce the proposed approach accuracy.	✗	Offensive Computing	✗	Automated	0	526,179	526,179
Nari and Ghorbani (2013)	Decision Tree	Network	Network features are extracted by a commercial traffic analyzer. The dataset used in the experimental evaluations is small.	✗	Communication Research Center Canada	✗	Automated	0	3,768	3,768
Ahmadi et al. (2015)	SVM, Random Forest, Gradient Boosting Decision Tree	Byte sequences, opcodes, APIs/system calls, Windows Registry, CPU registers, and PE file characteristics	Selected features can be further reduced to have a clearer view of the reasons behind sample classification.		Microsoft's malware classification challenge	✗	-	0	21,741	21,741

(*continued on next page*)

Table 3 (*continued*)

Paper	Algorithms	Features	Limitations	Dataset samples						
				Public	Source	Available	Labeling	Benign	Malicious	Total
Asquith (2015)	-[2]	APIs/System calls, memory, file system, PE file characteristics, and raised exceptions	-	-	-	-	-	-	-	-
Lin et al. (2015)	SVM	Byte sequences, APIs/system calls, file system, and CPU registers	Selected API/system call set could be not optimal. Evasion techniques and samples requiring user interactions reduce the accuracy of the proposed approach. The dataset is small.	✗	Own sandbox	✗	-	389	3,899	4,288
Kawaguchi and Omote (2015)	Decision Tree, Random Forest, k-NN, Naïve Bayes	APIs/System calls	This classification approach can be easily evaded by real-world malware. The dataset used in the experimental evaluations is very small.	✗	FFRI Inc.	✗	-	236	408	644
Mohaisen et al. (2015)	Decision Tree, k-NN, SVM, Clustering with with different similarity measures, Hierarchical clustering	File system, Windows Registry, CPU registers, and network	Evasion techniques reduce the accuracy of the proposed solution.	✗	AMAL system	✗	Manual and automated	0	115,157	115,157
Pai et al. (2015)	k-Means, Expectation Maximization	Opcodes	Obfuscation techniques reduce the effectiveness of the employed approach. The dataset is small.	✗	Cygwin utility files and Malicia	✗	-	213	8,052	8,265
Raff and Nicholas (2017)	k-NN with Lempel-Ziv Jaccard distance	Byte sequences	Obfuscation techniques reduce detection accuracy.	✗	Industry partner	✗	-	240,000	237,349	477,349

Table 4 – Characterization of surveyed papers having malware similarities detection as objective. [3]SVM is used only for computing the optimal values of weight factors associated to each feature chosen to detect similarities among malicious samples.

Paper	Algorithms	Features	Limitations	Dataset samples						
				Public	Source	Available	Labeling	Benign	Malicious	Total
Bailey et al. (2007)	Hierarchical clustering with normalized compression distance	APIs/System calls, file system, Windows Registry, and network	Evasion techniques and samples requiring user interactions reduce the accuracy of the proposed classification method. The dataset is small.	✗	Albor Malware Library and public repository	✗	Automated	0	8,228	8,228
Bayer et al. (2009)	Clustering with locality sensitive hashing	APIs/System calls	Evasion techniques and samples requiring user interactions reduce the approach accuracy.	✗	Anubis website	✗	Manual and automated	0	75,692	75,692
Rieck et al. (2011)	Prototype-based classification and clustering with Euclidean distance	Byte sequences and APIs/system calls	Evasion techniques and samples requiring user interactions reduce the accuracy of the proposed framework.	✗	CWSandbox and Sunbelt Software	✗	Automated	0	36,831	36,831
Palahan et al. (2013)	Logistic Regression	APIs/System calls	Evasion techniques and samples requiring user interactions reduce the accuracy of the proposed framework, while unknown observed behaviors are classified as malicious. The dataset used in the experimental evaluations is very small.	✗	Own honeypot	✗	-	49	912	961
Egele et al. (2014)	SVM[3]	APIs/System calls, memory, and CPU registers	The accuracy of computed PE function similarities drops when different compiler toolchains or aggressive optimization levels are used. The dataset is small.		`coreutils-8.13` program suite	✗	-	1,140	0	1,140

Table 5 – Characterization of surveyed papers having malware differences detection as objective.

Paper	Algorithms	Features	Limitations	Dataset samples						
				Public	Source	Available	Labeling	Benign	Malicious	Total
Bayer et al. (2009)	Clustering with locality sensitive hashing	APIs/System calls	Evasion techniques and samples requiring user interactions reduce the approach accuracy.	✗	Anubis website	✗	Manual and automated	0	75,692	75,692
Lindorfer et al. (2011)	Rule-based classifier	APIs/System calls and network	Sophisticated evasion techniques and samples requiring user interactions can still bypass detection processes. The dataset used in the experimental evaluations is small.	✗	Anubis Sandbox	✗	Automated	0	1,871	1,871
Rieck et al. (2011)	Prototype-based classification and clustering with Euclidean distance	Byte sequences and APIs/system calls	Evasion techniques and samples requiring user interactions reduce the accuracy of the proposed framework.	✗	CWSandbox and Sunbelt Software	✗	Automated	0	36,831	36,831
Palahan et al. (2013)	Logistic Regression	APIs/System calls	Evasion techniques and samples requiring user interactions reduce the accuracy of the proposed framework, while unknown observed behaviors are classified as malicious. The dataset used in the experimental evaluations is small.	✗	Own honeypot	✗	-	49	912	961
Santos et al. (2013a)	Decision Tree, k-NN, Bayesian Network, Random Forest	Opcodes	Opcode sequence is not optimal and the dataset size is small. The proposed method is not effective against packed malware. The dataset is small.		Own machines and VXHeavens	✗	Automated	1,000	1,000	2,000
Polino et al. (2015)	Clustering with Jaccard similarity	APIs/System calls	Evasion techniques, packed malware, and samples requiring user interactions reduce the accuracy of the proposed framework. API calls sequence used to identify sample behaviors is not optimal. The dataset size is small.	-	-	-	-	?	?	2,136

Table 6 – Characterization of surveyed papers having malware category detection as objective. [4]Instead of using machine learning techniques, these articles rely on Hidden Markov Models to detect metamorphic viruses (Attaluri et al., 2009; Wong and Stamp, 2006).

Paper	Algorithms	Features	Limitations	Dataset samples						
				Public	Source	Available	Labeling	Benign	Malicious	Total
Wong and Stamp (2006)	-[4]	Opcodes	Detection fails if metamorphic malware are similar to benign files. The dataset is extremely small.	✗	Cygwin and VX Heavens generators	✗	-	40	25	65
Attaluri et al. (2009)	-[4]	Opcodes	The proposed approach is not effective against all types of metamorphic viruses. The dataset size is very small.	✗	Cygwin, legitimate DLLs and VX Heavens generators	✗	-	240	70	310
Tian et al. (2008)	Rule-based classifier	Function length	Function lengths alone are not sufficient to detect Trojans and the dataset used in the experimental evaluations is very small.	-	-	-	-	0	721	721
Siddiqui et al. (2009)	Decision Tree, Random Forest	Opcodes	Advanced packing techniques could reduce detection accuracy. The dataset used in the experimental evaluations is small.		Windows XP and VX Heavens	✗	-	1,444	1,330	2,774
Chen et al. (2012)	Random Forest, SVM	Byte sequences	The proposed framework heavily relies on security companies' encyclopedias.	✗	Trend Micro	✗	-	0	330,248	330,248
Comar et al. (2013)	Random Forest, SVM	Network	Network features are extracted by a commercial traffic analyzer.	✗	Internet Service Provider	✗	Manual and automated	212,505	4,394	216,899
Kwon et al. (2015)	Random Forest	Network	Not able to detect bots with rootkit capabilities.		Symantec's Worldwide Intelligence Network Environment	✗	-	?	?	$24*10^6$
Sexton et al. (2015)	Rule-based classifier, Logistic Regression, Naïve Bayes, SVM	Byte sequences and opcodes	Obfuscation techniques reduce detection accuracy. The dataset used in the experimental evaluations is small.	-	-	-	-	4,622	197	4,819

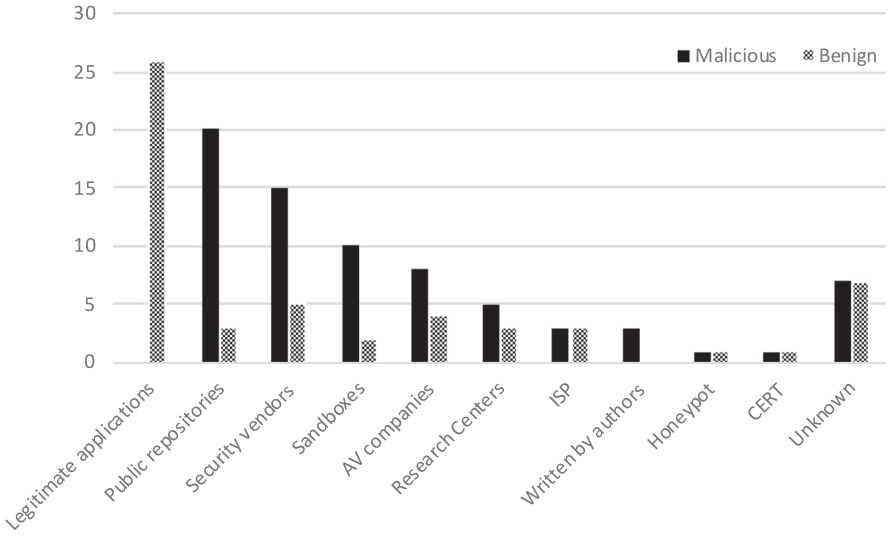

Fig. 2 – Frequency histogram showing how many reviewed papers use each type of source (e.g. public repositories, honeypot) to collect their datasets, and whether it is used to gather malware or benign samples.

and Seymour, 2018) and are thus likely to affect negatively the whole analyses. Approaches that improve the quality of generated disassembly and decompiled code, as in Schulte et al. (2018), can reduce the impact due to these errors.

5.3. *Datasets*

More than 72% of surveyed works use datasets with both malicious and benign samples, while about 28% rely on datasets with malware only. Just two works rely on benign datasets only (Caliskan-Islam et al., 2015; Egele et al., 2014), because their objectives are identifying sample similarities and attributing the ownership of some source codes under analysis, respectively.

Fig. 2 shows the dataset sources for malicious and benign samples. It is worth noting that most of benign datasets consists of legitimate applications (e.g. software contained in "Program Files" or "system" folders), while most of malware have been obtained from public repositories, security vendors and popular sandboxed analysis services. The most popular public repository in the examined works is VX Heavens (Vxheavens), followed by Offensive Computing (OffensiveComputing) and Malicia Project (MaliciaProject). The first two repositories are still actively maintained at the time of writing, while Malicia Project has been permanently shut down due to dataset ageing and lack of maintainers.

Security vendors, popular sandboxed analysis services, and AV companies have access to a huge number of samples. Surveyed works rely on CWSandbox, developed by ThreatTrack Security (ThreatTrack), and Anubis (Anubis). As can be observed from Fig. 2, these sandboxes are mainly used for obtaining malicious samples. Internet Service Providers (ISPs), honeypots and Computer Emergency Response Teams (CERTs) share with researchers both benign and malicious datasets. A few works use malware developed by the authors (Gharacheh et al., 2015; Khodamoradi et al., 2015), created using malware toolkits (Wong and Stamp, 2006) such as Next Generation Virus Construction Kit, Virus Creation Lab, Mass Code Generator and Second Generation Virus Generator, all available on VX Heavens (Vxheavens). A minority of analysed papers do not mention the source of their datasets.

Among surveyed papers, a recurring issue is the size of used dataset. Many works, including Kolter and Maloof (2006), Ahmed et al. (2009) and Firdausi et al. (2010), carry out evaluations on less than 1000 samples. Just 39% of reviewed studies test their approaches on a population greater than 10,000 samples.

When both malicious and benign samples are used for the evaluation, it is crucial to reflect their real distribution (Ahmed et al., 2009; Anderson et al., 2011; 2012; Bai et al., 2014; Bilge et al., 2012; Dahl et al., 2013; Elhadi et al., 2015; Eskandari et al., 2013; Feng et al., 2015; Firdausi et al., 2010; Ghiasi et al., 2015; Islam et al., 2013; Kawaguchi and Omote, 2015; Kirat et al., 2013; Kolter and Maloof, 2006; Lin et al., 2015; Mao et al., 2015; Pai et al., 2015; Palahan et al., 2013; Park et al., 2010; Raff and Nicholas, 2017; Santos et al., 2013a; 2013b; 2011; Saxe and Berlin, 2015; Schultz et al., 2001; Siddiqui et al., 2009; Srakaew et al., 2015; Uppal et al., 2014; Wüchner et al., 2015; Yonts, 2012). Indeed, there needs to be a huge imbalance because non-malware executables are the overwhelming majority. 48% of surveyed works do not take care of this aspect and use datasets that either are balanced between malware and non malicious software or, even, have more of the former than the latter. In Yonts (2012), Yonts supports his choice of using a smaller benign dataset by pointing out that changes in standard system files and legitimate applications are little. 38% of examined papers employ instead datasets having a proper distribution of malware and non-malware, indeed they are either unbalanced towards benign samples or use exclusively benign or malicious software. As an example, the majority of surveyed papers having malware similarities detection as objective (see Table 4) contains exclusively either malware or legitimate applications (Bailey et al., 2007; Bayer et al., 2009; Egele et al., 2014; Rieck et al., 2011). The remaining 14% does not describe how datasets are composed.

Differently from other research fields, no reference benchmark is available for malware analysis to compare accuracy and performance with other works. Furthermore, published results are known to be biased towards good results (Sanders, 2017). In addition, since the datasets used for evaluations are rarely shared, it is nearly impossible to compare works. Only two surveyed works have shared their dataset (Schultz et al., 2001; Upchurch and Zhou, 2015), while a third one plans to share it in the future (Mohaisen et al., 2015). It is worth mentioning that one of the shared dataset is from 2001, hence almost useless today. Indeed, temporal information is crucial to evaluate malware analysis results (Miller et al., 2015) and determine whether machine learning models have become obsolete (Harang and Ducau, 2018; Jordaney et al., 2017).

Given such lack of reference datasets, we propose three desiderata for malware analysis benchmarks.

1. Benchmarks should be labeled accordingly to the specific objectives to achieve. As an example, benchmarks for families selection should be labeled with samples' families.
2. Benchmarks should model realistically the sample distributions of real-world scenarios, considering the objectives to attain. For example, benchmarks for malware detection should contain a set of legitimate applications orders of magnitude greater than the number of malware samples.
3. Benchmarks should be actively maintained and updated over time with new samples, trying to keep pace with the malware industry. Samples should also be provided with temporal information, e.g., when they have been spotted first.

Datasets used in Schultz et al. (2001) and Upchurch and Zhou (2015) are correctly labeled according to malware detection and malware variants selection objectives, respectively. Both datasets are not balanced. In Schultz et al. (2001), the described dataset is biased towards malicious programs, while in Upchurch and Zhou (2015) diverse groups of variants contain a different number of samples, ranging from 3 to 20. Finally, analysed datasets are not actively maintained and do not contain any temporal information (in Schultz et al. (2001), the authors do not mention if such information has been included into the dataset).

6. Topical trends

This section outlines a list of topical trends in malware analysis, i.e. topics that are currently being investigated but have not reached the same level of maturity of the other areas described in previous sections.

6.1. *Malware development detection*

Malware developers can use online public services like VirusTotal (VirusTotal) and Malwr (Malwr) to test the effectiveness of their samples in evading most common antiviruses. Malware analysts can leverage such behaviour by querying these online services to obtain additional information useful for the analysis, such as submission time and how many online antiviruses classify a sample as malicious. Graziano et al. (2015) leverage submissions to an online sandbox for identifying cases where new samples are being tested, with the final aim to detect novel malware during their development process. Surprisingly, it turned out that samples used in infamous targeted campaigns had been submitted to public sandboxes months or years before.

With reference to the proposed taxonomy, advances in the state of the art in malware analysis could be obtained by analysing submissions to online malware analysis services, to extract additional machine learning features and gather intelligence on what next malware are likely to be.

6.2. *Malware attribution*

Another aspect of interest for malware analysts is the identification of who developed a given sample, i.e. the attribution of a malware to a specific malicious actor. There are a number of features in a binary to support this process: used programming language, included IP addresses and URLs, and the language of comments and resources. Additional, recently proposed features which can be used for attribution are the time slot when the malware communicates with a command and control centre and what digital certificates are used(Ruthven and Blaich, 2017). Features related to the coding style can also reveal important details on developer's identity, at least for arguing whether different malware have been developed by the same person or group. In Caliskan-Islam et al. (2015), the author's coding style of a generic software (i.e. not necessarily malicious) is accurately profiled through syntactic, lexical, and layout features. Unfortunately, this approach requires the availability of source code, which happens only occasionally, e.g. in case of leaks and/or public disclosures.

Malware attribution can be seen as an additional analysis objective, according to the proposed taxonomy. Progresses in this direction through machine learning techniques are currently hindered by the lack of ground truth on malware authors, which proves to be really hard to provision. Recent approaches leverage on public reports referring to APT groups and detailing what malware they are supposed to have developed: those reports are parsed to mine the relationships between malicious samples and corresponding APT group authors (Laurenza et al., 2017). The state of the art in malware attribution through machine learning can be advanced by researching alternative methods to generate reliable ground truth on malware developers, or on what malware have been developed by the same actor.

6.3. *Malware triage*

Given the huge amount of new malware that need to be analysed, a fast and accurate prioritisation is required to identify what samples deserve more in depth analyses. This can be decided on the basis of the level of similarity with already known samples. If a new malware resembles very closely other binaries that have been analysed before, then its examination is not a priority. Otherwise, further analyses can be advised if a new malware really looks differently from everything else observed so far. This process is referred to as *malware triage* and shares some aspects with malware similarity analysis, as they

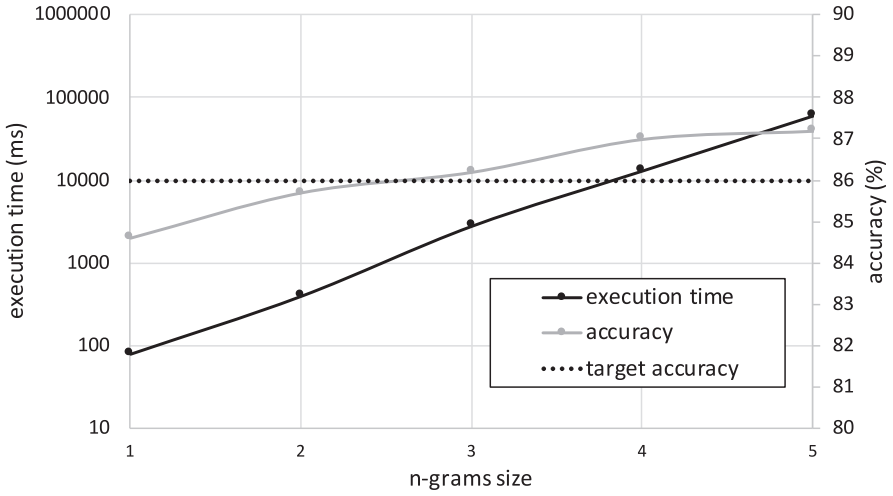

Fig. 3 – Relationship between execution time (in logarithmic scale) and detection accuracy as *n* varies. The target accuracy of 86% is also reported.

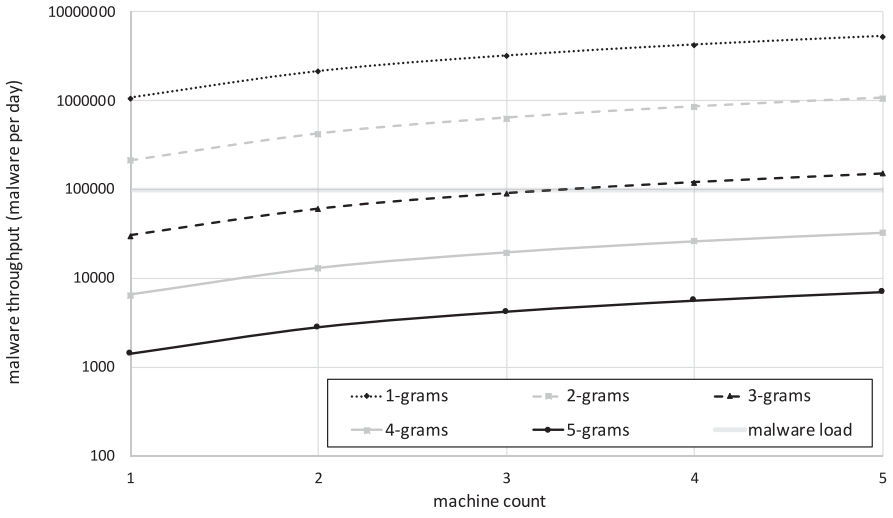

Fig. 4 – Relationship between machine count and malware throughput (in logarithmic scale) for different n-grams sizes. The one million malware per day to sustain is also reported.

both provide key information to support malware analysis prioritisation. Anyway, they are different because triage requires faster results at the cost of worse accuracy, hence different techniques are usually employed (Jang et al., 2011; Kirat et al., 2013; Laurenza et al., 2017; Rosseau and Seymour, 2018).

Likewise attribution, triage can be considered as another malware analysis objective. One important challenge of malware triage is finding the proper trade-off between accuracy and performance, which fits with the problems we address in the context of malware analysis economics (see Section 7).

6.4. *Prediction of future variants*

Compared to malware analysts, malware developers have the advantage of knowing current anti-malware measures and thus novel variants can be designed accordingly. A novel trend in malware analysis is investigating the feasibility to fill that gap by predicting how future malware will look like, so as to allow analysts to update anti-malware measures ahead. Howard et al. (2017) use machine learning techniques to model patterns in malware family evolutions and predict future variants.

This problem can be seen as yet another objective in the malware analysis taxonomy. It has not been investigated much yet, only a couple of works seem to address that topic (Howard et al., 2017; Juzonis et al., 2012). Given its great potential to proactively identify novel malware, and considering the opportunity to exploit existing malware families datasets, we claim the worthiness to boost the research on malware evolution prediction through machine learning techniques.

6.5. *Other features*

This section describes features different from those analysed in Section 3.2.2 and that have been used by just a few papers so

far. In view of advancing the state of the art in malware analysis, additional research is required on the effectiveness of using such features to improve the accuracy of machine learning techniques.

6.5.1. *Memory accesses*

Any data of interest, such as user generated content, is temporary stored in main memory, hence analysing how memory is accessed can reveal important information about the behaviour of an executable (Pomeranz, 2012). Kong and Yan (2013) rely on statically trace reads and writes in main memory, while Egele et al. (2014) dynamically trace values read from and written to stack and heap.

6.5.2. *Function length*

Another characterising feature is the function length, measured as the number of bytes contained in a function. This input alone is not sufficient to discriminate malicious executables from benign software, indeed it is usually combined with other features. This idea, formulated in Tian et al. (2008), is adopted in Islam et al. (2013), where function length frequencies, extracted through static analysis, are used together with other static and dynamic features.

6.5.3. *Raised exceptions*

The analysis of the exceptions raised during the execution can help understanding what strategies a malware adopts to evade analysis systems (Asquith, 2015; Santos et al., 2013b). A common trick to deceive analysts is throwing an exception to run a malicious handler, registered at the beginning of malware execution. In this way, examining the control flow becomes much more complex.

7. Malware analysis economics

Analysing samples through machine learning techniques requires complex computations for extracting desired features and running chosen algorithms. The time complexity of these computations has to be carefully taken into account to ensure they complete fast enough to keep pace with the speed new malware are developed. Space complexity has to be considered as well, indeed feature space can easily become excessively large (e.g., using n-grams), and also the memory required by machine learning algorithms can grow to the point of saturating available resources.

Time and space complexities can be either reduced to adapt to processing and storage capacity at disposal, or they can be accommodated by supplying more resources. In the former case, the analysis accuracy is likely to worsen, while, in the latter, accuracy levels can be preserved at the cost of providing more computing machines, storage and network. There exist therefore trade-offs between maintaining high accuracy and performance of malware analysis on one hand, and supplying the required equipment on the other. We refer to the study of these trade-offs as *malware analysis economics*, and in this section we provide some initial qualitative discussions on this novel topic.

The time needed to analyse a sample through machine learning is mainly spent in feature extraction and algorithm execution. While time complexity of machine learning algorithms is widely discussed in literature, the same does not apply for the study of feature extraction execution time. The main aspect to take into account is whether desired features come from static or dynamic analysis, which considerably affects execution time because the former does not require to run the samples, while the latter does.

To deepen even further this point, Table 7 reports for each feature type whether it can be extracted through static or dynamic analysis. It is interesting to note that certain types of features can be extracted both statically and dynamically, with significant differences on execution time as well as on malware analysis accuracy. Indeed, while certainly more time-consuming, dynamic analysis enables to gather features that contribute relevantly to the overall analysis effectiveness (Damodaran et al., 2015). As an example, we can consider the features derived from API calls (see Table 7), which can be obtained both statically and dynamically. Tools like IDA provide the list of imports used by a sample and can statically trace what API calls are present in the sample code. Malware authors usually hide their suspicious API calls by inserting in the source code a huge number of legitimate APIs. By means of dynamic analysis, it is possible to obtain the list of the APIs that the sample has actually invoked, thus simplifying the identification of those suspicious APIs. By consequences, in this case dynamic analysis is likely to generate more valuable features compared to static analysis. MazeWalker (Kulakov, 2017) is a typical example of how dynamic information can integrate static analysis.

Although choosing dynamic analysis over, or in addition to, static seems obvious, its inherently higher time complexity constitutes a potential performance bottleneck for the whole malware analysis process, which can undermine the possibility to keep pace with malware evolution speed. The natural solution is to provision more computational resources to parallelise analysis tasks and thus remove bottlenecks. In turn, such solution has a cost to be taken into account when designing a malware analysis environment, such as the one presented by Laurenza et al. (2016).

The qualitative trade-offs we have identified are between accuracy and time complexity (i.e., higher accuracy requires larger times), between time complexity and analysis pace (i.e., larger times implies slower pace), between analysis pace and computational resources (faster analysis demands using more resources), and between computational resources and economic cost (obviously, additional equipment has a cost). Similar trade-offs also hold for space complexity. As an example, when using n-grams as features, it has been shown that larger values of n lead to more accurate analysis, at cost of having the feature space grow exponentially with n (Lin et al., 2015; Uppal et al., 2014). As another example, using larger datasets in general enables more accurate machine learning models and thus better accuracy, provided the availability of enough space to store all the samples of the dataset and the related analysis reports.

We present a qualitative, simplified example of analysis that leverages on the trade-offs just introduced. The scenario we target regards detecting malware families of new malicious samples (Section 3.1.2) using as features n-grams computed over invoked APIs (Section 3.2.2), recorded through dynamic

Table 7 – Type of analysis required for extracting the inputs presented in Sections 3.2.2 and 6.5: strings, byte sequences, opcodes, APIs/system calls, file system, CPU registers, PE file characteristics, network, AV/Sandbox submissions, code stylometry, memory accesses, function length, and raised exceptions.

Analysis	Str.	Byte seq.	Ops	APIs sys. calls	File sys.	CPU reg.	PE file char.	Net.	Sub-mis.	Code stylo.	Mem.	Func. len.	Exc.
Static													
Dynamic													

Table 8 – Relationship between *n* and number of features.

n	Feature count
1	187
2	6,740
3	46,216
4	130,671
5	342,663

analysis (Section 3.2.1). We want here to explore the trade-offs between family detection accuracy, execution time, analysis pace and cost, in terms of required computational resources. For what concerns the scenario and qualitative numbers on the relationships between *n*, the number of features, accuracy and execution time, we take inspiration from the experimental evaluation presented by Lin et al. (2015). Table 8 shows the relationship between *n* and feature count. We introduce a few simplifying assumptions and constraints to make this qualitative example as consistent as possible. We assume that the algorithm used to detect families is parallelisable and ideally scalable, meaning that by doubling available machines we also double the throughput, i.e. the number of malware analysed per second. We want to process one million malware per day with an accuracy of at least 86%.

Fig. 3 highlights the trade-off between execution time (in logarithmic scale) and detection accuracy as *n* is varied. As *n* grows, the accuracy increases almost linearly while the execution time has an exponential rise, which translates to an exponential decrease of how many malware per second can be processed. It can be noted that the minimum n-grams size to meet the accuracy requirement of 86% is 3. The trade-off between analysis pace and cost can be observed in Fig. 4 where, by leveraging on the assumption of ideal scalability of the detection algorithm, it is shown that sustainable malware throughput (in logarithmic scale) increases linearly as the algorithm is parallelised on more machines. 4-grams and 5-grams cannot be used to cope with the expected malware load of one million per day, at least when considering up to five machines. On the other hand, by using four machines and 3-grams, we can sustain the target load and at the same time meet the constraint on detection accuracy.

The presented toy example is just meant to better explain how malware analysis economics can be used in practical scenarios. We claim the significance of investigating these trade-offs more in detail, with the aim of outlining proper guidelines and strategies to design a malware analysis environment in compliance with requirements on analysis accuracy and pace, while respecting budget constraints.

8. Conclusion

We presented a survey on existing literature on malware analysis through machine learning techniques. There are five main contributions of our work. First, we proposed an organization of reviewed works according to three orthogonal dimensions: *the objective of the analysis, the type of features extracted from samples*, the *machine learning algorithms used to process these features*. Such characterization provides an overview on how machine learning algorithms can be employed in malware analysis, emphasising which specific feature classes allow to achieve the objective(s) of interest. Second, we have arranged existing literature on PE malware analysis through machine learning according the proposed taxonomy, providing a detailed comparative analysis of surveyed works. Third, we highlighted the current issues of machine learning for malware analysis: anti-analysis techniques used by malware, what operation set to consider for the features and used datasets. Fourth, we identified topical trends on interesting objectives and features, such as malware attribution and triage. Fifth, we introduced the novel concept of *malware analysis economics*, concerning the investigation and exploitation of existing trade-offs between performance metrics of malware analysis (e.g., analysis accuracy and execution time) and economical costs.

Noteworthy research directions to investigate can be linked to those contributions. Novel combinations of objectives, features and algorithms can be investigated to achieve better accuracy compared to the state of the art. Moreover, observing that some classes of algorithms have never been used for a certain objective may suggest novel directions to examine further. The discussion on malware analysis issues can provide further ideas worth to be explored. In particular, defining appropriate benchmarks for malware analysis is a priority of the whole research area. The novel concept of malware analysis economics can encourage further research directions, where appropriate tuning strategies can be provided to balance competing metrics (e.g. accuracy and cost) when designing a malware analysis environment.

Acknowledgment

This work has been partially supported by a grant of the Italian Presidency of Ministry Council and by the Laboratorio

Nazionale of Cyber Security of the CINI (Consorzio Interuniversitario Nazionale Informatica).

REFERENCES

Ahmadi M, Giacinto G, Ulyanov D, Semenov S, Trofimov M. Novel feature extraction, selection and fusion for effective malware family classification. CoRR 2015. abs/1511.04317.

Ahmed F, Hameed H, Shafiq MZ, Farooq M. Using spatio-temporal information in APi calls with machine learning algorithms for malware detection. In: Proceedings of the 2nd ACM workshop on security and artificial intelligence. ACM; 2009. p. 55–62.

Allen FE. Control flow analysis. In: Proceedings of a symposium on compiler optimization. New York, NY, USA: ACM; 1970. p. 1–19.

Anderson B, Quist D, Neil J, Storlie C, Lane T. Graph-based malware detection using dynamic analysis. J Comput Virol 2011;7(4):247–58.

Anderson B, Storlie C, Lane T. Improving malware classification: bridging the static/dynamic gap. In: Proceedings of the 5th ACM workshop on security and artificial intelligence. ACM; 2012. p. 3–14.

Anubis. https://seclab.cs.ucsb.edu/academic/projects/projects/anubis/. Accessed: 2018-06-03.

Asquith M. Extremely scalable storage and clustering of malware metadata. J Comput Virol Hack Tech 2015;12:1–10.

Attaluri S, McGhee S, Stamp M. Profile hidden Markov models and metamorphic virus detection. J Comput Virol 2009;5(2):151–69.

AV-TEST. Security Report 2016/17. 2017. https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf.

Bai J, Wang J, Zou G. A malware detection scheme based on mining format information. Sci World J 2014.

Bailey M, Oberheide J, Andersen J, Mao ZM, Jahanian F, Nazario J. Automated classification and analysis of internet malware. In: Proceedings of the 10th international symposium on recent advances in intrusion detection. Springer; 2007. p. 178–97.

Baldoni R, Coppa E, D'Elia DC, Demetrescu C, Finocchi I. A survey of symbolic execution techniques. ACM Comput Surv 2018;51(3).

Barriga JJ, Yoo SG. Malware detection and evasion with machine learning techniques: a survey. Int J Appl Eng Res 2017;12(318):41:1–41:40.

Basu I. Malware detection based on source data using data mining: a survey. Am J Adv Comput 2016;3(1):18–37.

Bayer U, Comparetti PM, Hlauschek C, Kruegel C, Kirda E. Scalable, behavior-based malware clustering, 9; 2009. p. 8–11.

Bazrafshan Z, Hashemi H, Fard SMH, Hamzeh A. A survey on heuristic malware detection techniques. In: Proceedings of the 5th conference on information and knowledge technology (IKT). IEEE; 2013. p. 113–20.

Bilge L, Balzarotti D, Robertson W, Kirda E, Kruegel C. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In: Proceedings of the 28th annual computer security applications conference(ACSAC '12). ACM; 2012. p. 129–38.

Blazytko T, Contag M, Aschermann C, Holz T. Syntia: Synthesizing the semantics of obfuscated code. In: Proceedings of the 26th USENIX security symposium (USENIX Security 17). USENIX Association; 2017. p. 643–59.

Caliskan-Islam A, Harang R, Liu A, Narayanan A, Voss C, Yamaguchi F, Greenstadt R. De-anonymizing programmers via code stylometry. In: Proceedings of the USENIX Security '15. USENIX Association; 2015. p. 255–70.

Chau DH, Nachenberg C, Wilhelm J, Wright A, Faloutsos C. Polonium: Tera-scale graph mining for malware detection. In: Proceedings of the ACM SIGKDD conference on knowledge discovery and data mining; 2010. p. 131–42.

Chen L, Li T, Abdulhayoglu M, Ye Y. Intelligent malware detection based on file relation graphs. In: Proceedings of the IEEE international conference on semantic computing (ICSC),; 2015. p. 85–92.

Chen Z, Roussopoulos M, Liang Z, Zhang Y, Chen Z, Delis A. Malware characteristics and threats on the internet ecosystem. J Syst Softw 2012;85(7):1650–72.

Comar PM, Liu L, Saha S, Tan PN, Nucci A. Combining supervised and unsupervised learning for zero-day malware detection. In: Proceedings of the 32nd annual IEEE international conference on computer communications (INFOCOM); 2013. p. 2022–30.

Dahl GE, Stokes JW, Deng L, Yu D. Large-scale malware classification using random projections and neural networks. In: Proceedings of the 38th international conference on acoustics, speech and signal processing (ICASSP). IEEE; 2013. p. 3422–6.

Damodaran A, Di Troia F, Visaggio CA, Austin TH, Stamp M. A comparison of static, dynamic, and hybrid analysis for malware detection. J Comput Virol Hack Tech 2015:1–12.

Egele M, Scholte T, Kirda E, Kruegel C. A survey on automated dynamic malware-analysis techniques and tools. ACM Comput Surv (CSUR) 2012;44(2):6.

Egele M, Woo M, Chapman P, Brumley D. Blanket execution: dynamic similarity testing for program binaries and components. In: Proceedings of the 23rd USENIX Security Symposium. San Diego, CA: USENIX Association; 2014. p. 303–17.

Elhadi E, Maarof MA, Barry B. Improving the detection of malware behaviour using simplified data dependent api call graph. J Secur Appl 2015.

Eskandari M, Khorshidpour Z, Hashemi S. Hdm-analyser: a hybrid analysis approach based on data mining techniques for malware detection. J Comput Virol Hack Tech 2013;9(2):77–93.

Feng Z, Xiong S, Cao D, Deng X, Wang X, Yang Y, Zhou X, Huang Y, Wu G. Hrs: a hybrid framework for malware detection. In: Proceedings of the 2015 ACM international workshop on security and privacy analytics. ACM; 2015. p. 19–26.

Firdausi I, Lim C, Erwin A, Nugroho AS. Analysis of machine learning techniques used in behavior-based malware detection. In: Proceedings of the 2010 second international conference on advances in computing, control, and telecommunication technologies(ACT '10). IEEE; 2010. p. 201–3.

Gardiner J, Nagaraja S. On the security of machine learning in malware c&c detection: a survey. ACM Comput Surv 2016;49(3):59:1–59:39.

Gharacheh M, Derhami V, Hashemi S, Fard SMH. Proposing an HMM-based approach to detect metamorphic malware. In: Proceedings of the 4th Iranian Joint Congress on Fuzzy and Intelligent Systems Fuzzy and Intelligent Systems (CFIS); 2015. p. 1–5.

Ghiasi M, Sami A, Salehi Z. Dynamic VSA: a framework for malware detection based on register contents. Eng Appl Artif Intell 2015;44:111–22.

Graziano M, Canali D, Bilge L, Lanzi A, Balzarotti D. Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence. In: Proceedings of the 24th USENIX Security Symposium; 2015. p. 1057–72.

Guilfanov I. Decompilers and beyond. Black Hat USA 2008.

Harang R., Ducau F.. Measuring the Speed of the Red Queens Race. https://i.blackhat.com/us-18/Wed-August-8/us-18-Harang-Measuring-the-Speed-of-the-Red-Queens-Race.pdf, Last accessed: 2018-10-18; 2018.

Howard M, Pfeffer A, Dalai M, Reposa M. Predicting signatures of future malware variants. In: Proceedings of the 12th international conference on malicious and unwanted software (MALWARE); 2017. p. 126–32.

Hu X, Shin KG, Bhatkar S, Griffin K. Mutantx-s: Scalable malware clustering based on static features. In: Proceedings of the USENIX Annual Technical Conference; 2013. p. 187–98.
Huang K, Ye Y, Jiang Q. ISMCS: an intelligent instruction sequence based malware categorization system. In: Proceedings of the 3rd international conference on Anti-counterfeiting, security, and identification in Communication. IEEE; 2009. p. 509–12.
Islam R, Tian R, Batten LM, Versteeg S. Classification of malware based on integrated static and dynamic features. J Netw Comput Appl 2013;36(2):646–56.
Jang J, Brumley D, Venkataraman S. Bitshred: feature hashing malware for scalable triage and semantic analysis. In: Computer and communications security. ACM; 2011. p. 309–20.
Jordaney R, Sharad K, Dash SK, Wang Z, Papini D, Nouretdinov I, Cavallaro L. Transcend: Detecting concept drift in malware classification models. In: Proceedings of the 26th USENIX Security Symposium (USENIX Security 17). Vancouver, BC: USENIX Association; 2017. p. 625–42.
Juzonis V, Goranin N, Cenys A, Olifer D. Specialized genetic algorithm based simulation tool designed for malware evolution forecasting. Annales Univ Mariae Curie-Sklodowska sectio AI–Inf 2012;12(4):23–37.
Karpin J., Dorfman A.. Crypton - exposing malware's deepest secrets. https://recon.cx/2017/montreal/resources/slides/RECON-MTL-2017-crypton.pdf, Last accessed: 2018-10-18; 2017.
Kawaguchi N, Omote K. Malware function classification using APIs in initial behavior. In: Proceedings of the 10th Asia Joint Conference on Information Security (AsiaJCIS). IEEE; 2015. p. 138–44.
Khodamoradi P, Fazlali M, Mardukhi F, Nosrati M. Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms. In: Proceedings of the 18th CSI international symposium on computer architecture and digital systems (CADS). IEEE; 2015. p. 1–6.
Kirat D, Nataraj L, Vigna G, Manjunath B. Sigmal: A static signal processing based malware triage. In: Proceedings of the 29th Annual conference on computer security applications. ACM; 2013. p. 89–98.
Kolter JZ, Maloof MA. Learning to detect and classify malicious executables in the wild. J Mach Learn Res 2006;7:2721–44.
Kong D, Yan G. Discriminant malware distance learning on structural information for automated malware classification. In: Proceedings of the international conference on knowledge discovery and data mining. New York, NY, USA, KDD '13: ACM; 2013. p. 1357–65.
Kotov V, Wojnowicz M. Towards generic deobfuscation of windows API calls. Computing Research Repository 2018;abs/1802.04466.
Kruczkowski M, Szynkiewicz EN. Support vector machine for malware analysis and classification. In: Web intelligence (WI) and intelligent agent technologies (IAT). IEEE Computer Society; 2014. p. 415–20.
Kulakov Y.. Mazewalker. https://recon.cx/2017/montreal/resources/slides/RECON-MTL-2017-MazeWalker.pdf; 2017.
Kwon BJ, Mondal J, Jang J, Bilge L, Dumitras T. The dropper effect: Insights into malware distribution with downloader graph analytics. In: Proceedings of the 22nd SIGSAC Conference on Computer and Communications Security. ACM; 2015. p. 1118–29.
Laurenza G, Aniello L, Lazzeretti R, Baldoni R. Malware triage based on static features and public apt reports. In: Dolev S, Lodha S, editors. In: Cyber security cryptography and machine learning. Cham: Springer International Publishing; 2017. p. 288–305.
Laurenza G, Ucci D, Aniello L, Baldoni R. An architecture for semi-automatic collaborative malware analysis for CIs. In: Proceedings of the 46th annual IEEE/IFIP international conference on dependable systems and networks workshop. IEEE; 2016. p. 137–42.
LeDoux C, Lakhotia A. Malware and machine learning. In: Intelligent methods for cyber warfare. Springer; 2015. p. 1–42.
Lee T, Mody JJ. Behavioral classification. In: Proceedings of the EICAR Conference; 2006. p. 1–17.
Liang G, Pang J, Dai C. A behavior-based malware variant classification technique. Int J Inf Educ Technol 2016;6(4):291.
Liao Y, Cai R, Zhu G, Yin Y, Li K. Mobilefindr: Function similarity identification for reversing mobile binaries, 11098. Springer; 2018. p. 66–83. *Lecture Notes in Computer Science*
Lin CT, Wang NJ, Xiao H, Eckert C. Feature selection and extraction for malware classification. J Inf Sci Eng 2015;31(3):965–92.
Lindorfer M, Kolbitsch C, Comparetti PM. Detecting environment-sensitive malware. In: Recent advances in intrusion detection. Springer; 2011. p. 338–57.
Mao W, Cai Z, Towsley D, Guan X. Probabilistic inference on integrity for access behavior based malware detection. In: Proceedings of the international workshop on recent advances in intrusion detection. Springer; 2015. p. 155–76.
Malicia Project. http://malicia-project.com; Accessed: 2018-06-03.
Malwr. https://malwr.com; Accessed: 2018-06-03.
Miller B, Kantchelian A, Afroz S, Bachwani R, Faizullabhoy R, Huang L, Shankar V, Tschantz M, Wu T, Yiu G, et al. Back to the future: Malware detection with temporally consistent labels. Computing Research Repository 2015.
Miramirkhani N, Appini MP, Nikiforakis N, Polychronakis M. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In: Proceedings of the IEEE Symposium on Security and Privacy (SP); 2017. p. 1009–24.
Mohaisen A, Alrawi O, Mohaisen M. Amal: High-fidelity, behavior-based automated malware analysis and classification. Comput Secur 2015;52:251–66.
Nari S, Ghorbani AA. Automated malware classification based on network behavior. In: Proceedings of the international conference on computing, networking and communications (ICNC). IEEE; 2013. p. 642–7.
Offensive Computing. http://www.offensivecomputing.net. Accessed: 2018-06-03.
Pai S, Di Troia F, Visaggio CA, Austin TH, Stamp M. Clustering for malware classification. J Comput Virol Hack Tech 2015.
Palahan S, Babić D, Chaudhuri S, Kifer D. Extraction of statistically significant malware behaviors. In: Proceedings of the conference on computer security applications. ACM; 2013. p. 69–78.
Park HS, Jun CH. A simple and fast algorithm for K-medoids clustering. Expert Syst Appl 2009;36:3336–41.
Park Y, Reeves D, Mulukutla V, Sundaravel B. Fast malware classification by automated behavioral graph matching. In: Proceedings of the workshop on cyber security and information intelligence research. ACM; 2010. p. 45.
Polino M, Scorti A, Maggi F, Zanero S. Jackdaw: towards automatic reverse engineering of large datasets of binaries. In: Detection of intrusions and malware, and vulnerability assessment. Springer International Publishing; 2015. p. 121–43.
Pomeranz H.. Detecting malware with memory forensics. http://www.deer-run.com/~hal/Detect_Malware_w_Memory_Forensics.pdf, Last accessed: 2018-05-14; 2012.
Raff E, Nicholas C. An alternative to NCD for large sequences, Lempel-Ziv Jaccard distance. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2017. p. 1007–15.
Rieck K, Trinius P, Willems C, Holz T. Automatic analysis of malware behavior using machine learning. J Comput Secur 2011;19(4):639–68.

Rosseau A., Seymour R.. Finding Xori: malware analysis triage with automated disassembly. https://i.blackhat.com/us-18/Wed-August-8/us-18-Rousseau-Finding-Xori-Malware-Analysis-Triage-With-Automated-Disassembly.pdf, Last accessed: 2018-10-14; 2018.
Ruthven M., Blaich A.. Fighting targeted malware in the mobile ecosystem. https://www.blackhat.com/docs/us-17/wednesday/us-17-Ruthven-Fighting-Targeted-Malware-In-The-Mobile-Ecosystem.pdf, Last accessed: 2018-10-16; 2017.
Sahu MK, Ahirwar M, Hemlata A. A review of malware detection based on pattern matching technique. Int J Comput Sci Inf Technol (IJCSIT) 2014;5(1):944–7.
Sanders H.. Garbage in, garbage out - how purportedly great ML models can be screwed up by bad data. https://www.blackhat.com/docs/us-17/wednesday/us-17-Sanders-Garbage-In-Garbage-Out-How-Purportedly-Great-ML-Models-Can-Be-Screwed-Up-By-Bad-Data.pdf, Last accessed: 2018-10-18; 2017.
Santos I, Brezo F, Ugarte-Pedrero X, Bringas PG. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Inf Sci 2013a;231:64–82.
Santos I, Devesa J, Brezo F, Nieves J, Bringas PG. Opem: a static-dynamic approach for machine-learning-based malware detection. In: Proceedings of the international Joint Conference CISIS '12-ICEUTE' 12-SOCO' Special Sessions. Springer; 2013b. p. 271–80.
Santos I, Nieves J, Bringas PG. International symposium on distributed computing and artificial intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg; 2011. p. 415–22.
Saxe J, Berlin K. Deep neural network based malware detection using two dimensional binary program features. In: Proceedings of the 10th International Conference on Malicious and Unwanted Software (MALWARE). IEEE; 2015. p. 11–20.
Schulte E, Ruchti J, Noonan M, Ciarletta D, Loginov A. Evolving exact decompilation. Proceedings of the workshop on binary analysis research (BAR), 2018.
Schultz MG, Eskin E, Zadok F, Stolfo SJ. Data mining methods for detection of new malicious executables. In: Proceedings of the IEEE Symposium on Security and Privacy; 2001. p. 38–49.
Sexton J, Storlie C, Anderson B. Subroutine based detection of APT malware. J Comput Virol Hack Tech 2015:1–9.
Shabtai A, Moskovitch R, Elovici Y, Glezer C. Detection of malicious code by applying machine learning classifiers on static features: a state-of-the-art survey. Inf Secur Tech Rep 2009;14(1):16–29.
Siddiqui M, Wang MC, Lee J. Detecting internet worms using data mining techniques. J Syst Cybern Inf 2009:48–53.
Souri A, Hosseini R. A state-of-the-art survey of malware detection approaches using data mining techniques. Hum Cent Comput Inf Sci 2018;8(1):3.
Srakaew S, Piyanuntcharatsr W, Adulkasem S. On the comparison of malware detection methods using data mining with two feature sets. J Secur Appl 2015;9:293–318.
Tamersoy A, Roundy K, Chau DH. Guilt by association: large scale malware detection by mining file-relation graphs. In: Proceedings of the 20th international conference on Knowledge discovery and data mining(SIGKDD). ACM; 2014. p. 1524–33.
ThreatTrack. https://www.threattrack.com/malware-analysis.aspx. Accessed: 2018-06-03.
Tian R, Batten LM, Versteeg SC. Function length as a tool for malware classification. In: Proceedings of the 3rd international conference on malicious and unwanted software(malware); 2008. p. 69–76.
Upchurch J, Zhou X. Variant: a malware similarity testing framework. In: Proceedings of the 10th international conference on malicious and unwanted software (Malware). IEEE; 2015. p. 31–9.
Uppal D, Sinha R, Mehra V, Jain V. Malware detection and classification based on extraction of API sequences. In: Proceedings of the international conference on advances in computing, communications and informatics (ICACCI). IEEE; 2014. p. 2337–42.
Vadrevu P, Perdisci R. MAXS: Scaling malware execution with sequential multi-hypothesis testing. In: Proceedings of the conference on computer and communications security (CCS). New York, NY, USA: ACM; 2016. p. 771–82. ASIA CCS '16
Vadrevu P, Rahbarinia B, Perdisci R, Li K, Antonakakis M. Measuring and detecting malware downloads in live network traffic. In: Proceedings of the 18th European Symposium on Research in Computer Security. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. p. 556–73. Egham, UK, September 9–13, 2013
VirusTotal. https://www.virustotal.com. Accessed: 2018-06-03.
Vxheaven. https://github.com/opsxcq/mirror-vxheaven.org. Accessed: 2018-06-03.
Wong W, Stamp M. Hunting for metamorphic engines. J Comput Virol 2006;2(3):211–29.
Wüchner T, Ochoa M, Pretschner A. Robust and effective malware detection through quantitative data flow graph metrics. In: Detection of intrusions and malware, and vulnerability assessment. Springer; 2015. p. 98–118.
Ye Y, Li T, Adjeroh D, Iyengar SS. A survey on malware detection using data mining techniques. ACM Comput Surv (CSUR) 2017;50(3):41.
Ye Y, Li T, Chen Y, Jiang Q. Automatic malware categorization using cluster ensemble. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2010. p. 95–104.
Yonts J. In: Technical Report. Attributes of malicious files. The SANS Institute; 2012.

Daniele Ucci is a Ph.D. student in Engineering in Computer Science at Department of Computer, Control, and Management Engineering "Antonio Ruberti" at Sapienza University of Rome. He received the master degree with honors in Computer Engineering in Computer Science in 2014 A.Y.. His research interests mainly focus on Big Data and information security and privacy, with special regard to malware analysis. During his master thesis, he has investigated topics related to business intelligence and Big Data. Currently, he is working both on privacy-preserving data sharing of sensitive information in collaborative environments and malware analysis based on machine learning techniques.

Leonardo Aniello is a Lecturer in Cyber Security at the University of Southampton, where he is also a member of the Cyber Security Research Group. He obtained a Ph.D. in Engineering in Computer Science in 2014 from "La Sapienza" University of Rome, with a thesis about techniques for processing Big Data in large-scale environments by adopting a collaborative approach, and with the aim of improving the timeliness of the elaboration. His research studies are currently focused on cyber security aspects, including malware analysis, blockchain-based systems and privacy-preserving data sharing. Leonardo is author of more than 30 papers about these topics, published on international conferences, workshops, journals and books.

Roberto Baldoni is a full professor at the Sapienza University of Rome. He conducts research (from theory to practice) in the fields of distributed, pervasive and p2p computing, middleware platforms and information systems infrastructure with a specific emphasis on dependability and security aspects. Roberto Baldoni is director of the Sapienza Research Center for Cyber Intelligence and Information Security and, at national level, is director of the

Cyber Security National Laboratory. Recently, he has been appointed as coordinator of the National Committee for Cybersecurity Research born on February 2017 as an agreement between the Italian National Research Council and the Cyber Security National Laboratory. A partial list of his publications can be found at DBLP, at Scholar Google and at MIDLAB publication repository.