Survey on the Usage of Machine Learning Techniques for Malware Analysis

DANIELE UCCI, "La Sapienza" University of Rome
LEONARDO ANIELLO, "La Sapienza" University of Rome
ROBERTO BALDONI, "La Sapienza" University of Rome

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 for keeping pace with the speed of development of novel malware. This survey aims at providing an overview on the way machine learning has been used so far in the context of malware analysis.

We systematize surveyed papers according to their objectives (i.e., the *expected output*, what the analysis aims to), 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 problems concerning the *datasets* used in considered works, and finally introduce the novel concept of *malware analysis economics*, regarding the study of existing tradeoffs among key metrics, such as analysis accuracy and economical costs.

CCS Concepts: •**General and reference** → **Surveys and overviews;** •**Social and professional topics** → **Malware / spyware crime;** •**Computing methodologies** → **Machine learning;** •**Security and privacy** → Economics of security and privacy;

Additional Key Words and Phrases: malware analysis, machine learning, malware analysis economics

ACM Reference format:
Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2017. Survey on the Usage of Machine Learning Techniques for Malware Analysis. *ACM Trans. Web* 1, 1, Article 1 (October 2017), 34 pages.
DOI: 0000001.0000001

1 INTRODUCTION

Despite the significant improvement of security defence mechanisms and their continuous evolution, malware are still spreading and keeping to succeed in pursuing their malicious goals. Malware analysis concerns the study of malicious samples with the aim of developing a deeper understanding about several aspects of malware, including their behaviour, how they evolve over time, and how they intrude specific targets. The outcomes of malware analysis should allow security firms to update their defence solutions, in order to keep pace with malware evolution and consequently prevent new security incidents.

Author's addresses: D. Ucci, L. Aniello and R. Baldoni, Research Center of Cyber Intelligence and Information Security (CIS), Department of Computer, Control, and Management Engineering "Antonio Ruberti", "La Sapienza" University of Rome.

DOI: 0000001.0000001

Within the unceasing arm race between malware developers and analysts, each progress of security mechanisms is likely to be promptly followed by the realization of some evasion trick. The easiness of overcoming novel defensive measures also depends on how well they capture malicious traits of samples. For example, a detection rule based on the MD5 hash of a known malware can be easily eluded by applying standard obfuscation techniques, indeed they 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, as malware developers should apply more complex modifications.

Given the importance of producing defensive technologies as challenging as possible to overcome for malware producers, a major goal for malware analysis should be to capture aspects and traits having the broadest scope. In this way, resulting security measures would become harder to circumvent, and consequently the effort for attackers to adapt existing malware would result infeasible. Machine learning is a natural choice to support such a process of knowledge extraction. The plentiful availability of samples to analyse, and thus of really large training sets, has fostered the adoption of machine learning for malware analysis. Indeed, many works in literature have taken this direction, with a variety of approaches, objectives and obtained results.

This survey aims at reviewing and systematising existing academic works where machine learning is used to support malware analysis of Windows executables, i.e., Portable Executables (PEs). 57 recent papers have been reviewed, and their approaches have been systematised according to three fundamental dimensions

- the specific *objective* of the presented malware analysis (i.e., the *output*),
- what types of features they consider (i.e., the *input*),
- what machine learning *algorithm*s they consider.

Such a simple characterisation of academic works helps in understanding how machine learning can be used within the context of malware analysis, so as to also identify possible novel relevant objectives that have not been investigated yet. We distinguished 7 different objectives: *malware detection*, *malware variants detection* (variants selection and families selection), *malware category detection*, *malware novelty and similarity detection*, *malware development detection*, *malware attribution*, and *malware triage*.

The review of the features that can be gathered from a sample provides a comprehensive view of available information, and how they can be used with reference to identified malware analysis objectives. Smart combinations of these information can lead to extract additional knowledge to be used to achieve further objectives or refine existing ones. We grouped the features used by surveyed papers in 15 types: *strings*, *byte sequences*, *opcodes*, *APIs/System calls*, *memory accesses*, *file system accesses*, *Windows registry*, *CPU registers*, *function length*, *PE file characteristics*, *raised exceptions*, *network*, *AV/Sandbox submissions*, and *code stylometry*.

Examining used algorithms provides an effective overview about how selected inputs can be processed to achieve a specific malware analysis objective. The frequent employment of a particular algorithm to achieve a given objective means that such algorithm is likely to be really effective for that objective. On the other hand, observing that some class of algorithms has never been used for a certain objective may suggest novel directions to investigate further. We arranged algorithms in 4 classes: *signature-based* (malicious signature matching, malicious graph matching), *classification* (rule-based classifier, Bayes classifier, support vector machine, prototype-based classification, decision tree, k-Nearest neighbors, artificial neural network), *clustering* (clustering with locality sensitive hashing, clustering with distance and similarity metrics, k-Means clustering, density-based spatial clustering of applications with noise, hierarchical clustering, prototype-based

clustering, self-organizing maps), and others (expectation maximization, learning with local and global consistency, belief propagation).

The thorough study we have carried out has highlighted some interesting points that would deserve to be dealt with in more detail, indeed we claim they can be developed to extend and improve current academic research on the usage of machine learning for malware analysis.

A first point concerns a general lack of proper explanation about the reasons why a specific set of features enables to properly characterise the malicious traits of samples. The common approach is to take all the available features, feed them to the chosen machine learning algorithms, and compute accuracy metrics on obtained (usually good) results. Some works include a *feature selection* phase where the subset of most determining features is extracted. Except for a few papers, the vast majority does not delve into explaining the connection between considered features and achieved results, which seems to leave the whole analysis rather incomplete. We advocate the need to properly address this aspect whenever machine learning algorithms are used for malware analysis.

Another point regards the set of samples used for training and testing the chosen model. Most of reviewed papers do not describe in detail the employed dataset, nor they share it publicly, which prevents others from reproducing published results, and thus from properly comparing newly proposed solutions. This is obviously a significant obstacle to streamlining advancements in the field of malware analysis through machine learning. As a matter of fact, in other research areas where reference benchmarks are available, it is easy to prove (and also to disprove) that a novel technique is better than the state of the art, and thus to assert a progress. On the other hand, establishing a benchmark of samples acknowledged by the academic malware analysis community is extremely challenging. Indeed, benchmarks should be as stable as possible over time to be used as reference points for measurements, but malware are characterized by a strong intrinsic evolutionary nature. Novel and more advanced malicious samples are developed daily, hence each malware becomes less interesting from a research perspective as time goes by. Despite this, we believe that more effort should be spent along the direction of enabling researchers to reproduce published results, and thus to correctly compare different solutions. At this regard, we outline some desired properties that a dataset of samples should have to become a valid reference for research purposes.

A final point is about the novel concept of *malware analysis economics*. The final purpose of malware analysis is expanding the knowledge on malware, by the means of a learning process continuous over time, whose effectiveness can be measured along two dimensions. The first is the *pace* of knowledge growth, which relates to how fast this knowledge develops with respect to the evolution of malware over time. The second is the *accuracy* of the knowledge, which refers to the extent such knowledge matches the real characteristics of malware. Both pace and accuracy depend on several factors, some being hard to assess, others being easily measurable. When machine learning comes into play, these quantifiable factors include how many novel samples are considered, how many features are extracted from each sample, and what kinds of algorithms are used. Having bigger datasets at disposal (i.e., large number of samples) generally leads to learn more accurate malware knowledge, at the cost of greater effort for the collection, feature extraction, and elaboration of a larger number of samples. Required time is likely to increase too, which impacts negatively on the pace of malware knowledge growth. To keep this time as constant as possible, more physical resources should be employed to parallelise to some extent the whole malware analysis process, which in turn entails additional costs because of the higher provisioning requested. What emerges is the existence of a trade-off between the cost of the analysis from one hand, and the growth pace and accuracy of acquired knowledge from the other. Analogous

trade-offs also apply for what features are extracted from samples, and for what algorithms are used. Since the costs of malware analysis could easily rise to unaffordable levels in an attempt to achieve the highest accuracy and the fastest growth of malware knowledge, a careful understanding of the dynamics of such trade-offs turns out to be highly strategic. Thus, we claim the importance of investigating thoroughly the relationships between what is required to improve the effectiveness of malware analysis (i.e., growth pace and accuracy of obtained malware knowledge) and how much it costs (i.e., in terms of time, realization complexity, and needed resources). This would make it possible to define clear guidelines on setting up a malware analysis process able to meet specific requirements on effectiveness at the minimum cost.

In literature, some works have already addressed the problem of surveying contributions dealing with the usage of machine learning techniques for malware analysis. In [11], the authors analyse scientific papers on malware detection only. They identify three main methods for detecting malicious software, based on signatures, behaviors, and machine learning techniques. Gandotra *et al.* [29] survey papers that use machine learning techniques for malware analysis and only distinguish between malware detection and family classification. In [64], the authors focus on papers proposing techniques based on pattern matching to recognize malware. Basu *et al.* examine different works relying on data mining and machine learning techniques, whose primary objective is the detection of possibly malicious software [9]. The survey outlines the types of analysis that a malware analyst can carry out and discusses different types of inputs that can be potentially used (e.g. byte sequences, opcodes, PE file characteristics). Compared to our work, the above mentioned surveys focus just on very few malware analysis objectives, and their studies are limited to the descriptions of proposed approaches without any attempt of structuring surveyed contributions.

At the time of writing, the most similar work to ours is the one published by LeDoux and Lakhotia [47]. Their article points out the problems related to malware analysis and how machine learning can help in solving them. Similarly to our work, they provide a wider overview on machine learning concepts, list a set of features useful for analysing malware, and state the complexity of gathering a ground truth to evaluate analysis results. However, as final objective of malware analysis, they only consider the timely detection, removal, and recovery from the infection, while in this paper we identify 7 different possible objectives.

The rest of the paper is structured as follows. Section 2 introduces some basic notions on malware analysis. Section 3 outlines the possible objectives of malware analysis, Section 4 delves with what types of input data is used for the analysis, and Section 5 reports what machine learning methods are employed. The characterization of surveyed papers according to the inputs, outputs and algorithms described in previous sections is reported in Section 6. Section 7 describes the datasets used in each paper: it discusses sample collections and the issues related to experimental evaluation reproducibility. Malware analysis economics are investigated in Section 8. Finally, conclusions and possible future works are presented in Section 9.

2 BACKGROUND ON MALWARE ANALYSIS

With *malware analysis*, we refer to the process of studying a generic sample (i.e., a file), with the aim of acquiring knowledge about its potentially malicious nature. The analysis of a sample includes an initial phase where required data are extracted from the file, and an elaboration phase where these data are examined, and possibly correlated to some available knowledge base, to gain further added-value information. What information are mined depend on the specific objective to achieve. In the works considered in this survey, 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. Execution traces are indeed among the inputs used in examination and correlation phases when dynamic analysis is employed. For an extensive dissertation on dynamic analyses, refer to [22].

Malware development strategies are in line with software engineering recommendations for what concerns code reuse, in the sense that any specific malware is usually updated to the minimal extent required to evade latest detection techniques. Indeed, as a new malware is discovered by security firms and then neutralised by releasing the correspondent antivirus detection rule (e.g., its signature) or software patch, malware developers are likely to produce a *variant* of that malware, which keeps most of the code and characteristics of the original version but differs for a few aspects to guarantee its effectiveness in evading current recognition mechanisms. These mechanisms are commonly evaded by employing obfuscation and encryption techniques to automatically generate variants. These variants are referred to as *polymorphic* and *metamorphic* malware. Polymorphism changes the appearance of the original malicious code by means of encryption and data appending/prepending. These modifications are performed by *mutation engines*, usually bundled within the malware itself. The limitation of this variant generation approach is that malware code remains the same once decrypted by a mutation engine, which makes in-memory signature-based detection methods effective. On the other hand, metamorphic malware can still evade these recognition mechanisms thanks to more advanced morphing techniques. These include insertion of a number of No Operation (NOP) and garbage instructions, function reordering, control flow modification, and variation in data structure usage. Malicious software exploiting metamorphism automatically recodes itself before propagating or being redistributed by the attacker. This kind of variants can be detected by focussing on the semantics of an executable.

Variants originating from a same "root" malware are usually grouped in a *malware family*, which by consequence includes a set of samples sharing many similarities, yet being different enough among each other from the point of view of anti-malware tools.

3 MALWARE ANALYSIS OBJECTIVES

This section details the analysis goals of the surveyed papers, organized in 7 distinct objectives.

3.1 Malware Detection

The most common objective in the context of malware analysis is detecting whether a given sample is malicious. From a practical point of view, this objective is also the most important because knowing in advance that a sample is dangerous allows preventing it from being harmful for a system. Indeed, the majority of reviewed works has this as main goal [3, 4, 7, 15, 24, 25, 27, 28, 32, 42, 44, 66–68, 72, 73, 76, 78, 80, 81]. According to the specific machine learning technique employed into the detection process, the output generated by the analysis can be provided with a confidence value. The higher this value, the more the output of the analysis is likely to be correct. Hence, the confidence value can be used by malware analysts to understand if a sample under analysis needs further inspection.

3.2 Malware 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 continuous

development of new variants. Also this objective has been deeply studied in literature, and several papers included in this survey target the detection of variants. More specifically, we identify two slightly different variations of this objective

- *variants selection*: given a malicious sample *m*, select from the available knowledge base the samples that are variants of *m* [17, 31, 32, 40, 45, 49, 70, 75, 77]. Variants of a malicious samples can be obtained by employing metamorphism and polymorphism (see Section 2). Considering the huge number of malicious samples received daily from major security firms, recognizing variants of already known malware is crucial to reduce the workload for human analysts;
- *families selection*: given a malicious sample *m*, select from the available knowledge base the families that *m* belongs to [1, 19, 34–36, 39, 43, 48, 50, 54–56, 58]. 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.

3.3 Malware Category Detection

Malware can be categorized according to their prominent behaviours and objectives. As an example, malicious software can be interested in spying on users' activities and stealing their sensitive information (i.e., *spyware*), encrypting documents and asking for a ransom in some cryptocurrency (i.e., *ransomware*), or gaining remote control of an infected machine (i.e., *remote access trojans*). Even if more sophisticated malware fit more behaviours and objectives, using these categories is a coarse-grained yet significant way of describing malicious samples [16, 18, 69, 71, 74]. Although cyber security firms have not still agreed upon a standardized taxonomy of malware categories, effectively recognizing the categories of a sample can add valuable information for the analysis.

3.4 Malware Novelty and Similarity Detection

Along the line of acquiring knowledge about unknown samples by comparing them against known malware, it is really interesting to identify what are the specific similarities and differences of the binaries to analyse with respect to those already analysed and stored in the knowledge base. We can distinguish between two distinct types of novelty and similarity detection.

- *similarities detection*: discovering what parts and aspects of a sample are similar to something that has been already examined in the past enables to focus on what is really new, and hence to discard the rest as it does not deserve further investigation [8, 10, 23, 57, 61].
- *differences detection*: as a complement, also identifying what is different from everything that has been 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 [10, 51, 57, 59, 61, 65].

3.5 Malware Development Detection

An undeniable advantage for malware developers is the wide availability of the most used defence mechanisms, such as antiviruses, sandboxes, and online scanning services. Indeed, these tools can be used to test the evasion capabilities of the samples being developed. The latter can be consequently refined to avoid being detected by specific technologies, which can also depend on the actual targets of the attackers. Malware analysts can leverage this practice by analysing the submissions of samples to online services, like VirusTotal and public sandboxes, in order to identify those submissions that seem related to the test of novel malware [16, 33]. In particular, by analysing submissions and their metadata, researchers found out that malicious samples involved in famous targeted attacks, have been previously submitted to Anubis Sandbox [33].

3.6 Malware Attribution

Another aspect malware analysts are interested in regards the identification of who developed a given malware [14]. Anytime a cyber threat is detected, a three-level analysis can be carried out: *technical*, *operational*, and *strategic*. From a technical perspective, a malware analyst looks at specific indicators of the executable: what programming language has been used, if it contains any IP address or URL, and the language of comments and resources. Another distinctive trait, in case the malware exchanges messages with a command and control center, is the time slot where the attacker triggers the malware. The operational analysis consists in correlating technical information related to other cyber threats that share similarities with the malicious sample under analysis. During the strategic analysis, extracted technical and operational knowledge can be merged with intelligence information and political evaluations in the attempt of attributing a (set of) malware sample(s) to a cyber threat actor or group.

3.7 Malware Triage

A last objective addresses the need to provide a fast and accurate prioritization for new samples when they come at a fast rate and have to be analysed. This process is referred to as *triage*, and is becoming relevant because of the growing volume of new samples developed daily. Malware triage shares some aspects with the detection of variants, novelties and similarities, since they give key information to support the prioritization process. Nevertheless, triage should be considered as a different objective because it requires faster execution at the cost of possible worse accuracy, hence different techniques are usually employed [37].

4 MALWARE ANALYSIS FEATURES

This section provides an overview on the data used by reviewed papers to achieve the objectives outlined in Section 3. The features given as input to machine learning algorithms derive from these data. Since almost all the works we examined considered Windows executables, the inputs taken into account are extracted from the content of the PEs themselves or from traces and logs related to their execution.

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. Whenever possible, for each feature type we provide a table reporting what specific features are used, with proper references.

4.1 Strings

A PE can be inspected to explicitly look for the strings it contains, such as code fragments, author signatures, file names, system resource information. These types of strings have been shown [68] to provide valuable information for the malware analysis process (see Table 1). Once strings in clear are extracted, it is possible to gather information like number and presence of specific strings, which can unveil key cues to gain additional knowledge on a PE [36, 68]. In [1], the authors use histograms representing how string lengths are distributed in the sample.

String extraction tools. Strings[1] and *pedump*[2] are two well-known tools for extracting strings from a PE. While pedump outputs the list of the strings found in a Windows executable, Strings allows to use wild-card expressions and tune search parameters. Conversely to Strings, pedump is able to detect most common packers, hence it can be also used when the PE is packed. Both tools fail if

[1]Strings: https://technet.microsoft.com/en-us/sysinternals/strings.aspx
[2]pedump: https://github.com/zed-0xff/pedump

the strings contained in the executable are obfuscated. Another remarkable tool is FLOSS[3], which combines different static analysis techniques to deal with obfuscated string found in analysed samples.

Table 1. List of features employed in the surveyed papers for the input type *Strings*

Strings
Number of strings; presence of "GetProcAddress","CopyMemory", "CreateFileW", "OpenFile", "FindFirstFileA", "FindNextFileA", "RegQueryValueExW" [36]
Distribution of string lengths [1]

4.2 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 technique (see Table 2). A few works use chunks of bytes of specific sizes [68, 72], while many others rely on n-grams [1, 3, 4, 15, 19, 27, 37, 42, 50, 61, 69, 72, 75, 76, 80].

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. Different works use n-grams of diverse sizes. Most of them rely on sequences no longer than 3 (i.e., trigrams). Indeed, the number of features to consider grows exponentially with n.

Table 2. List of features employed in the surveyed papers for the input type *Byte sequences*

Byte sequences
Chunks either of 50, 100 KB, or equal to file size [72]
1-grams [1, 50, 72]
2-grams [3, 4, 50, 69, 72]
3-grams [19, 36, 50, 72]
4-grams [50, 76]
5-grams, 6-grams [50]

4.3 Opcodes

Opcodes identify the machine-level operations executed by a PE, and can be extracted by examining the assembly code [1, 3, 4, 31, 34, 40, 43, 56, 65, 66, 69–72]. As shown in Table 3, opcode frequency is a type of feature employed in some surveyed papers. It measures the number of times each specific opcode appears within the assembly of, or it is executed by, a PE [40]. Others [4, 40] count opcode occurrences by aggregating them by operation scope, e.g., math instructions, memory access instructions. Similarly to n-grams, also sequences of opcodes are used as features [31, 40, 72]. Given the executable, the Interactive DisAssembler[4] (IDA) is the most popular commercial solution that allows the extraction of the assembly code.

[3]FireEye Labs Obfuscated Strings Solver (FLOSS): https://github.com/fireeye/flare-floss
[4]Interactive DisAssembler: https://www.hex-rays.com/products/ida/index.shtml

Table 3. List of features employed in the surveyed papers for the input type *Opcodes*

Opcodes
Branch instruction, count, privileged instruction count, and memory instruction count [4]
Math instruction count, logic instruction count, stack instruction count, NOP instruction count, and other instruction count [4, 40]
Sequences of length 1 and 2 [65]
Instruction frequencies [1, 40]
Data define instruction proportions [1]
Count of instructions manipulating a single-bit, data transfer instruction count, data conversion instruction count, pointer-related instruction count, compare instruction count, jump instruction count, I/O instruction count, and set byte on conditional instruction count [40]

4.4 APIs/System calls

Similarly to opcodes, APIs and system calls enable the analysis of samples' behaviour, but at a higher level (see Table 4). They can be extracted by analysing either 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 in general what actions are executed by a sample [1, 7, 23, 36, 39, 43, 49, 70], looking at system call invocations provides a specific view on the interaction of the PE with the operating system [4, 6, 10, 19, 23, 24, 45, 48, 57, 58, 61, 66, 76]. 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. Next subsections give an overview on the data structures used in surveyed papers.

Table 4. List of features employed in the surveyed papers for the input type *APIs/System calls*

APIs/System calls
Process spawning [8, 17, 51]
Syscall dependencies [10, 24, 57]
Syscall sequences [6, 45, 58, 76]
MIST representation [61]
"RegOpenKeyEx" count, "RegOpenKeyExW" count, "RegQueryValueExW" count, "Compositing" count, "MessageBoxW" count, "LoadLibraryW" count, "LoadLibrary- ExW" count, "0x54" count [36]
Referred APIs count, Referred DDLs count [7]
Process activity [50]
Is API 'x' present in the analysed sample? [39]

4.4.1 Call graphs and data dependent call graphs. *Call graphs* allow to analyse how data is exchanged among procedure calls [63] by storing relationships among these procedures, and possibly including additional information on variables and parameters. Call graphs have been employed in [24, 33, 43, 45] to extract relationships among invoked functions. API call graphs have been subsequently extended in [17, 24, 58] by integrating data dependencies among APIs. Formally, two API calls are data dependent if the one invoked afterwards uses a parameter given in input to the other.

4.4.2 Control flow graphs, enriched control flow graphs, and quantitative data flow graphs. *Control flow graphs* allow compilers to produce an optimized version of the program itself [2]. Each graph models control flow relationships which can be used to represent the behaviour of a PE and extract

the program structure. Works employing control flow graphs for sample analysis are [4, 17, 33]. In [25], graph nodes are enriched with dynamic analysis information for deciding if conditional jumps are actually executed.Wüchner *et al.* [80] use *quantitative data flow graphs* to model the amount of data flowing among system resources. Analogously, Polino *et al.* leverage data flow analysis in order to track data dependencies and control operations in main memory [59].

4.4.3 Multilayer dependency chains. Represent function templates organized according sample behaviors [49]. Stored into different chains, function templates are characterized by six sub-behaviors capturing interactions between samples and the system in which they run. In turn, each chain contains a complete sequence of API calls invoked by a sample, along with API call metadata. All these information provide analysts a more detailed characterization of malware behavior.

4.4.4 Causal dependency graphs. They have been initially proposed by [8] for tracking the activities of a malware by monitoring persistent state changes in the target system. These persistent state changes enable to define malware behaviour profiles and recognize classes of malware exhibiting similar behaviours. In [43], causal dependency graphs are used to discover the entry point exploited by an attacker to gain access to a system.

4.4.5 Markov chains and Hidden Markov Models. Markov chains are memoryless random processes evolving with time. In [3, 4, 69], Markov chains are used to model binary content and execution traces of a sample to perform malware classification. Similarly to the approach designed in [25] for building enriched data flow graphs, instructions are categorized in 8 coarse-grained classes (e.g., math and logic instructions).

Hidden Markov models are probabilistic functions of Markov chains. States of hidden Markov models are unobservable, while the output of a state can be observed and it obeys to a probabilistic distribution (continuous or discrete). Pai *et al.* train various hidden Markov models having 2 hidden states, to recognize malware belonging to specific families [56].

4.5 Memory accesses

Any data of interest such as user generated content, Windows Registry key, configuration and network activity passes through main memory, hence analysing how memory is accessed can reveal important information about the behaviour of an executable [60] (see Table 5). Some works [43, 70] either rely on Virtual Machine Monitoring and Introspection techniques, or statically trace reads and writes in main memory. Egele *et al.* dynamically trace values read from and written to stack and heap [23].

Memory analysis tools. Different open-source tools are available to analyse memory during sample executions, such as Volatility[5] and Rekall[6], both included in the SANS Investigative Forensic Toolkit[7].

Table 5. List of features employed in the surveyed papers for the input type *Memory accesses*

Memory accesses
Performed read and write operations in main memory [43]
Values read/written from/to stack and heap [23]

[5]Volatility: http://www.volatilityfoundation.org/25
[6]Rekall: http://www.rekall-forensic.com/pages/at_a_glance.html
[7]SIFT: https://digital-forensics.sans.org/community/downloads

4.6 File system accesses

What samples do with files is fundamental to grasp evidence about the interaction with the environment and possible attempts to gain persistence. Features of interest mainly concern how many files are read or modified, what type of files and in what directories [8, 17, 33, 43, 48, 50, 54] (see Table 6). 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 under analysis. In [54], the size of these files is considered as well, while Lin *et al.* leverage the number of created hidden files [50].

File System analysis tools. Activities performed on file system can be monitored using programs like MS Windows Process Monitor[8] and SysAnalyzer[9]. While SysAnalyzer implements by default snapshots over a user-defined time interval to reduce the amount of data presented to malware analysts, Process Monitor has been designed for real-time monitoring. Nevertheless, SysAnalyzer can be also used in live-logging mode. ProcDOT[10] allows the integration of Process Monitor with network traces, produced by standard network sniffers (e.g. Windump), and provides an interactive visual analytics tool to analyse the binary activity.

Table 6. List of features employed in the surveyed papers for the input type *File system accesses*

File system accesses
Number of created/deleted/modified files, size of created files [54]
Number of created hidden files [50]

4.7 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 [48, 50, 54, 70] (see Table 7). Usually, works relying on file system inputs (see Section 4.6) monitor also the Windows Registry. In [70], changes to file system are used in conjunction with file system accesses.

Windows Registry analysis tools. Process Monitor, introduced in Section 4.6, takes also care of detecting changes to the Windows Registry. Similarly, RegShot[11] is an open-source lightweight software for examining variations in the Windows Registry by taking a snapshot before and after the sample is executed. Since it is based on snapshots, malware analysts are not overwhelmed with data belonging to the entire execution of the samples. As mentioned for memory and file system accesses, usually, sandboxes monitor Windows Registry key creations/deletions/modifications, reporting occurred changes.

Table 7. List of features employed in the surveyed papers for the input type *Windows Registry*

Windows Registry
Number of created/deleted/modified Registry keys [48, 50, 54, 70]

[8]Process Monitor: https://docs.microsoft.com/en-us/sysinternals/downloads/procmon
[9]SysAnalyzer: http://sandsprite.com/iDef/SysAnalyzer/
[10]ProcDOT: http://www.procdot.com
[11]RegShot: https://sourceforge.net/projects/regshot/

4.8 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 (see Table 8). Both [43] and [1] rely on static analysis of registers, whereas [23] and [32] employ a dynamic approach. Some works examine register use to detect malware variants [1, 32, 43]. While in [43] and [1] the objective is to identify samples belonging to one or more specific families, [32] aims to select the variants of the malware under analysis. Static analyses of CPU registers involve counting reads and writes performed on each register [43], tracking the number of changes to FLAGS [43], and measuring the frequency of register usage [1]. Conversely, [32] applies dynamic analysis to get features associated to the values contained in CPU registers. Instead, [23] monitors only returned values with the objective of detecting similarities among executables.

Table 8. List of features employed in the surveyed papers for the input type *CPU registers*

Registers
No. of changes to FLAGS register, register read/write count [43]
Returned values in the `eax` register upon function completion [23]
Registers usage frequency [1]
Registers values [32]

4.9 Function length

Another characterising feature is the length of functions. In particular, the frequency of function lengths is used [17] (see Table 9). This input alone is not sufficient to discriminate malicious executables from benign software, indeed it is usually employed in conjunction with other features. This idea, formulated in [74], is adopted in [36], where function length frequencies are combined with other static and dynamic features.

Table 9. List of features employed in the surveyed papers for the input type *Function length*

Function length
Function length frequencies [36]
Linearly/polynomially/exponentially spaced bins of length ranges [17]

4.10 PE file characteristics

A static analysis of a PE can provide a large set of valuable information such as sections, imports, symbols, used compilers (see Table 10). As malware generally present slight differences with respect to benign PEs, these information can be used to understand if a file is malicious or not [6, 7, 48, 81].

4.11 Raised exceptions

The analysis of the exceptions raised during the execution can help understanding what strategies a malware adopts to evade analysis systems [6, 66]. 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. This is the case of the Andromeda

Table 10. List of features employed in the surveyed papers for the input type *PE file characteristics*

PE file characteristics
Resource icon's checksum [6, 48]
Number of symbols, pointer to symbol table, PE timestamp, and PE characteristics flags [81]
Section count [7, 81]
Resource's directory table, items in `.reloc` section count and symbols in export table count [7]
Disassembled file size, sample size, number of lines in the disassembled file, first bytes sequence address, entropy, and symbol frequencies [1]
PE header checksum, resource strings' checksum, resource metadata checksum, section names, section sizes, import table location, import table size, and entry point offset [6]
Section attributes [1, 6]

botnet, version 2.08[12]. Even if such version is outdated, Andromeda is still active and targets victims with spam campaigns[13].

4.12 Network

A huge number of key information can be obtained by observing how the PE interacts with the network. Contacted addresses, generated traffic, and received packets can unveil valuable aspects, e.g., regarding communications with a command and control center. Statistics on used protocols, TCP/UDP ports, HTTP requests, DNS-level interactions are other features of this type (see Table 11). Many surveyed works require dynamic analysis to extract this kind of information [8, 10, 17, 33, 48, 49, 51, 54, 55]. Other papers extract network-related inputs by monitoring the network and analysing incoming and outgoing traffic [18, 44, 77]. A complementary approach consists in analysing download patterns of network users in a monitored network [78]. 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.

Table 11. List of features employed in the surveyed papers for the input type *Network*

Network
Connection count [8, 33, 54]
TCP flags, packet direction, total packets count, total packets with no-payload count, total transferred byte count, total transferred payload byte count, flow duration, average inter arrival time, size of i-th packet, inter arrival time of i-th packet, payload size of i-th packet, maximum payload size, minimum payload size, average payload size, payload size standard deviation, maximum packet size, minimum packet size, average packet size, and packet size standard deviation [55]
Download domain, download history, download request, and queried URLs [78]
Destination IP [44, 78]
Timestamp [44]
Unique IP count, protocol type, HTTP request/response type count, DNS `A/PTR/CNAME/MX/NS/SOA` record lookup count, request/response packet size [54]

[12]New Anti-Analysis Tricks In Andromeda 2.08: https://blog.fortinet.com/2014/05/19/new-anti-analysis-tricks-in-andromeda-2-08

[13]Andromeda Botnet Targets Italy in Recent Spam Campaigns: http://researchcenter.paloaltonetworks.com/2016/07/unit42-andromeda-botnet-targets-italy-in-recent-spam-campaigns/

4.13 AV/Sandbox submissions

Malware developers may use online public services like VirusTotal[14] and Malwr[15] to test the effectiveness of their samples in evading most common antiviruses. Querying these online services can provide additional information useful for the analysis: submission time, how many online antiviruses classify the same as malicious, and other data on the submission (see Table 12). Graziano *et al.* [33] leverage submissions to a sandbox for identifying cases of malware development. Surprisingly, samples used in infamous targeted campaigns have been found to be submitted to public sandboxes months or years before.

Table 12. List of features employed in the surveyed papers for the input type *AV/Sandbox submissions*

AV/Sandbox submissions
Occurred errors [6, 33, 66]
Created hidden files/registries, hidden connections, process/file activity, frequencies of specific words in the AV/Sandbox report [33, 50]
Count of registry types and registries modifications [33, 54]
PE file characteristics, timestamps, AV labels, submitting user information, and sandbox analysis results [33]

4.14 Code stylometry

Features related to the coding style used by an anonymous malware author can reveal important details about her identity. Unfortunately, this kind of features requires the availability of source code, which is very rare in malware analysis. However, in case of leaks and/or public disclosures, source codes can become available. In [14], the author's coding style of a generic software (i.e. not necessarily malicious) is captured through syntactic, lexical, and layout features (see Table 13). These are extracted both from the source code and from the abstract syntax tree, representing the executable. Syntactic features model keywords and the properties of the abstract syntax tree, while lexical and layout features allow to gather information about author's code writing preferences.

Table 13. List of features employed in the surveyed papers for the input type *Code stylometry*

Code stylometry
Source code's lexical, layout, and syntactic properties [14]

5 MALWARE ANALYSIS ALGORITHMS

In this section we briefly describe the machine learning algorithms used by reviewed papers. Different algorithms aim at a different goals, e.g., finding a match with respect to some available knowledge base, or classifying samples by assigning them labels, or clustering PEs on the basis of some metrics. We accordingly organize malware analysis algorithms in four categories: *signature-based* (Section 5.1), *classification* (Section 5.2), *clustering* (Section 5.3), and others (Section 5.4).

[14]VirusTotal: https://www.virustotal.com
[15]Malwr: https://malwr.com

5.1 Signature-based

Signature-based approaches are widely employed by commercial antiviruses to detect malicious samples. Signatures are computed by software analysts to find a pattern in potentially malicious samples under analysis. Found patterns should be also general enough to detect variants of the same malware. Obviously this task, performed by humans, is error-prone and time-consuming [21]. Moreover, due to the generality of patterns found by malware analysts, signature-based approaches suffer from a non-negligible of false positives. However, many surveyed works propose to extract signatures from call graphs, control flow graphs, and behavior profiles.

5.1.1 Malicious signature matching. Malicious signature matching approaches can be divided into two phases. First, malware signatures are collected inside a knowledge base (KB). In the second phase, signatures extracted samples to analyse are compared with those in the KB. If one or more matches are found, the consequence depends on the objective of the analysis, e.g., samples are marked as malicious, or are assigned a specific label. Malicious signature matching has been used both for malware detection in [27] and malware variants selection in [45] and [70].

5.1.2 Malicious graph matching. Signatures can be also computed from the graphs representing specific features or behaviours of the sample under analysis. Similarly to malicious signature matching, these approaches need an initial phase where graph representations are extracted from a dataset of samples and stored in the KB. The matching procedure, instead, varies from work to work. As an example, while in [24] signatures are extracted from data dependent call graphs (see Section 4.4.1) transformed into strings, Park *et al.* measure the similarity among data dependent graphs by calculating the maximal common subgraph distance [58]. Similarly to Park *et al.*, [45] represent samples behaviour with graphs as well and matching is performed by XORing matrix representations of behavioural graphs. Malicious graph matching is applied also in [70] to generate evasion-resistant signatures.

5.2 Classification

Classification is the task of assigning an observation to a specific category and it is performed through a statistical model called classifier. A classifier takes as input a vector of features representing measurable properties of an observation. In the following, several classifier implementations are discussed.

5.2.1 Rule-based classifier. Rule-based classification relies on a set of conditions consisting in a series of tests that, if successfully passed, allow the classifier to label the instances accordingly. These tests can be connected by logical conjunctions or more general logical expressions [79], as in [68] and [27]. Conditions can be also applied for modelling similarity [32, 49, 69] and distance thresholds exceeding [74], as well as scores. To this end, Lindorfer *et al.* use a rule-base classifier to compute the probability that a sample implements evasion techniques [51].

5.2.2 Bayes Classifier. Bayesian models are usually employed for document classification. Given a document and a fixed set of classes, Bayesian models outputs the predicted class of the document in input. Bayesian models perform best when features are completely independent, boolean, and not redundant. The more the redundant features, the more the classification is biased towards such features. Many surveyed works apply Bayesian models to malware analysis [39, 65, 66, 76, 80].

Naïve Bayes. The naïve Bayes classifier is the simplest among the probabilistic Bayes models. It assumes strong independence among features and normal distribution on feature values. While the first assumption can be modified by using other probability distributions (e.g. Bernoulli), the latter

needs to hold to avoid compromising classification results. [28, 39, 42, 68, 69, 76, 80] employ Naïve Bayes for analysing malware.

Bayesian Network. Conversely to naïve Bayes classifiers, Bayesian networks provide a tool for graphically representing probability distributions in a concise way [79]. Bayesian networks can be drawn as directed acyclic graphs, where nodes represent features and categories, while edges describe conditional dependence between nodes. Nodes are data structures storing a probability distribution over represented feature values. These probabilistic graphical models have been used in [25, 65, 66].

5.2.3 Support Vector Machine (SVM). Support vector machines are binary classifiers employed in a wide range of application fields ranging from control theory, medicine, biology, pattern recognition, and information security. In malware analysis, support vector machines have been applied in a large number of surveyed papers [1, 3, 16, 18, 27, 28, 36, 39, 42–44, 50, 54, 65, 66, 69, 76, 80]. In [23], SVM is employed to compute the optimal weights to associate to used features. These classifiers are able to deal with high-dimensional and sparse data [44]. In order to work with non-linearly separable data, support vector machines rely on kernel methods. Support vector machines are usually bundled with three default kernel functions: polynomial, sigmoid, and radial basis function.

5.2.4 Multiple Kernel Learning. Instead of using a single kernel, multiple kernel learning combines different kernel functions to classify observations. Chosen kernels may either capture complementary aspects of observations under analysis or aggregate features coming from different sources [30]. In [4], Anderson *et al.* combine six kernels, each one corresponding to a different data source (e.g. PE file information, system calls), and leverage multiple kernel learning for detecting malicious samples.

5.2.5 Prototype-based Classification. This approach relies on the concept of prototypes. They correspond to malware activity reports output by sandboxes or obtained by emulators, virtual or physical machines provided with monitoring tools. In [61], malware activity is extracted by means of system calls and by converting them in feature vectors. As discussed in Section 4.4, system calls are representative of samples behaviour. A prototype combines all the feature vectors in groups of homogeneous behaviours, according to the available reports. In the same work, Rieck *et al.* propose an approximation algorithm for extracting prototypes from a dataset of malware activity reports. Classification is performed by extracting the prototype from the sample under analysis and computing its nearest prototype in the dataset. The distance between prototypes is measured by using the Euclidean distance.

5.2.6 Decision Tree. Decision tree classifiers are widely employed in many fields. In general, nodes are meant for testing input features against some learned threshold [79]. One of the main strength of decision trees is that they can be trained using boolean, numeric, or nominal features. During the test phase, feature values guide the path to follow along the tree until a leaf node is reached. The specific instance is classified according to the category assigned to such leaf. In malware analysis, observations are typically related to samples. Works using decision trees are [7, 28, 36, 39, 40, 42, 54, 55, 65, 66, 71, 72, 76]. Interestingly, decision trees can be reduced to rule-based classifiers (see 5.2.1). Indeed, every path in the tree leading to a leaf node can be represented as a set of rules logically in "AND".

Random Forest. These classifiers are ensembles of decision trees, each fed with feature values independently sampled using the same distribution for all trees [13]. Usually classifier ensembles are obtained by means of bagging, boosting, and randomization. Random forest uses bagging

since it introduces randomness in the choice of the features to take into account. Random forest classifiers have been applied in [1, 18, 36, 39, 40, 71, 76, 80].

Gradient Boosting Decision Tree. Conversely to random forest classifiers, gradient boosting decision trees are tree ensembles built by using the boosting technique. They aim to minimize the expected value of a specified loss function on a training set. In [16] and [69], gradient boosting decision trees are used to detect the category of malicious samples.

Logistic Model Tree. Logistic model tree classifiers are decision trees having logistic regression models at their leaves. These models are linear and built on independent variables, representing the considered classes, weighted on the basis of the samples in the training set. Weights are computed by maximizing the log-likelihood. Graziano *et al.* employ logistic model trees for analysing malware submitted to a public sandbox [33], whereas [19, 57, 69] leverage logistic regression classifiers to detect, respectively, families of malware, their categories, and novelties or similarities with respect to other samples.

5.2.7 k-Nearest Neighbors (k-NN). For each labeled d-length feature vector contained in a training set of size n, a k-NN algorithm transforms them in d-dimensional points. Labels can, for example, report if a sample is malicious or benign. In the test phase, given a m-size dataset of samples under analysis, these are transformed into d-dimensional points to find what are their k nearest neighbours by means of a distance or similarity measure (e.g., Euclidean distance). Categories of unknown instances are chosen by picking the most popular class among them. The main advantage of these classifiers is that they require short training times. Using a worst-case analysis model, the time required to train a k-Nearest Neighbor classifier is $\Theta(n \cdot d)$. The test phase has $\Theta(m \cdot n \cdot d)$ time complexity [52]. Classic implementations of k-Nearest Neighbor can be further refined to improve their running time to logarithmic by employing KD-tree data structures.

5.2.8 Artificial Neural Network. Neural networks are computing systems formed by a set of highly interconnected processing units organized in layers. Each processing unit has an activation function and is linked to other units by means of weighted connections that are modified according to a specified learning rule. Artificial neural networks are widely employed in pattern recognition and novelty detection, time series prediction, and in medical diagnoses. They perform best if the system they model is error-tolerant and can be helpful when the relationships among inputs are not clear or difficult to describe with other models. Limitations strongly depend on the used activation function and applied learning rule. Dahl *et al.* take advantage of neural networks for detecting malware families, with unsatisfactory results [19].

Multilayer Perceptron Neural Network. Multilayer Perceptrons are neural networks whose connections are acyclic and each layer is fully connected with the next one. For this reason, they can be represented through directed graphs. These classifiers employ non-linear activation functions and, in the training phase, use backpropagation. In [28], Multilayer Perceptron Neural Networks are used for detecting malware.

5.3 Clustering

Clustering is the process of grouping similar elements. The ideal clustering should arrange similar elements together and they should be distant from other groups, also called clusters. The notion of distance is defined according specific distance or similarity metrics. Clustering methods can be divided into hierarchical, partitioning, soft-computing, density-based, and model-based [62].

5.3.1 Clustering with locality sensitive hashing. Local sensitive hashing is a dimensionality reduction technique for approximating clusters and neighbor search. It relies on locality-sensitive hash families, which map elements into bins. Similar elements are more likely to be mapped to same bucket. Locality-sensitive hash families and, hence local sensitive hashing, are defined only for specific similarity and distance measures such as cosine or Jaccard similarity and Hamming or Euclidean distance. Local sensitive hashing is the building block for grouping similar malicious sample and it has been applied in some works [10, 73, 75].

5.3.2 Clustering with Distance and Similarity Metrics. As already discussed, clustering can be performed by taking into account either distance or similarity among different samples. Several metrics have been used in malware analysis: Euclidean [54, 61] and Hamming distances [54], cosine [54] and Jaccard similarities [54, 59]. Distances can be also computed on signatures extracted by samples using tools such as *ssdeep*[16] and *sdhash*[17]. Both are fuzzy hashing algorithms based on blocks: anytime a sufficient amount of input is processed, a small block is generated. Each of the generated blocks is a portion of the final signature. Samples can be grouped together, in conjunction with other features, on the basis of their signatures obtained with block-based fuzzy hashing, as in [33] and [75].

Time complexity of algorithms based on distances and similarity metrics depends both on the used measures and their implementations. For commonly applied metrics, such as cosine similarity, Euclidean and Hamming distances, the required time to compute them between two d-dimensional points is $O(d)$. Thus, given a dataset of m samples, the time complexity to cluster them on these metrics is $O(d \cdot m^2)$.

5.3.3 k-Means Clustering. k-means is one of the simplest and most used clustering algorithm [62]. It is a variant of the Expectation Maximization algorithm, belongs to the class of partition algorithms and separates data into k clusters. The number of clusters k is chosen *a priori* and initial cluster centers, called centroids, are picked randomly. Iteratively, each instance of the dataset is assigned to its nearest centroid to minimize the least within-cluster sum of squares, that is the squared Euclidean distance. k-means can halt in two cases: the sum of squared error is under a threshold τ or no change occurs for the k clusters. This guarantees to reach a local optimum and convergence in a finite number of iterations. Even if theoretically has been proven that in the worst case k-means has an exponential running time, a relatively recent smoothed analysis has shown that the number of iterations are bounded by a polynomial in the number n of data points [5]. However, in practice, k-means running time is often linear in n. Both Huang *et al.* and Pai *et al.* use k-means clustering for performing family selection as objective of their analyses [35, 56].

5.3.4 Density-based Spatial Clustering of Applications with Noise. DBSCAN is a widely known density-based clustering algorithm, initially proposed by Ester *et al.* for grouping objects in large databases [26]. It is able to efficiently compute clusters of arbitrary shape through a two step process. The first step involves the selection of an entry having in its neighbourhood at least a certain number of other entries (i.e., the *core point*). Its neighbours can be obtained by transforming the database into a collection of points and by then measuring the distance among them. If the distance is lower than a chosen threshold, then the two points are considered neighbours. In the second step, the cluster is built by grouping all the points that are *density-reachable* from the core point. The notion of density-reachability has been defined in [26], and regards the constraints that allow a point to be directly reachable from another. The conditions impose that the former is a core point and the latter is in its neighbourhood. Rather than referring to two single points,

[16] ssdeep: http://ssdeep.sourceforge.net
[17] sdhash: http://roussev.net/sdhash/sdhash.html

density-reachability applies to a path in which points are directly reachable from each other. The algorithm runs on a database storing n entries that can be transformed in n points, and mainly serves neighbourhood queries. These queries can be answered efficiently in $O(\log n)$ (e.g. by using R^*-trees), thus the expected running time of DBSCAN is $O(n \log n)$. Vadrevu *et al.* use DBSCAN to detect variants of malware [77].

5.3.5 Hierarchical Clustering. A hierarchical clustering schema recursively partitions instances and constructs a tree of clusters called dendrogram. The tree structure allows the cluster exploration at different levels of granularity. Hierarchical clustering can be performed in two ways: *agglomerative* and *divisive*. Agglomerative approaches are bottom-up: they start with clusters each having a single element, then *closer* cluster pairs are iteratively merged until a unique cluster contains all the elements. Divisive approaches are top-down: all the elements are initially included in a unique cluster, then they are divided in smaller sub-clusters until clusters with only one element are obtained.

Closeness can be modelled using different criteria: complete-link, single-link, average-link, centroid-link, and Ward's-link [38, 54]. Agglomerative clustering is more used than divisive, mainly because in the worst case it has a running time $O(n^2 \log n)$, while divisive approach is exponential. In malware analysis, hierarchical clustering has been applied in [37, 54].

5.3.6 Prototype-based Clustering. Analogously to what described in Section 5.2.5, prototypes can be also used to cluster malware that are similar among each other [61]. In [61], Rieck *et al.* leverage hierarchical complete linkage clustering technique to group reports (see Section 5.2.5 for prototype/report mapping). The algorithm running time is $O(k^2 \log k + n)$, where k and n are the number of prototypes and reports, respectively. Thus, prototype-based clustering provides a $(n/k)^{\frac{1}{2}}$ time complexity with respect to exact hierarchical clustering running time.

5.3.7 Self-Organizing Maps. Self-organizing maps are useful data explorations tools that can be also used to cluster data. They are applied to a vast range of application fields going from industry, finance, natural sciences, to linguistics [41]. Self-organizing maps can be represented as an ordered regular grid in which more similar models are automatically arranged together in adjacent positions on the grid, far away from less similar models. Model disposition allows to get additional information from the data topographic relationships. This capability is fundamental when dealing with high-dimensional data. In a first phase, self-organizing maps are calibrated using an input dataset. Then, each time a new input instance feeds the map, it is elaborated by the *best-matching model*. Analogously to what described in Section 5.3.2, a model *best-matches* an instance on the basis of a specific similarity or distance measure. Many proposed self-organizing maps rely on different similarity or distance measures (e.g., dot product, Euclidean and Minkowski distances). Self-organizing maps have been used in [16] for analysing malware.

5.4 Others

This subsection presents Machine Learning algorithms that are neither signature-based, nor classification, nor clustering.

5.4.1 Expectation Maximization. Expectation-maximization is a general-purpose statistical iterative method of maximum likelihood estimation used in cases where observations are incomplete. It is often applied also for clustering. Given a set of observations characterizing each element of the dataset, an *expectation step* assigns the element to the most likely cluster, whereas a *maximization step* recomputes centroids. The main strengths of expectation-maximization are stability, robustness to noise, and ability to deal with missing data. Finally, the algorithm has been proven to converge to

a local maximum. Expectation maximization has been employed by Pai *et al.* for detecting malware variants belonging to the same families [56].

5.4.2 Learning with Local and Global Consistency. Learning with local and global consistency is a semi-supervised approach. Information on known labels are spread to neighbours until a global stable state is reached [82]. Learning with local and global consistency has been proved effective on synthetic data, in digit recognition and text categorization [82]. In malware analysis, this learning approach has been applied with good, but not excellent, results in [67].

5.4.3 Belief Propagation. Belief propagation is an approach for performing inference over graphical models (e.g. Bayesian networks) and, in general, graphs. It works iteratively. At each iteration, each pair of inter-connected nodes exchanges messages reporting nodes opinions about their probabilities of being in a certain state. Belief propagation converges when probabilities do not keep changing significantly or after a fixed number of iterations. Both [73] and [15] adapt belief propagation to malware analysis by proposing new mathematical representations. In [15], Chen *et al.* properly tune belief propagation algorithm and improve it with respect to the approach used in [73] and other classification algorithms (e.g., support vector machines and decision trees).

6 CHARACTERIZATION OF SURVEYED PAPERS

In this section we leverage on the discussed objectives (Section 3), feature classes (Section 4) and algorithm types (Section 5) to precisely characterize each reviewed paper. Such characterization is organized by objective: for each objective, we provide an overall view about which works aims at that objective, what machine learning algorithm they use and what feature classes they rely on.

6.1 Malware detection

Table 14 lists all the reviewed works having malware detection as objective. It can be noted that the most used inputs regard

- byte sequences, extracted from either the PE or its disassembled code, and organized in n-grams
- API/system call invocations, derived by executing the samples

Most of the works use more algorithms to find out the one guaranteeing more accurate results.

6.2 Malware variants detection

As explained in Section 3.2, variants detection includes two slightly diverse objectives: given a malware, identifying either its variants or its families. Tables 15 and Table 16 detail surveyed works aiming to identify variants and families, respectively. In both characterizations, APIs and system calls are largely employed as well as their interactions with the environment, modeled by memory, file system, Windows registries, and CPU registers. Some of the surveyed papers, instead, use byte sequences and opcodes, while others add to the analysis features related to sample network activity.

6.3 Malware category detection

These articles focus on the identification of specific threats and, thus, on particular inputs such as byte sequences, opcodes, executable binaries' function lengths, and network activity regarding samples. Table 17 reports the works whose objectives deal with the detection of the specific category of a malware sample.

Paper	Algorithms	Strings	Byte seq.	Ops	APIs Sys. calls	File system	Win. Reg.	CPU reg.	PE file char.	Raised excep.	Network
Schultz *et al* [68]	Rule-based classifier, Naïve Bayes	✓	✓								
Kolter and Maloof [42]	Decision Tree, Naïve Bayes, SVM		✓								
Firdausi *et al.* [28]	Decision Tree, Naïve Bayes, SVM, *k*-NN, Multilayer Perceptron Neural Network				✓	✓	✓				
Anderson *et al.* [3]	SVM		✓		✓						
Santos *et al.* [67]	Learning with Local and Global Consistency		✓								
Anderson *et al.* [4]	Multiple Kernel Learning		✓	✓	✓				✓		
Yonts [81]	Rule-based classifier								✓		
Eskandari *et al.* [25]	Bayesian Network				✓						
Santos *et al.* [66]	Bayesian Network, Decision Tree, *k*-NN, SVM			✓	✓					✓	
Vadrevu *et al.* [78]	Random Forest								✓		✓
Bai *et al.* [7]	Decision Tree, Random Forest								✓		
Kruczkowski and Szynkiewicz [44]	SVM										✓
Tamersoy *et al.* [73]	Clustering with locality sensitive hashing					✓					
Uppal *et al.* [76]	Decision Tree, Random Forest, Naïve Bayes, SVM		✓		✓						
Chen *et al.* [15]	Belief propagation					✓					
Elhadi *et al.* [24]	Malicious graph matching				✓						
Feng *et al.* [27]	Rule-based classifier, Malicious signature matching, SVM		✓								
Ghiasi *et al.* [32]	Rule-based classifier				✓			✓			
Srakaew *et al.* [72]	Decision Tree		✓	✓							
Wüchner *et al.* [80]	Naïve Bayes, Random Forest, SVM		✓		✓	✓	✓				

Table 14. Characterization of surveyed papers having malware detection as objective.

Paper	Algorithms	Byte seq.	Ops	APIs Sys. calls	Memory	File system	Win. Reg.	CPU reg.	PE file char.	Network
Kwon and Lee [45]	Malicious signature matching			✓						
Shosha *et al.* [70]	Malicious signature matching		✓	✓	✓			✓		
Chionis *et al.* [17]	Malicious signature matching			✓		✓	✓			✓
Gharacheh *et al.* [31]	-[18]		✓							
Ghiasi *et al.* [32]	Rule-based classifier			✓				✓		
Khodamoradi *et al.* [40]	Decision Tree, Random Forest		✓							
Upchurch and Zhou [75]	Clustering with locality sensitive hashing	✓								
Liang *et al.* [49]	Rule-based classifier			✓		✓	✓			✓
Vadrevu and Perdisci [77]	DBSCAN clustering			✓					✓	✓

Table 15. Characterization of surveyed papers having malware variants selection as objective. [18]Instead of using machine learning techniques, Gharacheh *et al.* rely on Hidden Markov Models to detect variants of the same malicious sample [31].

Paper	Algorithms	Str.	Byte seq.	Ops	APIs Sys. calls	Mem.	File sys.	Win. Reg.	CPU reg.	Func. length	PE file char.	Raised excep.	Net.
Huang *et al.* [35]	*k*-Means-like algorithm		✓										
Park *et al.* [58]	Malicious graph matching				✓								
Dahl *et al.* [19]	Logistic Regression, Neural Networks		✓		✓								
Hu *et al.* [34]	Prototype-based clustering			✓									
Islam *et al.* [36]	Decision Tree, *k*-NN, Random Forest, SVM	✓			✓					✓			
Kong and Yan [43]	SVM, *k*-NN			✓		✓	✓		✓				
Nari and Ghorbani [55]	Decision Tree												✓
Ahmadi *et al.* [1]	SVM, Random Forest, Gradient Boosting Decision Tree		✓	✓	✓			✓	✓		✓		
Asquith [6]	-[19]				✓	✓	✓				✓	✓	
Lin *et al.* [50]	SVM		✓		✓		✓		✓				
Kawaguchi and Omote [39]	Decision Tree, Random Forest, *k*-NN, Naïve Bayes				✓								
Mohaisen *et al.* [54]	Decision Tree, *k*-NN, SVM, Clustering with with different similarity measures, Hierarchical clustering						✓	✓	✓				✓
Pai *et al.* [56]	*k*-Means, Expectation Maximization			✓									

Table 16. Characterization of surveyed papers having malware families selection as objective.[19]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	Byte seq.	Ops	Func. length	Network
Tian *et al.* [74]	Rule-based classifier			✓	
Siddiqui *et al.* [71]	Decision Tree, Random Forest		✓		
Chen *et al.* [16]	Random Forest, SVM	✓			
Comar *et al.* [18]	Random Forest, SVM				✓
Sexton *et al.* [69]	Rule-based classifier, Logistic Regression, Naïve Bayes, SVM	✓	✓		

Table 17. Characterization of surveyed papers having malware category detection as objective.

Paper	Algorithms	Byte seq.	APIs Sys. calls	Mem.	File sys.	Win. Reg.	CPU reg.	Net.
Bailey *et al.* [8]	Hierarchical clustering with normalized compression distance		✓		✓	✓		✓
Bayer *et al.* [10]	Clustering with locality sensitive hashing		✓					
Rieck *et al.* [61]	Prototype-based classification and clustering with Euclidean distance	✓	✓					
Palahan *et al.* [57]	Logistic Regression		✓					
Egele *et al.* [23]	SVM[20]		✓	✓			✓	

Table 18. Characterization of surveyed papers having malware similarities detection as objective. [20]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	Byte seq.	Ops	APIs Sys. calls	Network
Bayer *et al.* [10]	Clustering with locality sensitive hashing			✓	
Lindorfer *et al.* [51]	Rule-based classifier			✓	✓
Rieck *et al.* [61]	Prototype-based classification and clustering, clustering with Euclidean distance	✓		✓	
Palahan *et al.* [57]	Logistic Regression			✓	
Santos *et al.* [65]	Decision Tree, *k*-NN, Bayesian Network, Random Forest		✓		
Polino *et al.* [59]	Clustering with Jaccard similarity			✓	

Table 19. Characterization of surveyed papers having malware differences detection as objective.

6.4 Malware novelty and similarity detection

Similarly to Section 6.2, this characterization groups works whose objective is to spot (dis)similarities among samples. According to the final objective of the analysis, one can be more interested in

finding out either similarities or differences characterizing a group of samples. All the analysed papers but [65] rely on APIs and system call collection (see Table 18 and Table 19).
While works aiming to spotting differences among PEs, in general, do not take into account interactions with the system in which they are executed, the ones that identify similarities leverage such information.

6.5 Malware development detection

Table 20 outlines the very few works that propose approaches to deal with malware development cases. While Chen *et al.* use just byte sequences for their analysis [16], in [33], both information related to malicious sample execution into a sandbox and their submission-related metadata are used.

Paper	Algorithms	Byte seq.	APIs Sys. calls	File sys.	Win. Reg.	Net.	Submissions
Chen *et al.* [16]	Gradient Boosting Decision Tree, Self-Organizing Maps, SVM	✓					
Graziano *et al.* [33]	Logistic Model Tree, Clustering using *ssdeep* tool		✓	✓	✓	✓	✓

Table 20. Characterization of surveyed papers having malware development detection as objective.

6.6 Malware attribution

Malware attribution is one of the most important tasks at the strategic level (see Section 3.6). The U.S. government has allocated research funds for the next decade to be able to attribute cyber threats to specific actors or groups, active in the cyberwarfare. In addition to the typical inputs used in malware analysis, Caliskan-Islam *et al.* rely on code stylometry [14] (refer to Table 21).

Paper	Algorithms	Code stylometry
Caliskan-Islam *et al.* [14]	Random Forest	✓

Table 21. Characterization of surveyed papers having malware attribution as objective.

6.7 Malware triage

Even if serious attention has been paid on malware detection in general, much less consideration is given to malware triage, as shown in Table 22. Jang *et al.* are the only ones, among the surveyed

Paper	Algorithms	Byte seq.	APIs Sys. calls
Jang *et al.* [37]	Clustering with locality sensitive hashing, full hierarchical clustering	✓	✓

Table 22. Characterization of surveyed papers having malware triage as objective.

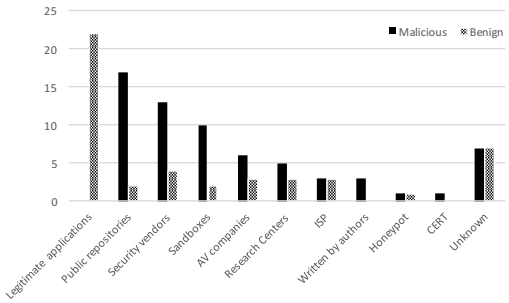

Fig. 1. Dataset sources for malicious and benign samples

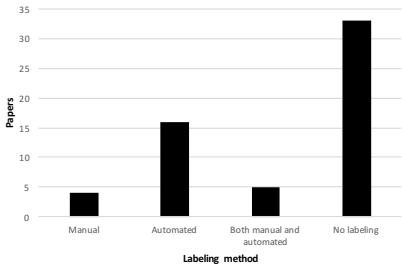

Fig. 2. Dataset labeling methods

works, that perform triage by using PE's byte sequences and API/system call invocations [37].

7 DATASETS

In the vast majority of surveyed works, datasets contain both malicious and benign samples. Nevertheless, several papers perform their experimental evaluations using datasets having solely malicious executables. The objectives of these works are *variants and families selection* [1, 34, 35, 43, 48, 49, 54, 55, 70, 75, 77], *category detection* [74], *malware novelty and similarity detection* [8, 10, 51, 61]. Just two works rely on benign datasets only [14, 23]. Since their objectives are identifying sample similarities and attributing the ownership of some source codes under analysis, respectively, then they do not necessarily need malware.

Figure 1 reports a summary of the employed sources for malicious and benign samples, respectively. Used datasets come from legitimate applications, public repositories, security vendors, sandboxes, AV companies and research centers, Internet Service Providers, honeypots, CERT and, in some cases, datasets are built by the researchers themselves.

It is worth noting that most of the benign datasets consist of legitimate applications, while most of malware have been obtained from public repositories, security vendors and popular sandboxed analysis services. Legitimate applications include PEs contained in the "Program Files" or "system" folders, and downloads from "trusted" (i.e. signed) companies. Examples of these benign programs are Cygwin, Putty, the Microsoft Office Suite, and Adobe Acrobat. The most popular public repository in the examined works is VX Heavens[21], followed by Offensive Computing[22] and Malicia Project[23]. The first two repositories are still actively maintained at the time of writing, while Malicia Project has been permanently shut down due to dataset aging 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 and Anubis sandboxed analysis services. CWSandbox is a commercial sandbox, now named Threat Analyzer. It is actively developed by ThreatTrack Security[24]. Anubis is no more maintained[25] by their creators and by iSecLab[26], which was the international laboratory hosting the sandbox. As can be observed from Figure 1, these

[21] VX Heavens: http://vxheaven.org
[22] Offensive Computing: http://www.offensivecomputing.net
[23] Malicia Project: http://malicia-project.com
[24] ThreatTrack: https://www.threattrack.com/malware-analysis.aspx
[25] Anubis: https://anubis.iseclab.org
[26] iSecLab: https://iseclab.org

sandboxes are mainly used for obtaining malicious samples. The same trend holds for security vendors, AV companies and research centers. Internet Service Providers (ISPs), honeypots, and Computer Emergency Response Teams (CERTs) share with researchers both benign and malicious datasets.

An interesting case is represented by samples developed by the researchers themselves. A few works use in their evaluations malicious samples developed by the authors [31, 40, 70]. These samples are created by also using malware toolkits such as Next Generation Virus Constrution Kit[27], Virus Creation Lab[28], Mass Code Generator[29], and Second Generation Virus Generator[30]. Finally, a minority of analysed papers do not mention the source of their datasets.

One of the most common problems of these datasets is that, very often, they are not balanced. A proper training of machine learning models require that each class contains an almost equal amount of samples, otherwise inaccurate models could be obtained. The same problem holds when also benign datasets are used, indeed malicious samples should be roughly as many as benign samples. In [81], Yonts supports his choice of using a smaller benign dataset by pointing out that changes in standard system files and legitimate applications are little. However, there are surveyed works that rely on benign datasets whose size is bigger than the size of malicious ones [12, 15, 40, 42, 69, 71, 73, 76].

Samples contained in the datasets used in considered papers are already labeled in general. Figure 2 reports statistics on whether considered works perform further labeling on these samples. The majority of reviewed papers does not perform any additional labeling operation to their already-labeled datasets. However, some works analyse samples again and recompute labels to check if they match with the ones provided together with the datasets. Label computation can be manual, automated, or both. Manual labeling is a highly time-consuming task, but provides more accurate results since this activity is usually performed by security experts. Consequently, the number of samples that can be labeled using this method is quite small compared to automated labeling techniques. Few works use manual labeling [19, 33, 59, 75], while others combine manual and automated methods [10, 34, 51, 54].

Differently from other research fields, in malware analysis there are no available reference benchmarks to compare accuracy and performance with respect to previous works. In addition, since the datasets used for experimental evaluations are rarely shared, it is difficult, if not impossible, to compare works. In the papers we have surveyed, only two works have shared the dataset used in their evaluations [68, 75], while a third one plans to provide a reference to the analysed malware samples in the future [54]. To this end, Upchurch *et al.* [75] share a golden standard test dataset for future works that aim to perform malware variants selection analyses. The dataset is imbalanced and only includes 85 samples, organized in 8 families containing trojans, information stealers, bots, keyloggers, backdoors, and potentially unwanted programs. All the above samples have been analysed by professional malware analysts and tested against 5 different malware variant detection approaches. Experimental evaluations report performance and accuracy of tested methodologies and compare obtained results with the ones published in the original papers. Sample metadata include MD5 hashes, but no temporal information, i.e., when a sample appeared first. Miller *et al.* have extensively proved that the lack of this critical information considerably affects the accuracy of experimental results [53].

[27]Next Generation Virus Construktion Kit: http://vxheaven.org/vx.php?id=tn02
[28]Virus Creation Lab: http://vxheaven.org/vx.php?id=tv03
[29]Mass Code Generator: http://vxheaven.org/vx.php?id=tm02
[30]Second Generation Virus Generator: http://vxheaven.org/vx.php?id=tg00

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, while benchmarks for category detection should label malware with their categories.
(2) Benchmarks should be balanced: samples of different classes should be in equal proportion to avoid issues on classification accuracy.
(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. In this way analysts would have at disposal a sort of timeline of malware evolution and they could also discard obsolete samples.

8 MALWARE ANALYSIS ECONOMICS

Analysing samples through machine learning techniques requires complex computations, both for extracting desired features and for running chosen algorithms. The time complexity of these computations has to be carefully taken into account because of the need to complete them 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 accuracy of the analysis is likely to worsen, while in the latter accuracy levels can be kept at the cost of providing additional means, e.g., in terms of computing machines, storage, network. There exist tradeoffs 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 tradeoffs 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. There exist in literature plenty of works discussing time complexity of machine learning algorithms. The same does not apply for the study of the execution time of the feature extraction process. The main aspect to take into account in such study 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. Table 23 categorizes surveyed works on the basis of the type of analysis they carry out, i.e., static, dynamic or hybrid. The majority of works relies on dynamic analyses, while the others use, in equal proportions, either static analyses alone or a combination of static and dynamic techniques.

Table 23. Surveyed papers arranged according the type of analysis performed on input samples.

Malware analysis	Papers
Static	[1, 7, 14–16, 27, 31, 34, 35, 40, 42, 43, 45, 56, 65, 67–69, 71–75, 78, 81]
Dynamic	[3, 8, 10, 18, 19, 24, 28, 32, 39, 44, 48–51, 54, 55, 57, 58, 61, 70, 76, 80]
Hybrid	[4, 17, 23, 25, 33, 36, 37, 59, 66, 77]

Table 24. Type of analysis required for extracting the inputs presented in Section 4.

Required analysis	Str.	Byte seq.	Ops	APIs Sys. calls	Mem.	File sys.	Win. Reg.	CPU reg.	Func. len.	PE file char.	Raised excep.	Net.	AV/Sand. submis-sions
Static	✓	✓	✓	✓				✓	✓	✓			✓
Dynamic				✓	✓	✓	✓	✓			✓	✓	✓

To deepen even further this point, Table 24 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 make the overall analysis improve its effectiveness [20]. As an example, we can consider the features derived from API calls (see Table 24), 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.

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.* [46].

The qualitative tradeoffs 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 tradeoffs 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. 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 claim the significance of investigating these tradeoffs more in details, with the aim of outlining proper guidelines and strategies to design a malware analysis environment in compliance with requirements on analysis accuracy and pace, and also by respecting budget constraints.*

9 CONCLUSION

We presented a survey on existing literature on malware analysis through machine learning techniques. There are three 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*. We identified 7 different malware analysis objectives (ranging from malware detection to malware triage), grouped features according to their specific type (e.g., strings and byte sequences), and organized machine learning algorithms for malware analysis in 4 distinct classes. 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. In this first contribution, we discussed the general lack of justifications for using a specific set of features to properly describe the malicious traits of samples: the majority of reviewed papers did not explain the correlation between considered features and obtained results.

Second, we highlighted some issues regarding the *datasets* used in literature and outlined three desiderata for building enhanced datasets. Currently, there is a shortage of publicly available datasets suitable for specific objectives. For example, datasets where samples are properly labelled by family are a rarity. Furthermore, usually, datasets employed in reviewed experimental evaluations are rarely shared. In the majority of examined papers, used datasets are not balanced, hence preventing the construction of really accurate models. When malware samples are to be used for evaluating novel analysis techniques, their fast obsolescence becomes an additional and relevant issue. Indeed, the effectiveness of new approaches should be tested on samples as much recent as possible, otherwise there would be the risk that such approaches turn out to be poorly accurate when applied in the real world. At today's malware evolution pace, samples are likely to become outdated in a few months, but reference datasets commonly include malware of a few years ago. Thus, we proposed three desired characteristics for malware analysis benchmarks: they should be (*i*) labeled accordingly to the specific objectives to achieve, (*ii*) balanced, (*iii*) actively maintained and updated over time.

Third, we introduced the novel concept of *malware analysis economics*, concerning the investigation and exploitation of existing tradeoffs between performance metrics of malware analysis (e.g., analysis accuracy and execution time) and economical costs. We have identified tradeoffs between accuracy, time complexity, analysis pace with respect to malware evolution, required computational resources, and economic cost. Similar tradeoffs also hold for space complexity.

Noteworthy research directions to investigate can be linked to each of the three contributions. The organization of malware analysis works along three dimensions can be further refined by better identifying and characterizing analysis objectives, extracted features, and used machine learning algorithms. Novel combinations of objectives, features and algorithms can be investigated to achieve better performance 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 datasets can drive academic works aimed at building new datasets in accordance with the three identified desiderata. Providing better datasets would enable better and fairer comparisons among results claimed by diverse works, hence would ease effective progresses in the malware analysis field. Finally, the initial set of general tradeoffs described in the context of malware analysis economics can be deepened to derive quantitative relationships among the key metrics of interest, which would allow defining effective approaches to design and setup analysis environments.

REFERENCES

[1] Mansour Ahmadi, Giorgio Giacinto, Dmitry Ulyanov, Stanislav Semenov, and Mikhail Trofimov. 2015. Novel feature extraction, selection and fusion for effective malware family classification. *CoRR* abs/1511.04317 (2015). http://arxiv.org/abs/1511.04317

[2] Frances E. Allen. 1970. Control Flow Analysis. In *Proceedings of a Symposium on Compiler Optimization*. ACM, New York, NY, USA, 1–19. https://doi.org/10.1145/800028.808479

[3] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. 2011. Graph-based malware detection using dynamic analysis. *Journal in Computer Virology* 7, 4 (2011), 247–258.

[4] Blake Anderson, Curtis Storlie, and Terran Lane. 2012. Improving malware classification: bridging the static/dynamic gap. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. ACM, 3–14.

[5] D. Arthur, B. Manthey, and H. Röglin. 2009. k-Means Has Polynomial Smoothed Complexity. In *FOCS '09*. 405–414. https://doi.org/10.1109/FOCS.2009.14

[6] Matthew Asquith. 2015. Extremely scalable storage and clustering of malware metadata. *Journal of Computer Virology and Hacking Techniques* (2015), 1–10.

[7] Jinrong Bai, Junfeng Wang, and Guozhong Zou. 2014. A malware detection scheme based on mining format information. *The Scientific World Journal* 2014 (2014).

[8] Michael Bailey, Jon Oberheide, Jon Andersen, Z Morley Mao, Farnam Jahanian, and Jose Nazario. 2007. Automated classification and analysis of internet malware. In *Recent advances in intrusion detection*. Springer, 178–197.

[9] Ishita Basu. 2016. Malware Detection Based on Source Data using Data Mining: A Survey. *American Journal Of Advanced Computing* 3, 1 (2016).

[10] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. 2009. Scalable, Behavior-Based Malware Clustering. In *NDSS*, Vol. 9. Citeseer, 8–11.

[11] Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, and Ali Hamzeh. 2013. A survey on heuristic malware detection techniques. In *Information and Knowledge Technology (IKT), 2013 5th Conference on*. IEEE, 113–120.

[12] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. 2012. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *ACSAC '12*. ACM, 129–138.

[13] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (Oct. 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[14] Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. 2015. De-anonymizing Programmers via Code Stylometry. In *USENIX Security '15*. USENIX Association, Washington, D.C., 255–270. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/caliskan-islam

[15] L. Chen, T. Li, M. Abdulhayoglu, and Y. Ye. 2015. Intelligent malware detection based on file relation graphs. In *Semantic Computing (ICSC), 2015 IEEE International Conference on*. 85–92. https://doi.org/10.1109/ICOSC.2015.7050784

[16] Zhongqiang Chen, Mema Roussopoulos, Zhanyan Liang, Yuan Zhang, Zhongrong Chen, and Alex Delis. 2012. Malware characteristics and threats on the internet ecosystem. *Journal of Systems and Software* 85, 7 (2012), 1650–1672.

[17] Ioannis Chionis, Stavros Nikolopoulos, and Iosif Polenakis. 2013. A Survey on Algorithmic Techniques for Malware Detection. (2013).

[18] P. M. Comar, L. Liu, S. Saha, P. N. Tan, and A. Nucci. 2013. Combining supervised and unsupervised learning for zero-day malware detection. In *INFOCOM, 2013 Proceedings IEEE*. 2022–2030. https://doi.org/10.1109/INFCOM.2013.6567003

[19] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. 2013. Large-scale malware classification using random projections and neural networks. In *Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 3422–3426.

[20] Anusha Damodaran, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H Austin, and Mark Stamp. 2015. A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques* (2015), 1–12.

[21] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. 2008. A Survey on Automated Dynamic Malware-analysis Techniques and Tools. *ACM Comput. Surv.* 44, 2, Article 6 (March 2008), 42 pages. https://doi.org/10.1145/2089125.2089126

[22] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. 2012. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)* 44, 2 (2012), 6.

[23] Manuel Egele, Maverick Woo, Peter Chapman, and David Brumley. 2014. Blanket Execution: Dynamic Similarity Testing for Program Binaries and Components. In *USENIX Security '14*. USENIX Association, San Diego, CA, 303–317. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/egele

[24] Erbiai Elhadi, Mohd Aizaini Maarof, and Bazara Barry. 2015. Improving the detection of malware behaviour using simplified data dependent api call graph. *Journal of Security and Its Applications* (2015).

[25] Mojtaba Eskandari, Zeinab Khorshidpour, and Sattar Hashemi. 2013. Hdm-analyser: a hybrid analysis approach based on data mining techniques for malware detection. *Journal of Computer Virology and Hacking Techniques* 9, 2 (2013), 77–93.

[26] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, 226–231.

[27] Zhentan Feng, Shuguang Xiong, Deqiang Cao, Xiaolu Deng, Xin Wang, Yang Yang, Xiaobo Zhou, Yan Huang, and Guangzhu Wu. 2015. HRS: A Hybrid Framework for Malware Detection. In *Proceedings of the 2015 ACM International Workshop on Security and Privacy Analytics*. ACM, 19–26.

[28] Ivan Firdausi, Charles Lim, Alva Erwin, and Anto Satriyo Nugroho. 2010. Analysis of machine learning techniques used in behavior-based malware detection. In *ACT '10*. IEEE, 201–203.

[29] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. 2014. Malware analysis and classification: A survey. *Journal of Information Security* 2014 (2014).

[30] Mehmet Gönen and Ethem Alpaydin. 2011. Multiple Kernel Learning Algorithms. *J. Mach. Learn. Res.* 12 (July 2011), 2211–2268. http://dl.acm.org/citation.cfm?id=1953048.2021071

[31] M. Gharacheh, V. Derhami, S. Hashemi, and S. M. H. Fard. 2015. Proposing an HMM-based approach to detect metamorphic malware. In *Fuzzy and Intelligent Systems (CFIS)*. 1–5. https://doi.org/10.1109/CFIS.2015.7391648

[32] Mahboobe Ghiasi, Ashkan Sami, and Zahra Salehi. 2015. Dynamic VSA: a framework for malware detection based on register contents. *Engineering Applications of Artificial Intelligence* 44 (2015), 111 – 122. https://doi.org/10.1016/j.engappai.2015.05.008

[33] Mariano Graziano, Davide Canali, Leyla Bilge, Andrea Lanzi, and Davide Balzarotti. 2015. Needles in a Haystack: Mining Information from Public Dynamic Analysis Sandboxes for Malware Intelligence. In *USENIX Security '15*. 1057–1072. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/graziano

[34] Xin Hu, Kang G Shin, Sandeep Bhatkar, and Kent Griffin. 2013. MutantX-S: Scalable Malware Clustering Based on Static Features. In *USENIX Annual Technical Conference*. 187–198.

[35] Kai Huang, Yanfang Ye, and Qinshan Jiang. 2009. Ismcs: an intelligent instruction sequence based malware categorization system. In *Anti-counterfeiting, Security, and Identification in Communication, 2009*. IEEE, 509–512.

[36] Rafiqul Islam, Ronghua Tian, Lynn M Batten, and Steve Versteeg. 2013. Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Applications* 36, 2 (2013), 646–656.

[37] Jiyong Jang, David Brumley, and Shobha Venkataraman. 2011. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Computer and communications security*. ACM, 309–320.

[38] Stephen C. Johnson. 1967. Hierarchical clustering schemes. *Psychometrika* 32, 3 (1967), 241–254. https://doi.org/10.1007/BF02289588

[39] Naoto Kawaguchi and Kazumasa Omote. 2015. Malware Function Classification Using APIs in Initial Behavior. In *Information Security (AsiaJCIS), 2015 10th Asia Joint Conference on*. IEEE, 138–144.

[40] Peyman Khodamoradi, Mahmood Fazlali, Farhad Mardukhi, and Masoud Nosrati. 2015. Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms. In *Computer Architecture and Digital Systems (CADS), 2015 18th CSI International Symposium on*. IEEE, 1–6.

[41] Teuvo Kohonen. 2013. Essentials of the Self-organizing Map. *Neural Netw.* 37 (Jan. 2013), 52–65. https://doi.org/10.1016/j.neunet.2012.09.018

[42] J. Zico Kolter and Marcus A. Maloof. 2006. Learning to Detect and Classify Malicious Executables in the Wild. *J. Mach. Learn. Res.* 7 (Dec. 2006), 2721–2744. http://dl.acm.org/citation.cfm?id=1248547.1248646

[43] Deguang Kong and Guanhua Yan. 2013. Discriminant Malware Distance Learning on Structural Information for Automated Malware Classification. In *ACM SIGKDD '13 (KDD '13)*. ACM, New York, NY, USA, 1357–1365. https://doi.org/10.1145/2487575.2488219

[44] Michal Kruczkowski and Ewa Niewiadomska Szynkiewicz. 2014. Support vector machine for malware analysis and classification. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*. IEEE Computer Society, 415–420.

[45] Jonghoon Kwon and Heejo Lee. 2012. Bingraph: Discovering mutant malware using hierarchical semantic signatures. In *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*. IEEE, 104–111.

[46] Giuseppe Laurenza, Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2016. An Architecture for Semi-Automatic Collaborative Malware Analysis for CIs. In *Dependable Systems and Networks Workshop, 2016 46th Annual IEEE/IFIP International Conference on*. IEEE, 137–142.

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

[48] Tony Lee and Jigar J Mody. 2006. Behavioral classification. In *EICAR Conference*. 1–17.

[49] Guanghui Liang, Jianmin Pang, and Chao Dai. 2016. A Behavior-Based Malware Variant Classification Technique. *International Journal of Information and Education Technology* 6, 4 (2016), 291.

[50] Chih-Ta Lin, Nai-Jian Wang, Han Xiao, and Claudia Eckert. 2015. Feature Selection and Extraction for Malware Classification. *Journal of Information Science and Engineering* 31, 3 (2015), 965–992.

[51] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. 2011. Detecting environment-sensitive malware. In *Recent Advances in Intrusion Detection*. Springer, 338–357.

[52] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[53] Brad Miller, Alex Kantchelian, S Afroz, R Bachwani, R Faizullabhoy, L Huang, V Shankar, MC Tschantz, Tony Wu, George Yiu, et al. 2015. Back to the future: Malware detection with temporally consistent labels. *CORR* (2015).

[54] Aziz Mohaisen, Omar Alrawi, and Manar Mohaisen. 2015. Amal: High-fidelity, behavior-based automated malware analysis and classification. *Computers & Security* (2015).

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

[56] Swathi Pai, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H. Austin, and Mark Stamp. 2015. Clustering for malware classification. (2015).

[57] Sirinda Palahan, Domagoj Babić, Swarat Chaudhuri, and Daniel Kifer. 2013. Extraction of statistically significant malware behaviors. In *Computer Security Applications Conference*. ACM, 69–78.

[58] Younghee Park, Douglas Reeves, Vikram Mulukutla, and Balaji Sundaravel. 2010. Fast malware classification by automated behavioral graph matching. In *Workshop on Cyber Security and Information Intelligence Research*. ACM, 45.

[59] Mario Polino, Andrea Scorti, Federico Maggi, and Stefano Zanero. 2015. Jackdaw: Towards Automatic Reverse Engineering of Large Datasets of Binaries. In *Detection of Intrusions and Malware, and Vulnerability Assessment (Lecture Notes in Computer Science)*, Magnus Almgren, Vincenzo Gulisano, and Federico Maggi (Eds.). Springer International Publishing, 121–143. http://link.springer.com/chapter/10.1007/978-3-319-20550-2_7 DOI: 10.1007/978-3-319-20550-2_7.

[60] Hal Pomeranz. 2012. Detecting Malware With Memory Forensics. http://www.deer-run.com/~hal/Detect_Malware_w_Memory_Forensics.pdf. (2012). Accessed: 2016-11-28.

[61] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. 2011. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security* 19, 4 (2011), 639–668.

[62] Lior Rokach and Oded Maimon. 2005. *Clustering Methods*. Springer US, Boston, MA, 321–352. https://doi.org/10.1007/0-387-25465-X_15

[63] B. G. Ryder. 1979. Constructing the Call Graph of a Program. *Transactions on Software Engineering* SE-5, 3 (May 1979), 216–226. https://doi.org/10.1109/TSE.1979.234183

[64] Manish Kumar Sahu, Manish Ahirwar, and A Hemlata. 2014. A Review of Malware Detection Based on Pattern Matching Technique. *Int. J. of Computer Science and Information Technologies (IJCSIT)* 5, 1 (2014), 944–947.

[65] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G Bringas. 2013. Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences* 231 (2013), 64–82.

[66] Igor Santos, Jaime Devesa, Félix Brezo, Javier Nieves, and Pablo Garcia Bringas. 2013. Opem: A static-dynamic approach for machine-learning-based malware detection. In *CISIS '12-ICEUTE´ 12-SOCO´*. Springer, 271–280.

[67] Igor Santos, Javier Nieves, and Pablo G. Bringas. 2011. *International Symposium on Distributed Computing and Artificial Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter Semi-supervised Learning for Unknown Malware Detection, 415–422. https://doi.org/10.1007/978-3-642-19934-9_53

[68] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo. 2001. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on*. 38–49. https://doi.org/10.1109/SECPRI.2001.924286

[69] Joseph Sexton, Curtis Storlie, and Blake Anderson. 2015. Subroutine based detection of APT malware. *Journal of Computer Virology and Hacking Techniques* (2015), 1–9. https://doi.org/10.1007/s11416-015-0258-7

[70] Ahmed F Shosha, Cong Liu, Pavel Gladyshev, and Marcus Matten. 2012. Evasion-resistant malware signature based on profiling kernel data structure objects. In *CRiSIS, 2012*. IEEE, 1–8.

[71] Muazzam Siddiqui, Morgan C. Wang, and Joohan Lee. 2009. Detecting Internet Worms using Data Mining Techniques. *Journal of Systemics, Cybernetics and Informatics* (2009), 48–53.

[72] Sathaporn Srakaew, Warot Piyanuntcharatsr, and Suchitra Adulkasem. 2015. On the Comparison of Malware Detection Methods Using Data Mining with Two Feature Sets. *Journal of Security and Its Applications* 9 (2015), 293–318.

[73] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. 2014. Guilt by association: large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*. ACM, 1524–1533.

[74] R. Tian, L. M. Batten, and S. C. Versteeg. 2008. Function length as a tool for malware classification. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*. 69–76. https://doi.org/10.1109/MALWARE.2008.4690860

[75] Jason Upchurch and Xiaobo Zhou. 2015. Variant: a malware similarity testing framework. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 31–39.

[76] Dolly Uppal, Roopak Sinha, Vishakha Mehra, and Vinesh Jain. 2014. Malware detection and classification based on extraction of API sequences. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on*. IEEE, 2337–2342.

[77] Phani Vadrevu and Roberto Perdisci. 2016. MAXS: Scaling Malware Execution with Sequential Multi-Hypothesis Testing. In *ASIA CCS '16 (ASIA CCS '16)*. ACM, New York, NY, USA, 771–782. https://doi.org/10.1145/2897845.2897873

[78] Phani Vadrevu, Babak Rahbarinia, Roberto Perdisci, Kang Li, and Manos Antonakakis. 2013. *Measuring and Detecting Malware Downloads in Live Network Traffic*. Springer Berlin Heidelberg, Berlin, Heidelberg, 556–573. https://doi.org/10.1007/978-3-642-40203-6_31

[79] Ian H. Witten and Eibe Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[80] Tobias Wüchner, Martín Ochoa, and Alexander Pretschner. 2015. Robust and Effective Malware Detection Through Quantitative Data Flow Graph Metrics. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 98–118.

[81] Joel Yonts. 2012. *Attributes of Malicious Files.* Technical Report. The SANS Institute.
[82] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16.* MIT Press, 321–328.