Computers & Security 128 (2023) 103177

Contents lists available at ScienceDirect

Computers & Security

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

RecMaL: Rectify the malware family label via hybrid analysis

Wang Yang[a,b], Mingzhe Gao[a,b,*], Ligeng Chen[c], Zhengxuan Liu[a,b], Lingyun Ying[d]

[a] School of Cyber Science and Engineering, Southeast University, China
[b] Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing, China
[c] State Key Laboratory for Novel Software Technology, Nanjing University, China
[d] QI-ANXIN Technology Research Institute, China

ARTICLE INFO

Article history:
Received 6 September 2022
Revised 30 January 2023
Accepted 7 March 2023
Available online 10 March 2023

Keywords:
Malware family
Hybrid analysis
AV labels
Behavioral semantics
Label rectification

ABSTRACT

Intelligent applications can be significantly impacted by incorrectly categorized data. Recently, artificial intelligence technology has been deployed in an increasing number of security-related scenarios, but the issue of data mislabeling has received little attention. We concentrate on the problem of malware mislabeling in this paper. Unfortunately, in the security field, the mislabeling issue of malware is not taken seriously. Existing work attempts to aggregate the AV labels to alleviate malware mislabeling. This will mislead the security analyst and pass the error to subsequent data-driven applications. Therefore, we conduct an in-depth analysis to explore the severity of the malware mislabel issue, and try to rectify the description of malware generated from anti-virus engines. We first propose a malware label correction tool called RecMaL. It employs hybrid analyses for malware label rectifying.

According to the thorough exploratory analysis, we figure out the core reasons for mislabeling issues and summarize them into 3 types. To verify the effectiveness and how RecMaL benefits the downstream applications (e.g., malware classification), we evaluate RecMaL through a series of experiments and show that the main components of RecMaL improve the performance, which proves our method effectively alleviates the mislabeling issue.

1. Introduction

Background of Malware Taxonomy. To facilitate the analysis, malware is taxonomized into different malware *families*. Specifically, malware with the same or similar malicious behavior and functionality is categorized into a single family.

Due to the rapid development of anti-virus engines, the family information of malware can be obtained from the anti-virus labels (abbreviated as *AV labels*[1]) generated by various vendors. A standard AV label consists of platform, malware type, malware family, variant component, and additional information that is prepended or appended to the name. Such as the malware label format used by Microsoft is as follows: $< Type : Platform/Family.Variant!Suffixes >$. The combination of the above profiles provides the malware repository, the malware analysis services, and the malware hunting services (e.g., JoeSandbox; Kaczmarczyck et al., 2020; VirusTotal, 2023b) to search and associate malware samples of interest to security researchers. Moreover, it can provide important context for network defenders and help them define and prioritize countermeasures.

The challenge of malware labelling. At present, the main way to obtain malware labels is the engine of security vendors. However, due to various reasons, the robustness and accuracy of the labels are affected. After our exploration, we find that there are mainly the following aspects leading to the problem.

1. There is no uniform naming convention for malware specimens (Maggi et al., 2011), which makes it possible that the same malware family has different representations.
2. Different vendors may employ different techniques to diagnose malware, leading to diverse detection results.
3. Malware mutating rapidly will quickly degrade the detection ability of vendors.
4. Malware developers deploy specific countermeasures over malware to disable the detection mechanism.

Perturbation on Learning Model. Although the power of artificial intelligence has been unleashed, some subtle pitfalls in machine learning will undermine the performance of the system and affect the practical deployment, especially the label inaccurate issue. Noisy labels are a common problem in machine learning and a source of bias (Northcutt et al., 2021b). Once mistakes

* Corresponding author.
E-mail address: mzgao@njnet.edu.cn (M. Gao).

[1] The scan result of the Anti-Virus engine.

https://doi.org/10.1016/j.cose.2023.103177
0167-4048/

are made, intelligent picture recognition and natural language processing may only impact the user's experience. However, in the scenario of malware detection, label mistakes that result in false positives or false negatives are likely to induce a variety of host infections or even paralysis. We hope that in the field of malware research, just like the mainstream field of artificial intelligence, after completing tag cleaning, more accurate data correspondence can be provided for downstream tasks (Northcutt et al., 2021b), so as to better serve downstream data-driven tasks.

Solution. Both academia and industry have made numerous attempts to tackle the aforementioned issues. Some recent works in academia such as AVCLASS (Sebastián et al., 2016), AVCLASS2 (Sebastián and Caballero, 2020) and Euphony (Hurier et al., 2017) try to summarize the profile by extracting family labels from different AV labels, which makes it feasible to classify or index large-scale malware samples. However, the extracted result is mainly maintained by the empirical threshold, inevitably flawed. Similarly, in industry, VirusTotal, an open-source threat intelligence query site affiliated with Google, gives suggested threat labels for the uploaded samples, such as *"worm.midie/vobfus"*. While this is similar to the functionality of the AVCLASS tool, in the end, it still does not address the issue of family aliases for malware. Meanwhile, according to the latest anti-virus engine measurement research (Zhu et al., 2020), the relationship between anti-virus engines remains convoluted and prone to blindness. Hence, it is urgent to design suitable automated tools to maintain the knowledge base of malware family names based on the invariance of the malware itself.

Goals and Approaches. To mitigate the impact of the aforementioned problems, we delve deeper and try to solve them. In this paper, we adopt an in-depth analysis via static analysis and dynamic analysis, aiming to explore the seriousness of AV labels error and try to rectify them. To meet this end, we propose a framework called RecMaL.[2] The core insight behind the key design is that the same behavior patterns on large-scale malware have unique mapping associations with family labels rather than other factors. By clustering different semantic patterns of malicious behaviors, it can identify malware sets in the same behavior pattern to find label errors in malware datasets.

Evaluation. For the experiment, we used BODMAS dataset (Yang et al., 2021a), the latest open source malware dataset in academia. The BODMAS dataset contains 57,293 malware samples collected from August 2019 to September 2020, with carefully curated family information (581 families). From a cross-comparison experiment, we found that close to 20% of the malware had shifted or incorrect family labels. And to simulate the ground truth of the BODMAS dataset, we introduced 6 different family label systems including the original BODMAS labels for evaluating the quality of RecMaL clustering (see Section 6.4 for details). The results of the ablation experiments show that the clustering quality of RecMaL is outperforming the state-of-the-art tools.

The main contributions of our work are as follows:

- Based on our daily work observation, we find that malware is always partially mislabeled by anti-virus engines. To figure out the root cause, we adopt an in-depth analysis to empirically study the problem. And we reduce the problem into 3 types of mislabeling issues, further deduce the root cause and raise some suggestions for the community.
- To overcome the aforementioned problem, we apply a framework called RecMaL via hybrid analyses, for rectifying the mislabeled part of malware based on the unique invariance of malicious behavior. We translate the underlying series of malware calls to a sequence of high-level behavioral semantics to limit the impact of the dynamic evolution of the same malware family, which is the phenomenon of conceptual drift.
- To evaluate the effectiveness of RecMaL, we apply a systematic experiment over a dynamically evolving malware dataset BODMAS. The RecMaL's clustering quality is superior to that of state-of-the-art techniques, according to the findings of cross-comparison trials.
- We will open-source our code and maintain an extensible interface. It not only provides support for subsequent research but also provides a powerful tool in the arms race between the security community and hackers.

2. The malware mislabelling problem

We illustrate the background and the motivation example in this section to inspire our work and guide our in-depth analysis.

2.1. Background and terminology

Background of Online Malware Diagnose. Nowadays, as an integral component of critical threat intelligence, malware family information can provide end-users, administrators, or security operators with vital information about possible types of attacks. The information can help define remediation procedures, identify possible root causes and evaluate the severity and potential consequences of the attack, which support the development of online anti-malware scanning services, such as VirusTotal (2023b), widely used by researchers and industrial practitioners. Since VirusTotal cooperates with 79 security vendors (Vendors) to provide malware scanning services, whenever a file is uploaded to VirusTotal, up to 79 anti-malware engine's detection results are available. These detection results (i.e., AV labels) denote whether the file is malicious or benign, and the attribute of the malware, i.e., family, type, platform, variant, etc. As a result, this capability of VirusTotal has been widely used to annotate malware datasets and provide system evaluation benchmarks (Chen et al., 2015; Ford et al., 2009; Hammad et al., 2018; Kharaz et al., 2016; Le Blond et al., 2014; Spreitzenbarth et al., 2013; Stringhini et al., 2014).

Diverse Naming Paradigm. Nevertheless, each vendor provides the AV labels with different structures and representations, since there is currently no universal naming convention (Maggi et al., 2011) to provide specifications for detection results. Referring to previous work, current vendors have 4 paradigms to provide AV labels for malware diagnose (Ducau et al., 2019a). We illustrate 4 instances (i.e., AV labels) to reclaim the situation.

- ❶*Worm:Win32/Mira.A*. These kinds of malware have obvious unique attributes, which typically have a large amount of identical original codes and often come from the same attacker or attack organization.
- ❷*HEUR:Trojan.Win32.Autoit.gen*. This type of naming paradigm differs only from that of ❶ with the existence of multiple authors or different sources. What's more, this example also contains the technique leveraged by the attacker, i.e., *Autoit*.
- ❸*Gen:Variant.Zusy.317885*. It is defined by the detection technique, for example, the result is detected by the *Zusy* technique.
- ❹*Trojan:Win32/Meterpreter*. The result is based on the method used to obfuscate or hide its payload.

Therefore, four different paradigms of AV labels given from different perspectives will further increase the difficulties of automatic marking tools. Adding the alias case and the multi-label case will increase the degree of confusion in the model.

Basic terms. We first define a few terms upfront.

- AV label denotes a set of intelligence information about the malicious sample, including type, platform, family, variant, etc. AV

[2] abbreviation of Rectify the Malware Family Label.

label denotes a set of intelligence information about the malicious sample, including type, platform, family, variant, etc.
- A malware family is a group of related programs that share enough "code overlap" to be considered a single entity. Instead of the typical notion of coarse-grained, all the research in this paper is based on the family concept of fine-grained.

2.2. Threat model

In the scenario of malware mislabelling, we mainly face the following 4 challenges.

❶***Non-uniform naming standards.*** Since the family label of malware is defined by security analysts, their perspectives and definition could be different. In particular, there is no uniform naming convention for malware specimens (Maggi et al., 2011). Hence, the security vendors can use their customized vocabulary (e.g., Kaspersky, 2020 and Microsoft, 2021), which makes it possible that different labels refer to the same malware family, or that the same label indicates different malicious behaviors. To address these issues, the Malware Attribute Enumeration and Characterization (abbreviated as MAEC) (MAEC) defines a standard language for sharing malware analysis results. However, MAEC does not include malware families but uses a strict predefined label table, which is not complete.

❷***Inconsistent detection methods.*** VirusTotal provides malware labels from a large set of anti-virus engines and is heavily used by researchers for malware annotation and system evaluation. However, security vendors on VirusTotal do not use the same detection techniques to scan malicious samples for family attributes, such as heuristics, artificial intelligence algorithms, Yara rule matching, etc., depending on specialties and accumulation of the security vendors.

❸***Malware mutates rapidly.*** According to the report of Av-Test, more than 1.2 billion unique malware samples were identified in 2021, an increase of more than 12 times compared with 2012. With the advent of the digital economy, the cyber security arms race is also in full swing. To steal more digital property, hackers try to mutate malware to achieve faster and more efficient attacks without being detected. Because variant malicious samples may include characteristics of several different families and their lineage is more convoluted, it becomes more challenging for security vendors to correctly characterize the variant malware in this scenario. The security vendor's final family attribution of the variation is determined by the malicious logic's prioritized relationship inside the vendor.

❹***Malware countermeasures.*** Packer and obfuscation. Malware often uses these methods to evade detection because low-cost packaging and obfuscation operations can achieve a great interference effect. Packaging can hide the internal structure and code resources of the software. Malware abuses this method to hide its malicious payloads. Aghakhani et al. (2020) find that although packers may preserve some information when packing programs that are "useful" for malware classification, such information does not necessarily capture the sample's behavior. In addition, such information does not help the classifier to generalize its knowledge to operate on previously unseen packers, or to be robust against trivial adversarial attacks. They also observed that static machine-learning-based products on VirusTotal produce a high false positive rate on packed binaries, possibly due to the influences of packers. Malware is also likely to abuse obfuscation techniques (Fass et al., 2019) such as randomization obfuscation, encoding obfuscation, logic structure obfuscation, and other obfuscation methods to avoid detection by AV-malware detectors and increase the workload for analysts. Malware may also modify software signatures to obfuscate (Kim et al., 2017) and signed malware, even a wrong signature, can interfere with the judgment of antivirus tools to a certain extent. These malware countermeasures greatly increase the difficulty of malware detection and analysis.

Table 1
Details on samples in the motivating example.

Sample MD5	Detected Threat Name
c9581ca3c7febdf1daa8755cacdf68a5	trojan.tofsee/invader
4a205e354b7d79837cbfcffd39835094	trojan.kryptik/brsecmone

2.3. Motivation

Motivating example. To further prove the severe malware mislabel situation, we illustrate a pair of malware examples in Table 1, denoting the MD5 and detection results. We find that both samples were highly consistent in their behavior and they both communicated with 43.231.4.7:443 in the Sandbox report, for which the IP both responded to the threat intelligence of Tofsee Botnet C&C activity. Therefore, judging from the perspective of dynamic behavior, these samples both belong to the *Tofsee* family, rather than one for *Tofsee* family and another one for *Kryptik* family.

Speculate the root cause. In response to the above-mentioned family misreports, and to dig deeper into the root causes, on the one hand, we use a dynamic detection method to completely characterize the malicious behavior, on the other hand, we try to stand from the perspective of the manufacturer and use rapid commercial detection to reproduce the false positive. Based on comparative experiments, we find that the static scanning method is easy to judge two samples as two families. However, through in-depth analysis, we find that the components of the two samples are the same, the malicious behaviors are both botnets, and both communicating with the same IP can be classified in the same family. Therefore, it is very likely that the false positive is caused by unidimensional feature analysis.

It is common for some researchers to annotate malware using only the detection result of a single security vendor (such as Kaspersky) to annotate malware, which is not particularly reliable. Moreover, some recent works (Fuller et al., 2021; Lee et al., 2021; Loi et al., 2021) show that the labels of most anti-malware engines are aggregated by segmentation of detection results, deletion of common tokens, replacement of aliases, voting, and others, while these methods are fundamentally dependent on the setting of empirical thresholds that observed on the malware dataset. Honestly, the different settings of experience thresholds and the quality of the malware dataset both have a significant impact on the generation of common tokens and aliases. As for the open algorithm, although it can generate malware knowledge bases without the involvement of experts, it relies considerably on the surface similarity of label words given by manufacturers rather than the deep similarity of the semantics behind them, hence it is not surprising when the study finds that this approach cannot handle the case where there are multiple sense words.

To sum up, the rationality of the above two situations needs further verification, which is the starting point of the central question addressed in this paper: whether the family labels obtained by these methods can match the malicious behavior behind the malware or not.

3. System design

In this section, we introduce the key design of RecMaL to figure out the inconsistency between the AV labels generated by the anti-virus engines and the actual malicious behavior.

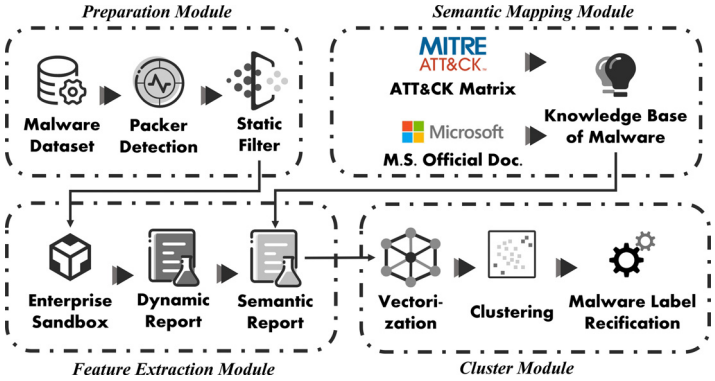

Fig. 1. Overall architecture of the RecMaL framework.

3.1. Overview

The overall architecture of the RecMaL is shown in Fig. 1. The input of the RecMaL framework is a well-labeled malware dataset. Note that the binary files contained in this dataset need to be intact and can be dynamically executed. The family labels of malware can be obtained by the anti-virus engines in VirusTotal community, and more stable labels can be obtained using tag aggregation tools, such as AVCLASS (Sebastián et al., 2016), AVCLASS2 (Sebastián and Caballero, 2020), Euphony (Hurier et al., 2017) et al. After running the RecMaL framework, the RecMaL framework will automatically rectify the possible label errors to reduce the ML algorithm benchmark problems caused by noisy labels.

In the **preparation module**, each executable file first enters the preparation module and sequentially passes through the packer detection and static filtering, thus discarding malware with the same malicious payload to improve the efficiency of subsequent analysis.

In the **semantic mapping module**, a semantic knowledge base of malware behavior is built offline, which is used to maintain the mapping information from system calls to semantic information. Whenever an executable file is run in the enterprise sandbox (technology institution), we extract key information from its dynamic operation reports and map them to behavioral semantics in the **feature extraction module**.

In the **cluster module**, we first deduplicate the behavior semantic sequences. Secondly, we train the paragraph vector model to convert the semantic sequence into paragraph embedding. Thirdly, we use a hierarchical clustering algorithm to merge malware with the same similar behavior. Finally, we rectify malware labels through inconsistent clue clusters in the malware dataset.

3.2. Preparation module

To reduce the number of dynamically executed malware samples and improve the overall efficiency of the RecMaL framework. We construct a preparation module consisting of two components: packer detection and static filtering, thus enabling a fast comparison of incoming malware samples at the static level to discard samples with the same malicious payload.

We studied different executable file filtering methods at the static level, such as various code similarity methods. The security community has widely studied how to detect code similarity. According to previous work research, namely testing and detecting binary and source code similarity testing detection (Ducau et al., 2019b; Mirzaei et al., 2021; Schleimer et al., 2003; Xu et al., 2017; Yang et al., 2021b; Yu et al., 2020) and (fuzzy) hashing (Li et al., 2015; Upchurch and Zhou, 2015), we noted that most methods build deep learning (or machine learning) models based on dynamic execution of recovery source code or disassembly instruction to complete code similarity detection. However, this similarity detection code detection has a high cost in the marking data and training model stage, which can not meet our expectations for static filters.

We observe in the wild that malware authors usually use a large number of samples with the same attack payload in attack activity. To avoid the direct filtering of the detection of the anti-malware engines, they usually modify the PE header structure or add some meaningless fields to the PE tail to bypass the capture of the full-text hash. Therefore, we try to ignore the second part of the executable file and calculate the hash value of the key part to filter malicious samples with the same attack payload.

At the same time, it is extremely difficult to extract the real payload of packed malware (Aghakhani et al., 2020; Cheng Binlin and Haotian, 2021). Therefore, we need to add a packer detector component before the static filter to avoid the interference of hash results caused by the packed sample.

Packer Detection. In recent years, many studies and tools have used entropy alone to classify whether a sample is packed or unpacked. However, just as researcher (Mantovani et al., 2020) have found, the size of entropy is not sufficient to conclude that the binary is packed or not. Therefore, we utilize the signature-based detection method rather than the entropy-based detection method to increase the stability of packer detection. DIE (Detect It Easy) (horsiccq) is a lightweight signature-based detection tool, which has the advantages of continuous maintenance, cross-platform, fully open signature architecture, and batch execution.

Static Filter. To filter malware samples with the same attack payload, we update the position of the hash algorithm and replace the entire file with the executable section where the original entry point (OEP) is located. In this way, the static filter component can only focus on the executable code of the sample, and the change of PE head and tail regions will not lead to the change in hash results. It is not sensitive to the non-core portion of malicious samples. Get the executable section code of OEP as shown in Algorithm 1.

In more detail, the starting and ending positions of each PE section are calculated after first obtaining the position of the program's original entrance point in the sample without packer. Each section's starting location corresponds to a virtual address in the section table. The operating system will assess if the virtual address can be aligned with the size of the memory block provided by the hardware condition when loading the executable file while computing the end position of the PE section. If not, 0 will be appended to align the memory. At this time, the calculation formula of the end position of PE section is shown in Eq. (1).

$$rva_{end} = rva_{start} + ((section.size//alignment) + 1) * alignment \tag{1}$$

Algorithm 1 Get the key section of binary file.

Input: binaryfile //the binary
Output: Contx //contents of the special section
  entrypoint ← *binary.optional_header.addressof_entrypoint*
  alignment ← *binary.optional_header.section_alignment*
  for *everysection* **do**
    rva ← *section.virtual_address*
    if *section.size%alignment* **then**
      rvaend ← *rva* + ((*section.size*//*alignment*) + 1) ∗ *alignment*
    else
      rvaend ← *rva* + *section.size*
    end if
    if *entrypoint* ≥ *rva* & *entrypoint* < *rvaend* **then**
      if *section.virtual_size* < *section.size* **then**
        Contx ← *section.contents*[: *section.virtual_size*]
      else
        Contx ← *section.content*
      end if
      return *Contx*
    end if
  end for

We used the classic Context-triggered hashing algorithm: SSDEEP (Kornblum, 2006) in the static filter. The piece-wise hash uses an arbitrary hashing algorithm to create many checksums for a file instead of just one. Rather than generating a single hash for the entire file, a hash is generated for many discrete fixed-size segments of the file. SSDEEP algorithm has non-propagation characteristics and alignment robustness, which ensures that the two binary files can still generate similar hash values in the case of small differences. We defined the static filtering method as SHASH.

Although static filters can filter malware with the same attack payload, it does not work for some packer types in malware. The reason is that these packers shift the position of the actual OEP to another section (not the section of the real OEP), which changes the action object of the static filter. Therefore, when the packer detection component detects the binary packed, we calculate the SSDEEP hash value of the whole file.

3.3. Feature extraction module

Running malware in sandboxes can observe its behavior and effect on the system. This strategy will examine various extracted information from the system while executing the malware, such as the registry key modification, the accessed/modified and dropped files, newly created and accessed processes (i.e., Application Programming Interfaces, APIs), and kernel-requested services. But malicious software to avoid sandbox execution and dynamic debugging, often adds some branch execution, delays execution, and other means of confrontation. For example, when malware detects that the runtime environment is a VMWare virtual machine, it will immediately end the program, or non-malicious code fragments will be executed. Therefore, to capture the behavior of malicious software as much as possible, we adopt an enterprise-level sandbox based on the out-of-box mechanism. The sandbox is developed based on hardware virtualization technology, software dynamic analysis technology, and control flow integrity analysis technology. It is superior to Cuckoo sandbox and other common sandbox based on in-box analysis mode in terms of analysis environment transparency, behavior analysis granularity, and sample analysis ability.

In the sandbox report, API is capable of holding enough information about programs and their behavior as it provides access to the essential resources that are available to the kernel system. API has two main parts, the function name, and parameters (arguments). The function name is a predefined list of APIs that developers can hook when developing sandboxes (Sandbox), and it belongs to different categories (i.e., administration and management, Windows user interface, and networking Microsoft, 2023a). However, function parameters are very complex and heterogeneous (e.g., integers, strings, and address pointers). Since it might be too difficult to analyze, the parameters generally are ignored in most API-based malware feature extraction studies (David and Netanyahu, 2015; Euh et al., 2020; Kolosnjaji et al., 2016; Pascanu et al., 2015). However, it still needs to be pointed out that models built only by APIs without parameters are blind. The reason is that the parameters of the same API function can express different semantics, but the parameters corresponding to different API functions can also express the same semantics. Therefore, it is not suggested to build only by the API function name, which cannot help to distinguish the real attack intention behind the malware author.

For example, we have been investigating the persistence of malicious samples in the Windows system, as shown in Table 2. It can be seen that the establishment of persistence is varied, the combination of different APIs and different parameters can produce the effect of persistence execution. Malicious samples can be persisted through the registry, file directory, process commands, Office default template, and auto-start scripts. Therefore, it is necessary to identify behavior semantics behind a single API.

To identify the real behavior semantics behind each API, we introduced the concept of the ontology (Smith and Welty, 2001) in the knowledge graph. Malware ontology is a knowledge model of the malware domain, it contains all relevant concepts related to malware individuals, malware behaviors, and computer system components. Therefore, we extracted caller names, function names, function parameters, and function return values in the behavioral report. And we further constructed a four-tuple as shown in Fig. 2. The caller name class defines which executable program calls the current API. Besides, the executable program may be the initial running malware, the software released/modified by malware, or normal software. The computer system component class defines the classification architecture of computer components, which includes all system component subclasses and individuals. The behavior class explains the classification architecture of malware behaviors, which includes different types of behaviors. The returned class defines the return value of the current API, which includes zero and non-zero. In this way, the behavioral semantics of a single API can be described as four-tuple: $< Caller\_name, API\_name, API\_exinfo, API\_ret >$. A self-reading example: <22703.file.exe, NtReadFile, C:\program\71733\22703.file.exe, 0>.

Abstractly, we used the semantic transformation method to obtain prior knowledge in the following two ways:

- Reverse, the feedback of Windows kernel API document (Microsoft, 2023b) and ATT&CK Matrix (MITRE).
- Positive, the feedback of real example in C language under the guidance of Windows user API document (Microsoft, 2023a).

Firstly, we processed and classified the API name (function name). The API name was represented by a string of words, such as “NtCreateFile”. In addition, some of the API names ended with various suffixes such as Ex, A, W, ExA, and ExW. Then we removed such suffixes to ensure that the extracted features were resilient against the conflict of using multiple versions of the same API call. To deal with heterogeneous API parameters, we classified API representing different domain functions, such as “InternetCrackUrl” and “DnsQuery” into the network function, “NtDeleteFile” and “NtOpenFile” into the file function. Secondly, we tried to identify the parameters of APIs with different functional areas, as

Table 2
Different expressions of persistence technology.

API Name	Parameters Values	Type
NtSetValueKey	HKEY_CURRENT_USER\... ... \Windows\CurrentVersion\Run	Register
CreateProcessInternal	C:\Windows\system32\reg.exe add Autorun_key	Process
NtWriteFile	C:\Users\jack\AppData\Roaming\... ... \Start Menu\Programs\Startup	Folder
MoveFileWithProgress	C:\Users\jack\AppData\Roaming\Microsoft\Templates\Normal.dotm	Office
CopyFile	C:\autorun.inf	Script

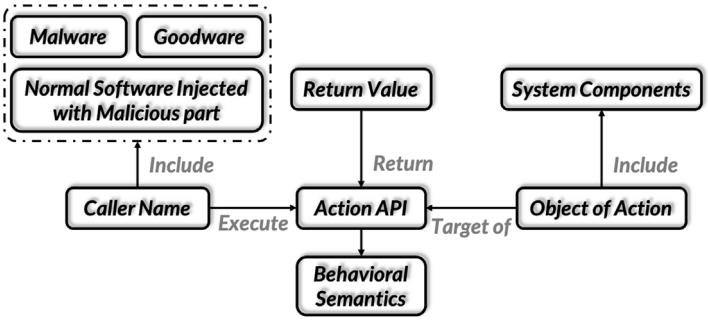

Fig. 2. Conceptual model of malware ontology.

Table 3
Part of the class structure of system components.

Functional Categories	Parameters Type	Instance
File	Prefetch File	C:\Windows\Prefetch
	Browser Privacy File	C:\Documents and Settings\Local Settings\Application Data\Chrome
	System Winhelp File	C:\Windows\winhelp.exe
	Email File	C:\Documents and Settings\Address book
Registry	AutoRun Key	HKEY_CURRENT_USER\SOFTWARE\... ... \CURRENTVERSION\RUN
	Hidden Key	HKEY_LOCAL_MACHINE\SOFTWARE\... ... \FOLDER\HIDDEN\SHOWALL
	Image Hijack Key	HKEY_LOCAL_MACHINE\SOFTWARE\... ... \IMAGE FILE EXECUTION OPTIONS
	System Software Setup Key	HKEY_LOCAL_MACHINE\SYSTEM\CONTROLSET001\SERVICES
Service	Message Service	Messenger
	Terminal Service	TermService
	Network Service	Rasman
	Sound Service	AudioSrv
Network	Local Net	127.0.0.1
		0.0.0.0
		localhost
	Inner Net	10.0.0.1
		192.0.0.1
	External Net	www.google.com
		221.226.65.138
Process	Batch cmd	C:\Windows\system32\cmd.exe
	Reg	C:\Windows\system32\reg.exe
	Rundll32	C:\Windows\system32\rundll32.exe
	CUI	C:\Windows\system32\conime.exe
	Register DLL	C:\Windows\system32\regsvr32.exe

shown in Table 3. We used the regular expression to transform specific heterogeneous parameters of system components into parameter types. The function parameters of some specific API were more than one, so the system component in the four-tuple is a list. Thirdly, we also processed the return value and gave it zero or non-zero forms. We adopted the above steps to extract useful and refined information from the four-tuple. At the same time, semantic matching is carried out with the extracted four-tuple information.

In addition, we also formulated some rules as shown below to assist the semantic translation.

- The self-reading, self-deleting, self-modifying, and a series of self-operating behaviors are defined which are mainly caused by functions such as NtReadFile, NtCreateFile, and NtDeleteFile. The key to this semantics is whether the caller name and parameter are the same.
- The rename semantics is defined, which is generally caused by the MoveFileWithProgress function. The key to this semantics is whether the directories of the two parameters in the parameter list are the same.
- The excessive behavioral semantics is defined, which means that the malware runs a single API more than 1500 times in the sandbox. According to the types of API, we divide the semantics into normal excessive behaviors and abnormal excessive behaviors. Some details are shown in Table 4.
- The two semantics of finding files and obtaining file attributes are mainly caused by functions such as NtQueryAttributesFile, NtQueryDirectoryFile, and NtQueryFullAttributesFile. The key to distinguishing the two semantics is whether the return value of the function is zero.
- To highlight the malicious behavior of samples, we define the calls that cannot be matched by rules as normal behavior.

Table 4
Partial API list of different excessive behavior types.

Type of Behavior	API Name
Normal too many behaviors	NtReadFile NtQueryAttributesFile LoadLibrary GetComputerName DnsQuery
Abnormal too many behavior	CreateProcessInternal RegOpenKey KiTrap0D NtAdjustPrivilegesToken VMDetect

3.4. Cluster module

After converting the dynamic behavior report into behavior semantic sequence, the RecMaL framework performed the feature transformation and the clustering on the behavior semantic sequence, and further analyzed and cleaned labels according to clustering results.

Preprocessing. After transforming the original sandbox report into a semantic sequence, we found a large number of repeated call sequences. We first performed [1,4]-gram sequences deduplication, retaining only the first consecutive sequence. At the same time, to avoid the uncertain conclusion caused by the small number of sample behaviors, which may be due to the lack of specific environments required by malicious samples in sandboxes, we filter the samples with the kinds of behavioral semantics less than 20 or run crashes.

Embedding. To better vectorize sandbox reports, we evaluated the advantages and disadvantages of a large number of language models that learn vectors from documents. For example, the word bag model will lose information about word order in the document and will not try to learn the meaning of potential words; Although the Word2Vec model can learn the meaning of potential words, calculating a vector for the entire document will introduce other defects. Therefore, we use the paragraph vector model (Le and Mikolov, 2014) to map the semantic sequence into the vector space. Specifically, the paragraph vector model can calculate a document as a low-dimensional vector representation, which is usually better than the word vector superposition or average of the Word2Vec model.

Clustering algorithm. Specifically, we implemented a highly parallel version of hierarchical agglomerative clustering (Müllner, 2011), which has been recently given theoretical support for its ability to generate high-quality clusters (Moseley and Wang, 2017). This clustering algorithm only needs to set one parameter and a distance threshold to determine when to cut the tree of clusters(i.e., when to stop creating smaller and smaller subclusters). However, the choice of deciding when to cut depends mainly on the similarity between the behaviors of malware families.

Distance threshold. Generally speaking, a higher value of the distance threshold generates fewer but larger clusters, whereas a smaller value yields more but smaller clusters. Therefore, the setting of distance threshold cannot rely on simple subjective experience. It should be noted that there must be a considerable part of sample label errors in the malware dataset, and there is a dynamic evolution of malware behavior in the same family. Therefore, if we use the maximum average distance as the distance threshold, more impurities will be introduced. To obtain a relatively scientific distance threshold, we use a method called Median Absolute Deviation (MAD) (Leys et al., 2013). As shown in Formula (2). Firstly, we calculate the average Euclidean distance of all samples under each family label and calculate the median of the average distance of all families. Secondly, we calculate the difference between the average distance to the median for each family. Finally, MAD is the median of the absolute value of these differences. As a measurement of statistical deviation, MAD is more suitable for capturing outliers in data sets than the standard deviation.

$$MAD = median(|X_i - median(X)|) \tag{2}$$

3.5. Label rectification

The type of label error. When we use an unsupervised clustering algorithm to get specific malware clusters, it means that we gather malicious samples with high behavioral similarities. Therefore, when the family labels in a cluster are not unique, we will discuss whether the labels in the cluster are appropriate. There are three specific cases:

- Label errors occur when a class exists in the dataset that is more appropriate for an example than its given class.
- Ontology issue is that the same malware belongs to different family names, while these family names can be replaced by each other. In other words, the ontology issue is the malware family alias problem. For example WannaCry and WannaCryptor.
- Multi-label malware has more than one label in the malware dataset. These labels belong to different categories of detection names. For example Lamber and Autorun.

Rectifying labels can improve the classification accuracy of machine learning. We first used RecMaL to perform family classification on samples of the BODMAS dataset. In the clustering results, if the samples in a certain cluster contain more than two different family labels, we think that some of the labels need to be corrected. After classifying samples from BODMAS using RecMaL, we found some labels that may need to be corrected. The detailed algorithm description is shown in Algorithm 2. To find and correct these labels, we mainly introduce three distances.

1. **The first distance.** The average Cosine distance of all samples under each family label
2. **The second distance.** The average Cosine distance of each sample in the inconsistent clue cluster to the other samples of its family.
3. **The third distance.** The average Cosine distance of each sample in the inconsistent clue cluster to the other samples under that cluster. The purpose of introducing the third distance is to find the most central part of the sample in the cluster.

❶ If the second distance of a sample is smaller than the first distance, it is considered that the original BODMAS label of the sample is more reasonable. Otherwise, it is considered that the sample may need to be revised. For every inconsistent clue cluster that needs to be relabeled, it is necessary to first judge the status of the cluster.

❷ If there is only one family with a reasonable sample label in the cluster, then use the label of this family to correct all samples that need to be revised. If there are multiple families with reasonably labeled samples in the family cluster, rank these samples from large to small according to the third distance of these samples. The corresponding ranking of each sample is their score, which means that the smaller the third distance of the sample, the higher the score. Count the sum of the scores of samples with reasonable labels under each family and use the label of the family with the highest score to correct all samples that need to be revised.

4. Implementation

The framework of RecMaL mainly consists of the following four modules.

Algorithm 2 Label Rectification.

Input: Samples //features and labels of BODMAS dataset
Input: cluster //clustering results output by RecMaL
Output: newlabel //sample labels corrected using the algorithm

```
for everycluster.everysample do
    if sample.secdis ≤ sample.firdis then
        sample.label ← True
    else
        sample.label ← False
    end if
end for
for everycluster do
    cluster.true_label ← dict()
    for everysample do
        if sample.label then
            cluster.true_label.add(sample.fam)
        end if
    end for
    cluster.true_label ← list(cluster.true_label)
end for
for everycluster.everysample do
    if sample.label == False then
        if len(cluster.true_label) == 1 then
            sample.newlabel ← cluster.truelabel[0]
        else if len(cluster.true_label) > 1 then
            Sort sample.rank By sample.thidis
            sample.score ← sample.rank
            for cluster.true_label do
                family.score ← Sum(current_family.sample.score)
            end for
            sample.newlabel ← cluster.maxscore_fam
        else
            sample.newlabel ← sample.fam
        end if
    end if
end for
```

4.1. Embedding and clustering details

To find the label errors in the malware dataset, we run the RecMaL framework in the BODMAS dataset. We progressively process the BODMAS dataset according to the RecMaL workflow. In the word embedding stage, we use the PV-DBOW model in the Doc2Vec model. Compared with PV-DM, this model can further omit the words in the window and use the paragraph vector to predict any words in the text. The PV-DBOW model is of great significance for highlighting the different behaviors of different malware families. Specifically, in the process of training the Doc2Vec model, we configure the frequency of random downsampling of high-frequency words as not 1e-5, the window size used is 10, the initial learning rate is set to 0.025, and the words with lower total frequency are never ignored. The training period is 20, and the hierarchical softmax method is used to train and finally map the document mapped to semantics to the document embedding representation of 2048.

In the hierarchical agglomerative clustering algorithm, we use the critical point of anomaly value automatically generated by the MAD method as the distance threshold, and finally, the RecMaL framework clusters the remaining 31,121 samples into 16,550 clusters. Statistics show that in all clusters, there are 3611 clusters with inconsistent family labels. Therefore, to measure the effectiveness of the clustering algorithm, we only manually check the behavior similarity of sandbox reports by a 1:10 random sampling ratio of clusters with inconsistent labels. Through our manual verification, the behavior similarity clustering method of RecMaL framework is effective and accurate.

4.2. Positioning label issue types

Firstly, we calculate the average Euclidean distance of all samples under each family label. Secondly, we calculate the average distance of each sample in the inconsistent clue cluster to the other samples of its family and compare this distance with the average distance of its family to determine whether the sample is an outlier.

- If all samples in inconsistent cues are not outliers, we consider that the case belongs to a multi-label or malware alias. When the average distance of these families does not exceed the distance threshold, that is, they belong to high cohesion family labels, they should be malware alias types, otherwise they are multi-label types.
- if there are samples in inconsistent cue clusters belonging to outliers, samples belonging to normal points, we consider that the case belongs to the label error, and rectify the label of the abnormal point family to the label of the normal point family;
- If all samples in inconsistent cues are outliers, the RecMaL framework cannot infer the ground truth of these samples according to the existing basis, so RecMaL does not do anything. But the RecMaL framework will throw out these outliers so that security analysts can pay more attention to these samples' labels.

According to the above steps, the RecMaL framework finds a total of 300 label errors, 58 malware family alias label pairs, and 6 general tokens of the multi-label.

5. Evaluation

We claim that static filter components in the framework can filter malware with the same attack payload, and feature and clustering modules can embed behavior semantic documents and generate high-quality clusters for label rectification. In this section, we conduct experiments to evaluate each component of RecMaL. We analyze the differences between the clusters and the differences between labels in the same cluster from the dataset. In addition, we verify the effectiveness of label rectification, via setting ablation experiments on malware family classification.

5.1. Datasets

Internal dataset. In our experiment, we investigate real malware (excluding normal software), which is collected from a large-scale security vendor (for anonymous). The dataset is collected from July 2020 to December 2020, including 208,124 malicious samples. It should be noted that if malicious samples are collected based on time, we may not able to guarantee that each SHASH is associated with enough samples, nor can we evaluate the effectiveness of the same SHASH. Therefore, we decided to take SHASH as the guide and to collect malicious samples to ensure that the number of each SHASH should be more than 100. At the same time, we got the VirusTotal detection reports of all samples and used the AVCLASS2 tool to label family labels and other attributes. The dataset included 67 different families (e.g. blackmoon, sytro, fearso), and all malicious samples were based on the Windows platform. The labels for this dataset only involve labels based on the traditional family category. We only evaluate the performance of Static filtering components on this dataset.

External dataset. We introduce a well-labeled external dataset BODMAS (Yang et al., 2021a), According to the authors, the BODMAS dataset contains 57,293 malware samples collected from August 2019 to September 2020, with carefully curated information

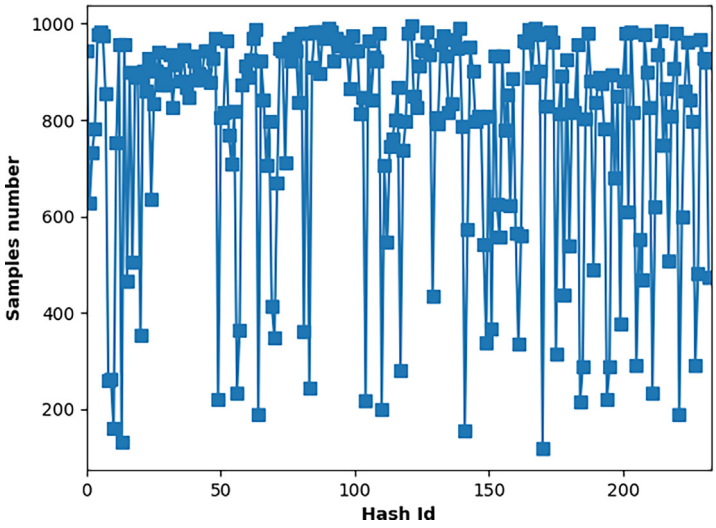

Fig. 3. Number of malware samples with each SHASH ID.

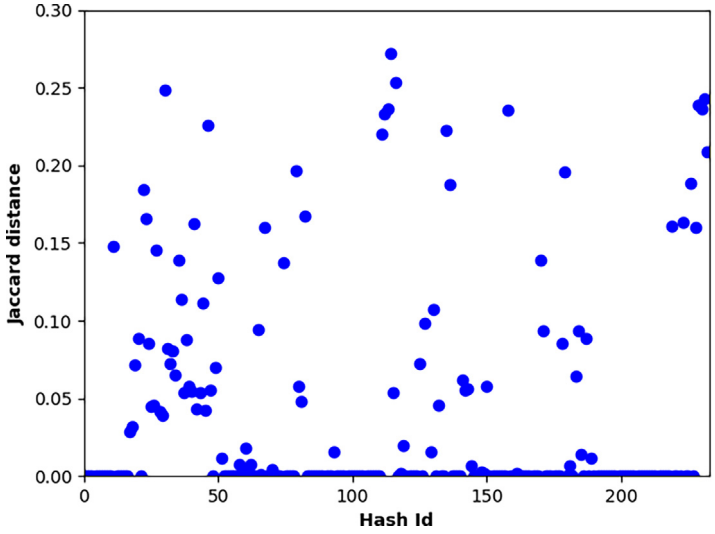

Fig. 4. Scatter plot of Jaccard distance of SHASH-related samples.

on 581 families. And the family label is mainly obtained by internal scripts similar to AVCLASS (Sebastián et al., 2016), but a small portion (about 1%) of malware was labeled via manual analysis of the binaries. We try to use the RecMaL to further analyze the label problems on the dataset.

5.2. Static filter performance

Filter effectiveness. In this section, we conduct the relevant experiment on our internal dataset to evaluate the effectiveness of the static filter. Please note that AV labels are not credible (maybe occur label errors), thus we use the distance of the dynamic behavior semantics kinds of relevant samples under the same SHASH to measure the effectiveness of static filters. We make statistics on the remaining samples after preprocessing. And the number of samples associated with each SHASH value is shown in Fig. 3. The majority of the malicious samples associated with SHASH are more than 200. We construct a bag of words model for all behavioral semantic sequences and then calculate the average Jaccard distance of the associated samples under each SHASH. The Jaccard distance formula is shown in Eq. (3). The Jaccard distance scatters plot is shown in Fig. 4. It can be seen that most of the SHASH associated with the behaviors of malicious samples are highly consistent, and occasionally there is a big difference because the sandbox does not run out of all the behaviors of the sample. In a words, it explained that the static filter in the RecMaL framework could effectively filter malware with the same attack payload.

$$Jaccard(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{3}$$

Filter efficiency. In this section, we conduct the relevant experiment on BODMAS dataset to evaluate the efficiency of the static filter. First we perform the packer detection on this dataset to exclude the effect of packed on the static filter. The detection results are shown in Fig. 5, in which more than 40% of the malware samples are packed. For these samples, the static filter does not process them because the real entry point of the program can not be determined. We also found a small number of samples with errors in the file format parsing, which were not processed by the static filter and are described separately in the Table 5. We performed static filtering on the remaining samples, and as shown in Table 5, the filtering efficiency of the uncased samples reached 82.0%, i.e., there were only 5736 groups of malware with different attack payloads among the 31,805 samples. Even when viewed on the entire dataset, the filtering efficiency reaches an impressive 45.5%.

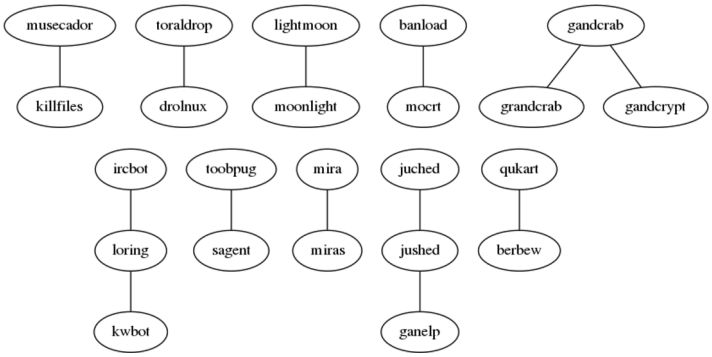

Fig. 5. Statistical result of packed samples in BODMAS.

Table 5
The detail of SHASH filtering effect.

Amount	Proportion	SHASH	Filter	Type
24,235	42.3%	24,235	0%	Packer
1251	2.2%	1251	0%	Unparse
31,805	55.5%	5736	82.0%	Unpacker
57,291	100%	31,222	45.5%	Total

Table 6
SHASH in terms of accuracy and efficiency compared to other methods.

Method\Metrics	Accuracy(Purity)	Hashes Number	Efficiency
ImpHash	0.81971	4036	92.955%
PeHash	0.88436	7029	87.731%
Vhash	0.89166	7632	86.679%
Authentihash	1.00000	57,072	0.382%
SHASH	0.97235	31,222	45.503%

Comparison with state-of-the-art methods. We compared the SHASH algorithm with several of the most widely used hashing methods in the current industry on the BODMAS dataset. We compare the purity metrics (Rao and Josephine, 2018) on different result sets to measure accuracy and the number of hash clusters to measure the efficiency of filtering.

- **ImpHash** (MANDIANT). The method is used specifically for Portable Executable (PE) files and is based on the PE import table contents. Since the behavioral capabilities of malware are indicated by imports function, it is hoped that hash values would be consistent across samples with related behavioral capabilities.
- **PeHash** (Wicherski, 2009). The method hashes the PE executable for selected fields that are not easily influenced by changes during compilation and packing, such as the initial stack size.
- **Vhash** (VirusTotal, 2023a). The method is a similarity clustering algorithm inside VirusTotal. It is based on a simple structural feature hashing algorithm for finding similar binary samples.
- **Authentihash** (VTAPI). The method used to verify that the relevant sections of a PE image file have not been altered by calculating the PE image excluding certificate related data and overlay.
- **SHASH.** Our method calculates the fuzzy hash of a binary file by analyzing its file structure and extracting its most important section representation.

Higher purity values indicate fewer anomalies in the clusters and higher accuracy of the hashing algorithm. The smaller number of hashes indicates that the algorithm can find fewer same attack payloads and the more efficient the filtering. As shown in Table 6, although the ImpHash method achieves 92.9% in filtering efficiency, its accuracy is the worst with only 0.82. Meanwhile, the Authentihash method, although its accuracy is the highest, its actual filtering efficiency is the worst at 0.38%. In a comprehensive comparison, our SHASH algorithm balances accuracy and efficiency. It maximizes the filtering efficiency while ensuring a high accuracy rate.

5.3. Dynamic clustering performance

To verify the performance of RecMaL, we compare the results of several representative malware classification methods on the BODMAS dataset and measure the Adjusted Mutual Information (AMI) (Vinh et al., 2010) metrics on different result sets. In the next part of this section, we briefly describe these selected methods and the details of the result set used to measure AMI metrics. The reason we use AMI as a judgment is that while the original tags of the samples are not entirely correct, most of them should be accurate. In the absence of accurate sample labels, we use the credible BODMAS dataset and the malware labeling tool AVCLASS to observe the clustering quality of different methods. In past studies, researchers have proposed different feature extraction methods to describe the behavior of malicious samples, but these features are mainly used for malware identification instead of malware classification. We tried to reproduce these feature extraction methods for family classification and compared the AMI of different feature extraction methods on different datasets.

- **Baseline.** This method extracts sequences of API call from malware reports and map them directly into feature vectors. These vectors are then learned through Doc2Vec to generate feature vectors.
- **Sample Hashing** (Zhang et al., 2020b). The method extracts key fields from malware sample running reports and map them to unique hash-like vectors. These vectors are filtered and combined to form the final sample features.
- **Malheur** (Rieck et al., 2011). The method extracts short behavioral patterns from malware samples to capture some of the underlying program semantic.
- **Malware Fusion** (Wang et al., 2021). The method performs sample filtering based on the API call sequence and use the n-gram model to remove API combinations with low occurrence rates.
- **Semantic Features.** Our approach analyzes API calls and parameter combinations to judge their behavior and generate semantic information needed for RecMaL.

Meanwhile, we not only compare the AMI information of different feature methods, we also add Fare framework (Liang et al.,

Table 7
The AMI metric results on all comparison experiments.

Method\Label	BODMAS	AVCLASS(direct)	AVCLASS_retrain-1	AVCLASS_retain-2	AVCLASS2	AVCLASS2_retrain
API+Agg.	0.642	0.654	0.665	0.663	0.658	0.665
API+Fare	0.399	0.321	0.325	0.323	0.324	0.323
HASH+Agg.	0.638	0.648	0.659	0.658	0.649	0.655
HASH+Fare	0.39	0.393	0.399	0.399	0.395	0.405
malheur+Agg.	0.687	0.712	0.719	0.720	0.714	0.718
malheur+Fare	0.428	0.436	0.436	0.441	0.438	0.444
Fusion+Agg.	0.781	0.746	0.745	0.745	0.752	0.754
Fusion+Fare	0.383	0.325	0.328	0.322	0.327	0.323
Semantic+Agg.	0.817	0.738	0.749	0.75	0.74	0.744
Semantic+Fare	0.387	0.323	0.322	0.321	0.328	0.324

2021) to compare the advantages and disadvantages of clustering algorithms. Fare is an aggregation framework for unsupervised classification models. It aims to perform classification on datasets with low-quality labels.

Ground-truth establishment. Since the labeling of the BODMAS dataset is not its ground-truth, we introduced a variety of different labeling results to simulate the most similar ground-truth. These label results are given by the AVCLASS tool and the AVCLASS2 tool, which are currently the most advanced tools in academia.

1. **AVCLASS (direct)** Label BODMAS samples using the AVCLASS default thesaurus.
2. **AVCLASS (retrain-1)** Use BODMAS samples with default nalias and talias parameters to retrain the thesaurus before AVCLASS labeling. These two parameters are used to control the association between label words.
3. **AVCLASS (retrain-2)** Use BODMAS samples to retrain the thesaurus before AVCLASS labeling. The nalias and talias parameters are changed to 100 and 0.98 to reduce the frequency of associated tags.
4. **AVCLASS2 (direct)** Label BODMAS samples using the AVCLASS2 default thesaurus.
5. **AVCLASS2 (retrain)** Use BODMAS samples with default parameters to retrain the thesaurus before AVCLASS2 labeling. We no longer adjust the parameters of AVCLASS2 because its retraining needs to provide the correct labels of the original samples to achieve better results than AVCLASS.

The experimental results are shown in Table 7. The method used by RecMaL has the highest AMI among all methods when compared with the BODMAS result set. Its AMI result on AVCLASS and AVCLASS2 result set is also higher than all methods except Malware Fusion.

Although the AMI does not directly represent the accuracy, the BODMAS dataset is partly manually verified which makes it more reliable. It should be noted that the AMI indicator is not the accuracy as the actual classification, we only use this indicator for auxiliary analysis. While Malware Fusion achieves a slightly higher AMI on AVCLASS, we prefer this because of the limitations of the tagging tool. One of the reasons is that the gap between AVCLASS and AVLCASS2 and their retraining results is very small, indicating that the tool does not learn enough useful information from the input samples. In contrast, the results of BODMAS contain some human-verified labels, and we have reason to believe that the results on the BODMAS set are more accurate than those on AVCLASS and AVCLASS2. Therefore, the Semantic Feature that achieved the highest AMI result on the BODMAS set also shows that RecMaL has a better performance.

Unsupervised model ensemble frameworks like Fare are not suitable for multi-class and few-shot tasks. Another notable result is that using the Fare clustering framework leads to lower AMI values for malware classification. We believe that this is due to the characteristics of Fare itself: Fare is suitable for more samples and Classification with fewer classes in the original implementation. BODMAS itself has more than 500 different malware family tags and 57,293 malicious samples. Compared with the experimental project data provided by Fare itself, there are too many types to be classified and too few samples are provided, which is an important reason for Fare's low AMI.

We also notice Malheur can achieve higher AMI results than other methods when using Fare for clustering. We think this is caused by the Malheur's characteristics having more dimensions and having traits that resemble one-hot encoding. This finding also suggests that sparse matrices may make input data more suitable for Fare.

5.4. Family label rectification

After correcting the labels, we used the original static features released by the authors of the BODMAS dataset to input a random forest algorithm for training. The results show that after correcting the label through RecMaL, the accuracy of the machine learning model trained using the original BODMAS data set has increased.

To verify the effect of label correction, we use the random forest classifier with default parameter settings to test on the original feature dataset of BODMAS. The experimental results using the five-fold verification method show that the accuracy of the classifier improved from 79.3% to 81.2%. This shows that even if the features provided by RecMaL are not applied, correcting the labels can slightly improve the accuracy of the Classification result.

6. Findings on BODMAS

In this section, we focus on the main label issues found when manually analyzing the inconsistency in family labels between RecMaL and BODMAS. To make it more clear, we follow the naming method of Northcutt et al. (2021a) for the malware label issues, including label error, ontology issue, and multi-label malware, as they have applied a thorough empirical study to figure out the label issues on the famous picture datasets, such as CIFAR-10. To provide a clearer visual illustration, we list the information of associated examples for each labeling issue shown in the Table 8. Column 1 denotes the ID of the case, column 2 denotes the MD5 of malware samples, column 3 denotes the packed techniques adopted by the attackers, column 4 denotes the label provided by BODMAS, column 5 denotes the ground truth of the malware family, and column 6 denotes the type of malware mislabel.

6.1. Label error

Observation: Based on our research, packed malware samples usually cause to be mislabeled. According to Table 8, the first merged row with ID '0' represents a pair of examples for label error, of which one is unpacked, and another is packed with technique *tElock*. To figure out the root cause of mislabeling issue, we

Table 8
Examples of each label issue.

ID	Sample MD5	Packed Detection	BODMAS Label	Ground Truth	Case Type
0	0fad413e6972bacbe6834816cc57b200	*unpacked*	wabot	wabot	Label error
	c40dba3c0dbf66c55328780fd79261e5	tElock	sfone		
1	0dca70cd568f819ef3e882430ae7fcab	VMProtect	lightmoon	\	Ontological
	b3be6e5663d07bd52d21b7653f28fdb4	VMProtect	moonlight		
2	a2ec2928668aa5463612658fd610ebba	Petite	trojan	trojan.upatre	Multi-label
	44375e1153f9b9044ffd7e9cfccedd6f	*unpacked*	upatre		

manually re-investigate the malicious behavior via static and dynamic analyses. According to the static scanning result, we find the PE section table of the packed sample has been changed compared to the unpacked sample, and 88 import functions in the import table have been reduced to 2 (i.e., function *GetModuleHandleA* and *MessageBoxA*). The limited static analysis information seems to tell us that the samples are different in content and structure, but the dynamic behavior sequences of the two samples are essentially the same, and the file named "sIRC4.exe" are both operated, which is consistent with the threat intelligence analysis result of Trend Micro, concluded as *wabot* (TRENDMICRO) family.

Finding #1: ***Packing techniques significantly limit the effectiveness of static scanning-only engines.*** To explore the impact of packing technology on the classification of malware families, besides the illustrated samples, we perform statistics and analysis on the packing of samples in the dataset.

According to Fig. 5, about one-third of malware samples are packed with 41 kinds of packing techniques, and the top 10 amount packing technique are presented respectively, of which *UPX* is the most leveraged packing technique. Due to the complexity of the packing technology, there is no automated unified unpacking tool at present, and the face of such a large number of complex packing technologies will indeed affect the detection results of the Anti-Virus engines. Besides packing techniques, obfuscation and other countermeasures reduce the risk of real malicious code being exposed to anti-virus engines, which are universally used by threat actors to evade engine detection. This also indicates that it is difficult for a single detection technology to give a reliable detection result.

Suggestion: Engines should adopt a multi-layered hybrid detection approach depending on the severity of the malware and its attributes. The scanners on VirusTotal only use part of the technologies from the actual anti-virus products. Rather than enforcing the files, it depends heavily on the detection signatures hit by them, without using the memory scanning techniques or dynamic execution techniques. As a consequence, relying solely on static scanning techniques to obtain detection results can produce a one-sided diagnosis of malware. On the premise of balancing performance and detection results, we recommend that these potentially high-threat samples can be subjected to a secondary hybrid analysis based on whether the malware has properties (i.e., packed) that affect static detection.

6.2. Ontological issue

Observation: Different malware has almost the same malicious behavior, but because the detection results come from different engines, they will produce different family classification results, that is the naming alias problem. In the malware sample examples belonging to the Ontological type in Table 8, it can be seen that the family label of the two samples is quite similar. Moreover, we confirm from the Microsoft's security intelligence report (MICROSOFT) that *lightmoon* and *moonlight* are two aliases of one malware family. To further explore the impact of the family alias issue, we enumerate some typical alias cases from BODMAS

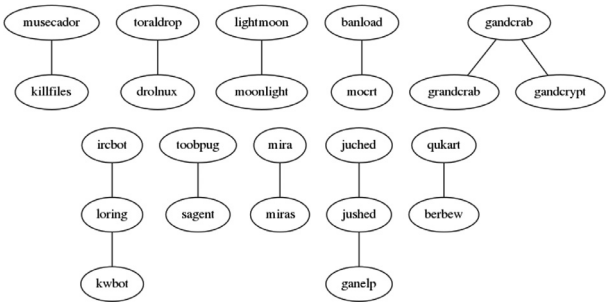

Fig. 6. Malware alias pairs.

in Fig. 6, from which we can intuitively find several pairs of labels that have a high degree of similarity to the literal view, such as *juched* and *jushed*. Moreover, we find the alias relationship between the label of *qukart* and *berbew* in the alias mapping list from the AVCLASS tool, which also confirms the existence of aliases on the other hand.

Finding #2: ***There is no standard naming paradigm for the taxonomy of the malware family.*** The main reason for the family alias problem is that, on the one hand, there is no authoritative naming system in the malware community for your reference, similar to the classification in biology. On the other hand, when defining family names, there is no authoritative naming system for your reference. Naming rules under different granularities will confuse classification levels.

Suggestion: It is essential to construct a malware family library as soon as possible to help eliminate malware family ambiguity. At present, since there is no standard malware family naming paradigm as well as no publicly accessible malware knowledge base, each anti-virus software vendor detects malware based on its own rules and signature corpus. As a result, the omissions and inconsistencies in family labels are difficult to eliminate, while malware family aliases are generally common, requiring us to build a malware family label library as soon as possible.

6.3. Multi-label issue

Observation: Malware is marked as multi-family (i.e., multi-label) with inclusion relationships, also with different granularity. Note that the multi-label issue is not the same as malware aliases (i.e., ontological issue). Ontological issue denotes that malware is labeled with family labels indicating the same behavior, but with different representations. As for the multi-label issue, the scope of malware covered by these labels is different and cannot be replaced by each other.

Therefore, the example denoted as multi-label issue in Table 8, one is labeled as *Trojan*, and another is labeled as *upatre*. We manually verify the ground truth of these two samples and find that both of the samples should be labeled as *upatre*. One of the samples can only be analyzed as *Trojan* via engine scanning due to the impediment of the *Petite* packing technology. Even, though the labels of both samples are correct, the hierarchy of the descriptions

Table 9
Top3 confidence for predicting *upatre* samples via 5 fold cross-validation.

	Recall@1	Recall@2	Recall@3	Trained?
Model-1	bluteal:0.60	upatre:0.40	kwbot:0.00	√
Model-2	fukru:0.60	upatre:0.40	win32:0.00	√
Model-3	bluteal:0.63	upatre:0.37	botgor:0.00	√
Model-4	ldpinch:0.63	upatre:0.37	kwbot:0.00	√
Model-5	**upatre:1.00**	ryzerlo:0.00	win32:0.00	✗

to which the labels belonged is not uniform. It is well known that family *Trojan*, described as one of the common malware types, has numerous different subordinate malicious families, including subfamily *upatre*. As a result, it is too coarse-grained as a family label.

Finding #3: ***There is no consensus on the layering of malicious families between the engines and within the engines.*** The detection mechanisms of different modules will hit different types of malicious features, and these features are not assigned different weights based on detection capabilities and malware classification, which leads to confusion in the level and granularity of detection results.

What's more, the non-unified granularity of malware family labels confuses the learning-based malware classification models. To verify our hypothesis, we measure the degree to which the inadvertent use of different hierarchies of family labels can confuse the model. We leverage the sample[3] in Table 8 as the experimental object.

We shuffle the data set and divide it into 5 parts equally, each time we take 4 of them for training and 1 for testing. To verify the degree of confusion of the model, we label the experimental object as *Trojan* (*upatre* in practice) into the data set. After training five times via a different combination of data, we use 5 different models to infer the malware family. The results are shown in the Table 9.

Column 2 to 4 separately denotes the Top3 family labels with the highest confidence and their concrete confidence. Column 5 denotes whether the experimental object sample is in the training set. According to the result, the prediction confidence of the *upatre* family is 100% when the samples are not partitioned into the training set, and conversely, the confidence of being predicated on the *bluteal, fukru* or even *ldpinch* families will exceed the confidence of the *upatre* family, not to mention the family *Trojan*. Intuitively, the same sample appears in the training set, and it is easy to succeed in inference, but the experimental results show that this is not the case. The mixed-use of family labels (e.g., *Trojan* and *upatre*) at different hierarchies can easily lead to confusion in the model, resulting in performance degradation.

Suggestion: The multi-hierarchical family label system should be included in the output of the detection engine and be valued by the community. From the above verification results, it can be seen that although the family labels of the parent level (i.e., *Trojan*) include the ones of the child level (i.e., *upatre*) in the natural language. But in the actual detection process, the malicious behavior and static representation of the malware may be quite different. Neglect to classify family labels at different hierarchies will not only hinder the accuracy and credibility of malware label clustering tools (e.g., AVCLASS) but may even hinder the development of intelligent virus analysis and detection.

7. Discussion

To better contribute to the community, we discuss the future direction of this issue and the limitation of RecMaL.

[3] MD5 of the sample is a2ec2928668aa5463612658fd610ebba.

❶ In this work, we focus on rectifying the family labels and pay less attention to static filter components. Previous studies have shown that both packing behavior and multi-labeling may reduce the efficiency of malware sample classification and identification. Since RecMaL's focus is mainly on semantic behavior extraction, appropriate static filtering and pre-processing of the samples will improve RecMaL's performance by reducing the difficulty of analyzing malware samples. For static analysis of malware, while balancing performance and efficiency, we could further try to consider more modalities (e.g., semantics, context) and more methods (e.g., data-driven) to improve the feature extraction results.

❷ How to reasonably apply advanced algorithm technology in the field of data mining and machine learning to malware analysis is also an urgent problem to be solved. Since we used a relatively simple statistical model in this work, some errors will be introduced. Due to the lack of absolutely accurate sample labels, we need to manually check some samples for further analysis. Some samples that have not been manually verified may have some extreme cases, such as unsuccessful runs, resulting in inaccurate sample reports. This will interfere with RecMaL's extraction of semantic behavior features in malicious samples. In addition, since the labels of the BODMAS dataset itself are not completely accurate, RecMaL's also learns some wrong label-related features from it. Due to the limitations of current malware labeling tools, when faced with samples labeled with multiple labels by these tools, RecMaL can only process labels associated with the behavior after extracting semantic information.

❸ The irregularity of malware labeling may greatly hinder the development of intelligent malware analysis. In this work, we try to solve it in a semi-automatic way, and we hope to rectify the label completely automatically or intelligently. Due to the instability and diversity of labeling tools, we decided to define malware by its semantic behavior. By comparing several malware feature extraction methods and classification methods, we believe that RecMaL has the best performance since it has the highest AMI metric on the human-verified BODMAS dataset. After using RecMaL to reclassify and correct the labels of malware samples in BODMAS, the accuracy of training with the original features of BODMAS is improved. This result shows that RecMaL can help machine learning improve the performance of malware classification and labeling tasks.

8. Related works

Malware clustering. Sebastián et al. (2016), Sebastián and Caballero (2020) proposed a tool to label malware at scale, which automatically extracted tags from AV labels. The accuracy of this approach on known families ranges from 67.5% to 96.3%. Although it works in a low-cost way, both its own experimental results and our evaluation results in this paper prove that the results can not in turn be used as ground truth for the malware sample. Kaczmarczyck et al. (2020) proposed a method named Spotlight, which uses a supervised learning method to filter known malware families, then clusters unknown malware by unsupervised method, and prioritizes them for further investigation using a scorer to obtain the related data that the security researchers wanted. Hu et al. (2013) proposed a malware clustering method based on static features. Bailey et al. (2007) and Bayer et al. (2009) describe the malware clustering method on dynamic behavior and its configuration file. They run their experiments on large-scale malware samples. Zhang et al. (2017) proposed an ensemble method to combine different features for automatic malware categorization. Li et al. (2010) have reported on their investigation of the impact that ground-truth selection might have on the accuracy reported for malware clustering techniques. Li et al. (2017) implemented an Android malware clustering system through iterative mining of

malicious payload and checking whether malware samples share the same version of the malicious payload. Pitolli et al. (2017) proposed a malware family clustering method based on hybrid features and birch algorithm. Rieck et al. (2011) proposed a framework named Malheur, it extracted short behavioral patterns from malware to capture underlying program semantics and analyse malware behavior. Furthermore, according to the experimental findings of Pitolli et al. (2017), Malheur can produce clusters of higher quality than AVCLASS (Sebastián et al., 2016).

Semantics-based malware features. Yang et al. (2019) surveyed the generation method of malware family features based on behavioral semantics, performed statistical analysis for the typical description in different aspects, and revealed the challenges and its future development prospects. Ding et al. (2019) applied ontology technique into the malware domain, and proposed the method for constructing malware behavioral knowledge base. Navarro et al. (2018) proposed an ontology-based framework to analyze the complex network and identify characteristics shared by malware samples. Zhang et al. (2020a) proposed a framework named APIGraph, which can enhance state-of-the-art malware classifiers with the similarity information among evolved Android malware in terms of semantically equivalent or similar API usages. Zhang et al. (2019) proposed a malware identification method that calculates the confidence of association rules between the abstracted API calls to form behavioral semantics to describe an application. Naval et al. (2015) proposed an approach for identifying real malware using asymptotic equipartition property (AEP) mainly applied in the information-theoretic domain to characterize the program semantics.

9. Conclusion

Based on the fact that the malware dataset has a large number of misstatements in family label description, we conduct the study on the plausibility of malware family labeling from the consensus that malware with similar behaviors should belong to the same malware family. For this, we design and construct an automated tool RecMaL to locate the family mislabeling problem types. Briefly, RecMaL locates the crucial location of malicious samples through static analysis to filter the number of samples, and it maps the sequence of calls from the underlying system in the sandbox report into the corresponding behavioral semantics, and then provides guidance to the clustering algorithm by calculating the similarity among behaviors to finally rectify malware labels. We find three different types of mislabeling issues, including label errors, ontology issues, and multi-labels. The three different types of labeling issues affect more than two thousand samples. When we rectify the mislabeling in the dataset, with the same features and models used, rectifying the label can lead to a 1.9% improvement in accuracy. More importantly, RecMaL is significant for complementing the malware family alias knowledge base. Our experiment indicates that RecMaL for reducing label noise can provide sufficient quality to mitigate label inaccuracy in practice.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Wang Yang: Conceptualization, Supervision, Writing – review & editing, Funding acquisition. **Mingzhe Gao:** Methodology, Software, Validation, Data curation, Writing – original draft. **Ligeng Chen:** Formal analysis, Data curation, Writing – original draft. **Zhengxuan Liu:** Software, Investigation, Visualization. **Lingyun Ying:** Resources, Funding acquisition, Writing – review & editing.

Data availability

Data will be made available on request.

References

Aghakhani, H., Gritti, F., Mecca, F., Lindorfer, M., Ortolani, S., Balzarotti, D., Vigna, G., Kruegel, C., 2020. When malware is packin' heat; limits of machine learning classifiers based on static analysis features. Network and Distributed Systems Security (NDSS) Symposium 2020.

Av-Test. Malware statistics and trends. https://www.av-test.org/en/statistics/malware/.

Bailey, M., Oberheide, J., Andersen, J., Mao, Z.M., Jahanian, F., Nazario, J., 2007. Automated classification and analysis of internet malware. In: International Workshop on Recent Advances in Intrusion Detection. Springer, pp. 178–197.

Bayer, U., Comparetti, P.M., Hlauschek, C., Kruegel, C., Kirda, E., 2009. Scalable, behavior-based malware clustering. In: NDSS, Vol. 9. Citeseer, pp. 8–11.

Chen, K., Wang, P., Lee, Y., Wang, X., Zhang, N., Huang, H., Zou, W., Liu, P., 2015. Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale. In: 24th {USENIX} Security Symposium ({USENIX} Security 15), pp. 659–674.

Cheng Binlin, E.A.L., Jiang Ming, Haotian, Z., 2021. Obfuscation-resilient executable payload extraction from packed malware. 30th USENIX Security Symposium (USENIX Security 21.

David, O.E., Netanyahu, N.S., 2015. DeepSign: deep learning for automatic malware signature generation and classification. In: 2015 International Joint Conference on Neural Networks (IJCNN). IEEE, pp. 1–8.

Ding, Y., Wu, R., Zhang, X., 2019. Ontology-based knowledge representation for malware individuals and families. Comput. Secur. 87, 101574.

Ducau, F. N., Rudd, E. M., Heppner, T. M., Long, A., Berlin, K., 2019a. Automatic malware description via attribute tagging and similarity embedding. arXiv preprint arXiv:1905.06262.

Ducau, F. N., Rudd, E. M., Heppner, T. M., Long, A., Berlin, K., 2019b. SMART: semantic malware attribute relevance tagging. arXiv preprint arXiv:1905.06262.

Euh, S., Lee, H., Kim, D., Hwang, D., 2020. Comparative analysis of low-dimensional features and tree-based ensembles for malware detection systems. IEEE Access 8, 76796–76808.

Fass, A., Backes, M., Stock, B., 2019. HideNoSeek: camouflaging malicious javascript in benign ASTs. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1899–1913.

Ford, S., Cova, M., Kruegel, C., Vigna, G., 2009. Analyzing and detecting malicious flash advertisements. In: 2009 Annual Computer Security Applications Conference. IEEE, pp. 363–372.

Fuller, J., Kasturi, R.P., Sikder, A., Xu, H., Arik, B., Verma, V., Asdar, E., Saltaformaggio, B., 2021. C3PO: large-scale study of covert monitoring of C&C servers via over-permissioned protocol infiltration. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 3352–3365.

Hammad, M., Garcia, J., Malek, S., 2018. A large-scale empirical study on the effects of code obfuscations on Android apps and anti-malware products. In: Proceedings of the 40th International Conference on Software Engineering, pp. 421–431.

horsiccq. Detect it easy. https://github.com/horsicq/Detect-It-Easy.

Hu, X., Shin, K.G., Bhatkar, S., Griffin, K., 2013. MutantX-S: scalable malware clustering based on static features. In: 2013 {USENIX} Annual Technical Conference ({USENIX}{ATC} 13), pp. 187–198.

Hurier, M., Suarez-Tangil, G., Kumar, S., Bissyandé, T., Cavallaro, L., 2017. Euphony: harmonious unification of cacophonous anti-virus vendor labels for android malware. In: IEEE/ACM 14th International Conference on Mining Software Repositories.

JoeSandbox. Joe sandbox. https://www.joesandbox.com/.

Kaczmarczyck, F., Grill, B., Invernizzi, L., Pullman, J., Procopiuc, C.M., Tao, D., Benko, B., Bursztein, E., 2020. Spotlight: malware lead generation at scale. In: Annual Computer Security Applications Conference, pp. 17–27.

Kaspersky, 2020. Kaspersky threats. https://threats.kaspersky.com/en/threat/?view=hierarchy.

Kharaz, A., Arshad, S., Mulliner, C., Robertson, W., Kirda, E., 2016. {UNVEIL}: a large-scale, automated approach to detecting ransomware. In: 25th {USENIX} Security Symposium ({USENIX} Security 16), pp. 757–772.

Kim, D., Kwon, B.J., Dumitraş, T., 2017. Certified malware: measuring breaches of trust in the windows code-signing PKI. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1435–1448.

Kolosnjaji, B., Zarras, A., Webster, G., Eckert, C., 2016. Deep learning for classification of malware system call sequences. In: Australasian Joint Conference on Artificial Intelligence. Springer, pp. 137–149.

Kornblum, J., 2006. Identifying almost identical files using context triggered piecewise hashing. Digital Invest. 3, 91–97.

Le, Q., Mikolov, T., 2014. Distributed representations of sentences and documents. In: International Conference on Machine Learning. PMLR, pp. 1188–1196.

Le Blond, S., Uritesc, A., Gilbert, C., Chua, Z.L., Saxena, P., Kirda, E., 2014. A look at targeted attacks through the lense of an {NGO}. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14), pp. 543–558.

Lee, S., Jung, W., Lee, W., Oh, H., Kim, E.T., 2021. Effective dataset construction method using dexofuzzy based on android malware opcode mining. ICT Express.
Leys, C., Ley, C., Klein, O., Bernard, P., Licata, L., 2013. Detecting outliers: do not use standard deviation around the mean, use absolute deviation around the median. J. Exp. Soc. Psychol. 49 (4), 764–766.
Li, P., Liu, L., Gao, D., Reiter, M.K., 2010. On challenges in evaluating malware clustering. In: International Workshop on Recent Advances in Intrusion Detection. Springer, pp. 238–255.
Li, Y., Jang, J., Hu, X., Ou, X., 2017. Android malware clustering through malicious payload mining. In: International Symposium on Research in Attacks, Intrusions, and Defenses. Springer, pp. 192–214.
Li, Y., Sundaramurthy, S.C., Bardas, A.G., Ou, X., Caragea, D., Hu, X., Jang, J., 2015. Experimental study of fuzzy hashing in malware clustering analysis. 8th Workshop on Cyber Security Experimentation and Test ({CSET} 15).
Liang, J., Guo, W., Luo, T., Vasant, H., Wang, G., Xing, X., 2021. Fare: enabling fine–grained attack categorization under low-quality labeled data. In: Proceedings of The Network and Distributed System Security Symposium (NDSS).
Loi, N., Borile, C., Ucci, D., 2021. Towards an automated pipeline for detecting and classifying malware through machine learning. arXiv preprint arXiv:2106.05625.
MAEC. Malware attribute enumeration and characterization. http://maecproject.github.io/.
Maggi, F., Bellini, A., Salvaneschi, G., Zanero, S., 2011. Finding non-trivial malware naming inconsistencies. In: International Conference on Information Systems Security. Springer, pp. 144–159.
MANDIANT. Tracking malware with import hashing. https://www.mandiant.com/resources/blog/tracking-malware-import-hashing.
Mantovani, A., Aonzo, S., Ugarte-Pedrero, X., Merlo, A., Balzarotti, D., 2020. Prevalence and impact of low-entropy packing schemes in the malware ecosystem. Network and Distributed System Security (NDSS) Symposium, NDSS, Vol. 20.
Microsoft. Overview of the windows API. https://docs.microsoft.com/en-us/previous-versions//aa383723(v=vs.85).
Microsoft. Windows kernel API. https://opdhsblobprod02.blob.core.windows.net/contents/72a50b11a1b74f26a8d45bfae9461268/4595a9810a2114ee29054688270d62f8?sv=2018-03-28&sr=b&si=ReadPolicy&sig=Mvbwe9yQ7g1BrlCfYttykDX4FdQnemJectFwr%2FYYScA%3D&st=2021-06-07T14%3A43%3A40Z&se=2021-06-08T14%3A53%3A40Z.
MICROSOFT. Worm:win32/lightmoon.h. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm:Win32/Lightmoon.H&threatId=-2147347757.
Microsoft, 2021. Malware names. https://docs.microsoft.com/en-us/windows/security/threat-protection/intelligence/malware-naming.
Mirzaei, O., Vasilenko, R., Kirda, E., Lu, L., Kharraz, A., 2021. Scrutinizer: detecting code reuse in malware via decompilation and machine learning.
MITRE. Att&ck. https://attack.mitre.org/.
Moseley, B., Wang, J., 2017. Approximation bounds for hierarchical clustering: average linkage, bisecting k-means, and local search. Adv. Neural Inf. Process. Syst. 30, 3094–3103.
Müllner, D., 2011. Modern hierarchical, agglomerative clustering algorithms. arXiv preprint arXiv:1109.2378.
Naval, S., Laxmi, V., Rajarajan, M., Gaur, M.S., Conti, M., 2015. Employing program semantics for malware detection. IEEE Trans. Inf. Forensics Secur. 10 (12), 2591–2604. doi:10.1109/TIFS.2015.2469253.
Navarro, L.C., Navarro, A.K., Gregio, A., Rocha, A., Dahab, R., 2018. Leveraging ontologies and machine-learning techniques for malware analysis into android permissions ecosystems. Comput. Secur. 78, 429–453.
Northcutt, C., Jiang, L., Chuang, I., 2021. Confident learning: estimating uncertainty in dataset labels. J. Artif. Intell. Res. 70, 1373–1411.
Northcutt, C. G., Athalye, A., Mueller, J., 2021b. Pervasive label errors in test sets destabilize machine learning benchmarks. arXiv preprint arXiv:2103.14749.
Pascanu, R., Stokes, J.W., Sanossian, H., Marinescu, M., Thomas, A., 2015. Malware classification with recurrent networks. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp. 1916–1920.
Pitolli, G., Aniello, L., Laurenza, G., Querzoni, L., Baldoni, R., 2017. Malware family identification with birch clustering. In: 2017 International Carnahan Conference on Security Technology (ICCST), pp. 1–6.
Rao, K.R., Josephine, B.M., 2018. Exploring the impact of optimal clusters on cluster purity. In: 2018 3rd International Conference on Communication and Electronics Systems (ICCES), pp. 754–757. doi:10.1109/CESYS.2018.8724114.
Rieck, K., Trinius, P., Willems, C., Holz, T., 2011. Automatic analysis of malware behavior using machine learning. J. Comput. Secur. 19 (4), 639–668.
Sandbox, C.. Hooked APIs and categories in Cuckoo. https://github.com/cuckoosandbox/cuckoo/wiki/Hooked-APIs-and-Categories.
Schleimer, S., Wilkerson, D.S., Aiken, A., 2003. Winnowing: local algorithms for document fingerprinting. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 76–85.
Sebastián, M., Rivera, R., Kotzias, P., Caballero, J., 2016. AVclass: a tool for massive malware labeling. International Symposium on Research in Attacks, Intrusions, and Defenses.
Sebastián, S., Caballero, J., 2020. AVClass2: massive malware tag extraction from AV labels. In: ACSAC '20: Annual Computer Security Applications Conference.
Smith, B., Welty, C., 2001. Ontology: towards a new synthesis. In: Formal Ontology in Information Systems, Vol. 10. ACM Press, pp. 3–9.
Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T., Hoffmann, J., 2013. Mobile-sandbox: having a deeper look into android applications. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 1808–1815.
Stringhini, G., Hohlfeld, O., Kruegel, C., Vigna, G., 2014. The harvester, the botmaster, and the spammer: on the relations between the different actors in the spam landscape. In: Proceedings of the 9th ACM symposium on Information, computer and communications security, pp. 353–364.
technology institution, Q. A. X.. Tianqiong sandbox. https://research.qianxin.com/sandbox.
TRENDMICRO. Trojan.win32.wabot.disc. https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/trojan.win32.wabot.disc/.
Upchurch, J., Zhou, X., 2015. Variant: a malware similarity testing framework. In: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, pp. 31–39.
Vendors, V.. Virustotal vendors. https://support.virustotal.com/hc/en-us/articles/360001385857-Identifying-files-according-to-antivirus-detections.
Vinh, N.X., Epps, J., Bailey, J., 2010. Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for chance. JMLR.org (95).
VirusTotal. Files information about files. https://developers.virustotal.com/reference/files.
VirusTotal. Virustotal. https://www.virustotal.com.
VTAPI. authentihash. https://developers.virustotal.com/reference/authentihash.
Wang, P., Tang, Z., Wang, J., 2021. A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling. Comput. Secur. 106, 102273.
Wicherski, G., 2009. peHash: a novel approach to fast malware clustering. 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET 09). USENIX Association, Boston, MA. https://www.usenix.org/conference/leet-09/pehash-novel-approach-fast-malware-clustering
Xu, X., Liu, C., Feng, Q., Yin, H., Song, L., Song, D., 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 363–376.
Yang, L., Ciptadi, A., Laziuk, I., Ahmadzadeh, A., Wang, G., 2021. BODMAS: an open dataset for learning based temporal analysis of PE malware. 4th Deep Learning and Security Workshop.
Yang, P., Shu, H., Xiong, X., Kang, F., 2019. Semantic-based malware behavior description: Past and future. In: Proceedings of the 2019 the 9th International Conference on Communication and Network Security, pp. 11–19.
Yang, S., Cheng, L., Zeng, Y., Lang, Z., Zhu, H., Shi, Z., 2021. Asteria: deep learning-based AST-encoding for cross-platform binary code similarity detection. In: 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, pp. 224–236.
Yu, Z., Cao, R., Tang, Q., Nie, S., Huang, J., Wu, S., 2020. Order matters: semantic-aware neural networks for binary code similarity detection. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, pp. 1145–1152.
Zhang, H., Luo, S., Zhang, Y., Pan, L., 2019. An efficient android malware detection system based on method-level behavioral semantic analysis. IEEE Access 7, 69246–69256. doi:10.1109/ACCESS.2019.2919796.
Zhang, X., Zhang, Y., Zhong, M., Ding, D., Cao, Y., Zhang, Y., Zhang, M., Yang, M., 2020. Enhancing state-of-the-art classifiers with API semantics to detect evolved android malware. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 757–770.
Zhang, Y., Rong, C., Huang, Q., Wu, Y., Yang, Z., Jiang, J., 2017. Based on multi-features and clustering ensemble method for automatic malware categorization. In: 2017 IEEE Trustcom/BigDataSE/ICESS. IEEE, pp. 73–82.
Zhang, Z., Qi, P., Wang, W., 2020. Dynamic malware analysis with feature engineering and feature learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, pp. 1210–1217.
Zhu, S., Shi, J., Yang, L., Qin, B., Zhang, Z., Song, L., Wang, G., 2020. Measuring and modeling the label dynamics of online anti-malware engines. In: 29th {USENIX} Security Symposium ({USENIX} Security 20), pp. 2361–2378.

Wang Yang is an Lecturer in the School of Cyber Science and Engineering at Southeast University. He received his PhD from the School of Computer Sciences and Technology at the Southeast University. His research area covers malware and binary analysis, threat intelligence analysis.

Mingzhe Gao received the MS degree from the Southeast University in 2022. His current research interests include system security, software security, and binary analysis. He works on solving security problems using data-driven and artificial intelligence approaches.

Ligeng Chen received the BE degree in Computer Science and Technology from Chongqing University in 2017. He is currently pursuing the PhD degree in the Department of Computer Science and Technology, Nanjing University. His current research interests include software analysis, malware analysis and data mining.

Zhengxuan Liu received the BE degree in information security from the Nanjing University of Information Science and Technology in 2021. He is currently pursuing the MS degree in Southeast University. His current research interests include malware detection and binary analysis.

Lingyun Ying is a research fellow of QI-ANXIN Technology Research Institute. He received his PhD degree from the Graduate University of Chinese Academy of Sciences. His research area covers software security, system security, malware and binary analysis.