Humans vs. Machines in Malware Classification

Simone Aonzo
EURECOM

Yufei Han
INRIA

Alessandro Mantovani
EURECOM

Davide Balzarotti
EURECOM

Abstract

Today, the classification of a file as either benign or malicious is performed by a combination of deterministic indicators (such as antivirus rules), Machine Learning classifiers, and, more importantly, the judgment of human experts.

However, to compare the difference between human and machine intelligence in malware analysis, it is first necessary to understand how human subjects approach malware classification. In this direction, our work presents the first experimental study designed to capture which 'features' of a suspicious program (e.g., static properties or runtime behaviors) are prioritized for malware classification according to humans and machines intelligence. For this purpose, we created a malware classification game where 110 human players worldwide and with different seniority levels (72 novices and 38 experts) have competed to classify the highest number of unknown samples based on detailed sandbox reports. Surprisingly, we discovered that both experts and novices base their decisions on approximately the same features, even if there are clear differences between the two expertise classes.

Furthermore, we implemented two state-of-the-art Machine Learning models for malware classification and evaluated their performances on the same set of samples. The comparative analysis of the results unveiled a common set of features preferred by both Machine Learning models and helped better understand the difference in the feature extraction.

This work reflects the difference in the decision-making process of humans and computer algorithms and the different ways they extract information from the same data. Its findings serve multiple purposes, from training better malware analysts to improving feature encoding.

1 Introduction

The evolution of technology and digitization has transformed our lives and our society, but it has also given birth to a wide range of new cyber-criminal activities. These activities, independently of their goal, are often carried out using different types of malicious software, also known as malware. Over the past three decades, the steady rise of malware attacks has created the need for specialized security experts trained to study, recognize, and classify new malware strains: the *malware analysts*. Their goal is to examine malicious software, such as bots, worms, and trojans, to understand the nature of their threat. This task usually requires examining how the suspicious sample interacts with its environment, observed by executing the sample inside a malware analysis sandbox. Sandboxes come in different flavors and with different functionalities. However, they all provide the analysis with a *dynamic* (i.e., behavioral) report that summarizes all operations performed by the program, together with some *static* properties extracted from the executable file.

In turn, such static and dynamic components are divided into different fine-grained feature groups, which we will henceforth call *features* for readability. For example, the Import Table of a Portable Executable file is a static feature, while the DNS queries it performed during its execution are part of the network activity, which is a dynamic feature.

The role of a malware analysis report is to provide the raw information required to *classify* the unknown program as either benign or malicious. This malware classification task can be challenging, as it requires reasoning about the possible purpose and intent of the unknown software. Because of this, in the past, this task was usually performed manually by expert malware analysts. However, today a typical antivirus company collects and analyses over 350,000 suspicious programs per day [8], and these are not just minor variations of known families: for instance, 41% of the families observed in the wild in 2019 were never observed before [7]. It is clear that the number of security experts cannot scale to cope with these numbers.

Machine learning (ML) offers an easy-to-deploy and scalable solution to massive-scale malware classification applications. A vast amount of research has been conducted on ML-based malware classification [14, 16, 17, 28, 29, 33, 34, 39, 43, 47, 49, 51, 52, 64, 67, 68, 70, 71, 77, 92–94, 96], often with very promising results. However, unlike applications such as speech and text recognition, where pronunciations and character shapes remain relatively constant over time, malware constantly changes to evade detection. Therefore, it is unclear how robust these previous results are and, more importantly, which features really influences the accuracy of classification in the previously reported results. In other words, no study to date has examined whether human experts and ML-based so-

lutions use the same information to decide whether a sample is benign or malicious. We do not even know if different humans rely on the same features or if the choices of the used features change as the analyst's background and experience vary.

This paper tries to answer all these questions by conducting the first experiments designed to compare 110 humans with different expertise (72 *novices* and 38 *experts*) vs. different state-of-the-art ML algorithms. Our goal is not to measure who performs better but to understand *how* each group uses the data extracted from malware analysis reports to make its decision.

To collect the data, we designed an online game "*Detect Me If You Can!*", which asks the participants to classify 20 suspicious files based on their sandbox reports. Players have to 'buy' each individual set of features to add them to their report, allowing us to capture which information they considered more valuable and how many features they needed to reach a decision. Even though we observed a general agreement among the features preferred by all human subjects in our experiments, experts and novices showed distinctive traits (especially speed and accuracy). For example, while dealing with goodware, experts used many more features, and novices made more mistakes. This is because experts know that they need to rule out any possible signs of bad intentions in order to classify a sample as goodware.

Finally, to compare humans with machine intelligence, we collected 21,944 reports, equally distributed among malware and goodware, to train two Machine Learning-based malware classification models based on the encoding techniques recently proposed in state of the art. Both *ML players* performed in the game like an average human expert, but with the exceptional advantage of accessing all the features simultaneously. In a second test, we use the same algorithms to replay some experts' games using the same features that human subjects used to classify each sample. In this case, the accuracy of the two ML players was worse than random guess. This brings us to one of the main findings of our study: the two ML algorithms do *NOT* use the same features as human analysts do. Quite the opposite: they often rely on static features, whereas human subjects prefer mostly dynamic behavior features. ML even ranked the resource section's entries of executables as the most relevant feature, which was constantly ranked in the last place – and almost never used – by humans.

2 Related Work

For over two decades, the automated analysis of malicious code has been one of the top research areas in system security. Because of the large number of samples that need to be analyzed, previous studies have mainly focused on fully automated techniques or approaches requiring minimal interaction with the analyst. These solutions range from malware analysis sandboxes [18, 24, 45, 74, 79] to machine learning classifiers [53, 54, 57, 65, 75].

For example, Darshan et al. [27] focus their vectorization approach on the syscalls reported by the *Cuckoo* sandbox logs. In the same year, Miller et al. [62] proposed four main techniques to provide proper vectorized features for the use of ML-based detectors. Other recent approaches are presented by Karbab et al. [50], where the authors implement a natural language processing methodology, and Ijaz [44], who shows a comparison between the accuracy obtained by dynamic features versus static features (retrieved thanks to static analysis approaches). Jindal et al. [49] developed an end-to-end malware detection tool based on techniques borrowed from the field of text classification.

Xiong et al. [87] performed a set of experiments to detect malicious network traffic, where non-skilled users can communicate with the framework and provide feedback with respect to their feeling about the regularity of network activity. Researchers have also studied the use of graphical visualization techniques to represent malware samples and help analysts recognize specific behaviors and features [38, 72, 81].

More recently, in 2016, Yakdan et al. [88] proposed a work about enhancing decompilers from a malware analyst point of view. In the same year, Miller et al. [62] showed an innovative way to integrate the human analysts into an automated malware analysis pipeline that increased the detection rate by 12%. Ugarte Pedrero et al. [82] have also asked malware analysts to manually label previously-unknown malware clusters.

While today most of the analysis is delegated to machines, part of the job (in particular the interpretation of the analysis results) is still performed by malware analysts [82].

Wong et al. [95] interviewed 21 malware analysts to understand the different aspects of this profession. The main results of their study are a taxonomy that classifies malware analysts into three different groups based on their high-level objectives, the identification of five common workflows when an analyst decides to analyze a malware sample in detail, and the factors they consider when setting up a dynamic analysis system. The key difference between our work and the latter is that we conducted a user study on binary classification, while their semi-structured interview yielded observations about the goals and workflows in analyzing samples already classified as malicious. Therefore, we argue that our work should be considered complementary, and, as we will show below, we have confirmed some observations.

Very few works in the system security area have compared experts and novices in the context of malware analysis. In 2015, Hibshi et al. [40] conducted a study in which they assessed the role of the experience in the decision-making model of experts and novices for vulnerability mitigation in the source code. The effects of knowledge and expertise in security are also the topic presented by Ben-Asher et al. [19], where the authors develop a simple intrusion detection system to study how individuals with different backgrounds identify malicious events. More recently, Votipka et al. [84] carried out a set of interviews about how experts approach reverse engineering problems, while Mantovani et al. [58] studied the different strategies adopted by expert and beginner reverse engineers during the analysis of x86 disassembly code: a typical static reverse engineering task that is often part of the malware analysis process. Finally, several user studies about phishing have been conducted to examine how individuals

with different levels of experience react to malicious web pages [63, 97], aiming at building a mental model of the investigated subjects when performing phishing-related tasks.

Researchers in the program comprehension community often compare expert and novice subjects outside the security field when reading the source code to capture the role of expertise and the adopted abstractions [22, 30, 36, 56, 86].

3 Methodology

Our study requires access to a large and diverse set of malware analysts. To remove biases introduced by the training provided in a given workplace or by the workflow adopted by all analysts in the same group, we wanted our participants to come from a broad range of companies and have different day-to-day activities.

To accommodate these constraints and collect participants from all over the world, we decided to implement a web-based platform specifically designed to conduct our experiments. Moreover, we decided to adopt a gamification approach because scientific studies found that it has a positive motivational effect on individuals and their overall performances [37, 69]. Therefore, our platform implements a custom game, which we call *Detect Me If You Can!* (DMIYC, from now on). From a gaming perspective, the typical design elements used in DMIYC are points and a leader board. Points provide a way to numerically represent a player's outcome [85] while leader boards rank players according to their relative success, thus measuring them against a specific success criterion and showing who performed best in a certain activity [25].

While a useful tool to create social pressure that can increase the player's level of engagement [21], the leader board can also introduce tension and stress in the participants. To soften this aspect, the participants of our game were completely anonymous, and players were only identified by their arbitrary usernames. Moreover, the player's position on the leader board was only visible when the entire game was completed.

It is worth emphasizing that our study is not dedicated to organizing a competitive gaming scenario for human subjects or between human analysts and machine intelligence. In fact, players need to independently select the features required to reach their own decisions without knowing or interfering with other participants' scores while playing the game.

The study was piloted in collaboration with other members of our group, who helped us improve the game's rules and the GUI. Moreover, the game included an initial demo phase to ensure each player was familiar with the rules and interface.

3.1 Game Rules

DMIYC players must correctly classify the higher number of samples, some of which are malicious and some benign, using as few features as possible. Our objective is to understand which features humans inspect before reaching a decision. When an analyst first encounters a new sample in our game, she has no prior knowledge of its nature. However, the more she analyzes its different characteristics, the more (in some sort of mental bayesian process) she updates her belief of whether the program is benign or malicious. While she might never be entirely convinced, once she reaches sufficient confidence about her decision, she would classify the sample and move on to the next one. To capture this process, we decided to present each sample with an initial blank report. The player is then instructed to add new features to the report by choosing them from a pre-defined catalog until she has gained enough information to make a *confident* binary classification.

DMIYC is divided into 20 rounds, and a player has 20 *potential* points for each round; however, each new set of features added to the report decreases the potential points by one, while if the player buys an "empty" feature (e.g., a sample without network activity), no points are subtracted. We set the cost of a feature to prevent players from clicking on every piece of information without any specific order and precisely capture which set of features are preferred by a given user.

If the sample is correctly classified, the player gets the remaining potential points; otherwise, no points are given in case of a wrong classification. Since there are 15 sets of features to choose from in each round, and the score starts from 20, the game guarantees that every correct answer always scores a positive amount of points between 5 and 20. The final score is computed by summing up all points obtained in each round multiplied by the number of correct answers (to increase the importance of correct classification). This makes $19 \cdot 20 \cdot 20 = 7600$ the highest possible score, corresponding to 19 points (because the player must buy at least one feature) for every 20 rounds in which the player has answered all 20 times correctly. As described in Section 4, we removed from our analysis the players who submitted at least one answer without inspecting any feature (some beginners who probably decided not to complete the assignment).

Given that the score, the number of correct answers, and the leaderboard are only visible at the end of the experiment, a participant has no additional information to optimize her strategy based on the result of other participants. For this reason, we did not employ any particular game-theoretical design.

Finally, to mimic the professional pressure that analysts encounter in their career, we added an overall limit of 60 minutes to complete the task over 20 samples, corresponding to three minutes per sample on average. In comparison, Ugarte-Pedrero et al. [82] reported that professional malware analysts often classify samples based on static and dynamic features in less than 30 seconds.

3.2 Game UI

A shortened version of the user interface of DMIYC is depicted in Figure 1. The control panel on the left of the game UI contains information about the game's progress, including the sample counter compared with the total number of samples to classify, the remaining time, and how many features the player has already added to the current report.

The sidebar also shows the list of available features, divided into two main groups: static and dynamic. The interface shows

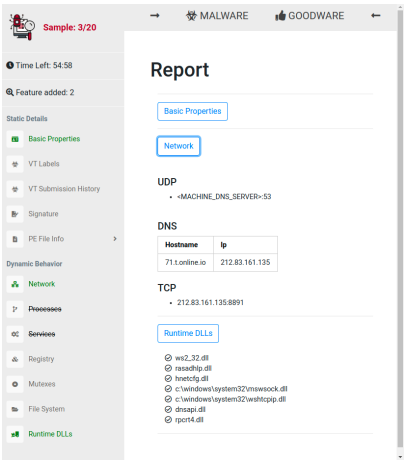

Figure 1: Shortened User Interface of "Detect Me If You Can!"

in green the features already added to the report and those that the player has not inspected in gray. Finally, a strike-through name means that the player has already tried to buy that feature, but it turned out to be empty. The top of the UI contains the player's two main buttons to classify the current sample as either malicious or benign, while the central part of the screen shows the current report. The 'Basic Properties' feature is collapsed in the picture, while 'Network' and 'Runtime DLLs' are expanded. This functionality is useful when the player deals with verbose outputs.

3.3 Features

Our game uses all features that are generally extracted from the file (static features) or the runtime behavior collected by a malware analysis sandbox (dynamic features). We summarized the complete list of features available to DMIYC players in Appendix 10.2.

Static captures different aspects of the file extracted by parsing the Portable Executable (PE) format file. PE is the format used by Microsoft Windows operating systems to store executables, object code, and DLLs. Our study is focused on PE because it is the most common file format of malware [7] in the desktop/server ecosystem.

Authenticode is a Microsoft code signing technology designed to guarantee the origin and integrity of an executable. Once an executable is signed, its code cannot change without breaking the envelope integrity. In this way, the user is guaranteed that the only code they are executing is created by the software publisher that signed it. Hence, the *Signature* feature contains the signer's certificate and if the certificate is valid or not.

The *Header Metadata* feature gathers together PE header info and metadata, like the target architecture (32 or 64 bit), the compilation timestamp, the release version, copyright strings, number of sections, and the total entropy of the file. A PE file consists of many headers and sections that tell the dynamic linker how to map the file into memory. For example, usually the `.text` section, which holds program code, is mapped as execute/readonly, and the `.data` section, holding global variables, is mapped as no-execute/readwrite. The *Section* feature contains all this information, plus the entropy of each section.

Software frequently depends on the functions exposed by the operating system. Windows exports most of its functions, called Application Programming Interfaces (API), required for these interactions in Dynamic Link Libary (DLL) files. Executables import and call these functions typically from various DLLs that provide different functionality. The functions an executable imports from other files (mostly DLLs) are called imported functions or, in short, *imports*.

The resources required by the executable file, such as icons, menu, dialog, etc., are stored in the resource section `.rsrc` of an executable file. Often, attackers store additional binary, decoy documents, and configuration data in the resource section, so examining the resource can reveal valuable information about a binary. The *Resources* feature contains the information of the resource section's entries, plus each resource's entropy.

Strings are ASCII and Unicode-printable sequences of characters embedded within a file. Extracting strings can give clues about the program functionality and indicators associated with a suspect binary. *Strings* feature contains, indeed, the strings extracted from the binary, and they can contain references to filenames, URLs, domain names, IP addresses, shell commands, registry keys, etc.

Finally, our static features also include results from VirusTotal engines, namely *VT Labels*. VirusTotal aggregates many antivirus products and online scan engines. Therefore, its reports contain how many engines analyzed the sample and how many classified the sample as malicious with the corresponding label. Detection labels by antivirus engines can be seen as a serialization of the tags an engine assigns to the sample, for example, the family name, the class of malware (e.g., ransomware, spyware, adware), file properties (e.g., packed, themida, bundle) and behaviors (e.g., spam, ddos, infosteal). Moreover, we also included the *VT Submission History*, which includes the first time it was submitted, and the submission names, i.e., a list containing the filename of the sample whenever it was submitted.

Dynamic features are extracted from the execution of the target sample in a controlled environment that logged all the interactions between the sample and the operating system. In this case, the names of the feature are self-explanatory: *Network* traffic summarized in UDP, TCP and HTTP; *Processes* created, terminated, or invoked shell commands; *Services* started, stopped or created; any modification to Windows *Registry* keys; *Mutexes* created or opened; *File System* operations (files read, written, and deleted); and the list of *Runtime DLLs*, i.e., dynamic libraries loaded at runtime.

3.4 Game Samples

We choose the samples to include in our experiments among real-world benign and malicious PE files. Each sample was submitted to VirusTotal, and the game information was extracted from its static and dynamic reports (we did not consider samples with an empty dynamic report). For the sake of precision, given the fact that Virus Total offers different

Table 1: Samples of the game

Sample	M|G	Malware Family	Description
1	M	hematite	file infector
2	M	kryptik	trojan
3	M	onlineio	adware
4	G	-	*Dell Backup & Recovery*
5	G	-	*TeamViewer*
6	M	sysn	dropper
7	G	-	*Google Chrome installer*
8	M	nanolocker	ransomware
9	M	doomjuice	worm
10	M	zbot	spyware
11	G	-	*Fallout 4 component*
12	G	-	*custom Autohotkey*
13	M	nitol	backdoor
14	G	-	*DOSBox*
15	M	zbot	packed spyware
16	M	nanocore	RAT
17	G	-	*WinDirStat*
18	G	-	*Java Update Checker*
19	G	-	*Media Player Classic*
20	M	zdowbot	keylogger downloader

dynamic reports from different sandboxes, we rely on the results generated by VirusTotal Jujubox [11]. Each report was also anonymized to prevent players from guessing the correct answer just by searching on the internet, for example, by looking for the sample's hash. We selected 11 malware and nine goodware; the list is summarized in Table 1, in the same order as they appear to the players. The rationale behind our choice was to create a representative dataset of the most common types of malware with their typical behaviors, while the goodware is composed of carefully verified third-party benign applications. Moreover, we verified that each sample report contained sufficient indicators of whether it was benign or malicious. For instance, sample #17 (WinDirStat, a disk usage statistics viewer) report contains many file operations, which might confuse it with ransomware, although all operations are read, and the sample does not write any files. On the other hand, sample #3 is signed, but the signature is invalid.

To mimic the scenario in which analysts need to classify previously-unknown samples based on their features (and not just on the AV labels), we removed the antivirus features for all except four reports. Among the remaining four, two were correctly classified by the antiviruses (one benign and one malicious), and one was a benign file misclassified as malicious. The last was a `zbot` (sample number 15) repacked with the PEtite file compressor. In this case, we manually reset all the antivirus engines' results to create a case of a malicious file with zero detection. In this way, we tried to reproduce a typical scenario in which malware authors re-release a new version of their software in a compressed form to bypass static signatures. Our game also includes another version of the `zbot` family (sample number 10), this time unpacked.

Finally, sample number 12 was custom-developed by us. It is a benign program that uses AutoHotKey (AHK) [2], a popular Windows scripting language that provides easy keyboard shortcuts and software automation. AHK is used by benign software but also, as reported by Trend Micro researchers [5, 10], by malware detected in the wild. What makes this sample interesting from a classification point of view is that the PE executables generated by AutoHotkey rely on Windows APIs to intercept events by using a hooking technique, the same also employed by some keyloggers. We did not develop a dummy example but a complete tool with useful functionalities: our program was designed to associate special key shortcuts to open different websites in the default browser.

It is important to note that all reports contained enough information to perform a correct classification, and we ensure that all signs of malicious behavior or harmless activity were captured in the respective features (e.g., we did not include malicious files that did not work correctly).

4 Participants

To recruit a sufficient number of participants with different backgrounds and expertise levels, we invited the participants by means of an invitation key that we sent to specific groups of candidates. Our *experts* have been recruited among companies and academic researchers in the malware analysis field, while *novices* were recruited among master students and beginner CTF players. Our participants are employed in seven renowned cybersecurity companies (a minimum of three to five participants for each company), while students (MSc and Ph.D.) and researchers were recruited from four universities located in two different countries. However, it is important to stress that we ensure that each participant we invited had some background in malware analysis. Apart from those who work in the sector, academic researchers have written papers related to malware analysis, students have attended at least a course on malware classification, and CTF players had experience reversing Windows malware.

More specifically, we generated different invite keys for the different groups. Then, when accessing the website for the registration, the participants had to enter some information about their job type, age, and years of experience in malware analysis. The reason we did not just rely on self-evaluating questions to classify experts or novices is that participants may adjust their answers to portray themselves as more or less skilled if they are concerned with the interviewer's perception [42, 80]. However, the fact that our novices are beginners has been confirmed by the fact that they reported dedicating zero years to malware analysis.

Cumulatively, we collected 145 registrations to the website, but we discarded 13 users who did not complete the game. Furthermore, we also removed the 35 players who did not show a sufficient amount of meaningful and reasonable activity, e.g., the ones who provided an answer in one or more rounds without watching any feature. We did not warn players about those post-processing controls as we wanted to use this information to identify those who were not genuinely interested in our experiment. Finally, 110 players were

Table 2: Data required for the registration

	Job			
	Student	**Researcher**	**Industry**	**Other**
Experts	-	7	27	4
Novices	72	-	-	-
	Age			
	[20-25]	**[26-30]**	**[31-40]**	**[40+]**
Experts	-	7	21	10
Novices	61	13	-	-
	Years of experience			
	[0]	**[1-3]**	**[4-6]**	**[7-9]**	**[10+]**
Experts	-	13	11	9	5
Novices	72	-	-	-	-

considered eligible, divided into 72 novices and 38 experts.

We summarize the results from the registration of each player in Table 2. When the job does not match any of the proposed categories (namely: student, academic researcher, and industry), we label it as *Other*. Consequently, such a division reflects the participants' age, with a broad group of people, most likely the students, aged 20-25. Remarkably, in our final dataset of experts, 9 participants have worked in malware analysis for 7-9 years, and five have worked in this field for more than ten years.

Human Study and Ethical Aspects – The methods adopted in this research are consistent with our institution's ethical guidelines, and the data collection and storage are compliant with the law in which our institution resides. All participants were informed about the purpose of our study and gave their explicit consent to take part in our experiment and to provide their age and employment information (no name or other personally identifiable information was collected in our study).

5 Analysis of humans' results

For each statistical test that we executed in this section, we collected the resulting p-values, and we used the Bonferroni correction method to correct them with an input α of $5e-2$. In the Bonferroni correction method, α determines the type-I error (rejecting the null hypothesis while it is actually true). All p-values we report in the rest of the paper already take into account this correction. It is worth noting that the conclusion of a hypothesis test in our study, i.e., whether a null hypothesis should be rejected, is drawn based on the statistical significance unveiled by the samples collected in the game. Lastly, we use σ and M to denote respectively the standard deviation and the median.

Table 3 reports a summary of the results of the 110 human players, grouped by their skill level. If we consider the average time needed to complete the game, the final score, and the number of correct answers, experts outperform novices,

Table 3: An overall view of the results

Metric	**E**xperts **N**ovices	**Min**	**Max**	**Avg**	**Std**	**Median**
Time	**E**	7:48	56:48	29:04	08:53	26:51
	N	8:14	59:58	44:31	10:05	46:32
Score	**E**	2310	5339	4103	742	4329
	N	1072	6042	3072	1054	2991
Right Answers	**E**	13	19	16.1	1.4	16
	N	8	19	13.7	2.4	14
Total Used Features	**E**	42	165	82.0	35.1	70
	N	37	146	81.7	27.5	68.5
Unique Used Features	**E**	7	16	13.4	2.6	14
	N	7	16	14.1	2.1	15

unsurprisingly. On average, experts took less time (29 minutes versus 44), scored more points (4103 versus 3072), and answered more questions correctly (16.1 versus 13.7). Since the classification is binary, even when the analyst is unsure about the nature of a program, she would still have at least a 50% probability of choosing the right category by chance. In fact, the standard deviation of the number of novices' correct answers (whom we expect to rely more on luck) is considerably higher than among experts (2.4 vs. 1.4).

Therefore, we performed a Welch's t-test to assess whether any statistically-significant difference exists between the two groups when considering I) the time needed to complete the game, II) the final score, and III) the number of correct answers. We assumed as a null hypothesis that there was no difference between the two groups, performed the test, and obtained a t-statistic value of I) -9.4, II) 6.4, and III) 5.8. The three values were supported by a p-value $< 1e-3$, which allows us to reject the null hypothesis and conclude that there are statistically significant differences in how the two groups of participants performed in our experiment.

On the contrary, if we look instead at the total and the unique number of features used by the two groups, we found no statistically significant difference (p-value $> 1e-3$). In other words, both experts and novices used a similar amount of features to reach a decision for each sample. Despite this, experts completed the game considerably faster than novices. In fact, if we look at the time spent on each feature, experts looked at each newly acquired information for an average of 11 seconds before either taking a decision or acquiring a new feature ($\sigma = 17, M = 6$). This time increases to 20 seconds for novices ($\sigma = 30, M = 10$), showing that inexperienced users take more time to spot the relevant part of an analysis report.

Finally, none of the 110 humans was able to classify all 20 samples correctly, and experts have an average accuracy of 80%. The line that separates malware from goodware can be very thin and, as we will see later in this section, some cases can easily deceive even the experts' eyes, especially when participants did not have the entire report at their disposal. In fact, even the five experts with 10+ years of experience in malware analysis misclassified $\langle 2,2,2,3,3 \rangle$ samples, respectively.

We also tested whether there was a correlation (Pearson) between years of experience and results (score or number of correct answers), and we found no statistically significant

Table 4: Correct answers when dealing with T/F P/N

Sample №	2	4	17	15
Type	TP	TN	FP	FN
Class	Malicious	Benign	Benign	Malicious
VT Matches	10	0	5	0
Experts	29/31 (93%)	27/29 (93%)	23/28 (82%)	28/28 (100%)
Novices	59/64 (92%)	57/63 (90%)	20/61 (32%)	49/61 (80%)

correlation.

5.1 VirusTotal Impact

In Section 3.4 we described how we have deliberately included in our game four samples with altered VirusTotal engines' reports to study how our participants react in response to correct and incorrect antivirus engine results.

The results, computed only on the users who have looked at the *AV labels* feature, are summarized in Table 4. Both experts and novices obtained good results when dealing with correct AV detections. However, the accuracy of novices significantly decreased when the antivirus reports were misleading, particularly for the benign file with five detections. Experts were less affected by this, probably because they are more aware that both false positives and false negatives are pretty frequent in VirusTotal. In fact, in 93% of the cases, expert analysts correctly flagged sample 4 as a benign sample. However, the behavioral analysis reports many file readings, and we manually inserted some misleading AV labels implying that it is generic ransomware. Nevertheless, the false positive (sample 17) is `WinDirStat`, an open-source graphical disk usage analyzer; as a matter of fact, its *FileSystem* feature does not show any writings on files as real ransomware (e.g., `nanolocker`, sample 8) would do. Surprisingly, *all* the experts correctly classified the false negative (sample 15), showing that this type of error does not affect their decision-making, while the same cannot be said for the false positive.

5.2 Self Evaluation

At the end of the exercise, we asked each participant to indicate the number of samples that they found difficult to classify and for which the user was unsure about the final choice.

Overall, experts have wrongly classified on average 4.0 ($\sigma = 1.4, M = 4$) samples, and they reported to be unsure about 3.6 ($\sigma = 3.2, M = 4$) of their answers. Novices have responded incorrectly for 6.3 ($\sigma = 2.4, M = 6$) samples while reporting being unsure about 6.7 ($\sigma = 6.1, M = 5.5$) answers. It is interesting to note that, on average, the number of mistakes is lower than the number of samples that users found difficult to classify. We can expect some of those complex cases to be classified correctly by luck. However, roughly half of the participants (49% of experts and 54% of novices) overestimated their performance by providing many wrong answers greater than the number of samples they were unsure about the choice. In one extreme case, one expert made seven mistakes while reporting to be unsure about only three of her choices.

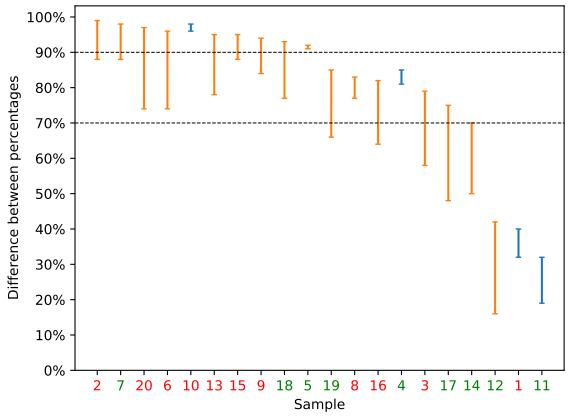

Figure 2: Comparison of correct answers experts/novices

5.3 Samples Difficulty

As we already mentioned in Section 3, we intentionally selected samples of different complexity for our experiment. Based on the experts' results, we can identify three classes: 10 samples were classified correctly by over 90% of the experts, seven samples were classified correctly by 70-90% of the experts, and three samples posed severe problems to most of our participants. These samples, one malicious and two benign (number 1, 11, 12 in Table 1), were, in fact, misclassified by over 60% of our experts.

One is a file infector, which could explain why its report may contain signs of both malicious and benign behaviors. The second is part of a popular videogame, a category that often employs obfuscation techniques often associated with malicious files. The last is an executable we generated using AutoHotKey to create shortcuts to open different URLs in the browser – and that therefore requires intercepting keyboard events.

Figure 2, inspired by the finance candlestick chart, shows for each sample the difference in classification accuracy between experts and novices (the orange line corresponds to cases in which experts answered more correctly than novices and the blue line vice versa). The samples are sorted in descending order with respect to the correct answers of the experts. We also plotted two dashed lines to indicate the 90% and 70% thresholds we used to differentiate samples' difficulty and the sample's number (in direct correspondence with Table 1) in the X-axis is in red for malware and green for goodware. The graph shows that for most of the samples, experts outperformed novices. In two cases (samples 10 and 4), the novices had a slight advantage, which became much more significant for two out of three problematic samples, probably because most experts were misled and responded consistently; instead, novices were not sure and responded more randomly.

5.4 Feature Ranking

The main goal of our work is not to measure how well humans perform at malware classification but, in particular, to understand which features they rely upon for making their decisions. Therefore, we computed how often our participants

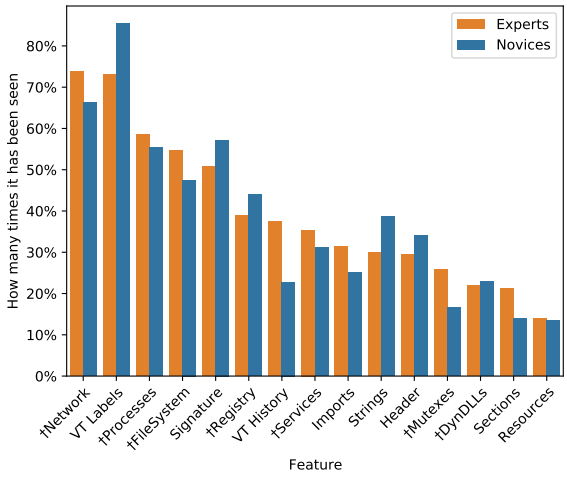

Figure 3: Features prevalence

used a given feature during the game. We recall that a player is not aware whether a feature is present or not until she clicks on it (as described in Section 3), therefore in this particular discussion, we are considering the features irrespective of their presence or absence, but by the fact that the player decided to look at them. The result, which is one of the main findings of this study, was quite surprising: independently of their level of expertise, all players relied on the same set of features.

The complete list of features with their prevalence for experts and novices is shown in Figure 3; the features marked with a dagger (†) are the dynamic ones. Percentages are computed over the total number of classified samples, i.e., 760 for the 38 experts and 1440 for the 72 novices. The features are ordered by the prevalence among experts.

We also investigated the five most frequent entries for experts and novices, broken down for the samples that were classified correctly and those that were misclassified (summarized in Table 7 in the Appendix 10.1). All lists are strikingly similar irrespective of the correctness of the classsification. For the experts are: †Network, VT labels, †Processes, †FileSystem, and Signature. While for novices: VT labels, †Network, Signature, †Processes, and †FileSystem.

Even if VT labels was only present on 4/20 samples, participants often tried to consult it. We remind the reader that at the beginning of the round, the player could not know if it was an empty feature or not until she tried to click on it. Therefore, we counted whether the feature was present or not because in this measurement, we are interested in the number of times the player wanted to view a specific feature.

By observing Figure 3, static file features (i.e., related to the PE file format and therefore without considering VT-relative features) are more often located on the right side of the graph, thus suggesting a human tendency to focus on the dynamic behavior. The Welch's t-tests supported this intuition (p-values < 0.001), but just for experts, who relied more heavily on dynamic features between the two groups (static file/dynamic behavior – t-statistic value of -3.92). In Section 7 we will show how this was the exact opposite for ML algorithms.

5.5 Malware Vs. Goodware

Looking at Figure 2, it seems that the goodware (characterized by green numbers on the X-axis) is more shifted towards the right side of the graph, thus suggesting that our participants had more trouble classifying them w.r.t. malware (red numbers). Therefore, for each class (experts/novices), we counted how many times they misclassified each group (malware/goodware). First, we measured their distribution, and we found that experts, on average, misclassified 1.7 ($\sigma = 1.0$, $M = 1$) malicious samples and 2.3 ($\sigma = 2.3, M = 2$) benign ones. Novices, instead, 2.8 ($\sigma = 1.6, M = 2$) and 3.9 ($\sigma = 1.7, M = 3$).

When Ugarte Pedrero et al. [82] asked malware analysts to help them classify previously unknown clusters, they noticed that malicious samples required less effort to classify than benign applications. Therefore, we further investigated and measured the average time spent and the number of watched features while analyzing each misclassified sample. However, maybe because there was a maximum amount of time to complete the game, on average, we did not notice a significant difference in the time: experts spent roughly 80 seconds per sample and novices 120 seconds, regardless of the sample's nature. While for the number of watched features, on average, experts used 14.8 ($\sigma = 11.1, M = 12$) features when analyzing malware and 17.4 ($\sigma = 11.3, M = 15$) for goodware. For novices the difference was less pronounced, 11.2 ($\sigma = 1.6$, $M = 10$) and 12.6 ($\sigma = 2.9, M = 11$) respectively.

We performed a Welch's t-test to assess whether any statistically-significant difference exists between the two groups (malware/goodware in this case) when considering I) the number of misclassified samples, II) the average time spent on each misclassified sample, and III) the number of watched features when they misclassified a sample. We assumed as a null hypothesis that there was no difference between the two groups and performed the test, and we repeated the tests for each class (novices/experts). The results we obtained allow us to draw the following conclusions. I) Novices made more errors classifying goodware (t-statistic -2.8, p-value $< 1e-3$), II) for both classes, analyzing goodware took no longer than malware (p-values $> 1e-3$), and III) experts used more features when analyzing goodware (t-statistic -0.6, p-value $< 1e-3$). The last point seems particularly reasonable because, in order to be confident that a sample is benign, one has to rule out *all* possible 'signs' of malicious behavior.

6 Machine Learning Players

Quite surprisingly, we found a consensus among humans about the key features used for malware classification, independently of their background and experience level. We further introduce the machine learning-based classifiers and compare their modus operandi with human experts. It is crucial to point out that we excluded two features, *VT labels* and *VT history*, when dealing with ML players because we used them to select and label the malware/goodware samples to form our benchmark dataset. Therefore, they must be removed to avoid overfitting

the ML models. In total, ML players have 13 features at their disposal, while humans have 15.

6.1 Benchmark Dataset

We created a well-balanced dataset for training and validation by downloading 21,944 VirusTotal complete reports of PE executables. We only selected the reports that contain a complete behavioral analysis of the sample, which VirusTotal obtains by running the samples in Jujubox [11] sandbox.

For malware, we randomly selected samples submitted to VirusTotal between 2018 and 2020 and classified them as malicious by more than 21 antivirus engines, as suggested by [59]. We ensured that no malware families were over-represented in our dataset using AVClass [73]. In total, we obtained a set of 10,972 malicious samples, in which the most frequent family had 125/10,972 occurrences (namely the 1.1% of the dataset).

We took a conservative strategy to build a goodware dataset based on the fresh installation of all the community-maintained packages (ensuring a rigorous moderation review process in order to avoid pollution) of Chocolatey [4] in a clean machine running Windows 10. After all, packages were installed, we extracted all the executable files present on the hard disk; therefore, our dataset contains a combination of third-party and Windows system files. Finally, we removed from our goodware dataset the files with more than three detections on VT. This allowed us to discard borderline cases (i.e., benign files with characteristics very similar to malware) found in Chocolatey's repository (e.g., hacking and scanning tools). Using this procedure, we have collected 10,972 goodware, and about 95% (10,374/10,972) of the samples in this set have zero detection.

6.2 Choice of ML-based Classification Models

Most of the features contained in the VT report are categorical. Therefore, we vectorized all VT sandbox reports based on the encoding scheme recently proposed by [44, 49, 62]. For completeness, in Appendix 10.3 we provide all the necessary details for the reproducibility of our work, and we will also share the source code used to conduct this experiment alongside all our dataset reports. We report this content in the Appendix because we used state-of-the-art approaches, and it is not a core contribution of our paper.

The literature contains various ML-based malware classification methods using static and dynamic analysis features. In [14, 29, 39, 43, 47, 51, 64, 70, 71, 92–94, 96], traditional models, e.g. Decision Trees (DT) and Random Forests (RF), were applied to the static analysis features. Advanced models, such as Deep Neural Networks [52, 68], treated raw binaries as gray-scale images and employed computer vision methods for malware classification. Recently, Graph Neural Network (GNN) [15, 20, 89] was applied to control flow graphs of malware. Long Short-Term Memory (LSTM) [41], Gated Recurrent Units (GRU) [23], and Recurrent Neural Nets (RNN) [49] were used over dynamic analysis reports containing features, e.g., system call traces, network activities, changes to the registry, and file actions executed by malware for malware classification. The success of these approaches confirms the applicability of ML-based approaches in extracting useful features of malware.

Our goal is to explain the contributions of the features extracted from static/dynamic analysis reports (described in Section 3.3) in Machine-Learning based malware classification. We opted for two popular Machine Learning models. The first is *Random Forest* [43, 47, 64, 70, 71, 94]. The other model is Convolution Neural Network (CNN) [13, 52, 90, 91]. The use of CNN is summarized in the Appendix. In our study, we first compress the categorical attributes into low-dimensional numerical embedding vectors, i.e., word2vec embeddings (as seen in Word2Vec [61]). The embeddings are treated as features of the CNN-based classifier. We use RF and CNN as two different players in the game, providing diversified perspectives of the classification boundary. RF uses the greedy tree-branch split strategy to divide the feature space and locate the classification boundary. In contrast, CNN inclines to directly fit the classification boundary in high-dimensional feature space by minimizing the correntropy loss. While the two ML players achieve high accuracy, their results are not always consistent. One may misclassify samples correctly classified by the other and vice versa.

6.3 Validation

We first evaluate the performances of the two ML players over the 21,944 training samples via a 5-fold cross-validation (CV) test. After that, we apply the ML models trained using the 21,944 samples to the 20 game samples. The CV test's role is to evaluate the ML player's classification accuracy using the encoded features.

In each fold of the CV test, we randomly select 80% of the benign and malicious samples to tune the model parameters of the ML players. The remaining 20% samples are used as the test set to evaluate the classification accuracy. The train-test split is repeated five times. We use the averaged and standard deviation of the *AUC-ROC* to measure the overall performance metric, as given by Table 8 in the Appendix, due to the space limit.

The reported AUC-ROC scores are achieved by utilizing 500 trees for Random Forest. For the CNN-based model, we choose a 4-layered CNN architecture implemented using PyTorch. The first layer is a word2vec embedding module, which compresses the categorical attributes in a static- or dynamic-analysis-based VT report into a 528 dimensional embedding vector. The second layer is composed of the first convolution filters with a size of 20 and the number of the output channels as 4. Following the convolution operation, a max pooling function is enforced over the convolution output. The third and fourth layers are given as fully connected layers to linearly transform the pooled convolution responses to a 64-dimensional and 2-dimensional vector, respectively. Finally, a sigmoid function is adopted to produce the decision score in $[0,1]$ of the binary classification. The averaged AUC scores of the RF and CNN-based participant derived from the test are *0.9962* and *0.9950*, indicating the effectiveness of the two models in capturing the feature-label correlation in the classification task.

7 Humans vs. Machines

Consistently with the observation in the CV test, the two ML players produce similar accuracy during the game. Both of them misclassified the game sample **3** (malicious) and **17** (benign). Besides, the RF-based players misclassified the benign game sample number **12**. In contrast, the CNN-based player misclassified the malicious game samples **4** and **15**. RF-player achieved slightly better accuracy than the average human expert (17/20), while the CNN-based acted like the average expert (16/20).

Sample **17** is a curious case because we used it to study how humans reacted to false positives, and, in a bizarre turn of events, both ML players made a false-positive decision. We remind the reader that ML players had no access to VT information, and among the 18 experts who did not inspect VT-related features, five (33%) made the same mistake.

Sample **3** is malware that directs to a malicious domain (`71.t.oneline[.]io`). Although we have no data to prove that humans did an internet search (we have set no limits on this), all experts who have correctly classified this sample have looked at the Network feature. Human subjects can proactively introduce additional knowledge beyond the training data set to help their decision-making. On the other hand, the ML-driven models are constrained by the statistical associations in the training data. Thus, without checking the domain's maliciousness, the ML models fail to recognize the file correctly.

Besides, most of the misclassified game files by the ML players and the human subjects are different. Only sample **12**, misclassified by 57% of the experts, is also misrecognized by the RF-based model. This shows that humans and ML players do not share the same idea of what constitutes a *difficult* sample to classify.

The unveiled difference is related to the human's ability to change the decision strategy depending on the situation. Indeed, according to cognitive psychology research [66], the human decision is usually made based on an ensemble of logical rules and concepts, such as permissions, obligations, prohibitions, and heuristics. On the one hand, human subjects can adjust the decision policies on a case-by-case basis, according to the available information. On the other hand, once trained, ML models are restricted to using the same set of features adopted by the learning paradigm. They cannot flexibly extend the feature space to enrich the description of the suspicious files, as the human subjects did in the game. As a result, different ML models would generate similar classification boundaries with the same training samples. Furthermore, the ML models are likely to fail without sufficient supporting information on the feature, whereas the human subjects can look for additional evidence to extend their knowledge base.

7.1 Feature Ranking

We adopt *SHAP* [12] as a model-agnostic model explanation tool to measure the contribution of each type of feature encoded from the VT reports. SHAP [78] computes the Shapley value of each type of feature and ranks the features in descending order. A *higher* Shapley value denotes *stronger* relevance of the corresponding feature in the malware classification task. The Shapley value origins from game theory and extends to evaluate feature-wise contributions in a given learning task [31]. It is represented as the weighted average of the marginal contribution of a feature concerning a machine learning model as in Definition II.2 and Definition II.3 in [32].

Table 5: Ranked features by the two ML players and experts

#	RF	CNN	Expert Humans
1	Resources	Resources	†Network
2	†Services	Sections	†Processes
3	Header Metadata	†Network	†FileSystem
4	†Network	†Runtime DLLs	Signature
5	Signature	Header Metadata	†Registry
6	†Runtime DLLs	Signature	†Services
7	Strings	†Services	Imports
8	Sections	†FileSystem	Strings
9	Imports	Strings	Header Metadata
10	†Mutexes	†Registry	†Mutexes
11	†Registry	†Mutexes	†Runtime DLLs
12	†FileSystem	Imports	Sections
13	†Processes	†Processes	Resources

Note that Random Forest also has a built-in feature importance evaluation tool, i.e., recursively feature elimination using out-of-bag error (OOB) evaluation. Nevertheless, the OOB-based feature importance measurement inclines to overestimate the importance of high-cardinality categorical variables. This makes not trustable the OOB-based feature importance computation in our study. Table 5 shows the ranked features in descending order according to the feature-wise Shapley scores derived from SHAP and expert humans (without considering the VT labels/history features as discussed in Section 6). The features marked with a dagger (†) are dynamic.

The first finding is that the two ML players share similar important features in the classification task. *Resources*, †*Network* and *Header Metadata* appear in the top 5 features for both ML players according to the feature-wise Shapley values, despite the difference of learning mechanisms of RF and CNN. Interestingly, also †*Network* and *Signature* appear within human experts' top five features.

It can also be noted that the two ML players prefer static features rather than the dynamic ones that are favored by human subjects. In the first half of the ranking (1-7), human participants have chosen five dynamic features and two static. In the second half (8-13), the two ML players have chosen two static features and four dynamic ones. In fact, ML players made significant use of static analysis features, and, amazingly, PE *Resources* is the most crucial for ML algorithms while the least important for both human experts and novice players.

As humans, we have tried to understand why ML models prefer these features with an in-depth look at the reports, and when necessary, we manually inspected the sample. We found that malware tends to: embed executables/DLLs or big raw data among resources, some PE header metadata contains random strings or non-ASCII characters, does not sign or use invalid signatures, and uses sections with non-standard names.

Some of our samples also had these three peculiarities; for example, in the same order discussed above, samples number 6, 20, 3, and 15. Respectively, the 96%, 89%, 61%, and 91% of expert humans who classified correctly those samples did not watch *Resources*, *Header Metadata*, *Signature* and *Sections*. As it turns out, humans and machines agree that *Signature* is a crucial feature, while for the remaining static features, humans did not need them to make a correct classification.

While more research works may be needed to underpin the actual reason behind such differences between humans and machines, two possible explanations come to mind. First, dynamic attributes are not always present in all the reports. Over the 21,944 malware and goodware samples, 83% of them have at least one dynamic feature containing missing values (i.e., NULL value). It is worth noting that such missing features are ordinary; a program does not need to use all the operating system's capabilities. For example, ransomware just needs to interact with the file system and the network, while a browser has no apparent need to create services.

In contrast, static features are rarely missing. Misobservations in the dynamic features weaken the statistical feature-label association and therefore downgrade the usefulness of the dynamic features in the ML models. This is also related to the difference in the decision logic of humans [66, 83] and ML-based models [26]. Human subjects' judgments are made based on a combination of heuristics and rules. Missing some features/attributes does not prevent human subjects from pursuing complementary rules to reach a correct conclusion. Nevertheless, the ML-based models, though they are usually superior in capturing the complex statistical correlation, they do not encode the causality reasoning rules [26]. Consequently, ML-based models are more prone to misobservations in training data than humans.

The feature encoding is more complicated for dynamic attributes, often containing semantically rich categorical data types (w.r.t. static features containing more numerical features). To handle the high dimensional and sparse one-hot encoded representation of these categorical features, the *Word2Vec* technique [60] was developed to compress the one-hot encoded features into a low-dimensional embedding space. Nevertheless, missing features induce unneglected negative impacts on the representation stability of the Word2Vec based embeddings [46, 60], which can cause model overfitting as a consequence. It has been a challenging and open problem to handle incomplete observations of discrete features.

Second, the feature space of the ML models *cannot* be enriched in an on-the-fly manner by incrementally collecting additional evidence of the security incidents. We witnessed this limit by analyzing sample **3**, misclassified by both ML-based players. Another example is the game file **16**, which resolves the domain `phone2347.ddns[.]net`: a human subject can easily note that it is a dynamic DNS resolver, while ML does not have access to this knowledge. Moreover, by checking only the *Gibberish Score* used in our encoding process (see Appendix A for more details), the domain is not random enough to be categorized as automatically generated. However, it still looks suspicious to the human subjects' eyes; overall, not what an analyst would expect from a sample that has nothing to do with phone calls. This is additional semantic information that is difficult for an automated ML algorithm to capture proactively.

7.2 Game Replay with the ML players

The comparison is still not fair enough: the ML players had access to ALL features for each sample, while human experts had to progressively select only a subset of them. This raises a new question: *If the ML players were to use the same set of features used by the human subjects in the game, how would they perform?* To answer this question and help us better unveil the difference in decision-making between human subjects and ML models, we conducted two new experiments. In the first, we selected *the five most used features* according to their frequency among human experts' gameplay records. In the second, we extended *the list to the top seven* (except for the VirusTotal features as discussed in Section 6.2).

We then use only these features as the input to the two ML players and retrain them with the 21,944 file samples. After that, we went over each round played by the top five human experts, and we asked the retrained ML models to classify the game samples involved in the rounds played by the top five experts as per the following settings.

We give the retrained ML players the same features that each human expert used in the game for each selected game sample. To perform a fair comparison, we only selected those game samples where the features used by the top human experts were all within the range of the most used 5 and 7 features selected to train the classifiers. Consequently, both the human experts and the ML players involved in the replay test had access to the same feature pool for each game sample. This limited our experiments to *8* game samples for the five-feature experiment and *13* game samples for the seven-feature experiment. It is possible that individual human experts use only a fraction of the most used 5 and 7 features when they play the game. Given a game sample i, for the features in the top-ranked list that were not used by the human subject, we consider them as missing features in the input to the ML models for classifying this sample. To handle the missed feature values, both RF and CNN-based models first conduct imputations by completing them with the most frequently observed values in the corresponding feature dimensions. Then the classifiers are applied over the completed input feature vectors.

At first, for the two ML players using only the 5 and 7 features, we conducted the 5-fold CV test on the 21,944 samples to check the overall accuracy of the ML players with the restricted feature space. We then apply the retrained models with the top 5 and 7 features over the 20 game samples and report the number of misclassified game files. Table 9 in Appendix.10.6 reports the *average* and *standard deviation* of the ROC-AUC scores produced by the restricted ML models in the 5-fold CV test. While the overall accuracy of the two ML players is lower than when trained using all features, the ROC-AUC remains high. With the top 7 features, RF and CNN reach an ROC-AUC value of 0.95 and 0.97 respectively.

Table 6 reports the number of the *correctly* classified game

Table 6: Game Replay Accuracy w.r.t. ML players

Methods	Top 5 features	Top 7 features
RF	4/8	5/13
CNN	3/8	5/13
Human Experts	8/8	13/13

samples by the two ML players in the replay test. In the table, we use x/y to denote that x game samples out of the whole y samples are correctly classified. Interestingly, the game classification accuracy dropped significantly when the ML players replayed experts' games. First, the game performance of the two ML players is worse than a random guess. In contrast, human experts did a perfect job, with all the samples correctly classified. Second, the accuracy score with the top 7 features was even worse than that derived with the top 5 features (e.g., $4/8$ v.s. $5/13$ for the RF model). The main reason is that there were many more missing feature values in the seven features, which caused the performance deterioration of the ML models. In contrast, human experts rarely used more than three features in their choices.

The cause of the performance degradation is twofold. **First**, human experts do not necessarily use all the features for decisions. As discussed before, human experts' judgment is made based on their own set of heuristics and rules learned from prior knowledge and experience of malware analysis practices [66, 83]. In the game replay test, 80% of the human experts used no more than three features while reaching the correct malware classification. 50% of them used less than five features. In contrast, with fewer features used, the less information the ML models can employ to learn the statistical association between features and labels. The classification accuracy over the game samples of both ML models thus becomes significantly lower than using the complete feature set. **Second**, while human experts can use partially observed information for their inference, any test inputs with incomplete features to a trained ML model inevitably harm the model's accuracy. The ML-driven models are thus less usable compared to manual investigation if training data and/or testing data are partially observed.

8 Key Takeaways

We conducted the first empirical study comparing humans (experts and novices) with machine intelligence in malware classification based on sandbox reports.

First, we found that both experts and novices base their decisions on the same set of features, with a slight difference in the order. As expected, the critical difference is that experts can provide a more accurate and fast classification with the same information. This finding holds among experts from different companies and countries and students from different universities. Moreover, we also found that humans and machines agree that network traffic and a valid signature are among the most important features.

Second, novices make the majority of mistakes during goodware classification. In fact, experts analyze more features when dealing with benign samples, confirming a recent observation [82] regarding the fact that benign files are the hardest to classify correctly for analysts because analysts have to rule out all possible malicious indicators. This difficulty with goodware for novices is probably due to the fact that they might be well-trained at detecting signs of potentially malicious behavior, while finding the absence of such signs is much more error-prone. Therefore, we need to improve this aspect during teaching. On the other hand, ML classifiers are designed to capture the data distribution of both benign and malicious samples in the feature space. It implies that the classification boundary established by ML classifiers may also help capture the common profiles of goodware; therefore, machines know distinctive traits for goodware and malware. We also found no agreement between human players and machine learning algorithms about which samples are more difficult to classify, and this fact suggest that machines and humans have complementary skills and a lot to still teach to each other.

Third, experts classify samples by using less than 1/3 of the available features, with a clear preference for dynamic attributes. Also, interviews of a recent work [95] bolsters the fact that malware analysis is trending towards behavior-based detection, because this can protect against undiscovered malware samples and remain effective for more extended periods of time [6]. This reflects the difference between human and machine intelligence. A human's decision depends on a flexible combination of candidate heuristics. Misobservations of the features do not prevent human experts from trying alternative heuristics. However, missing observations and a lack of semantic understanding of the features weaken the trustworthiness of the machine intelligence's decision. Echoed by other previous studies [44, 49], how to properly extract semantically meaningful logic is still a challenging task for ML models. As recently proposed in [35, 76], one potentially promising approach that could be used to address this issue could be the integration of generative models, such as AutoEncoder, into the classifiers and the use of causality inference in the decision process. The goal is to produce a semantically meaningful reconstruction of missing features and encourage exploring causality relations between features to deliver explainable classification. While these are exciting research directions, we emphasize once again that for our study, we used only state-of-the-art feature vectorization strategies, as described in Section 6.2.

Fourth, human analysts can consolidate the decision beyond the given behavioral reports by collecting additional evidence and knowledge. The process of proactive knowledge enriching is much more complex for ML models because the training strictly follows the definition of the feature space, which makes the derived models a passive process of knowledge encoding. By comparison, human subjects can strengthen their classification by checking suspicious records with additional information sources, even if the time limit imposed in our game limited this process to not much more than a quick query to a search engine. However, even if this is an inherent problem of ML, this particular case highlights the importance of enhancing ML models with open-source intelligence results.

Fifth, none of the human participants or computer algo-

rithms were able to classify all the 20 game samples correctly. Human experts classified correctly on average 16.1 ± 1.4 with a median of 16, i.e., an average accuracy of 80%. This number is further supported by the median value of the number of choices experts reported being unsure of, which is 4. Even if some samples were specifically crafted to reflect complex real-world situations (e.g., false positive/negative for AV products), we manually selected the samples to ensure that malicious or harmless activity indicators were always captured in the reports. However, it was fascinating to observe that the False Positive case posed many difficulties even to experts, while no expert made any mistakes on the False Negative – among those who looked at VT's labels. As FPs could irreparably damage the system functionality, as has happened several times before [1, 3, 9], antivirus companies pay close attention and perform extensive tests to reduce their number. Analysts, on the other hand, are accustomed to new malware variants and thus pay close attention to FNs, whereas in the presence of a positive AV match they tend to believe the machines' verdict.

As a last point, the results of this work can have a significant impact on the human-computer interaction for malware analysis during the review of sandbox reports. The computer must make all the data collected through OSINT available to human beings. Considering the domain name we discussed in Section 7 as an example, if the human had seen that it was a domain associated with malicious activity, he would have immediately known it was malware. Then, the use of ML models that precisely indicate what are the most significant parts of the features that helped classify the sample (RF for example) would help the human to point out which static malware parts are critical to the machine, so that she can focus on the behavior and bridging the cognitive gap seen throughout this paper.

9 Limitations and Conclusions

We introduced some design choices into the experimental setup, which may have caused biases in the final outcome as well as may have prevented us from unveiling further findings.

The granularity of the data collected with our web-based platform did not allow us to understand further the exact component of the feature that ignites the spark in the human mind. However, we argue that it is difficult to achieve such granularity. A sufficiently precise eye tracking device could provide excellent accuracy, but the experiment cannot be conducted remotely. As an alternative, the use of a restricted focus viewer to capture the part of the screen a user is currently focusing on is a standard methodology in comprehension experiments [48]. However, it would significantly impact the participants' speed, and it would require more time to conduct the experiments.

As Mantovani et al. [58], we did not offer money or goods to participate in this study because many participants were not allowed to receive compensation for their effort, and we did not want to discriminate among different classes of users. However, for some students involved in the study, the experiment gave them credits for their malware analysis exam, and experts voluntarily participated in the game.

The features list presented in the sidebar of our web UI always follows the same ordering, and it might have introduced a bias in the way our participants visited the features. However, we underline that by comparing the features list in the web-UI with the top-5 used features of table 7 only a tiny part of these follow the order of the UI list. Moreover, we introduced this design choice to mimic the VT interface that was needed not to affect the user experience and usability, especially for experts who work mainly with that interface.

Since we wanted to study existing ML solutions (and not design a new one for our experiments), we based our dynamic feature vectorization approach on what was described in recent papers [14, 39, 49, 96]. However, our tests show that more research is needed in this area (in accordance with [44]): dynamic features usually generate a very sparse representation that can lessen the stability of classification, especially when dynamic features contain misobservations. This is a possible reason why the classification performance deteriorates if we train an ML model on the same dynamic features chosen by humans. In general, the performance of ML-based classification and the features favored by the ML model depend significantly on the choice of the model and the training dataset.

Finally, expert analysts in our study likely misclassified some samples because of the game mechanics. For example, the cost of buying features might have reduced the accuracy; however, this has affected both experts and novices. In any case, our goal was to study which features were used more often by humans and compare them with those of machine learning algorithms. We want to stress that real-world classification is much more challenging, involving reports that did not capture the relevant features of the program or samples that did not run correctly in the sandbox, thus resulting in incomplete information. To cope with that, as extensively discussed in [82] and [95], malware analysts may use various techniques, including manual reverse engineering.

In the spirit of open science, we release [1] the source code we used to create the ML players (a tool to vectorize VirusTotal reports) and our game (that could have exciting teaching applications). While we cannot share the complete VirusTotal reports due to legal restrictions, we share the hashes of the samples we used in our dataset.

Acknowledgements

We sincerely thank the anonymous reviewers for their constructive feedback that has helped to improve this paper significantly, and Slasti Mormanti for his tireless support to our research group.

This work has benefited from a government grant managed by the National Research Agency under France 2030 with reference "ANR-22-PECY-0007," and the European Research Council (ERC) under the Horizon 2020 research and innovation program (grant agreement No 771844 BitCrumbs).

[1] https://gitlab.eurecom.fr/saonzo/DetectMeIfYouCan_ML

References

[1] Antivirus software webroot bricks pcs by deleting windows system files. https://liliputing.com/2017/04/whoops-antivirus-software-webroot-bricks-pcs-deleting-windows-system-files.html, Accessed September 20, 2022.

[2] Autohotkey, the scripting language for windows. https://www.autohotkey.com/, Accessed September 20, 2022.

[3] Catastrophic avira antivirus update bricks windows pcs. https://www.theregister.com/2012/05/16/avira_update_snafu/, Accessed September 20, 2022.

[4] Chocolatey, the package manager for windows. https://chocolatey.org/, Accessed September 20, 2022.

[5] Credential stealer targets us, canadian bank customers. https://www.trendmicro.com/en_us/research/20/l/stealth-credential-stealer-targets-us-canadian-bank-customers.html, Accessed September 20, 2022.

[6] How antivirus softwares are evolving with behaviour-based malware detection algorithms. https://analyticsindiamag.com/how-antivirus-softwares-are-evolving-with-behaviour-based-malware-detection-algorithms/, Accessed September 20, 2022.

[7] M-trends 2020. https://content.fireeye.com/m-trends/rpt-m-trends-2020, Accessed September 20, 2022.

[8] Malware statistics. https://www.av-test.org/en/statistics/malware/, Accessed September 20, 2022.

[9] Panda antivirus mistakenly flags itself as malware, bricks pcs. https://www.zdnet.com/article/panda-antivirus-mistakenly-flags-itself-as-malware-breaks-pcs/, Accessed September 20, 2022.

[10] Potential targeted attack uses autohotkey and excel. https://www.trendmicro.com/en_us/research/19/d/potential-targeted-attack-uses-autohotkey-and-malicious-script-embedded-in-excel-file-to-avoid-detection.html, Accessed September 20, 2022.

[11] Revamping in-house dynamic analysis with virustotal jujubox sandbox. https://blog.virustotal.com/2019/10/in-house-dynamic-analysis-virustotal-jujubox.html, Accessed September 20, 2022.

[12] Shapley additive explanations. https://shap.readthedocs.io/en/latest/index.html, Accessed September 20, 2022.

[13] Rakshit Agrawal, Jack W. Stokes, Karthik Selvaraj, and Mady Marinescu. Attention in recurrent neural networks for ransomware detection. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3222–3226, 2019.

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

[15] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. In *International Conference on Learning Representations*, 2018.

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

[17] Srilatha Attaluri, Scott McGhee, and Mark Stamp. Profile hidden markov models and metamorphic virus detection. *Journal in computer virology*, 5:151–169, 2009.

[18] Ulrich Bayer, Engin Kirda, and Christopher Kruegel. Improving the efficiency of dynamic malware analysis. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1871–1878, 2010.

[19] Noam Ben-Asher and Cleotilde Gonzalez. Effects of cyber security knowledge on attack detection. *Computers in Human Behavior*, 48:51–61, 2015.

[20] David Bieber, Charles Sutton, H. Larochelle, and Daniel Tarlow. Learning to execute programs with instruction pointer attention graph neural networks. *NeurIPS 2020*, abs/2010.12621, 2020.

[21] Juan C Burguillo. Using game theory and competition-based learning to stimulate student motivation and performance. *Computers & education*, 55(2):566–575, 2010.

[22] Jean-Marie Burkhardt, Françoise Détienne, and Susan Wiedenbeck. Mental representations constructed by experts and novices in object-oriented program comprehension. In *Human-Computer Interaction INTERACT'97*, pages 339–346. Springer, 1997.

[23] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[24] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. Understanding linux malware. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 161–175. IEEE, 2018.

[25] Christian Crumlish and Erin Malone. *Designing social interfaces: Principles, patterns, and practices for improving the user experience*. " O'Reilly Media, Inc.", 2009.

[26] Wang-Zhou Dai, Qiuling Xu, Yang Yu, and Zhi-Hua Zhou. Bridging machine learning and logical reasoning by abductive learning. In *Advances in Neural Information Processing Systems*, volume 32, pages 2815–2826, 2019.

[27] SL Shiva Darshan, MA Ajay Kumara, and CD Jaidhar. Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm. In *2016 11th International Conference on Industrial and Information Systems (ICIIS)*, pages 534–539. IEEE, 2016.

[28] Yuxin Ding, Xuebing Yuan, Ke Tang, Xiao Xiao, and Yibin Zhang. A fast malware detection algorithm based on objective-oriented association mining. *Computers and Security*, 39:315 – 324, 2013.

[29] Parvez Faruki, Vijay Laxmi, M. S. Gaur, and P. Vinod. Mining control flow graph as api call-grams to detect portable executable malware. In *Proceedings of the Fifth International Conference on Security of Information and Networks*, SIN '12, page 130–137. Association for Computing Machinery, 2012.

[30] Vikki Fix, Susan Wiedenbeck, and Jean Scholtz. Mental representations of programs by novices and experts. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 74–79, 1993.

[31] Christopher Frye, Colin Rowat, and Ilya Feige. Asymmetric shapley values: incorporating causal knowledge into model-agnostic explainability. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1229–1239. Curran Associates, Inc., 2020.

[32] Daniel Fryer, Inga Strümke, and Hien Nguyen. Shapley values for feature selection: The good, the bad, and the axioms, 2021.

[33] Daniel Gibert, Carles Mateu, and Jordi Planes. Hydra: A multimodal deep learning framework for malware classification. *Computers and Security*, 95:101873, 2020.

[34] Daniel Gibert, Carles Mateu, Jordi Planes, and Ramon Vicens. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, 15:15–28, 2019.

[35] David Grangier and Iain Melvin. Feature set embedding for incomplete data. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1*, NIPS'10, page 793–801, Red Hook, NY, USA, 2010. Curran Associates Inc.

[36] Leo Gugerty and Gary Olson. Debugging by skilled and novice programmers. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 171–174, 1986.

[37] Juho Hamari, Jonna Koivisto, and Harri Sarsa. Does gamification work?–a literature review of empirical studies on gamification. In *2014 47th Hawaii international conference on system sciences*, pages 3025–3034. Ieee, 2014.

[38] KyoungSoo Han, Jae Hyun Lim, and Eul Gyu Im. Malware analysis method using visualization of binary files. In *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, pages 317–321. 2013.

[39] Mehadi Hassen and Philip K. Chan. Scalable function call graph-based malware classification. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, page 239–248. Association for Computing Machinery, 2017.

[40] Hanan Hibshi, Travis Breaux, Maria Riaz, and Laurie Williams. Discovering decision-making patterns for security novices and experts. *Inst. for Softw. Research., Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-ISR-15-101*, 2015.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[42] Allyson L Holbrook, Melanie C Green, and Jon A Krosnick. Telephone versus face-to-face interviewing of national probability samples with long questionnaires: Comparisons of respondent satisficing and social desirability response bias. *Public opinion quarterly*, 67(1):79–125, 2003.

[43] Xin Hu, Sandeep Bhatkar, Kent Griffin, and Kang G. Shin. Mutantx-s: Scalable malware clustering based on static features. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, USENIX ATC'13, page 187–198. USENIX Association, 2013.

[44] Muhammad Ijaz, Muhammad Hanif Durad, and Maliha Ismail. Static and dynamic malware analysis using machine learning. In *2019 16th International bhurban conference on applied sciences and technology (IBCAST)*, pages 687–691. IEEE, 2019.

[45] Daisuke Inoue, Katsunari Yoshioka, Masashi Eto, Yuji Hoshizawa, and Koji Nakao. Automated malware analysis system and its sandbox for revealing malware's internal and external activities. *IEICE transactions on information and systems*, 92(5):945–954, 2009.

[46] Hemant Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone, and Michael S. Lauer. Random survival forests. *The Annals of Applied Statistics*, 2(3):841 – 860, 2008.

[47] Sachin Jain and Yogesh Kumar Meena. Byte level n–gram analysis for malware detection. In *International Conference on Information Processing 2011*, volume 157, pages 51–59, 2011.

[48] Anthony R Jansen, Alan F Blackwell, and Kim Marriott. A tool for tracking visual attention: The restricted focus viewer. *Behavior research methods, instruments, & computers*, 35(1):57–69, 2003.

[49] Chani Jindal, Christopher Salls, Hojjat Aghakhani, Keith Long, Christopher Kruegel, and Giovanni Vigna. Neurlux: Dynamic malware analysis without feature engineering. In *Proceedings of the 35th Annual Computer Security Applications Conference*, page 444–455, 2019.

[50] ElMouatez Billah Karbab and Mourad Debbabi. Maldy: Portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports. *Digital Investigation*, 28:S77–S87, 2019.

[51] Joris Kinable and Orestis Kostakis. Malware classification based on call graph clustering. *Journal in Computer Virology*, 7:233–245, 2011.

[52] Bojan Kolosnjaji, Ghadir Eraisha, George Webster, Apostolis Zarras, and Claudia Eckert. Empowering convolutional networks for malware classification and analysis. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3838–3845, 2017.

[53] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep learning for classification of malware system call sequences. In *Australasian Joint Conference on Artificial Intelligence*, pages 137–149. Springer, 2016.

[54] Deguang Kong and Guanhua Yan. Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1357–1365, 2013.

[55] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[56] SeolHwa Lee, Andrew Matteson, Danial Hooshyar, SongHyun Kim, JaeBum Jung, GiChun Nam, and HeuiSeok Lim. Comparing programming language comprehension between novice and expert programmers using eeg analysis. In *2016 IEEE 16th International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 350–355. IEEE, 2016.

[57] Liu Liu, Bao-sheng Wang, Bo Yu, and Qiu-xi Zhong. Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering*, 18(9):1336–1347, 2017.

[58] Alessandro Mantovani, Simone Aonzo, Yanick Fratantonio, and Davide Balzarotti. Re-mind: a first look inside the mind of a reverse engineer. In *31st USENIX Security Symposium (USENIX Security 2022)*. USENIX Association, August 2022.

[59] Alessandro Mantovani, Simone Aonzo, Xabier Ugarte-Pedrero, Alessio Merlo, and Davide Balzarotti. Prevalence and impact of low-entropy packing schemes in the malware ecosystem. In *Network and Distributed System Security (NDSS) Symposium NDSS 20*, 2020.

[60] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[61] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[62] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, et al. Reviewer integration and performance measurement for malware detection. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 122–141. Springer, 2016.

[63] Daisuke Miyamoto, Takuji Iimura, Gregory Blanc, Hajime Tazaki, and Youki Kadobayashi. Eyebit: Eye-tracking approach for enforcing phishing prevention habits. In *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, pages 56–65. IEEE, 2014.

[64] Robert Moskovitch, Dima Stopel, Clint Feher, Nir Nissim, Nathalie Japkowicz, and Yuval Elovici Elovici. Unknown malcode detection via text categorization and the imbalance problem. In *2008 IEEE International Conference on Intelligence and Security Informatics*, pages 156–161, 2008.

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

[66] Jean Oh Oh, Felipe Meneguzzi, and Katia Sycara. Chapter 11 - probabilistic plan recognition for proactive assistant agents. In Gita Sukthankar, Christopher Geib, Hung Hai Bui, David V. Pynadath, and Robert P. Goldman, editors, *Plan, Activity, and Intent Recognition*, pages 275 – 288. 2014.

[67] Jonas Pfoh, Christian Schneider, and Claudia Eckert. Leveraging string kernels for malware detection. In *Proceedings of International Conference on Network and System Security*, pages 206–219, 2013.

[68] Edmar Rezende, Guilherme Ruppert, Tiago Carvalho, Fabio Ramos, and Paulo De Geus. Malicious software classification using transfer learning of resnet-50 deep neural network. In *The16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1011–1014, 2017.

[69] Michael Sailer, Jan Ulrich Hense, Sarah Katharina Mayr, and Heinz Mandl. How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction. *Computers in Human Behavior*, 69:371–380, 2017.

[70] Ashkan Sami, Babak Yadegari, Hossein Rahimi, Naser Peiravian, Sattar Hashemi, and Ali Hamze. Malware detection based on mining api calls. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, page 1020–1025. Association for Computing Machinery, 2010.

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

[72] Josh Saxe, David Mentis, and Chris Greamo. Visualization of shared system call sequence relationships in large malware corpora. In *Proceedings of the ninth international symposium on visualization for cyber security*, pages 33–40, 2012.

[73] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 230–253. Springer, 2016.

[74] Giorgio Severi, Tim Leek, and Brendan Dolan-Gavitt. M alrec: compact full-trace malware recording for retrospective deep analysis. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–23. Springer, 2018.

[75] Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Daiki Chiba, and Takeshi Yada. Efficient dynamic malware analysis based on network behavior using deep learning. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2016.

[76] Marek Smieja, undefinedukasz Struski, Jacek Tabor, Bartosz Zieliński, and Przemysław Spurek. Processing of missing data by neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 2724–2734, Red Hook, NY, USA, 2018. Curran Associates Inc.

[77] Curtis Storlie, Blake Anderson, Scott Vander Wiel, Daniel Quist, Curtis Hash, and Nathan Brown. Stochastic identification of malware with dynamic traces. *Annuals of Applied Statistics*, 8:1–18, 2014.

[78] Erik Strumbelj. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41:647–655, 2014.

[79] Bo Sun, Akinori Fujino, Tatsuya Mori, Tao Ban, Takeshi Takahashi, and Daisuke Inoue. Automatically generating malware analysis reports using sandbox logs. *IEICE TRANSACTIONS on Information and Systems*, 101(11):2622–2632, 2018.

[80] Roger Tourangeau and Ting Yan. Sensitive questions in surveys. *Psychological bulletin*, 133(5):859, 2007.

[81] Philipp Trinius, Thorsten Holz, Jan Göbel, and Felix C Freiling. Visual analysis of malware behavior using treemaps and thread graphs. In *2009 6th International Workshop on Visualization for Cyber Security*, pages 33–38. IEEE, 2009.

[82] Xabier Ugarte-Pedrero, Mariano Graziano, and Davide Balzarotti. A close look at a daily dataset of malware samples. *ACM Transactions on Privacy and Security (TOPS)*, 22(1):1–30, 2019.

[83] J Van der Pligt. Decision making, psychology of. In *International Encyclopedia of the Social and Behavioral Sciences*, pages 3309 – 3315. 2001.

[84] Daniel Votipka, Seth Rabin, Kristopher Micinski, Jeffrey S Foster, and Michelle L Mazurek. An observational investigation of reverse engineers' processes. In *29th USENIX Security Symposium (USENIX Security 2020)*, pages 1875–1892, 2020.

[85] Kevin Werbach and Dan Hunter. *The gamification toolkit: dynamics, mechanics, and components for the win*. Wharton School Press, 2015.

[86] Susan Wiedenbeck, Vikki Fix, and Jean Scholtz. Characteristics of the mental representations of novice and expert programmers: an empirical study. *International Journal of Man-Machine Studies*, 39(5):793–812, 1993.

[87] Huijun Xiong, Prateek Malhotra, Deian Stefan, Chehai Wu, and Danfeng Yao. User-assisted host-based detection of outbound malware traffic. In *International Conference on Information and Communications Security*, pages 293–307. Springer, 2009.

[88] Khaled Yakdan, Sergej Dechand, Elmar Gerhards-Padilla, and Matthew Smith. Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 158–177. IEEE, 2016.

[89] Jiaqi Yan, Guanhua Yan, and Dong Jin. Classifying malware represented as control flow graphs using deep graph convolutional neural network. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 52–63, 2019.

[90] Chun Yang, Yu Wen, Jianbin Guo, Haitao Song, Linfeng Li, Haoyang Che, and Dan Meng. A convolutional neural network based classifier for uncompressed malware samples. In *Proceedings of the 1st Workshop on Security-Oriented Designs of Computer Architectures and Processors*, SecArch'18, page 15–17, New York, NY, USA, 2018. Association for Computing Machinery.

[91] Chun Yang, Jinghui Xu, Shuangshuang Liang, Yanna Wu, Yu Wen, Boyang Zhang, and Dan Meng. Deepmal: maliciousness-preserving adversarial instruction learning against static malware detection. *Cybersecurity*, 4:2523–3246, 2021.

[92] Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. An intelligent pe-malware detection system based on association mining. *Journal in Computer Virology*, 283(5), 2008.

[93] Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. Sbmds: an interpretable string based malware detection system using svm ensemble with bagging. *Journal in Computer Virology*, 283(5), 2008.

[94] Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. A survey on malware detection using data mining techniques. *ACM Comput. Surv.*, 50(3), June 2017.

[95] Miuyin Yong Wong, Matthew Landen, Manos Antonakakis, Douglas M Blough, Elissa M Redmiles, and Mustaque Ahamad. An inside look into the practice of malware analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3053–3069, 2021.

[96] Ding Yuxin and Zhu Siyi. Malware detection based on deep learning algorithm. *Neural Comput. Appl.*, 31(2):461–472, 2019.

[97] Olga A Zielinska, Allaire K Welk, Christopher B Mayhorn, and Emerson Murphy-Hill. Exploring expert and novice mental models of phishing. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 59, pages 1132–1136. SAGE Publications Sage CA: Los Angeles, CA, 2015.

10 Appendix

10.1 Most used features

In Table 7 we report the five most frequent entries for experts and novices broken down for the samples that were classified correctly and those that were misclassified.

Table 7: Most used five features

	All	Correct	Misclassified
Experts	†Network	†Network	†Network
	VT labels	VT labels	VT labels
	†Processes	†Processes	†Processes
	†FileSystem	†FileSystem	†FileSystem
	Signature	Signature	Signature
Novices	VT labels	VT labels	VT labels
	†Network	†Network	†Network
	Signature	Signature	†Processes
	†Processes	†Processes	Signature
	†FileSystem	†FileSystem	†FileSystem

10.2 List of the features in DMIYC

The list of features, divided into static and dynamic, available to DMIYC players.

Static Properties: Virus Total Labels, Virus Total Submission History, Signature, Header Metadata, Sections, Imports, Resources, Strings.

Dynamic Behavior: Network, Processes, Services, Registry, Mutexes, File System, Runtime DLLs.

10.3 Feature Vectorization

We present more details about how we performed feature vectorization (see Section 6.2).

First, we must introduce how and why, we used a *Gibberish Score* (GScore, in short) – gibberish means spoken or written words that have no meaning. Sandbox reports are rich with strings, like process names, file paths, and URLs, and legitimate ones are often meaningful names because such strings come from good programming practices where a name should make people understand why it exists, what it does, and how it is used. Hence, in addition to well-known practices for feature vectorization discussed above, we also involve our GScore that works as follows. We built a Markov Chain of character to character transition probabilities from the words in the English dictionary, files and folder names of a clean Windows installation, and the top 500 domain names of the Alexa ranking. To calculate the GScore of a given string, we walk through the Markov Chain and compute the product of the transition probabilities. Hence, our GScore is a function that takes a string and returns a real number between 0 and 1 (not included), i.e., $GScore : String \rightarrow (0,1)$. A high - close to 1 - GScore means that it is very likely that it is a meaningful string, while a lower - close to 0 - GScore indicates a high chance of a gibberish string.

Moreover, we make another little digression to explain how we managed the file paths, i.e., a file system location. When dealing with disk accesses, the sample under analysis transfers data between the main and a target file in the persistent memory. Even starting a process or dynamically loading a DLL involves disk access to retrieve the executable or the DLL. Since we encounter file paths very frequently, we have generalized the way we handle them. Given a string containing the file path, we consider the: parent folder, extension, and GScore of the file name. The parent folder suggests if the file resides in a reliable folder, like the system ones, while the extension on the Windows operating system defines the file type (e.g., `.exe` for executables). We consider the parent folder and the extension as categorical, while the GScore as numerical.

We discuss below the techniques we used to convert static and dynamic features into numerical feature vectors. Raw static and dynamic analysis reports are encoded into computable feature vectors to train Machine Learning based models in the malware classification task. Most of the raw attributes in the reports are categorical variables without an ordinal relationship. We follow a one-hot encoding scheme for each of these attributes: we first count all the unique categories that one such attribute can carry. Then each attribute is transformed into a bit vector, with each bit representing a possible category. One bit in the vector is set to 1, only if the attribute takes the corresponding category value. Otherwise, it is set to 0. We conclude by illustrating the remaining details about the feature encoding scheme for each feature class below.

Signature. We treat the signers (e.g., Microsoft, Dell, Google, etc.) as a categorical attribute, and we also dedicate one bit to indicate if the signature has been verified or not.

Header Metadata. First, we consider the section's name containing the entry point and the sub-system (if it is a command line or a GUI executable, and the target architecture) as categorical features. The timestamp, the size of the file, and the size of possible overlay data are used as numerical features.

Sections. We compute the summary statistics [2] of the size and entropy scores for each section, considering them as numerical features. Instead, we consider the names of the sections as a categorical feature.

Imports. For each entry, we merge the DLL's name with the function's name, and we consider this new string as a categorical feature.

Resources. We compute the summary statistics[2] of the entropy scores as numerical features. Besides, we consider the string that indicates the type of resource as categorical.

Strings. We filter all the string, using regular expressions, and we keep: URLs, registry keys, file paths, and domain names. Then, each remaining string is treated as a categorical attribute, while we calculate the summary statistics[2] of the GScores of the extracted strings.

[2] We refer to *summary statistics* to indicate min, max, average, median, and standard deviation of a series of numbers.

Network. We count the number of UDP, TCP, and DNS connections as numerical features separately while considering TCP and UDP ports as categorical. For each domain involved in the DNS protocol, we consider the top-level domain as categorical, while we compute the summary statistics[2] of the GScores on the remaining domain name.

Processes. We count the number of the injected, created, terminated, and shell commands as separated numerical features. Moreover, if we encounter a file path, we manage it as a categorical attribute as described before.

Services. We count the number of created and stopped services; for each service, we consider the summary statistics[2] of the GScores of the service names and each of the names as categorical.

Registry. Actions on the Windows registry can be summarized in: 'set' and 'delete.' Each action operates on a certain key; hence, we treat the keys involved in set and delete action separately, considering them as categorical.

Mutexes. We consider the count the number of mutexes created or opened (opening a mutex is often used to check if it exists), and the summary statistics[2] of the GScores calculated on mutex names as numerical, while mutexes' name as categorical.

File System. There are three types – written, read, and deleted – of operations on a file path; thus, we consider them separately. For example, for every written file, we have the parent folders, the extensions, and the summary statistics[2] of the GScores of the file names.

Runtime DLLs. Processes can load dynamically DLLs, and they can also control the location from which a DLL is loaded by specifying a full path. Therefore, we count separately how many DLLs are loaded: without specifying a path, a path within the Windows system folder, a path outside system folders. Then, every file path is treated as a categorical attribute in our standard way.

10.4 Validation of the ML participants

We first evaluate the classification performances of the two ML-based participants over the 21,944 training samples via a 5-fold cross-validation (CV) test. After that, we train the two classifiers with the 21,944 training samples and apply the final models to the 20 game samples. The cross-validation test's role is to evaluate the ML algorithms' classification capability when using the encoded feature vectors as input features.

In each fold of the CV test, we randomly select 80% of the benign and malicious samples from each class of the training data set in order to tune the model parameters of the ML participants. The remaining 20% samples are used as the validation set to evaluate the classification accuracy. We choose *AUC-ROC* score to evaluate the classification accuracy with well-balanced training and testing samples. The train-test split is repeated for five times. We use the averaged and standard deviation of the AUC-ROC scores to measure the overall performance metric as given in Table.8.

Table 8: 5-fold cross-validation test of the ML participants

	Random Forest	**Convolution Neural Network**
AUC-ROC	0.9962(4.324e-4)	0.9950(1.225e-4)
TPR with **FPR**=1%	0.9427(3.120e-4)	0.9386 (1.542e-4)

10.5 Choice of ML-based Malware Classifiers

The literature is full of ML-based classification methods applied to static and/or dynamic analysis reports of malware samples [14,16,17,28,29,39,43,47,51,64,67,70,71,77,92–94,96], for both malware detection and malware family classification.

In our study, we opted for two popular Machine Learning models. The first is *Random Forest*.According to previous works [14, 16, 29, 39, 43, 47, 51, 64, 70, 71, 92, 94], *Random Forest* can provide accurate detection using both static and dynamic analysis-based features. As a tree-structured classification mechanism, *Random Forest* can by-design handle the categorical attributes in the static/dynamic analysis reports, which can not be applied as a direct input to many other ML models, such as Support Vector Machine and Deep Learning models. The other model is Convolution Neural Network (CNN). As a modern machine learning model, CNN has been applied in the use of malware detection [13, 52, 90, 91], the first two [52, 90] transfer the binaries into gray images by treating bytes as the intensity values of each pixel. Then CNN is deployed to differentiate malware from benign executables in a similar way as in image classification [55]. [13] builds a CNN-based detector using the API call traces extracted from the dynamic analysis of malware. In our study, we use CNN with the categorical features provided in the VirusTotal reports. We first compress the categorical attributes into low-dimensional numerical embedding vectors, i.e., word2vec embeddings (as seen in Word2Vec [61]). The embedding vectors are treated as features of the CNN-based detector. Another related topic [15, 20, 89] is to apply Graph Neural Networks (a CNN extension to graph-structured data) on control flow graphs of malware. However, these GNN-based methods require the complete chain of operations to constitute the graph-structured feature representation, which are not provided in the VT reports.

10.6 CV test results using the restricted feature space

Table 9 reports the average and standard deviation of the ROC-AUC scores produced by the restricted ML models in the 5-fold CV test.

Table 9: ML players with the restricted feature space

Methods	**Top 5 features**	**Top 7 features**
RF	0.9554 (3.1013e-4)	0.9747 (9.7544e-4)
CNN	0.9410 (1.5000e-4)	0.9598 (1.0327e-4)