Debugging Malware Classification Models Based on Event Logs with Explainable AI

Joon-Young Gwak[1], Priti Wakodikar[1], Meng Wang[1], Guanhua Yan[1], Xiaokui Shu[2], Scott Stoller[3], and Ping Yang[1]

[1]State University of New York at Binghamton, Binghamton, NY, USA
[2]IBM Research, Yorktown Heights, NY, USA
[3]State University of New York at Stony Brook, Stony Brook, NY, USA

***Abstract*—As machine learning models find broader applications in cybersecurity, the importance of model explainability becomes more evident. In the area of malware detection, where the consequence of misclassification can be severe, explainability becomes crucial. AI explainers not only help understand the reasons behind malware classifications but also assist in fine-tuning models to improve detection accuracy. Additionally, AI explainers can serve as a valuable tool for error detection, ensuring accountability, and mitigating potential biases. In this paper, we demonstrate how AI explainers can play a vital role in identifying issues in data collection and enhancing our comprehension of the model's classification results. Our analysis of explanation results reveals several issues within the data collection process, including event loss and the presence of environment-specific information. Additionally, we have identified mislabelled samples based on the explanation results and shared lessons learned from our data collection efforts.**

I. Introduction

The efficacy of machine learning models depends on the quality of the training dataset. When training malware classification models, three biases can impact their accuracy. Firstly, as benign and malware samples are labelled by human based on their domain knowledge, the distinction between malware and benign samples can be ambiguous, which could cause labelling bias. Secondly, malware execution logs are typically collected in controlled environments such as inside virtual machines, which may not accurately represent real-world scenarios. Logs collected may also contain machine or environment specific information, leading to potential environment bias. Lastly, it is often challenging to determine which events to log during the data collection process, and the events selected can significantly influence the classification model's detection performance.

Evaluating the training dataset's quality and determining whether a model's classification results are based on behavioral differences between benign and malicious samples or external factors/biases can be intricate. It is also challenging to assess a model's potential for generalization to other datasets, as the training and testing dataset may lack the necessary diversity and may only be suitable for specific types of malware. AI explainers can serve as a diagnostic tool in the data collection and model analysis process, offering insights into the model's performance and the quality of the dataset. They may also shed light on the decision-making process of the model by identifying top features that are key for understanding the model's classification results [1]. The top features identified can help reveal potential biases and issues in the dataset, and assess the model's generalizability.

In this paper, we demonstrate how AI explainers can help identify issues in data collection and enhance our understanding of the model's malware classification results. We gathered event logs of benign and malicious applications running on the Windows operating system using the Event Tracing for Windows (ETW) tool. Two datasets were collected: the first consists of event logs of Windows PE (Portable Executable) malware collected from VirusShare [2] and benign PE applications including Microsoft Word, PowerPoint, Google Chrome, etc., whereas the second comprises logs of benign and malicious PowerShell Scripts obtained from [3]–[22]. Our analysis of explanation results reveals significant event loss when utilizing the ETW for log collection. This problem arises when the event generation rate substantially outpaces the event processing rate, causing new logs to overwrite old ones before they can be saved to disk. We devised multiple solutions to address event loss, including utilizing Elasticsearch [23] as our log storage solution, developing a selective collection method that gathers only logs of the target process and processes executed after it, and excluding unimportant features during the log collection process.

Our analysis of explanation results also reveals the presence of machine or environment-specific data within ETW logs. Such data could potentially be used by models to differentiate between benign and malicious samples, especially when benign and malicious logs are collected from different machines. Therefore, it is crucial to collect both benign and malicious samples from a large and diverse pool of computers to enhance the model's robustness and generalizability.

In a nutshell, our contributions are summarized below.

- We collected two datasets containing logs from both benign and malicious applications running on the Windows operating system using ETW. Our Random Forest classification model achieves an accuracy of 99.6% for one dataset and 85% for the other.
- We utilized the TreeSHAP [24] explainer to analyze the model's prediction results to gain insights into the model's performance and the data collection process.

Through the analysis, we identified several issues within the data collection process, including event loss and the presence of environment-specific information, and proposed solutions to address these issues.
- We used the explainer to identify mislabelled samples and shared lessons learned from our data collection efforts.

Organization: The rest of the paper is organized as follows. Section II provides a brief overview of ETW and the SHAP explainer. Section III presents the general workflow for debugging the malware classification model using explainable AI and highlights common biases encountered in practice. Section IV provides detailed information about our datasets and the machine learning algorithm we employed. Section V illustrates how TreeSHAP helps identify issues and biases in the two datasets and shares lessons learned during our data collection efforts. Related works are discussed in Section VI. Section VII concludes the paper.

II. Background

This section presents background on the SHAP explainer and ETW.

A. SHAP Explainer

SHAP (SHapley Additive Explanation) [25] is a type of additive feature attribution methods, which measure the contributions made by simplified input features to the prediction results by a target machine learning model. Simplified input features, such as superpixels in images, are commonly used by XAI (Explainable AI) methods to explain model predictions as they are usually more concise and human-understandable than the raw features. Formally speaking, given a blackbox prediction model f and M simplified input features, an additive feature attribution method uses a linear explanation model g as defined below:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i, \tag{1}$$

where $z' \in \{0,1\}^M$ and $\phi_i \in \mathbb{R}$ provides the effect of the i-th simplified input feature. Such a model can be used to explain locally prediction made by model f at data point x, which can be mapped from a simplified input x' with a mapping function h_x (i.e., $x = h_x(x')$) while having $x' \approx z'$.

Let $f_x(z') = f(h_x(z'))$ and $z' \setminus i$ denote setting $z'_i = 0$ in z'. It has been proven that when additive feature attribution methods are considered, the only explanation model that satisfies local accuracy, missingness, and consistency properties must have feature attributions defined below:

$$\phi_i(f,x) = \sum_{z' \in x'} \frac{|z'|!(M-|z'|-1)!}{M!}[f_x(z') - f_x(z' \setminus i)], \tag{2}$$

where $|z'|$ is the number of non-zero entries in z'. The ϕ_i in Eq. (2) are known as Shapley values in cooperative game theory.

The SHAP explainer uses conditional expectations to define the simplified input mapping function as follows: $f_x(z') = f(h_x(z')) = \mathbb{E}[f(z)|z_S]$, where S is the set of non-zero indices in z' and z_S means that features not in set S are assumed to have missing values. Moreover, $\phi_0 = f(h_x(\mathbf{0}))$ is the same as $\mathbb{E}[f(z)]$, where all simplified inputs are assumed to be missing or unknown.

Eq. (2) can be used to explain locally the classification result made by model f at data point x with $\phi_i(f,x)$ giving the attribution score of the i-th feature. By averaging the absolute Shapley values of each individual feature among all the data in dataset X, we can obtain its global importance as follows:

$$\Phi_i(f) = \frac{1}{|X|} \sum_{x \in X} |\phi_i(f,x)|. \tag{3}$$

A significant hurdle in applying the SHAP explainer in practice is its high computational burden. A straightforward implementation of SHAP shown in Eq. (2) requires to iterate all feature permutations, which can be prohibitive if the classification model uses a large number of features. When the classification model is tree-based, such as random forests and gradient boosted trees, a variant of SHAP explainer called TreeSHAP can significantly reduce the computational complexity [24]. In this paper, we utilize the Random Forest model for malware classification. While Random Forest provides measures for computing the global importance of features using model internals [26], TreeSHAP can be used to generate both local and global explanations. TreeSHAP also offers both positive and negative attributions for each feature in the prediction, enabling effective model debugging.

B. Event Tracing for Windows (ETW)

ETW is a logging and tracing software for Windows operating system developed by Microsoft, offering real-time monitoring, diagnostics, and performance analysis. It comprises three main components: producers, controllers, and consumers. Producers generate detailed event data from the operating system and applications, controllers manage logging configurations, and consumers analyze and process log messages. The Windows operating system features numerous essential built-in event providers, such as Microsoft-Windows-Kernel, Microsoft-Windows-Security-Auditing, Microsoft-Windows-Diagnostics-Performance, Microsoft-Windows-EventLog, Microsoft-Windows-PowerShell, Microsoft-Windows-TaskScheduler, Microsoft-Windows-Networking-Correlation, and Microsoft-Windows-Sysmon. These providers generate event traces that capture a wide range of critical system activities, including kernel-level events, security-related actions, performance metrics, event log changes, PowerShell usage, task scheduling, network events, and detailed system activity. System administrators and security analysts rely on event providers, especially kernel-level providers such Microsoft-Windows-Kernel-Process, to monitor system behaviors, diagnose issues, optimize performance, and enhance security. These kernel-level providers offer insights into core system operations, including process creation, thread management, and other crucial low-level activities.

(15, {'EventHeader': {'Size': 128, 'HeaderType': 0, 'Flags': 576, 'EventProperty': 0, 'ThreadId': 2552, 'ProcessId': 4136, 'TimeStamp': 132957191320371848, 'ProviderId': '{EDD08927-9CC4-4E65-B970-C2560FB5C289}', 'EventDescriptor': {'Id': 15, 'Version': 1, 'Channel': 16, 'Level': 4, 'Opcode': 0, 'Task': 15, 'Keyword': 9223372036854776096}, 'KernelTime': 14, 'UserTime': 15, 'ActivityId': '{00000000-0000-0000-0000-000000000000}'}, 'Task Name': 'READ', 'ByteOffset': '0x1AE000', 'Irp': '0xFFFFBB8E1D419C88', 'FileObject': '0xFFFFBB8E1CFCC890', 'FileKey': '0xFFFFBB8E1CE0AD60', 'IssuingThreadId': '2552', 'IOSize': '0x10000', 'IOFlags': '0x60043', 'ExtraFlags': '0x0', 'Description': ''})

Fig. 1. An example of an ETW event log entry.

Figure 1 gives an example of an ETW log entry collected by the Microsoft-Windows-Kernel-File provider identified by 'ProviderId' {EDD08927-9CC4-4E65-B970-C2560FB5C289} [27]. It contains a key called "EventHeader" [28], which specifies the event's origin, timing, nature, and other information that can aid users in interpreting and analyzing the event. Providers are associated with various keys. Some keys are shared across all providers, while some are specific to individual providers. For example, the 'ProviderId' key is shared across all providers, while 'FileKey' is specific to the File provider. In ETW logs, system activities are categorized by the 'TaskName' and 'Opcode' keys. 'TaskName' (e.g., 'READ') is a human-readable description of an activity, and 'Opcode' (e.g., 0) serves as a numeric identifier for a specific operation. Both keys can be valuable features for training machine learning models.

III. Debugging Workflow Based on Explainable AI

Malware classification models trained from event logs can be applied for both offline and online malware detection. In offline malware detection, suspicious applications (e.g., Windows PE files or PowerShell scripts) are dynamically executed within a controlled environment, where ETW logs are collected. These logs are used to predict whether these applications are benign or malicious based on pre-trained machine learning models. In online malware detection, ETW logs are continuously collected as audit logs, from which classification models can be trained and used to detect whether there are suspicious activities in a real computer system.

The solid-lined box in Figure 2 illustrates the common practice for training malware classification models for both scenarios, which includes the following four steps: (1) obtain labelled benign/malicious samples; (2) set up controlled execution environments and collect ETW event logs from sample executions; (3) extract features from ETW event logs; and (4) train malware classification models. During this process, three types of biases can occur in practice:

- **Labelling bias:** Malware samples are usually humanly labelled based on domain knowledge. However, the distinction between malware and benign samples can sometimes be blurry. The same file deletion PowerShell script can be malicious if it is used by the attacker to cause damage to the computer system, or can be benign if it is executed by a normal user to clean her files. When a malicious program executes as a process, it may have subsequent malicious activities not directly issued by the malicious process. For example, it may spawn another proprietary process or a PowerShell process to implement malicious activities or utilize system services such as Remote Desktop Services to execute commands on another host. It is challenging to define the the boundary of activities between benign and malicious programs. Labelling bias could be introduced in such scenarios.
- **Environment bias:** Due to the destructive nature of malware executions, execution logs are usually collected within a controlled environment (e.g., virtual machines) to prevent offensive malware traffic from being leaked into the operational network. However, these environments for collecting event logs to train the malware classification models may differ significantly from those where the models are deployed, particularly for online malware detection. In addition, the data collected may contain environment-specific information, leading to potential inaccuracies in the classification result.
- **Logging bias:** Depending upon the providers used, there can be a large volume of events recorded by ETW. The ETW event tracing framework allows to capture only certain events of interest by limiting the set of providers for event logging. During development of malware classification models, however, it is unclear what providers should be used or what events should be recorded from the selected providers. Such logging bias can affect the performance of the classification models later trained from the event logs to a large degree.

As illustrated in Figure 2, this work enhances the standard ML practice in training malware classification models by leveraging model explanation (dotted box). Assumed to be blackbox to the explainer, a malware classification model under development provides sample predictions to the explainer. Using the explanation results from the explainer, the human analyst identifies biases in the previous model training cycle

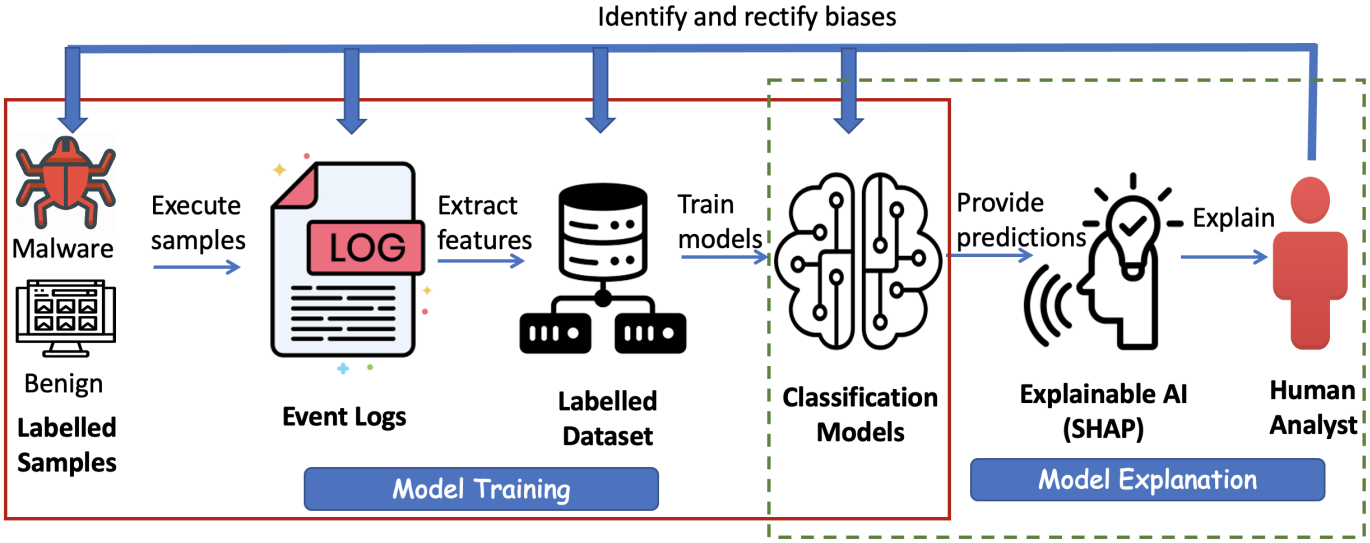

Fig. 2. Debugging workflow based on Explainable AI

and rectifies the issues in the relevant steps. While some of the same issues can be identified with an ad-hoc combination of other tools, this becomes less efficient when the datasets are larger. XAI methods offer a coherent approach to expose various kinds of biases in the datasets, thus enabling effective debugging of the malware classification models.

In this work we use the SHAP explainer to debug malware classification models for the following reasons. First, SHAP provides a unifying approach to interpreting model predictions with a strong theoretic foundation. This enables the proposed framework seen in Figure 2 to be a generic solution to debugging different kinds of malware classification models. Second, as discussed in Section II-A, the SHAP explainer can be used for both local explanations and global explanations. Such capabilities offer the human analysts more insights into model predictions than other explainers focusing on local explanations alone. Last but not least, the various options and plotting tools available with the SHAP explainer allow the human analysts to find the explanation method that suits best the classification model and the datasets in the task. For example, since the malware classification model uses the Random Forest classifier in this work, we use the TreeSHAP method to speed up explanation.

IV. Dataset Collection and Model Development

A. Datasets

We collected two datasets for malware classification using ETW, namely **Dataset I** and **Dataset II**. The logs in both datasets were collected from four event providers: [27]: (1) *Microsoft-Windows-Kernel-Process*, which records process and thread activities; (2) *Microsoft-Windows-Kernel-File*, which captures all file-related activities; (3) *Microsoft-Windows-Kernel-Registry*, which records Windows registry information; and (4) *Microsoft-Windows-Kernel-Network*, which records processes' and threads' network activities.

We utilized the pywintrace library [29] to gather event data from various ETW providers. This library offers a Python interface for interacting with ETW providers. During the data collection, we noted a considerable delay in the writing of ETW logs to the disk. Consequently, we allowed for a significant accumulation period for each data collection session.

Dataset I comprises 700 Windows PE (Portable Executable) malware samples collected from VirusShare [2] and 699 benign PE applications. Logs of malware samples were collected within a virtual machine (VM) running Windows 10, configured with a fake network environment [30] to isolate any potential impact of the malware within the VM. Benign logs were collected on Windows 10 machines to reflect the realistic setting. We collected logs from both user applications and system processes, with each sample corresponding to logs collected from a specific application.

Dataset II comprises ETW event logs of 148 benign and 148 malicious PowerShell scripts obtained from [3]–[22].

Table I gives the mean and the standard deviation of the log size (i.e., the number of log entries) of benign and malware samples in Datasets I and II. Malware samples exhibit a higher average number of log entries, suggesting that malware processes are generally more active than the benign applications. Additionally, both malicious and benign samples show great variability in log sizes, as indicated by their high standard deviations.

B. Machine Learning Algorithm and Data Pre-processing

We selected Random Forest [31], a tree ensemble model [32]–[34], as our malware detection model. During data pre-processing, our primary focus was to identify which ETW log attributes to use, ensuring that we selected features suitable for the model. Towards this end, we conducted a comprehensive analysis of ETW log attributes. We observed that certain ETW log attributes, such as the 'FileName'

TABLE I
THE AVERAGE NUMBER OF LOG ENTRIES OF BENIGN OR MALWARE SAMPLES.

	Dataset I (PE Executables)		Dataset II (PowerShell Scripts)	
	Mean	**Standard deviation**	**Mean**	**Standard deviation**
Benign	8920.67	26474.06	27741.41	25650.74
Malware	13134.27	24584.44	36428.23	32646.52

attribute, exhibit virtualization artifacts that are readily distinguishable. For instance, malicious logs collected from a VM consistently included 'HarddiskVolume2' in their 'FileName' attribute (e.g., /Device/HarddiskVolume2/Users/puma-4/AppData/Local/Temp/nsb83A0.tmp), whereas benign logs obtained from a physical machine consistently featured 'HarddiskVolume3'. Therefore, we excluded such attributes from consideration. We also observed attributes related to memory addresses, such as 'KeyObject', which represent hexadecimal values corresponding to the memory addresses of registry keys. Such attributes were excluded because address information is not generalizable across different systems. Based on our analysis, we concluded that the two most suitable choices for attributes are 'TaskName' and 'Opcode'. These attributes are universally provided by all event providers and convey essential information about the event type in a human-readable format, as discussed in Section II-B. Since the information provided by 'TaskName' and 'Opcode' are complementary, we merged them into a collection of features, each corresponding to a TaskName-Opcode combination. Our model uses only these features; other keys in log entries are unused. In our datasets, each data point corresponds to the logs generated by a specific process and its child processes. Similar to Yewale and Singh [35], we calculate the frequency of our Taskname-Opcode features from logs associated with a data point. These frequencies serve as the values for our features.

V. MODEL DEBUGGING

Our model debugging results involve both global and local explanation, which are discussed in Section II-A. The global importance of features (global explanation) offers a comprehensive overview of the features the model depends on at the dataset level. This information guides us in determining which features to prioritize for further analysis. Global explanation is visualized using global bar plot [36] where the x-axis represents the global importance value $\Phi_i(f)$ in Eq. (3), and the y-axis represents the features.

Explanations on local predictions (local explanation) offer sample-level insights into each feature's contribution to the prediction. Through these local explanations, we identified issues within our datasets that caused model predictions to be predominantly driven by external factors rather than behavioral distinctions between malicious and benign applications. We also identified samples that should be removed from our dataset due to mislabeling. Local explanation is visualized using waterfall plot [37] where the x-axis corresponds to the predicted probability of the malware class when considering the feature values along the y-axis for a specific sample.

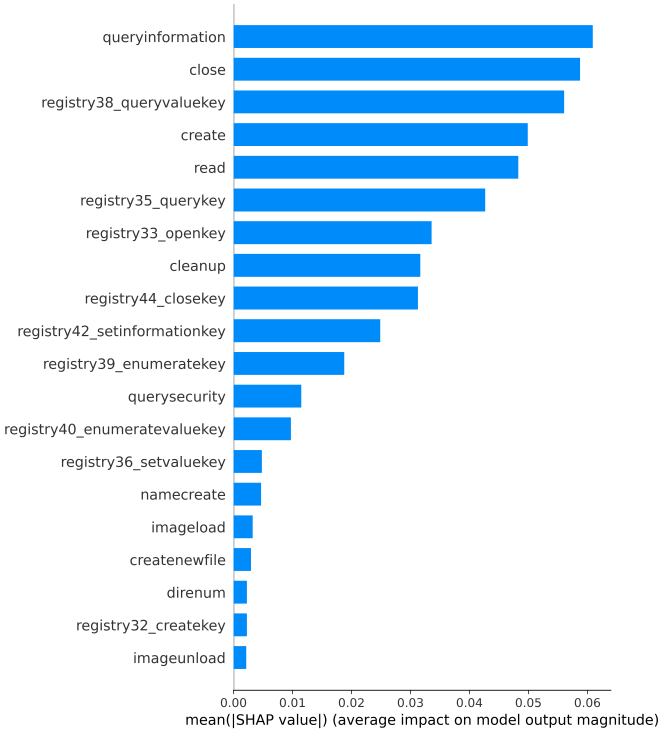

Fig. 3. Global explanation of model results for Dataset I

Blue bars with left-pointing arrows indicate feature values pushing for a benign classification, while red bars represent the opposite effect. The bar length denotes the extent of impact and corresponds to ϕ_i in Eq. (2). $E[f(X)]$ on the x-axis, which is the base value, is equivalent to $E[f(z)]$ in Section II-A. The base value represents the predicted probability of malware class if we did not have any information of the feature values of this sample [25]. $f(X)$ on the x-axis is equivalent to the sum of ϕ_i across all features, denoted as $\sum_i \phi_i(f, x)$, added by the base value. This represents the extent to which a sample resembles a malware sample.

A. Missing logs

The model trained on our initial collection of Dataset I achieved a perfect accuracy of 100%, and the global explanation (Figure 3) reveals that this exceptional performance results from contributions of various file and registry features. 'Query information', 'close', 'create', 'read', and 'cleanup' are file-related features, while 'query value key', 'query key', 'open key', 'close key', 'set information key', and 'enumerate key' are registry-related features. Local explanations on benign test samples of Dataset I reveal that the model's perfect prediction

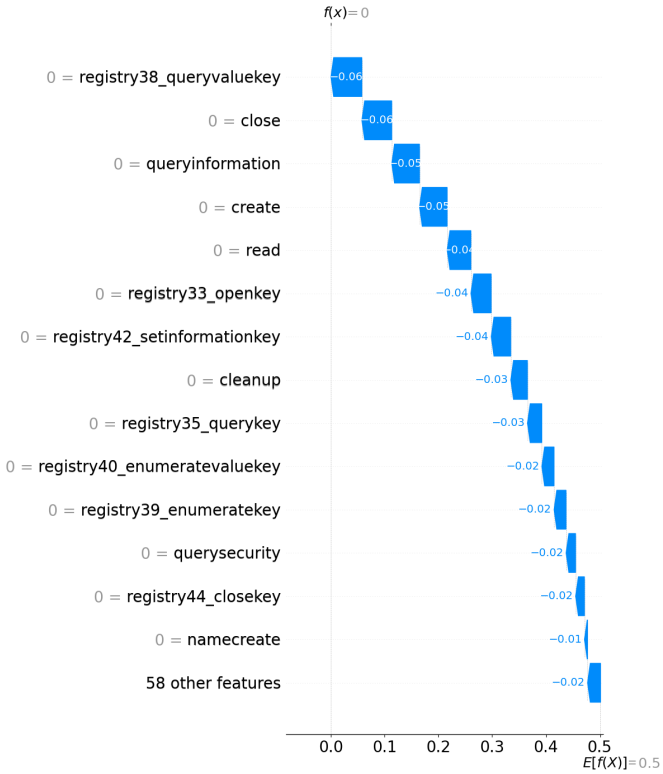

Fig. 4. Local explanation of correctly predicted benign sample (Dataset I)

result on benign samples are primarily due to all benign samples having zero feature values for all file and registry related features, as illustrated along the y-axis of Figure 4. This observation raises concerns about the correctness of the collected benign logs. A closer examination of the dataset shows that logs of many benign applications contained a large number of process and thread events such as 'thread work on behalf update', 'cpu priority change', and 'image load', accounting for approximately 50% of the total events. This issue is more noticeable in larger applications. While the excessive occurrence of 'thread work on behalf update' events is not inherently problematic, such a high frequency can serve as an indicator of missing events during the log collection process [38]. Further investigation revealed that this issue could stem from either (1) the slower disk I/O speed compared to the faster logging rate or (2) the real-time consumer's inability to keep up with the event generation rate, resulting in the overwriting of old logs by new ones before they could be saved on disk. To address (1), we stream the logs into Elasticsearch [23] rather than storing them on the disk. To tackle (2), we developed a selective log-collection method that collect only logs of the target process and processes executed after it. Subsequently, we recollected logs for benign applications. These strategies enabled us to capture the majority of events, although some event loss persisted as logs were collected from four providers and each provider generates events at a high rate.

Figure 5 presents the test accuracy of the Random Forest

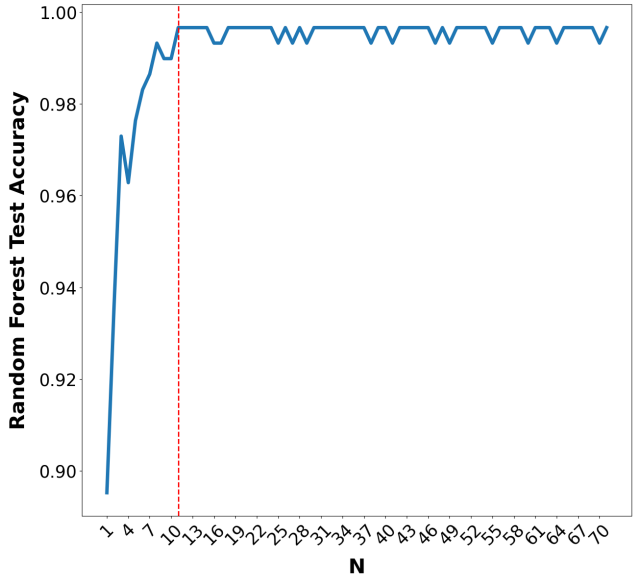

Fig. 5. Classification accuracy with top N features (Dataset I)

model trained using the top N features ($1 \leq N \leq 71$) based on the recollected Dataset I. The x-axis denotes the value of N and the y-axis represents the accuracy of the Random Forest model when utilizing the top $1-N$ features. The N features are arranged in descending order of their global importance. The Random Forest model achieves the overall accuracy of 99.6% with just its top 11 features. This suggests there are cases where not all features are critical for the model's performance. Therefore, for this specific dataset, we can leverage the information provided by the explainer to selectively subscribe to the identified subset of features by configuring the ETW session to emit only events of specific types from providers [39]. Alternatively, we can exclude the *Microsoft-Windows-Kernel-Network* provider without significantly impacting detection accuracy, because none of the top 11 features ('set information key', 'read', 'enumerate value key', 'name create', 'enumerate key', 'query ea', 'fsctl', 'thread stop', 'create', 'query security', and 'query value key') are generated by this provider. These approaches can help mitigate event loss in real-time detection by reducing both the volume of events to process and the associated event logging overhead.

Lesson 1: Data collection through ETW can result in data loss due to event generation rate significantly exceeding event processing rate.

B. Environment bias

The model trained on our initial collection of Dataset II also achieved perfect accuracy of 100%. Figure 6 gives the global importance bar plot for Dataset II, which shows that 'queryea', 'fsctl', 'query security', and 'name create' are the features with the greatest influence on overall predictions. 'queryea', 'query

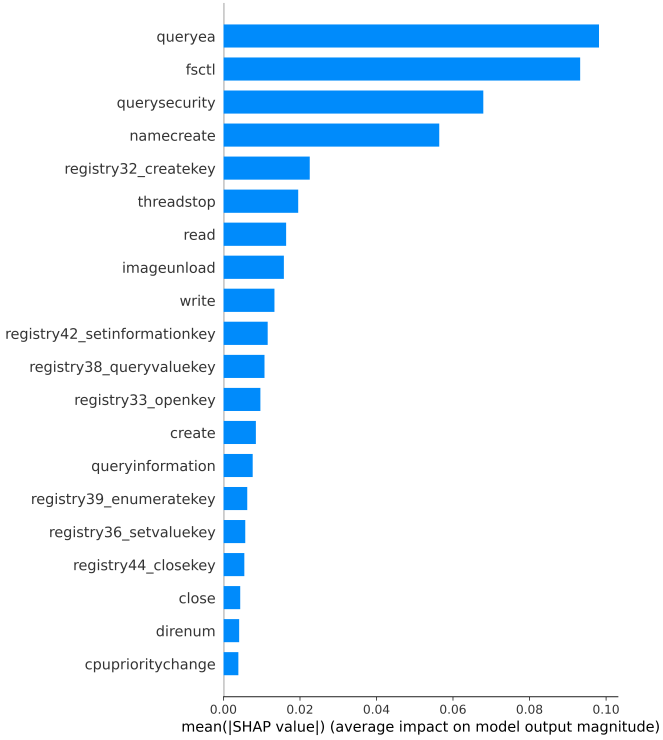

Fig. 6. Global explanation (Dataset II)

security', 'fstcl', and 'name create' are file I/O operations used to retrieve extended attributes of a file [40], inquire a file's security information [41], [42], request file system I/O control [43], [44], and create new files [45], respectively. Local explanations consistently indicate consistent low values for these four features in malware samples and high values in benign samples, as shown under the "With Env. Bias" column in Table II. This raises concerns regarding the accuracy of the collected dataset. As a result, we re-collected the benign and malware samples from the same environment. As shown under the "Without Env. Bias" column in Table II, both benign and malicious samples have more similar values for these 4 features. Collecting benign and malware samples from a single environment can potentially result in logs that contain environment/machine specific information (e.g. hardware and resource utilization). To prevent classification based on environment/machine specific information, we re-collected both benign and malware samples from multiple environments. Based on the re-collected Dataset II, Random Forest achieves 85% accuracy without exhibiting any consistent and significant contrast in features between benign and malware samples.

Lesson 2: Malicious and benign samples should be executed in a diverse pool of environments to prevent classification based on environment-specific features.

TABLE II
TOP 4 FEATURE STATISTICS: MALWARE VS. BENIGN (DATASET II)

	With Env. Bias				Without Env. Bias			
	Benign		Malware		Benign		Malware	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
queryea	78.9	22.4	1.5	7.2	4.4	15.9	6.4	17.1
fsctl	235	64.7	23.4	13.8	34.7	39.2	35.6	43.3
querysec.	293	417	31.8	17.4	46.4	43.4	48.2	50.7
namecreate	244	329	33.2	50.4	51.9	55.9	151	160

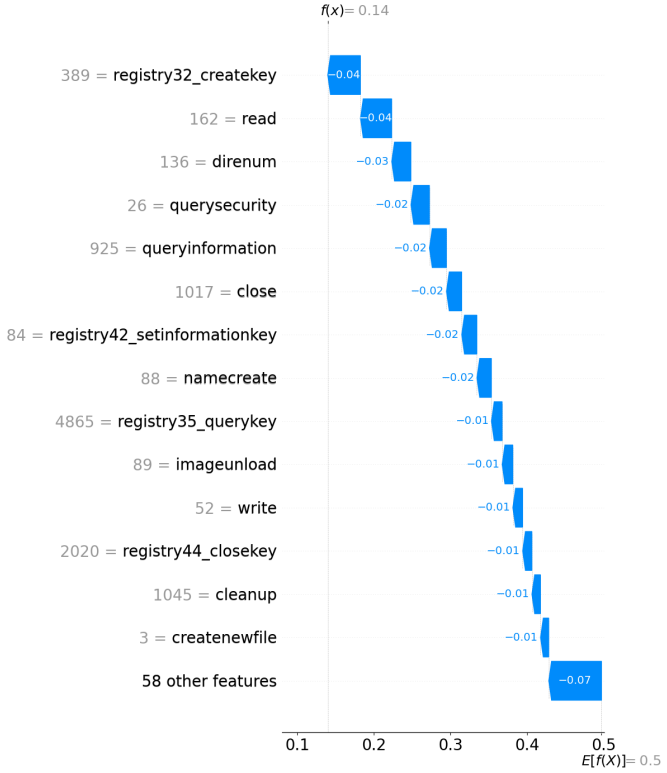

Fig. 7. Local explanation of mispredicted malware sample (Dataset II)

C. Labelling Bias

Malicious samples are typically labelled by humans based on domain knowledge. However, the distinction between malicious and benign samples can sometimes be blurry, leading to labelling bias. Local explanations on mispredicted samples can offer insights into how the model interprets these cases. Figure 7 gives the local explanation of a mispredicted malware sample from Dataset II. The predominance of blue bars with left arrows shows that all of its important features point to a benign classification, even though the sample is labeled as malicious. We examined the corresponding Powershell script [46], which includes a single line of code: `Get-EventLog -LogName * | ForEach { Clear-EventLog $_.Log }`. This script retrieves event logs in the system and clears them. It is labeled as malicious as it can be exploited by attackers to maliciously delete event logs of a system. However, computer systems may also routinely delete old event logs without malicious intent. We chose to

exclude these scripts from the training dataset instead of introducing a third label "ambiguous," as doing so would require a substantial amount of samples labeled as "ambiguous" to achieve high accuracy in multi-class classification.

Similarly, a Powershell script `Write-Host 'My voice is my passport, verify me.'` [47], which simply prints the specified string, has also been labeled as malicious. The local explanation also shows a large number of left-pointing blue bars suggesting that this sample is benign. As this script does not perform any malicious activities, it is unclear why it is labeled as malicious. We could upload these ambiguous samples to some malware classification services such as VirusTotal [48] for label confirmation. However, the labeling results from these services can be inconsistent or sometimes wrong [49], [50]. We thus opt for a more practical approach by excluding them from the training datasets.

Lesson 3: Local explanations can help identify mislabeled samples and thus improve the accuracy of the malware classification models.

VI. Related Work

Nadeem et. al. [51] systematically studied how to utilize explainable AI methods for security domains and identified three cybersecurity stakeholders, including model users, model designers, and adversaries. The work in [52]–[56] showed that explainable AI can offer decision support to model users, enabling them to prioritize threats, reduce workloads, and enhance efficiency by concentrating on high-level threats. Additionally, users can leverage explainable AI to identify false alarms by examining the explanations provided by the model for its predictions [57]–[61].

Explainable AI can also be leveraged by adversaries for offensive purposes. It provides adversaries with insights into the machine learning model, enabling them to execute attacks that compromise data confidentiality [62]–[65], as well as integrity and availability [64], [66]–[70].

Explainable AI can also serve as a tool for model designers to debug and validate the correctness of a model. Angelini et al. [71] propose to enhance the decision-making process in malware detection by providing visual insights into data, model decisions, and potential issues. Lemna [72] is a model-agnostic tool for explaining the decisions of deep learning security models. Kyadige et al. [73] proposed a multi-view deep neural network which produces a detection score based on feature vectors from PE file content and their respective file paths. They used the LIME explainer to perform an interpretability analysis, which aimed at verifying the classifier has learned a sensible representation and examining how the file path impacts changes in the classifier's score. Becker et al. [74] proposed a visual analytics system for deep learning models in multi-class DGA classification. Their approach addresses explainable AI challenges in understanding algorithmically generated data with complex semantics, particularly in cybersecurity. CADE [75] is a method designed for detecting and explaining concept drift samples in security applications. It helps model designers enhance model performance by understanding shifts in data patterns over time, allowing for timely adjustments to the model to maintain its effectiveness in dynamic environments. Dolej et al. [76] introduced a method to enhance the interpretability of machine learning-based results in malware detection by generating a set of rules that explain the decision-making process of the model. The above-mentioned works assume the correctness of the dataset used in machine learing. Our work, in contrast, utilizes explainable AI to identify issues and biases in the collected data, thereby improving the quality of the training data.

Several explainers were developed for traditional machine learning models, including SHAP [25], LIME [77], DALEX [78], and Grad-CAM [79]. Techniques have also been developed to explain the prediction results of Graph Neural Networks (GNNs), including GNNExplainer [80], PGExplainer [81], SubgraphX [82], CFGExplainer [83], PROVEXPLAINER [84], XGNN [85], RelEx [86], and GraphLIME [87]. Many of these methods generate explanations by identifying sub-graphs that have the most impact on malware classification results. In contrast, this paper does not introduce new explanation methods. Instead, it focuses on utilizing the SHAP explainer to detect potential issues within the datasets for malware classification.

VII. Conclusion

In this paper, we demonstrate how explainable AI (XAI) methods can help identify issues in the data collection process for malware detection and enhance understanding of the detection model's classification results. We utilized TreeSHAP to analyze the model predication results, through which we identified issues within the data collection process, including event loss and the presence of environment-specific information, and proposed solutions to address these issues. We also identified mislabelled samples and shared lessons learned from our data collection and classifier training efforts.

In the future we plan to extend this work as follows. First, this work considers only malware classification models trained from tabular feature data. Existing research suggests that graph-represented data extracted from static and dynamic analysis of malware samples, such as control flow graphs and provenance graphs, can also be used to build malware classification models with high predictive power. As SHAP cannot be applied to explain predictions by such models directly, we plan to investigate XAI methods for debugging graph-based malware classification models. Second, the effectiveness of XAI methods hinges upon how well they can assist human analysts in identifying critical issues that affect the performance of malware classification models. This challenge is particularly pronounced in malware classification tasks, where interpretation of explanation results hinges upon domain knowledge, unlike other classification tasks such as image classification where the explanations can be inherently human-understandable. We plan to develop methods that can bridge the gap between the explanations

produced by XAI tools and explanations expected by human analysts. We believe that such human-computer interactions are crucial to the success of XAI methods in cybersecurity applications, including malware detection. Additionally, we plan to investigate methods that leverage the human-in-the-loop process to enhance the efficiency of the explanation process [88]. Last but not least, in this work we have demonstrated that XAI methods are valuable to model developers in training malware classification models. We also plan to investigate how XAI methods can assist model users in understanding detection results from machine learning models. It is also interesting to study how XAI methods can be abused by attackers to develop more evasive malware variants.

Acknowledgement: This work is supported in part by a SUNY-IBM AI Research Alliance grant. We thank the anonymous reviewers for their constructive comments.

REFERENCES

[1] B. Pfeifer, A. Holzinger, and M. G. Schimek, "Robust random forest-based all-relevant feature ranks for trustworthy ai," *Studies in Health Technology and Informatics*, vol. 294, pp. 137–138, 2022.

[2] Virusshare. [Online]. Available: https://virusshare.com/

[3] C. Ross. (2019, May) Empire. [Online]. Available: https://github.com/EmpireProject/Empire/tree/master/data

[4] RiskyDissonance. (2022, March) poshc2. [Online]. Available: https://github.com/nettitude/PoshC2/tree/master/resources/modules

[5] N. Mittal. (2023, Feb) nishang. [Online]. Available: https://github.com/samratashok/nishang

[6] Will. (2020, Aug) Powersploit. [Online]. Available: https://github.com/PowerShellMafia/PowerSploit

[7] J. Russell. (2021, Feb) powershell-scripts. [Online]. Available: https://github.com/jrussellfreelance/powershell-scripts

[8] S. Sutherland. (2023, April) Powershellery. [Online]. Available: https://github.com/nullbind/Powershellery

[9] C. Ross. (2017, Dec) Randomps-scripts. [Online]. Available: https://github.com/xorrior/RandomPS-Scripts

[10] RaouzRouik. (2022, Aug) smallposh. [Online]. Available: https://github.com/RaouzRouik/smallposh

[11] BlackSnufkin. (2022, mar) pt-toolkit. [Online]. Available: https://github.com/BlackSnufkin/PT-ToolKit/tree/f78567ce9b4701acfd6af21196b95eef44bbc9c5/PowerShell-Scripts

[12] G. Tworek. (2023, Aug) Psbits. [Online]. Available: https://github.com/gtworek/PSBits

[13] (2023, May) Offsec-powershell. [Online]. Available: https://github.com/IAMinZoho/OFFSEC-PowerShell

[14] D. Nefedov. (2023, Aug) Sophia Community. [Online]. Available: https://github.com/farag2/Utilities

[15] J. Briggs. (2023, May) Psscripts. Actuarial Open Source Community. [Online]. Available: https://github.com/jimbrig/PSScripts

[16] B. Olin. (2023, Aug) Terminal-icons. [Online]. Available: https://github.com/devblackops/Terminal-Icons

[17] N. Rodriguez. (2023, feb) Powershell-scripts. [Online]. Available: https://github.com/nickrod518/PowerShell-Scripts

[18] E. G. Godoy. (2021, Oct) Sysadmin-survival-kit-scripts. [Online]. Available: https://github.com/ErickRock/Sysadmin-Survival-Kit-Scripts

[19] M. Fleschutz. (2023, Aug) Powershell. [Online]. Available: https://github.com/fleschutz/PowerShell/tree/master/Scripts

[20] P. Larrubia. (2023, April) Win-debloat-tools. [Online]. Available: https://github.com/LeDragoX/Win-Debloat-Tools/tree/main/src/scripts

[21] J. Hochwald. (2023, July) Powershell-collection. [Online]. Available: https://github.com/jhochwald/PowerShell-collection

[22] stevencohn. (2023, June) Windowspowershell. [Online]. Available: https://github.com/stevencohn/WindowsPowerShell

[23] P. Shukla and S. Kumar, *Learning elastic stack 7. 0 : distributed search, analytics, and visualization using elasticsearch, logstash, beats, and kibana.* Mumbai, India: Packt Publishing, 2019.

[24] S. M. Lundberg, G. G. Erion, and S.-I. Lee, "Consistent individualized feature attribution for tree ensembles," *arXiv preprint arXiv:1802.03888*, 2018.

[25] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 4768–4777.

[26] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," *Advances in neural information processing systems*, vol. 26, 2013.

[27] I. Yoshizaki. (20022, sept) Providers. github. [Online]. Available: https://gist.github.com/guitarrapc/35a94b908bad677a7310

[28] Microsoft Team. (2022, August) Microsoft. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/evntcons/ns-evntcons-event_header

[29] (2023, Mar) pywintrace. fireeye. [Online]. Available: https://github.com/fireeye/pywintrace/tree/master

[30] Fakenet-ng. [Online]. Available: https://github.com/mandiant/flare-fakenet-ng

[31] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[32] E. Mushtaq, F. Shahid, and A. Zameer, "A comparative study of machine learning models for malware detection," in *2022 19th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2022, pp. 677–681.

[33] H. Rathore, S. Agarwal, S. K. Sahay, and M. Sewak, "Malware detection using machine learning and deep learning," in *Big Data Analytics: 6th International Conference, BDA 2018, Warangal, India, December 18–21, 2018, Proceedings 6*. Springer, 2018, pp. 402–411.

[34] C. D. Morales-Molina, D. Santamaria-Guerrero, G. Sanchez-Perez, H. Perez-Meana, and A. Hernandez-Suarez, "Methodology for malware classification using a random forest classifier," in *2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, 2018, pp. 1–6.

[35] A. Yewale and M. Singh, "Malware detection based on opcode frequency," in *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, 2016, pp. 646–649.

[36] S. Lundberg. (2020, Sep) Shap global bar plot. [Online]. Available: https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/bar.html#Global-bar-plot

[37] ——. (2020, Jul) Shap waterfall plot. [Online]. Available: https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/waterfall.html#waterfall-plot

[38] Karl-Bridge-Microsoft. (2021, Jan) About event tracing - missing events. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/etw/about-event-tracing#missing-events

[39] M. Baranauskas. (2020, June) Etw: Event tracing for windows 101. [Online]. Available: https://www.ired.team/miscellaneous-reversing-forensics/windows-kernel-internals/etw-event-tracing-for-windows-101

[40] Microsoft. (2023, March) IRP-MJ-QUERY-EA (FS and filter drivers). [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/irp-mj-query-ea

[41] ——. (2023, March). [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/irp-mj-query-security

[42] ——. (2021, Oct) Getsecurityinfo function (aclapi.h). [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/aclapi/nf-aclapi-getsecurityinfo

[43] ——. (2023, March) About FSCTLs. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/about-fsctls

[44] ——. (2023, March) IRP-MJ-FILE-SYSTEM-CONTROL (FS and filter drivers). [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/irp-mj-file-system-control

[45] ——. (2021, Oct) Ntcreatefile function (winternl.h). [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/winternl/nf-winternl-ntcreatefile

[46] RaouzRouik. (2022, Aug) smallposh/cleareventlogs.ps1. [Online]. Available: https://github.com/RaouzRouik/smallposh/blob/main/cleareventlogs.ps1

[47] nullbind. (2014, July) Powershellery/runme.ps1. [Online]. Available: https://github.com/nullbind/Powershellery/blob/master/Brainstorming/runme.ps1

[48] Virustotal. [Online]. Available: https://www.virustotal.com/

[49] F. Maggi, A. Bellini, G. Salvaneschi, and S. Zanero, "Finding non-trivial malware naming inconsistencies," in *Information Systems Security: 7th International Conference, ICISS 2011, Kolkata, India, December 15-19, 2011, Procedings 7*. Springer, 2011, pp. 144–159.

[50] G. Yan, N. Brown, and D. Kong, "Exploring discriminatory features for automated malware classification," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013, pp. 41–61.

[51] A. Nadeem, D. Vos, C. Cao, L. Pajola, S. Dieck, R. Baumgartner, and S. Verwer, "Sok: Explainable machine learning for computer security applications," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2023, pp. 221–240.

[52] E. Holder and N. Wang, "Explainable artificial intelligence (xai) interactively working with humans as a junior cyber analyst," *Human-Intelligent Systems Integration*, vol. 3, no. 2, pp. 139–153, 2021.

[53] A. Nadeem, S. Verwer, S. Moskal, and S. J. Yang, "Alert-driven attack graph generation using s-pdfa," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 731–746, 2021.

[54] T. Van Ede, H. Aghakhani, N. Spahn, R. Bortolameotti, M. Cova, A. Continella, M. van Steen, A. Peter, C. Kruegel, and G. Vigna, "Deepcase: Semi-supervised contextual analysis of security events," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 522–539.

[55] G. De La Torre Parra, L. Selvera, J. Khoury, H. Irizarry, E. Bou-Harb, and P. Rad, "Interpretable federated transformer log learning for cloud threat forensics," *NDSS 22*, 2022.

[56] H. Li, J. Wu, H. Xu, G. Li, and M. Guizani, "Explainable intelligence-driven defense mechanism against advanced persistent threats: A joint edge game and ai approach," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 757–775, 2021.

[57] A. Sopan, M. Berninger, M. Mulakaluri, and R. Katakam, "Building a machine learning model for the soc, by the input from the soc, and analyzing it for the soc," in *2018 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE, 2018, pp. 1–8.

[58] B. Mahbooba, M. Timilsina, R. Sahal, and M. Serrano, "Explainable artificial intelligence (xai) to enhance trust management in intrusion detection systems using decision tree model," *Complexity*, vol. 2021, pp. 1–11, 2021.

[59] M. Szczepański, M. Choraś, M. Pawlicki, and R. Kozik, "Achieving explainability of intrusion detection system by hybrid oracle-explainer approach," in *2020 International Joint Conference on neural networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[60] K. de Bie, A. Lucic, and H. Haned, "To trust or not to trust a regressor: Estimating and explaining trustworthiness of regression predictions," *arXiv preprint arXiv:2104.06982*, 2021.

[61] M. Kinkead, S. Millar, N. McLaughlin, and P. O'Kane, "Towards explainable cnns for android malware detection," *Procedia Computer Science*, vol. 184, pp. 959–965, 2021.

[62] X. Zhao, W. Zhang, X. Xiao, and B. Lim, "Exploiting explanations for model inversion attacks," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 682–692.

[63] R. Shokri, M. Strobel, and Y. Zick, "On the privacy risks of model explanations," in *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, 2021, pp. 231–241.

[64] A. Kuppa and N.-A. Le-Khac, "Adversarial xai methods in cybersecurity," *IEEE transactions on information forensics and security*, vol. 16, pp. 4924–4938, 2021.

[65] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 399–414.

[66] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Explaining vulnerabilities of deep learning to adversarial malware binaries," *arXiv preprint arXiv:1901.03583*, 2019.

[67] G. Severi, J. Meyer, S. Coull, and A. Oprea, "Explanation-guided backdoor poisoning attacks against malware classifiers," in *30th USENIX security symposium (USENIX security 21)*, 2021, pp. 1487–1504.

[68] X. Wu, W. Guo, H. Wei, and X. Xing, "Adversarial policy training against deep reinforcement learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1883–1900.

[69] J. Xu, M. Xue, and S. Picek, "Explainability-based backdoor attacks against graph neural networks," in *Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning*, 2021, pp. 31–36.

[70] Z. Shu and G. Yan, "EAGLE: Evasion attacks guided by local explanations against android malware classification," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[71] M. Angelini, L. Aniello, S. Lenti, G. Santucci, and D. Ucci, "The goods, the bads and the uglies: Supporting decisions in malware detection through visual analytics," in *2017 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE, 2017, pp. 1–8.

[72] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 364–379.

[73] A. Kyadige, E. M. Rudd, and K. Berlin, "Learning from context: A multi-view deep learning architecture for malware detection," in *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2020, pp. 1–7.

[74] F. Becker, A. Drichel, C. Müller, and T. Ertl, "Interpretable visualizations of deep neural networks for domain generation algorithm detection," in *2020 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE, 2020, pp. 25–29.

[75] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "CADE: Detecting and explaining concept drift samples for security applications," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2327–2344.

[76] J. Dolejš and M. Jureček, "Interpretability of machine learning-based results of malware detection using a set of rules," in *Artificial Intelligence for Cybersecurity*. Springer, 2022, pp. 107–136.

[77] M. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," 02 2016, pp. 97–101.

[78] P. Biecek, "Dalex: Explainers for complex predictive models in r," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 3245–3249, 2018.

[79] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[80] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, *GNNExplainer: Generating Explanations for Graph Neural Networks*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[81] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Pgexplainer," 2020. [Online]. Available: https://github.com/flyingdoog/PGExplainer

[82] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji, "On explainability of graph neural networks via subgraph explorations," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 241–12 252.

[83] J. D. Herath, P. P. Wakodikar, P. Yang, and G. Yan, "Cfgexplainer: Explaining graph neural network-based malware classification from control flow graphs," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022, pp. 172–184.

[84] K. Mukherjee, J. Wiedemeier, T. Wang, M. Kim, F. Chen, M. Kantarcioglu, and K. Jee, "Interpreting gnn-based ids detections using provenance graph structural features," 2023.

[85] H. Yuan, J. Tang, X. Hu, and S. Ji, "Xgnn: Towards model-level explanations of graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 430–438.

[86] Y. Zhang, D. Defazio, and A. Ramesh, "Relex: A model-agnostic relational model explainer," in *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, 2021, pp. 1042–1049.

[87] Q. Huang, M. Yamada, Y. Tian, D. Singh, and Y. Chang, "Graphlime: Local interpretable model explanations for graph neural networks," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[88] A. Gevaert, A. Saranti, A. Holzinger, and Y. Saeys, "Efficient approximation of asymmetric shapley values using functional decomposition," in *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, 2023, pp. 13–30.