Computers & Security 133 (2023) 103409

Contents lists available at ScienceDirect

Computers & Security

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

XMal: A lightweight memory-based explainable obfuscated-malware detector

Mohammed M. Alani [a,b,*], Atefeh Mashatan [a], Ali Miri [c]

[a] Cybersecurity Research Lab, Toronto Metropolitan University, Toronto, Canada
[b] School of IT Administration and Security, Seneca College of Applied Arts and Technology, Toronto, Canada
[c] Computer Science Department, Toronto Metropolitan University, Toronto, Canada

ARTICLE INFO

Keywords:
Malware
Obfuscated malware
Malware detection
Machine learning
Explainable machine learning

ABSTRACT

An average of 560,000 new malware instances are being detected every day. Malware detection is becoming one of the biggest challenges in the field of computer security. The use of code obfuscation techniques by malicious actors is gaining popularity, further complicating the process of detection. In this paper, we introduce a lightweight obfuscated-malware detector based on machine learning that is also explainable. The proposed method, based on extreme gradient boost, employs only five features extracted from memory dumps, achieving a detection accuracy of over 99%. These five features were selected using recursive feature elimination, based on feature importance. Through testing, we demonstrated that the system was capable of detecting malware instances in just 0.413 µs. The model was explained using Shapley additive explanations.

1. Introduction

In recent years, there has been a significant rise in malware attacks. As reported by Statista (2023a), the first half of 2022 alone saw approximately 2.8 billion instances of malware attacks. Not only are these attacks increasing in number, but they are also becoming more sophisticated. With a daily detection rate of 560,000 new malware instances and a 62% year-over-year increase in the number of malware variants, malware has emerged as one of the most prominent cybersecurity threats (DataProt, 2022).

Malware stands out as an exceptionally effective attack vector employed by malicious actors, capable of delivering a wide range of payloads and accomplishing diverse objectives. According to CheckPoint's Cyber Security Report 2021 (Check Point Software, 2021), the categories of malware responsible for the highest volume of attacks on corporate networks in 2021 were:

- Botnets: Malware that infects devices and turns them into bots, controlled by a malicious actor to launch attacks on other targets. Bots join together to form a botnet, under the control of the same malicious actor Xing et al. (2021).
- Banking malware: Specifically designed to gain unauthorized access to banking information, which is then sent back to the malicious actor. Advanced versions of this malware can even conduct banking transactions without the target's knowledge or consent.
- Spyware (Infostealers): Malware aimed at collecting information about the infected device and its user(s). This information can include login credentials, financial details, keystrokes, and other private information.
- Cryptominers: Specialized malware that mines cryptocurrencies, consuming significant processing power and resulting in elevated electricity consumption.
- Mobile malware: Designed to infect mobile devices and operating systems, with Android and iOS experiencing a surge in malware infections due to increased reliance on mobile devices for daily activities.
- Ransomware: Malware that encrypts a user's data or locks their device, demanding a ransom fee for its release. These attacks have witnessed a rapid increase in recent years. Ransomware attacks have seen a rapid increase, with 304 million reported attacks in Ransomware (2022).

While detecting malware poses challenges, machine learning offers a promising solution (Alani, 2021). However, the task becomes even more complex when dealing with obfuscated malware. Obfuscated malware refers to a type of malware that modifies its source code to make

* Corresponding author.
E-mail address: m@alani.me (M.M. Alani).

https://doi.org/10.1016/j.cose.2023.103409
Received 5 November 2022; Received in revised form 22 July 2023; Accepted 24 July 2023
Available online 28 July 2023

Table 1
Summary of Obfuscation Techniques.

Technique	Description
Dead code insertion	Inserting useless code that changes the appearance of the program without affecting its functionality.
XOR	Applying XOR logical operation to encrypt data using a selected key, which can be reversed by XORing the obfuscated data with the key.
Register reassignment	Swapping the values stored in registers between generations while preserving the program's code and behavior.
Subroutine reordering	Rearranging the order of subroutines within the program to introduce obfuscation. This technique can create $n!$ possible arrangements if the program has n subroutines.
Instruction substitution	Substituting certain instructions with alternative instructions that perform the same task. This may involve using multiple instructions to accomplish the desired functionality.
Code transposition	Reordering blocks of instructions by introducing unconditional jumps or moving independent code blocks without impacting the program's overall functionality.
Base 64 encoding	Converting the code's encoding from ASCII to Base64 by grouping three adjacent characters into a 24-bit string, which is then divided into four 6-bit chunks representing Base64 characters.
Control-flow flattening	Disrupting the program's structured flow by enclosing program blocks within a loop controlled by a single switch statement (Sophos News, 2022).
Code integration	Decompiling the infected program, inserting the malware code between existing code blocks, and reassembling the injected code into a new variant with a distinct signature.
Packers	Encrypting malware code and encapsulating it within a packing application, resulting in a new executable file. The packer decrypts the malware code only when loaded into memory (Mohanta and Saldanha, 2020).
API hashing	Generating hashes of the APIs used by the malware to obscure these function calls from static analysis techniques.

it unrecognizable to humans, enabling it to evade traditional detection methods. Signature-based detection relies on a database of known malware signatures, which can include partial code scripts, hashes, or specific indicators extracted from the malware to facilitate detection. However, obfuscation has become a favored technique among malicious actors to evade detection, making the detection process more difficult.

Obfuscation techniques employed in malware vary in their levels of sophistication, ranging from basic methods like dead code insertion to more advanced techniques such as packing (Monnappa, 2018). Table 1 provides a list of some of these obfuscation techniques.

The prevalence of obfuscated malware utilized by malicious actors is increasing. According to Katz (2022), over 25% of JavaScript-based malware is obfuscated. This alarming malware obfuscation trend indicates the urgent need for efficient and highly accurate detection techniques. In light of the difficulty detecting obfuscated malware faced by signature-based malware detection solutions, machine learning has emerged as a reliable approach for obfuscated-malware detection. Machine learning-based solutions generally outperform signature-based methods in detecting "never seen before" malware (Qiu et al., 2020; Singh and Singh, 2021; Alani and Awad, 2022; Aslan and Samet, 2020).

According to Cisco's cyber security threat trends report (Cisco Umbrella, 2022), certain malware families have the potential to spread rapidly if not detected promptly. This underscores the need for fast and lightweight detection systems that can quickly identify malware with minimal information.

One challenge that machine learning–based malware detection solutions face is adversarial attacks that are intentionally designed to evade detection (Maiorca et al., 2019). According to Grosse et al. (2017), most adversarial attacks target the static features extracted from malware samples without executing them. This suggests that incorporating dynamic analysis features can enhance the accuracy of detection (Suciu et al., 2019).

Another challenge faced by machine learning–based solutions is the acceptance and trust in these solutions within the cybersecurity community. This concern has been a driving force behind our research, as we aim to develop an explainable and efficient machine learning–based solution to address this issue. According to a survey published in Lemos (2022), 46% of cybersecurity professionals thought that artificial intelligence (AI)-based cybersecurity solutions generate excessive noise, making it difficult to understand and sift through the results. The same survey found that only 11% of cybersecurity professionals reported having no challenges with their organization's machine learning–based cybersecurity solutions. The lack of interoperability is identified as a key reason behind this skepticism. Cybersecurity professionals are less confident in systems that operate as "black boxes" without clear explanations of their decision-making processes. Hence, our work focuses on providing an explainable model where predictions are based on interpretable conditions, aiming to enhance the transparency and trustworthiness of machine learning–based cybersecurity solutions.

Other works have investigated the application of explainable machine learning in various areas of cybersecurity. Khoulimi et al. conducted an overview of explainable AI for cybersecurity, with a particular emphasis on intrusion detection, as discussed in Khoulimi et al. (2022). Giudici and Raffinetti (2022) focused on the application of explainable machine learning in cybersecurity risk management, presenting an explainable AI model that incorporates Shapley values and statistical normalization based on Lorenz Zonoids, especially suitable for assessing cyber risk with ordinal measurement variables. Kabir et al. (2022) explored the use of explainable artificial intelligence in addressing cybersecurity challenges in smart cities. Their work focuses on the transition from black-box models to white-box models using explainable AI techniques.

1.1. *Research contribution*

This research makes the following key contributions:

1. Introduction of a machine learning–based system for detecting obfuscated Windows malware with a high accuracy rate exceeding 99%.
2. Implementation of Recursive Feature Elimination (RFE) for feature selection, enabling the identification of a minimal set of highly effective features without compromising the overall accuracy of the detection system. This approach reduces the number of features acquired and used for classification, streamlining the data acquisition process in real-life deployments.
3. Provision of detailed model explainability, which enhances the trustworthiness of the system. By providing clear and interpretable conditions for classification decisions, the model's accuracy is derived from transparent and understandable factors rather than operating as a black-box system.
4. Creation of a reduced version of the dataset specifically designed for obfuscated malware detection research.

1.2. *Structure of paper*

The rest of this paper is organized as follows. The next section reviews and summarizes previous research conducted to address the problem of obfuscated malware. Section 3 provides an overview of the proposed system. In Section 4, we introduce the dataset used in our experiments. Section 5 explains the experimentation environment, the performance metrics used in the experiments, the experiment design, and the results. The explainability aspect of the model is presented in Section 6. A comprehensive discussion and comparative analysis of the results obtained in this research are presented in Section 7. The final section of the paper provides concluding remarks and outlines potential directions for future work.

2. Related work

As our proposed system is based on machine learning, we review related works that have utilized machine learning–based solutions to address the challenge of obfuscated-malware detection. Bacci et al. (2018) investigated the influence of obfuscation on machine learning–based malware detection systems within the Android ecosystem. They put forth a mitigation proposal, primarily focused on the Android operating system. While their study focused on the Android operating system, the authors shed light on the potential hindrances posed by obfuscation to machine learning–based malware detection systems. This highlights the necessity for more robust detection systems to effectively combat the escalating use of obfuscated malware by malicious actors.

In 2018, Ali and Soomro proposed an obfuscated-malware detection system that employed particle swarm intelligence optimization (PSO) for feature selection and a random forest (RF) classifier (Ali and Soomro, 2018). The proposed system extracts features from packed and non-packed portable executable (PE) files. The main contribution of their research lies in the use of the PSO approach for feature selection. This approach prioritizes the removal of redundant, irrelevant, and noisy features from the extracted set. It is worth noting that the dataset used in their experiments was initially introduced in 2008, although the research itself was published in 2018. Consequently, the dataset did not encompass samples from the most recent malware families. When tested, the proposed system achieved an accuracy of 0.9960 by leveraging packed PE features.

In 2019, Li et al. introduced a machine learning–based malware detection system called Obfusifier (Li et al., 2019). This system was specifically designed to maintain high accuracy in detecting obfuscated malware by utilizing obfuscation-resistant features extracted from unobfuscated applications and represented in method graphs. The method graphs were used to capture the calling relationships between different methods and subroutines in the malware structure. Experimental testing of the proposed system demonstrated an accuracy of 0.95 in detecting obfuscated malware when employing an RF classifier.

In 2020, Park et al. introduced an obfuscated-malware detection system specifically designed for low-power devices, such as Internet-of-Things devices (Park et al., 2020). The proposed method used the extraction of features from Markov matrices constructed from opcode traces. These features were then used to train a machine learning–based classifier. Experimental testing demonstrated that the proposed system achieved an accuracy of 0.926 in detecting non-obfuscated malware and an accuracy of 0.9318 in detecting obfuscated malware. However, the proposed system was proved to consume less power when compared with other methods.

In 2020, Jahromi et al. introduced an improved stacked long-term short-memory (LSTM) based method for malware detection in safety- and time-critical systems (Jahromi et al., 2020). The proposed method presented an enhanced version of stacked LSTM that surpassed the performance of the original stacked LSTM in terms of accuracy and efficiency. The system was evaluated on multiple datasets, including various Windows malware datasets, and achieved accuracy levels ranging from 80% to 100%. However, it is important to note that the Windows malware datasets used in the study had a relatively small number of samples compared to the dataset utilized in the current research. Furthermore, these datasets did not specifically address the detection of obfuscated malware.

Darabian et al. presented, in 2020, a deep learning–based approach for detecting malware that combined static and dynamic features (Darabian et al., 2020). While the study specifically focused on detecting cryptomining malware, the approach of integrating static and dynamic features showed promise for malware detection in general. The proposed system achieved a high accuracy rate of 99% using various types of deep neural networks (DNNs).

In 2021, Darem et al. introduced a semi-supervised approach to detect obfuscated malware using deep learning (Darem et al., 2021). The proposed method used image representations of extracted features and a convolutional neural network classifier. The authors treated the problem as a semi-supervised learning task, where labels were generated for training files and predicted for testing files. The method focused on utilizing OpCode as the primary identifier of malware behavior. Experimental testing demonstrated that the proposed approach achieved an accuracy of 0.9912. Notably, the research created its own dataset specifically for this study, instead of relying on publicly available datasets from previous works.

In 2021, Demetrio et al. conducted an experimental evaluation focusing on evasion techniques employed by attackers to bypass machine learning–based malware detection systems that utilize static analysis features (Demetrio et al., 2021). The study examined features extracted from PE files through static analysis, without executing the actual executables, which are typically relied upon by detection systems. The authors discussed evasion techniques employed by attackers in this context. Furthermore, the research introduced three novel attacks named Full DOS, Extend, and Shift, which involved practical manipulations that preserve the functionality of the malware. These attacks specifically targeted machine learning–based malware detection systems, aiming to evade their detection capabilities.

In 2022, Kim et al. introduced a hybrid deep generative model designed to detect malware variants by leveraging both global and local features (Kim and Cho, 2022). The proposed model used image representations of malware to capture global features, while local features were extracted from binary code sequences. These features were then used to train a DNN for the purpose of malware detection. The impact of the proposed method was evaluated using class activation maps. For experimentation, the Kaggle Microsoft Malware Classification Challenge dataset from 2015 was employed, indicating that it does not encompass the most recent malware families. Testing of the proposed model demonstrated an accuracy of 0.9747.

In 2022, Carrier et al. introduced an ensemble machine learning–based system for detecting obfuscated malware (Carrier et al., 2022). The system utilized features extracted from memory dumps that contained obfuscated malware. During testing, the proposed system achieved an accuracy of 0.99 by employing an ensemble approach with naive Bayes (NB), RF, and decision tree (DT) as base learners, and logistic regression (LR) as meta learners. The publication also introduced the dataset used in our research.

Ratcliffe et al. introduced another machine learning–based detection system for obfuscated malware in Ratcliffe et al. (2022). The authors tested multiple classifiers and created their own dataset using a tool named VolMemLyzer, which is the same tool used to extract data from the dataset used in our research.

Table 2 provides a concise overview of the related works that were reviewed and compared to our proposed system.

3. Overview of proposed system

3.1. Design goals

The proposed Windows malware detection system was designed to meet the following goals:

1. High accuracy
   Given the rising number of obfuscated-malware attacks, the system aimed to achieve a high level of accuracy in detecting such malware.

Table 2
Summary of Related Works.

Research	Method	Classifier	Accuracy	Obfuscation	Explainability
Ali and Soomro Ali and Soomro (2018)	Static features	RF	0.9960	✓	×
Li et al. Li et al. (2019)	Method graph	RF	0.95	✓	×
Park et al. Park et al. (2020)	Markov matrices	XGB-Tree	0.9318	✓	×
Darem et al. Darem et al. (2021)	Image representation	CNN	0.9912	✓	×
Kim et al. Kim and Cho (2022)	Image representation	CNN+LSTM	0.9747	✓	×
Carrier et al. Carrier et al. (2022)	Memory dump features	Ensemble	0.99	✓	×
Hamid et al. Darabian et al. (2020)	Static features	LSTM	0.99	×	×
Jahromi et al. Jahromi et al. (2020)	Static features	En Stck LSTM	0.8851	×	×
Ratcliffe et al. Ratcliffe et al. (2022)	Memory dump features	MLP	0.9997	mixed	×
Proposed work	Memory dump features	XGB	0.9989	✓	✓

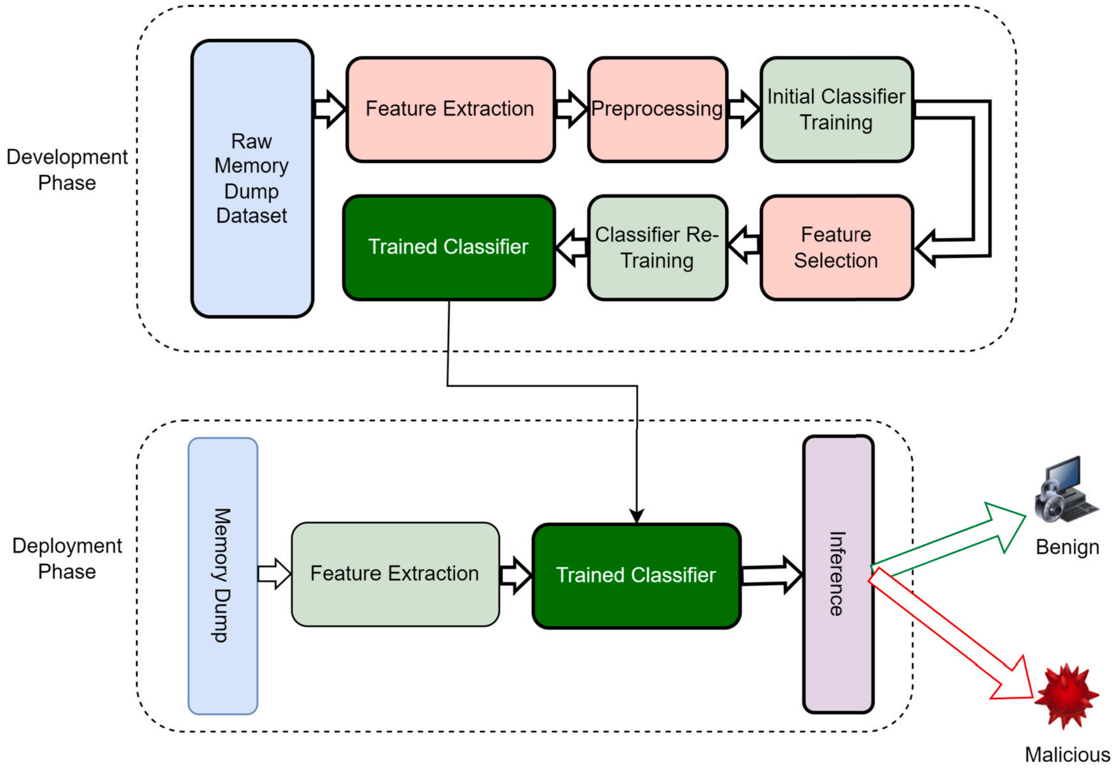

Fig. 1. Overview of Proposed System.

2. Lightweight
   The system was designed to be lightweight, minimizing resource consumption and processing requirements. By reducing the number of features needed for detection, it aimed to optimize performance and efficiency.
3. Explainability
   The selected features and detection mechanisms of the system were designed to be explainable. This means that the system's decisions and classifications were based on transparent and interpretable conditions.

3.2. System description

The proposed malware detector, XMal, is designed to detect obfuscated malware based on features extracted from memory dumps. It is intended to address the limitations of classical signature-based detection systems, which often struggle to identify obfuscated malware. Fig. 1 provides an overview of the proposed system.

As shown in Fig. 1, the proposed system comprises two main phases: the development phase and the deployment phase. In the development phase, the system processes the raw memory dumps through the feature extraction unit to extract a comprehensive set of features, resulting in the initial version of the dataset. Specifically, 56 features are extracted from each memory dump, and these features are extracted to form a dataset consisting of 58,596 samples. Subsequently, the preprocessed dataset undergoes further preprocessing steps to prepare it for initial classifier training. The subsequent stage involves training and testing a pipeline of diverse classifiers to identify the best-performing classifier. This classifier is then utilized in the feature selection process to select the most effective features and generate a reduced dataset with a smaller feature set. More comprehensive information regarding the feature selection process is provided in Section 5. During the next stage, the dataset containing the selected features is utilized to train the final classifier that will be deployed during the deployment phase. Following training and testing, the classifier is stored and prepared for deployment.

During the deployment phase, the system captures a memory image, which is then processed through the feature extraction unit using the VolMemLyzer tool (Ahlashkari, 2022). The tool extracts the limited number of features that were selected during the development phase. These extracted feature values are subsequently fed into the pre-trained classifier, which generates a prediction indicating whether the memory dump is infected with malware or not.

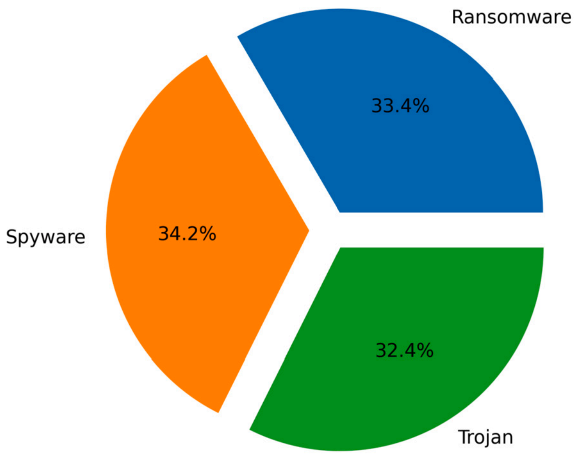

Fig. 2. Distribution of Malware Categories in the MalMem Dataset.

4. Dataset

The dataset used in our experiments was introduced in Carrier et al. (2022). It comprises a collection of features extracted from memory captures containing both malware and benign samples. The dataset includes 58,596 instances, with an equal distribution of 29,298 benign samples and 29,298 malware-infected samples. In the dataset, each instance corresponds to the features extracted from a single memory dump file. Each data record consists of 55 features extracted from a particular memory capture file. Additionally, an extra feature was included to provide information about the specific malware category and family, as annotated by the dataset authors. The tool used to extract these features was VolMemLyzer (Ahlashkari, 2022), which uses the Volatility Framework (The Volatility Foundation, 2022) to extract the 55 features from each memory image. These features are obtained by analyzing the memory image and capturing various information, including the number of running processes, average number of threads per process, number of open files, number of open dynamic-link libraries (DLLs), etc. For a comprehensive list of these features, please refer to The Volatility Foundation (2022); Ahlashkari (2022).

As described in Carrier et al. (2022), the dataset included malicious samples that were categorized into three types: trojans, spyware, and ransomware. Fig. 2 provides an overview of the distribution of these malware categories within the dataset, displaying the respective percentages for each category. Within each of the malware categories, there were several specific malware families present in the dataset. Examples of these families include Zeus, Emotet, Gator, Transponder, Conti, MAZE, and Shade. Fig. 3 provides a breakdown of the percentage distribution for each malware family within the dataset. The figures indicate that the different categories and families are adequately represented, without significant discrepancies that could introduce bias into the detection model towards any particular malware family. The analyzed malware families in the dataset exhibited varying levels of obfuscation techniques. Some older malware families like Shade and Gator employed simpler obfuscation techniques such as subroutine reordering and dead code insertion. On the other hand, more advanced and prominent malware families like Emotet and Zeus utilized multiple complex obfuscation techniques. Emotet, known for its significant impact on global botnet attacks in 2021 (Statista, 2023b), employed advanced obfuscation techniques such as API hashing and control-flow flattening. Similarly, Zeus, another prominent malware family (Statista, 2023b), employed a combination of packers, API hashing, and code transposition. These combinations of obfuscation techniques made the static analysis of the malware highly challenging and complex (Nayyar and Bueno, 2010). Considering the complexity and effectiveness of these obfuscation techniques, it becomes apparent that solely relying on

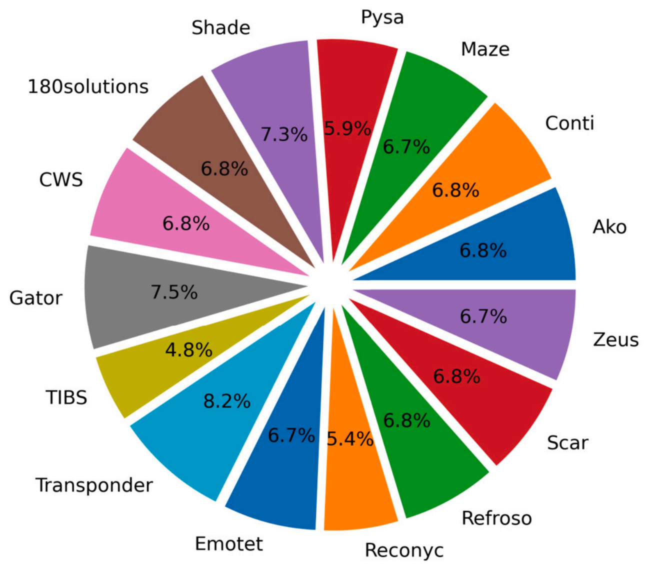

Fig. 3. Distribution of Malware Families in MalMem Dataset.

static analysis of malware code may not be sufficient for the effective detection of obfuscated malware (Or-Meir et al., 2019).

As the focus of our research is on the detection of obfuscated malware, we removed the specific malware family label leaving the dataset with "benign" and "malware" classes.

After conducting a thorough analysis of the dataset, we discovered that it exhibited a balanced distribution. Fig. 2 demonstrated that the dataset contained a relatively equal representation of the three types of malware, with no significant discrepancies in their proportions. Similarly, Fig. 3 revealed a balanced distribution of various malware families, ensuring that no specific family was over-represented compared to others. Furthermore, our examination confirmed that the dataset was complete, containing no missing data. This indicated that the dataset was well-prepared for the training phase and did not require any additional preprocessing steps.

5. Experiments and results

In this section, we will provide a comprehensive explanation of the experiments that were conducted as part of our research, along with their corresponding results.

5.1. Experimentation environment

The experimentation environment utilized for our research consisted of the following hardware and software specifications:

- Processor: AMD Ryzen 5 3600 4.2GHz
- RAM: 128GB
- GPU: Nvidia RTX 3060Ti
- Operating System: Windows 10 Professional
- Python: v3.8.5 (Python, 2022)
- Sci-Kit Learn: v1.0.1 (Scikit-Learn, 2022)
- Matplotlib: v3.3.4 (Matplotlib, 2022)

5.2. Performance metrics

To evaluate the performance of the binary classifiers in our experiments, we employed four fundamental performance metrics: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). These four measures, when combined together, generate the confusion matrix. These metrics are utilized to construct the confusion matrix, which provides an overview of the classifier's performance.

Additionally, these metrics can be used to derive comprehensive performance measures, including:

1. Accuracy: This measures the ratio of correct predictions using the equation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

2. Precision: This measures the ratio of the accuracy of positive predictions using the equation:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

3. Recall: This measures the ratio of positive instances that are correctly detected by the classifier using the equation:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

4. F_1 Score: This measures the harmonic mean of precision and recall using the equation:

$$F_1 Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

5.3. *Experiment design*

The experiments conducted in this research can be divided into the following steps:

1. Initial training and testing: The dataset is initially split randomly, with proper stratification, into a 66.66% training subset and a 33.33% testing subset. A pipeline of five classifiers is trained and tested to determine the best-performing classifier among them.
2. Feature selection: The goal of this step is to select the minimum number of effective features in order to improve system efficiency and create a lightweight system. The RFE algorithm based on feature importance is used for feature selection. Algorithm 1 outlines the steps of the RFE algorithm, which involves iteratively eliminating the feature with the least importance, retraining the model, and recalculating feature importance until a desired number of features is reached while maintaining a minimum compromise in accuracy (Guyon et al., 2002).
3. Post-selection training and testing: The dataset with the selected features is used to retrain and retest the five classifiers, assessing the impact of the feature selection process on the performance metrics.
4. Validation: The model undergoes 10-fold cross-validation in this step. Algorithm 2 describes the steps involved in 10-fold cross-validation. The dataset is randomly divided into 10 folds, with each fold used for testing once while the remaining 9 folds are used for training. If the performance metrics obtained from the 10 folds are consistent and exhibit a small standard deviation, it indicates that the classifier is generalizing well and is not suffering from overfitting.
5. Model's explainability: In this step, the model undergoes an explainability analysis to ensure that the model's predictions are not based on black-box operations. Further details about the explainability technique employed are provided in Section 6.

5.4. *Initial results*

As described in Section 5.3, the first stage of the experimentation involved the creation and evaluation of five different classifiers to identify the best-performing one. The chosen classifier algorithms chosen for the initial experiments were:

1. RF
2. LR
3. DT
4. Gaussian Naive Bayes (GNB)
5. Extreme gradient boost (XGB)

Algorithm 1: Recursive Feature Elimination Using Feature Importance.

```
Input: Dataset with n features
Output: Dataset with m features
$Array \leftarrow Dataset$
$model \leftarrow Classifier$
$TargetFeatures \leftarrow m$
while $Features(Dataset) > TargetFeatures$ do
    $RandomSplit(Array) \rightarrow Train\_Array, Test\_Array$
    train $model$ with $Train\_Array$
    $importance \leftarrow FeatureImportance(model)$
    $i \leftarrow$ index of feature with lowest importance
    $Array.DeleteFeature(i)$
end
$Dataset \leftarrow Array$
```

Algorithm 2: 10-Fold Cross-Validation.

```
Input: Dataset
Output: testing results for 10 runs; $results$
$Array \leftarrow Dataset$
Split $Array$ into 10-folds randomly to produce $fold(1to10)$
$model = RandomForestClassifier$
for $i = 1to10$ do
    $TestingData = fold(i)$
    $TrainingData = Null$
    for $j = 1to10$ do
        if $j <> i$ then
            $TrainingData = TrainingData.Append(fold(j))$
        end
    end
    train $model(TrainData)$
    test $model(TestData)$ to produce $results(i)$
end
print $results$
```

These five algorithms were chosen over deep learning methods due to the computationally intensive nature of deep learning compared to the listed algorithms, as indicated Raschka et al. (2022). As mentioned in Section 3, one of the main objectives of the proposed system is to create a lightweight system. Therefore, opting for less resource-intensive classification algorithms aligns with our design goals and facilitates their achievement.

The 55-feature dataset was randomly divided into a training subset comprising 66.66% of the data and a testing subset comprising 33.33% of the data, with stratification, to maintain the balance between malicious and benign samples. The performance metrics obtained from the testing phase are presented in Table 3.

As shown in the table, all five classifiers demonstrated high accuracy scores, surpassing 0.99. Among them, the RF classifier achieved slightly higher accuracy. However, when considering both accuracy and timing, the XGB classifier exhibited superior performance.

5.5. *Feature selection*

We decided to use RFE for feature selection instead of other statistical dimensionality reduction algorithms, such as principal component analysis (PCA), singular value decomposition (SVD), and linear discriminant analysis (LDA) for several reasons. Unlike dimensionality reduction algorithms such as PCA, SVD, and LDA, RFE has certain advantages in the context of feature selection. While the mentioned dimensionality reduction algorithms reduce the number of features used as input to the classifier, they still require capturing the same number of features at the data acquisition stage in real-life deployments. Additionally, these algorithms involve mathematically intensive operations during prepro-

Table 3
Performance Metrics for 55-Feature Dataset.

Metric	Accuracy	Precision	Recall	F_1 Score	Train Time (s)	Test Time (µs)
RF	0.998862	0.998865	0.998859	0.998862	1.242281	5.224347
LR	0.995656	0.995658	0.995652	0.995655	0.061013	0.651723
DT	0.998397	0.998397	0.998396	0.998397	0.023005	0.717035
GNB	0.995139	0.995118	0.995171	0.995138	0.005002	0.603433
XGB	0.998552	0.998556	0.998548	0.998552	0.518116	0.569026

Table 4
Selected Features.

Feature	Description
`svcscan.nservices`	Total number of services registered in a memory image
`dlllist.avg_dlls_per_proc`	Average number of DLLs called per process
`handles.nmutant`	Total number of mutant handles detected
`svcscan.kernel_drivers`	Total number of processes running in kernel mode that have an associated driver name
`svcscan.shared_process_services`	Total number of services that share one or more process with other services

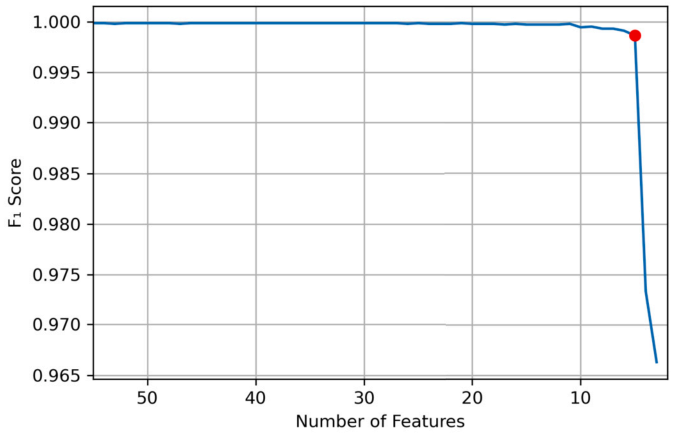

Fig. 4. Change in F_1 Score with Feature Reduction.

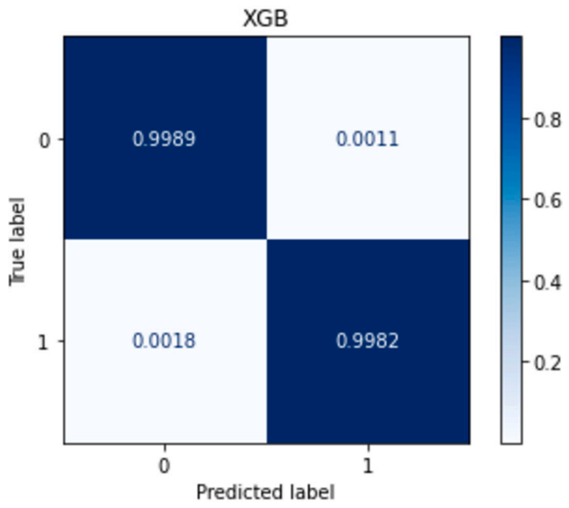

Fig. 5. Confusion Matrix Plot for XGB Classifier.

cessing to generate the new features at deployment. On the other hand, RFE not only reduces the number of features fed into the classifier but also reduces the number of features captured during data acquisition without the need for additional preprocessing. This results in improved efficiency and reduced processing time.

The steps of the RFE algorithm, as presented in Algorithm 1, were followed to perform feature selection. The algorithm successfully identified five features in the final version of the dataset as the most effective features in the detection process. The relationship between the number of features and the F_1 score is illustrated in Fig. 4. The threshold for selecting the number of features was determined to be five, as the model experienced significant performance degradation when the number of features fell below this threshold.

Table 4 presents the description of the five selected features. These features can be readily extracted from a memory image using the VolMemLyzer tool (Ahlashkari, 2022). The table provides a clear overview of the selected features and their corresponding descriptions.

5.6. Post-selection results

The resulting 5-feature dataset obtained after feature selection was further divided into a training subset (66.66%) and a testing subset (33.33%). The training subset was utilized to retrain the same set of five classifiers mentioned in Section 5.4. The performance metrics obtained from the subsequent tests conducted after training are presented in Table 5.

The table demonstrates that the performance of each classifier remained consistently high even with the reduction in the number of features from 55 to only 5. This indicates that the selected five features hold significant importance in the classification process. Notably, when considering both accuracy and timing parameters, XGB outperformed the other classifiers slightly in terms of timing, while maintaining a similar level of accuracy.

The testing time of the XGB classifier exhibited a significant improvement, as anticipated, after feature selection. The testing time for a single instance decreased from 0.569 µs with 55 features to 0.413 µs with the 5-feature predictions. This represents a 26% improvement in timing, further highlighting the practical advantage of the feature selection process.

Fig. 5 displays the confusion matrix plot of the XGB classifier when evaluated using the five-feature dataset. As shown in the figure, the FP rate was only 0.11%, while the FN rate was 0.18%.

5.7. Cross-validation

As a final step to validate the proposed model, we performed 10-fold cross-validation as described in Algorithm 2. Table 6 shows the results of this process.

As shown in the table, the mean value of the performance metrics obtained from the 10-fold cross-validation aligns closely with the earlier results presented in Section 5.6. Moreover, the low standard deviation

Table 5
Performance Metrics for 5-Feature Dataset.

Metric	Accuracy	Precision	Recall	F_1 Score	Train Time (s)	Test Time (μs)
RF	0.998914	0.998917	0.998910	0.998914	1.268830	5.224335
LR	0.995656	0.995658	0.995652	0.995655	0.061013	0.516630
DT	0.998397	0.998397	0.998396	0.998397	0.027005	0.551735
GNB	0.995139	0.995118	0.995171	0.995138	0.006000	0.510723
XGB	0.998552	0.998556	0.998548	0.998552	0.477106	0.413808

Table 6
Results of 10-Fold Cross-Validation for XGB Classifier.

Fold	Accuracy	Precision	Recall	F_1 Score
1	0.998635	0.999657	0.997604	0.998630
2	0.998123	0.997877	0.998230	0.998053
3	0.998976	0.999303	0.998608	0.998955
4	0.998123	0.998660	0.997657	0.998158
5	0.999317	0.999309	0.999309	0.999309
6	0.998294	0.999331	0.997329	0.998329
7	0.998976	0.998977	0.998977	0.998977
8	0.998805	0.999665	0.997991	0.998827
9	0.998635	0.999657	0.997607	0.998631
10	0.998976	0.998649	0.999324	0.998986
μ	0.998686	0.999108	0.998264	0.998686
σ	0.000382	0.000546	0.000709	0.000383

across the performance metrics signifies the robustness and reliability of the proposed model. These findings indicate that the model's performance extends beyond the specific training dataset, suggesting a strong generalization capability.

To prevent information leakage and ensure the integrity of the cross-validation process in the MalMem dataset, a specific strategy was employed to handle the potential issue of multiple memory dumps from the same malware family. The strategy involved ensuring that the 10-fold cross-validation splits were carefully designed to avoid any intersection of malware families within each fold. This precautionary step was implemented before initiating the first cycle of cross-validation, thereby mitigating any possibility of information leakage and maintaining the integrity of the evaluation process.

6. Model's explainability

Our research aimed to explain the factors that influence the prediction decisions made by the trained model. The goal was to enhance trust in the model by providing transparency in its operation. By gaining insights into the inner workings of the model and uncovering the conditions and factors that contribute to its high accuracy, we aimed to ensure that the model's performance is based on explainable and interpretable processes, rather than relying on opaque“black-box” operations.

To explain our proposed model, we utilize Shapley additive explanation (SHAP). SHAP is a model-agnostic method introduced in 2017 that aims to explain machine-learning models (Lundberg and Lee, 2017). SHAP provides explanations derived from game theory. The SHAP process involves quantifying the impact of each feature by comparing the model's performance with and without the presence of that feature in the data. This approach allows us to gain insights into the influence of individual features on the model's prediction outcomes. In our experiment, we employed the TreeExplainer as the explainer type.

Fig. 6 presents a summary plot of SHAP values for the five selected features, arranged in descending order based on their impact on the prediction decisions. The feature with the highest influence is positioned at the top, while the feature with the lowest influence is at the bottom.

The values displayed on the left side of the axis have a downward drag on the prediction value, resulting in it being closer to the “benign” category. Conversely, the values on the right side of the axis exert an upward push on the prediction value, bringing it closer to the “malware” category. In the plot, red dots correspond to high feature values, while blue dots correspond to low feature values. In the case of binary features, red dots represent a value of 1, while blue dots represent a value of 0.

As depicted in Fig. 6, the feature `svcscan.nservices` has the most significant impact on the prediction. This feature represents the total number of services detected in the memory image. The presence of red dots on the left side of the plot indicates that a high number of services is associated with a lower prediction value, indicating a higher likelihood of being classified as “benign.” On the other hand, the presence of blue dots indicates that a lower value of the registered service feature leads to an increase in the prediction value towards “malware.” This observation is consistent with the behavior patterns observed in recent malware. Instead of running as independent services, modern obfuscated-malware families often employ techniques such as process hijacking or fileless execution (Cybertriage, 2021). Process hijacking involves malicious code tricking other processes into running them or taking control of them. As a result, the number of processes running independently may be lower in the presence of malware. One common target for process hijacking in the Windows operating system is “svchost.exe” (Mohanta and Saldanha, 2020). Additionally, obfuscated-malware code may be “fileless,” meaning it is stored within registry entries or executed through scripting languages like PowerShell, rather than being present as a standalone file. This technique aims to evade detection by traditional file-based scanning approaches (Cybertriage, 2021).

The next two features that have a significant impact on the prediction are called `dlllist_avg_dlls_per_proc` and `handles.nmutant`. These features have a similar effect, where higher values bring the prediction closer to being classified as benign, while lower values indicate a higher likelihood of being classified as “malware.” Similar to `svcscan.nservices`, lower values of these two features are associated with a malware infection. When a computer is infected with malware, the average number of DLL files invoked by a process tends to decrease. This is because the malware hijacks normal processes, preventing them from functioning properly. However, the obfuscated malware itself may still call other DLLs, particularly those used in modifying the Windows registry. Nevertheless, on average, the malware uses fewer DLLs compared to a benign process. The same behavior of the malware that leads to lower values of `handles.nmutant` is observed when a computer is infected.

The fourth feature depicted in the figure is referred to as `svcscan.kernel_drivers`. The figure illustrates that there is a specific range of high values for this feature that tends to associate the prediction more closely with malware. Additionally, there is a distinct range of low values on the right side of the figure that also contributes to the prediction leaning towards malware. This suggests that this particular feature alone is not highly conclusive in supporting the decision and is more effective when considered in combination with other features. The reason for the feature's limited standalone effectiveness is that many newly developed obfuscated-malware families adopt a strategy of running their processes in kernel mode using a driver name, rather than operating in user mode with a recognizable EXE identifier. This approach allows malware to evade detection since most malware detection software focuses on identifying user mode processes with distinct EXE identifiers. Consequently, a slightly higher number of processes running in kernel mode would logically indicate a potential malware infection.

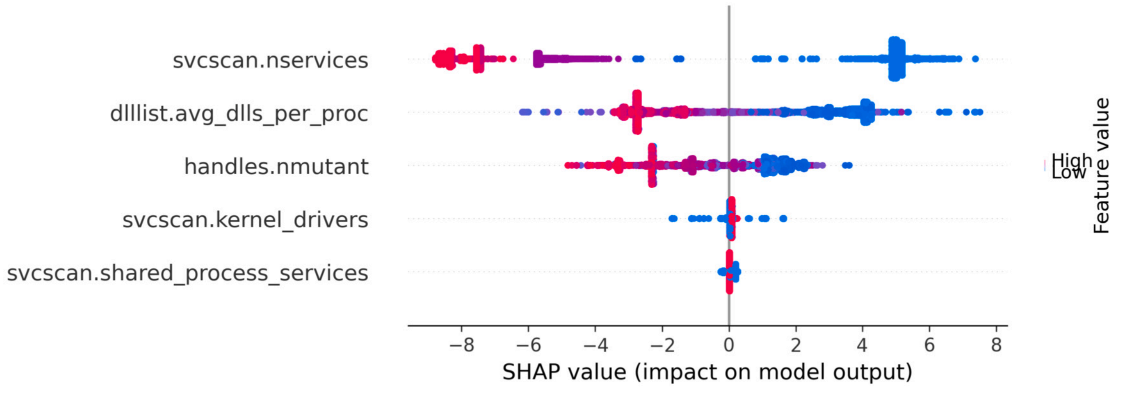

Fig. 6. SHAP Values Plot for Selected Features. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 7
Comparison of Proposed System with Related Works.

Research	Dataset	Instances	Balanced	Features	Classifier	Explainable	Accuracy	F_1 Score	Test Time
Ali and Soomro (2018)	Malfese2008	5,498	✓	9	RF	×	0.9960	0.9960	-
Darem et al. (2021)	Microsoft2015	10,868	×	4,358+ images	CNN	×	0.9912	—	17 s
Kim and Cho (2022)	Microsoft2015	10,868	×	4,358+ images	CNN + LSTM	×	0.9747	0.9229	—
Li et al. (2019)	Malgenome and VirusShare2017	45,112	✓	28	RF	×	0.95	0.95	35.06 s
Darabian et al. (2020)	VirusTotal 2018	3,000	✓	—	LSTM	×	0.99	0.98	—
Park et al. (2020)	VirusShare2020	5,415	✓	—	XGB-Tree	×	0.9318	—	—
Jahromi et al. (2020)	XVHaven	3,300	✓	256	En Stck LSTM	×	0.8851	—	0.0027 s
Carrier et al. (2022)	MalMem2022	58,596	✓	55	Ensemble	×	0.99	0.99	8 µs
Ratcliffe et al. (2022)	Ratcliffe et al. (2022)	6,656	×	55	MLP	×	0.9997	—	—
Proposed work	MalMem2022	58,596	✓	5	XGB	✓	0.9989	0.9985	0.413 µs

As indicated in Fig. 6, the feature with the lowest impact is `svcscan.shared_process_services`. This feature exhibits low values on both sides of the figure. However, these low values do not provide a definitive indication of whether the sample is benign or malware. On the other hand, high values of this feature suggest a higher likelihood of a malware infection. This observation aligns with the fact that most malware processes often interact with other processes, attempting to exploit or gain administrator access through them (Cybertriage, 2021). Although certain benign processes may exhibit similar behavior, it is not a characteristic shared by all benign processes.

7. Discussion

As explained in Section 3, our proposed system was designed with three primary goals in mind: high accuracy, lightweight implementation, and explainability. Firstly, in terms of high accuracy (discussed in Section 5.6), we carefully selected highly effective features and utilized them to train a classifier that demonstrated superior accuracy in malware detection. Secondly, to ensure a lightweight implementation, we employed RFE as detailed in Section 5.3. RFE helped us reduce the number of required features to just five. This not only streamlined the detection process but also minimized the number of features that need to be captured and extracted during system deployment. Lastly, to fulfill the goal of explainability, we employed SHAP as described in Section 6. SHAP values played a crucial role in providing a meaningful interpretation of the impact of different feature values on the model's predictions. This allowed us to gain a deeper understanding of the inner workings of the model, enhancing the system's overall explainability.

Table 7 provides a detailed comparison of the proposed system with related works. As observed in the table, other works have achieved high accuracy levels that are comparable to those achieved by our proposed system. However, when considering timing and efficiency measures, it becomes evident that the proposed system outperforms all of the related works, with an impressive detection time of 0.569 µs. This detection time is approximately $\frac{1}{15}$ of the time required by the closest competitor. The superior performance of our system can be attributed to the utilization of a highly effective set of five features, carefully selected to optimize both accuracy and efficiency.

In the landscape of related works, it is worth noting that some studies, such as Carrier et al. (2022) and Li et al. (2019), have utilized larger datasets similar to ours. However, many other related works have relied on significantly smaller datasets, which poses challenges in terms of generalizability, particularly for studies that employed highly imbalanced datasets like Microsoft2015. Moreover, when evaluating these smaller datasets, several works have used outdated samples that do not adequately represent the latest developments in obfuscated malware. Notable exceptions to this include Carrier et al. (2022), Ratcliffe et al. (2022), and Park et al. (2020). While Carrier et al. (2022) demonstrated high accuracy, it incurred a significantly higher processing time per sample due to the use of ensemble learning and a larger number of features. On the other hand, Ratcliffe et al. (2022) did not exclusively focus on obfuscated samples, and the work primarily revolved around classification rather than detection, leading to a noticeable class imbalance in the training samples. Hence, the achieved accuracy may not be directly comparable to our proposed system. Although Park et al. (2020) included more recent samples, the accuracy achieved in that study was noticeably lower than what our proposed system achieved.

As the comparison shows, none of the other works have presented explainability of their models, while our model was explained using SHAP values in Section 6.

The proposed model successfully attained its design objectives by delivering both high accuracy and efficiency, while also providing explainability. Rigorous testing, conducted through 10-fold cross-validation, consistently demonstrated the model's robust performance. This substantiates the model's ability to generalize effectively across different datasets and scenarios.

Although our proposed system has demonstrated high accuracy in detecting obfuscated malware, it is essential to acknowledge that malicious actors employ various evasion techniques beyond obfuscation. The use of such techniques to evade detection might impact our proposed system's accuracy. One such technique is opportunistic translation to WebAssembly in JavaScript malicious code obfuscation, as presented by Romano et al. (2022). This technique involves leveraging WebAssembly and JavaScript to obfuscate malicious code, adding another layer of complexity to detection efforts. Additionally, the technique known as Living-off-the-Land (LotL) has gained popularity among malicious actors. LotL involves utilizing legitimate binaries and tools already present in the system to carry out malicious activities. This technique is particularly prevalent in advanced persistent threats, as highlighted in Barr-Smith et al. (2021).

8. Conclusions and future directions

This paper introduced a lightweight obfuscated-malware detector based on explainable machine-learning techniques The system was evaluated using the MalMem2022 dataset and achieved a remarkable accuracy of over 99.8% by utilizing only five carefully selected features. The feature selection method, RFE, effectively reduced the number of features while maintaining high accuracy. This streamlined approach significantly enhanced the efficiency of the system. Additionally, the detection time of the proposed system was impressively fast, measuring 0.413 μs. To ensure transparency and interpretability, the proposed system was explained using SHAP values. The obtained explanations were found to be consistent with modern malware behavior. The utilization of SHAP values not only justified the high accuracy achieved by the classifier but also ensured that the system's performance was based on explainable reasoning rather than operating as a black-box approach.

An interesting future direction would involve exploring updated datasets that include both obfuscated and non-obfuscated samples. Another potential avenue for future research involves investigating the deployment of the system on low-resource devices and addressing the task of categorizing malware into specific families to facilitate forensic analysis.

CRediT authorship contribution statement

Mohammed M. Alani: Conceptualization, Data curation, Investigation, Methodology, Software, Writing – original draft. **Atefeh Mashatan:** Conceptualization, Methodology, Supervision, Writing – review & editing. **Ali Miri:** Conceptualization, Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgements

This research was funded by NSERC, Canada.

References

Ahlashkari, 2022. VolMemLyzer. https://github.com/ahlashkari/VolMemLyzer. (Accessed 14 June 2022).

Alani, M.M., 2021. Big data in cybersecurity: a survey of applications and future trends. J. Reliab. Intell. Environ. 7 (2), 85–114.

Alani, M.M., Awad, A.I., 2022. Paired: an explainable lightweight Android malware detection system. IEEE Access 10, 73214–73228. https://doi.org/10.1109/ACCESS.2022.3189645.

Ali, Z., Soomro, T.R., 2018. An efficient mining based approach using pso selection technique for analysis and detection of obfuscated malware. J. Inf. Assur. Cyber Secur. 2018, 1–13.

Aslan, O.A., Samet, R., 2020. A comprehensive review on malware detection approaches. IEEE Access 8, 6249–6271. https://doi.org/10.1109/ACCESS.2019.2963724.

A Not-So-Common Cold: Malware Statistics in 2022, 2022. https://dataprot.net/statistics/malware-statistics. (Accessed 7 June 2022).

Bacci, A., Bartoli, A., Martinelli, F., Medvet, E., Mercaldo, F., Visaggio, C.A., 2018. Impact of code obfuscation on Android malware detection based on static and dynamic analysis. In: ICISSP, pp. 379–385.

Barr-Smith, F., Ugarte-Pedrero, X., Graziano, M., Spolaor, R., Martinovic, I., 2021. Survivalism: systematic analysis of windows malware living-off-the-land. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 1557–1574.

Carrier, T., Victor, P., Tekeoglu, A., Lashkari, A., 2022. Detecting obfuscated malware using memory feature engineering. In: Proceedings of the 8th International Conference on Information Systems Security and Privacy - ICISSP, INSTICC. SciTePress, pp. 177–188.

Cisco Umbrella, 2022. Cybersecurity threat trends: phishing, crypto top the list. https://umbrella.cisco.com/info/2021-cyber-security-threat-trends-phishing-crypto-top-the-list. (Accessed 17 July 2022).

Cyber Security Report - Check Point Software, 2021. https://www.checkpoint.com/pages/cyber-security-report-2021. (Accessed 19 January 2023).

Darabian, H., Homayounoot, S., Dehghantanha, A., Hashemi, S., Karimipour, H., Parizi, R.M., Choo, K.-K.R., 2020. Detecting cryptomining malware: a deep learning approach for static and dynamic analysis. J. Grid Comput. 18 (2), 293–303.

Darem, A., Abawajy, J., Makkar, A., Alhashmi, A., Alanazi, S., 2021. Visualization and deep-learning-based malware variant detection using opcode-level features. Future Gener. Comput. Syst. 125, 314–323.

Demetrio, L., Coull, S.E., Biggio, B., Lagorio, G., Armando, A., Roli, F., 2021. Adversarial exemples: a survey and experimental evaluation of practical attacks on machine learning for windows malware detection. ACM Trans. Priv. Secur. 24 (4). https://doi.org/10.1145/3473039.

Giudici, P., Raffinetti, E., 2022. Explainable ai methods in cyber risk management. Qual. Reliab. Eng. Int. 38 (3), 1318–1326.

Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P., 2017. Adversarial examples for malware detection. In: Foley, S.N., Gollmann, D., Snekkenes, E. (Eds.), Computer Security – ESORICS 2017. Springer International Publishing, Cham, pp. 62–79.

Guyon, I., Weston, J., Barnhill, S., Vapnik, V., 2002. Gene selection for cancer classification using support vector machines. Mach. Learn. 46, 389–422.

How to Detect Running Malware - Intro to Incident Response Triage (Part 7), 2021. https://www.cybertriage.com/blog/training/how-to-detect-running-malware-intro-to-incident-response-triage-part-7. (Accessed 15 June 2022).

Jahromi, A.N., Hashemi, S., Dehghantanha, A., Parizi, R.M., Choo, K.-K.R., 2020. An enhanced stacked lstm method with no random initialization for malware threat hunting in safety and time-critical systems. IEEE Trans. Emerg. Top. Comput. Intell. 4 (5), 630–640. https://doi.org/10.1109/TETCI.2019.2910243.

Kabir, M.H., Hasan, K.F., Hasan, M.K., Ansari, K., 2022. Explainable Artificial Intelligence for Smart City Application: A Secure and Trusted Platform. Springer International Publishing, Cham, pp. 241–263.

Katz, O., 2022. Over 25% of Malicious JavaScript Is Being Obfuscated. Akamai Technologies. https://www.akamai.com/blog/security/over-25-percent-of-malicious-javascript-is-being-obfuscated.

Khoulimi, H., Lahby, M., Benammar, O., 2022. An Overview of Explainable Artificial Intelligence for Cyber Security. Springer International Publishing, Cham, pp. 31–58.

Kim, J.-Y., Cho, S.-B., 2022. Obfuscated malware detection using deep generative model based on global/local features. Comput. Secur. 112, 102501.

Lemos, R., 2022. AI for Cybersecurity Shimmers with Promise, but Challenges Abound. Dark Reading. https://www.darkreading.com/analytics/ai-cybersecurity-promise-challenges.

Li, Z., Sun, J., Yan, Q., Srisa-an, W., Tsutano, Y., 2019. Obfusifier: obfuscation-resistant Android malware detection system. In: International Conference on Security and Privacy in Communication Systems. Springer, pp. 214–234.

Lundberg, S.M., Lee, S.-I., 2017. A unified approach to interpreting model predictions. Adv. Neural Inf. Process. Syst. 30.

Maiorca, D., Biggio, B., Giacinto, G., 2019. Towards adversarial malware detection: lessons learned from pdf-based attacks. ACM Comput. Surv. 52 (4), 1–36.

Matplotlib — Visualization with Python, 2022. https://matplotlib.org. (Accessed 30 March 2022).

Mohanta, A., Saldanha, A., 2020. Malware Analysis and Detection Engineering: A Comprehensive Approach to Detect and Analyze Modern Malware. Apress.

Monnappa, K., 2018. Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware. Packt Publishing Ltd.

Nayyar, H., Bueno, P., 2010. Clash of the titans: Zeus v spyeye. SANS Institute InfoSec Reading Room.

Or-Meir, O., Nissim, N., Elovici, Y., Rokach, L., 2019. Dynamic malware analysis in the modern era—a state of the art survey. ACM Comput. Surv. 52 (5), 1–48.

Park, D., Powers, H., Prashker, B., Liu, L., Yener, B., 2020. Towards obfuscated malware detection for low powered iot devices. In: 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, pp. 1073–1080.

Welcome to Python.org, 2022. https://www.python.org. (Accessed 30 March 2022).

Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., Xiang, Y., 2020. A survey of Android malware detection with deep neural models. ACM Comput. Surv. 53 (6). https://doi.org/10.1145/3417978.

Topic: Ransomware, 2022. https://www.statista.com/topics/4136/ransomware. (Accessed 7 June 2022).

Raschka, S., Liu, Y., Mirjalili, V., 2022. Machine Learning with PyTorch and Scikit-Learn. Packt Publishing.

Ratcliffe, C., Bokolo, B.G., Oladimeji, D., Zhou, B., 2022. Detection of anti-forensics and malware applications in volatile memory acquisition. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. Springer, pp. 516–527.

Romano, A., Lehmann, D., Pradel, M., Wang, W., 2022. Wobfuscator: obfuscating javascript malware via opportunistic translation to webassembly. In: 2022 IEEE Symposium on Security and Privacy (SP), pp. 1574–1589.

Singh, J., Singh, J., 2021. A survey on machine learning-based malware detection in executable files. J. Syst. Archit. 112, 101861. https://doi.org/10.1016/j.sysarc.2020.101861. https://www.sciencedirect.com/science/article/pii/S1383762120301442.

Scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation, 2022. https://scikit-learn.org/stable. (Accessed 30 March 2022).

Attacking Emotet's Control Flow Flattening, 2022. https://news.sophos.com/en-us/2022/05/04/attacking-emotets-control-flow-flattening. (Accessed 20 January 2023).

Statista, 2023a. Number of malware attacks per year 2022. https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide. (Accessed 19 January 2023).

Statista, 2023b. Top botnets worldwide 2021. https://www.statista.com/statistics/1238993/top-botnets-worldwide. (Accessed 20 January 2023).

Suciu, O., Coull, S.E., Johns, J., 2019. Exploring adversarial examples in malware detection. In: 2019 IEEE Security and Privacy Workshops (SPW). IEEE, pp. 8–14.

The Volatility Foundation, 2022. Open source memory forensics. https://www.volatilityfoundation.org. (Accessed 17 June 2022).

Xing, Y., Shu, H., Zhao, H., Li, D., Guo, L., 2021. Survey on botnet detection techniques: classification, methods, and evaluation. Math. Probl. Eng. 2021.